NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2025-31

August, 2025

Time Series Foundation Models and Their Applications to Scientific Discoveries

Weijian Li

Abstract

The advent of large foundation models like ChatGPT and the concept of artificial general intelligence have shifted the machine learning paradigm from "one model per task" to "one large model, many tasks." On one hand, the prevalent LLM-based foundation models excel at many tasks that can be described by natural language, programming language, and mathematical language. On the other hand, they still struggle with tasks of other modalities, such as DNA data and continuous time-series data. This limitation has led researchers to pursue specialized foundation models for other data domains by transferring the techniques from LLM-based foundation models. This thesis is to look at the concept of foundation models with a focus on the track with less attention from the research community, which is about time series foundation models and their applications to scientific discoveries. Initial attempts of time series foundation models (TSFMs) have demonstrated superior efficacy across various benchmark datasets, outperforming traditional one-model-per-task approaches in forecasting tasks. While these models are pre-trained on hundreds of gigabytes of multi-domain time series data and achieve good performance on benchmarks, there still exist limitations and challenges to developing a better TSFM with more comprehensive abilities. Besides, their potential contribution to scientific discoveries remains largely unexplored. Particularly, questions persist regarding their ability to handle irregularly sampled scientific time series and their effectiveness in domain-specific downstream tasks such as variable star classification in astrophysics.

This thesis looks at the realm of time series foundation models from three perspectives. i) Starting from the oracle of foundation models, LLM-based foundation models, I study the adaptive batch size technique to improve the pre-training efficiency for large models. ii) I study better methodologies for each step in the time series foundation models' pipeline that contribute to developing a better time series foundation model. iii) I study time series foundation models' applications to accelerate scientific discoveries in astrophysics.

Keywords

time series foundation models, scientific application, astrophysics, large language models, pre-train, fine-tune, zero-shot

NORTHWESTERN UNIVERSITY

Time Series Foundation Models and Their Applications to Scientific Discoveries

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Weijian Li

EVANSTON, ILLINOIS

August 2025

Dissertation Committee

Professor Han Liu (Chair),

Department of Computer Science, Northwestern University

Department of Statistics and Data Science, Northwestern University

Professor Yan Chen,

Department of Computer Science, Northwestern University

Professor Zhaoran Wang,

Department of Industrial Engineering & Management Sciences

Department of Computer Science, Northwestern University

Professor Manling Li,

Department of Computer Science, Northwestern University

Statement of Funding Source

During his Ph.D. years (2020 Winter - 2025 Winter), Weijian Li is funded by Teaching assistantships (TA), Research assistantships (RA), and SkAI Ph.D. Fellowship.

ABSTRACT

Time Series Foundation Models and Their Applications to Scientific Discoveries

Weijian Li

The advent of large foundation models like ChatGPT and the concept of artificial general intelligence have shifted the machine learning paradigm from “one model per task” to “one large model, many tasks.” On one hand, the prevalent LLM-based foundation models excel at many tasks that can be described by natural language, programming language, and mathematical language. On the other hand, they still struggle with tasks of other modalities, such as DNA data and continuous time-series data. This limitation has led researchers to pursue specialized foundation models for other data domains by transferring the techniques from LLM-based foundation models. This thesis is to look at the concept of foundation models with a focus on the track with less attention from the research community, which is about time series foundation models and their applications to scientific discoveries. Initial attempts of time series foundation models (TSFMs) have demonstrated superior efficacy across various benchmark datasets, outperforming traditional one-model-per-task approaches in forecasting tasks. While these models are pre-trained on hundreds of gigabytes of multi-domain time series data and achieve good performance on benchmarks, there still exist limitations and challenges to developing a better TSFM with more comprehensive abilities. Besides, their potential contribution to scientific discoveries remains largely unexplored. Particularly, questions persist regarding their ability to handle irregularly sampled scientific time series and their effectiveness in domain-specific downstream tasks such as variable star classification in astrophysics.

This thesis looks at the realm of time series foundation models from three perspectives. i) Starting from the oracle of foundation models, LLM-based foundation models, I study the adaptive batch size technique to improve the pre-training efficiency for large models. ii) I study better methodologies for each step in the time series foundation models' pipeline that contribute to developing a better time series foundation model. iii) I study time series foundation models' applications to accelerate scientific discoveries in astrophysics.

Acknowledgements

All of the work presented in this dissertation would not have been possible without the support of many people. I would like to use this chapter to express my sincere gratitude to them.

I would like to express my utmost gratitude to my advisor, Prof. Han Liu. Over the past five years working with him, I had the privilege of learning closely from his example. His passion and dedication to research have always been a profound source of inspiration for me. I still remember our first meeting: as we walked down the hallway of the Mudd Building, he told me that no matter how difficult or uncertain a problem is, if we remain focused and dedicated, we will achieve something meaningful and gain valuable insights along the way. Entering the PhD program without much research experience, I benefited tremendously from Prof. Liu's hands-on mentoring. I learned how to write high-quality academic papers, how to make effective presentations to communicate with the community, and how to identify valuable research problems. Most importantly, I learned how to work with resilience and effectiveness in navigating the uncertainties and ups and downs of the pursuit of scientific understanding. These lessons will continue to guide me long after graduation.

I am also grateful to my dissertation committee, Prof. Han Liu (Chair), Prof. Yan Chen, Prof. Zhaoran Wang, and Prof. Manling Li, for their time, guidance, and insightful feedback, which greatly strengthened this work.

I am also thankful to all the collaborators I have worked with during these years. I would like to thank Prof. Adam Andrew Miller from the Department of Physics and Astronomy, our collaboration extended my research on time series foundation models into a new scientific domain, which was both meaningful and exciting. I also thank Dr. Wei Yu for the guidance in our research on high-performance computing and data processing frameworks. My thanks also go to Prof. Yan

Chen, Prof. Zhaoran Wang, Prof. Wan Zhang, Mr. Qinjie Lin, Dr. Tim Tsz-Kit Lau, Dr. Zhihan Zhou, Mr. Jerry Yao-Chieh Hu, Mr. Dennis Wu, and Mr. Robin Luo. Our collaborations have led to many meaningful research outcomes and publications.

To my dear friends who have shared the highs and lows of this PhD journey, thank you for your support and companionship: Tim Tsz-Kit Lau, Zhihan Zhou, Liqian Ma, Guanlin Liu, Peirui Yang, Qinjie Lin, Yukun Ma, Guo Ye, Jiayi Wang, Hongyu Chen, Jerry Yao-Chieh Hu, Weimin Wu, Chenwei Xu, Yiqi Liu, Yuchen Wang, Chenjian Lin, Ning Wang, Peiyao Zhu, Baihong Zeng, Ruochen Jiao, Weixin Fan, Tingting Tang, Lu Liu, Zhiyuan Huang, Xinyi Kuang, Jianyang Chen, Haozheng Luo, Mattson Thieme, Ammar Gilani, Dennis Wu and many others. We have shared so many unforgettable moments together.

I am deeply grateful to Jingya Xun for standing by me through the challenges and triumphs of this journey, and for her love, kindness, inspiration, and encouragement. The little bottle of ginger shot will always shine in my memory. I also thank my two beloved golden doodles, Bailey and Porky, whose companionship brought joy and comfort throughout my doctoral years. I am also grateful to Mr. Xun and Ms. Wang for their kindness, hospitality, and the valuable lessons I learned from them over the years.

Last but not least, my deepest gratitude goes to my parents, Guohui Li and Xiaohong Zeng. Their unconditional love, support, and encouragement have given me the curiosity and courage to explore the world and chase my dreams. I remember starting first grade, afraid to ask questions in class, my mom encouraged me to raise my hand confidently, and my dad taught me to be independent from a young age. Their guidance has profoundly shaped my life. I am also deeply grateful to my grandparents, my great-grandmother, my wonderful cousins, and my uncles and aunts.

This adventure began long before my PhD, in the moment I first stepped off a bus far from my hometown, breathing in the different breeze and feeling the thrill of a new beginning. From that day to today, many people have supported and inspired me along the way. To all of you, including those whose names are not written here, I am deeply grateful. Thank you all.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1. Overview

The emergence of ChatGPT (OpenAI, 2022; Achiam et al., 2023; Floridi and Chiriatti, 2020) and other comparable models such as Claude fundamentally transformed the machine learning landscape, introducing the concept of large foundation models and artificial general intelligence (AGI). These models, built upon large language model architecture, support inputs and outputs in the form of natural language, programming language, and mathematics language. Pre-trained through next-token prediction on massive datasets encompassing natural language, programming code, and mathematical formulae, these models can solve a diverse array of tasks expressed in the supported languages, from writing essays and research reports to generating computer programs and constructing mathematical proofs. This demonstration of emergent capabilities and signs of general intelligence earned them the designation ”foundation models” (Bommasani et al., 2021). These LLM-based foundation models represent a paradigm shift from the traditional one-model-one-task approach to a revolutionary one-large-model-all-tasks framework.

On one hand, researchers keep working on the field of LLM-based foundation models from the aspects pre-training and fine-tuning efficiency, benchmark performance, inference efficiency, etc. On the other hand, despite their versatility, LLM-based foundation models exhibit limitations when processing data types of other modalities such as DNA/RNA sequences and time series. This has prompted researchers across scientific domains to develop specialized foundation models. In biology, for example, researchers have created DNA foundation models (Ji et al., 2021; Zhou et al., 2023, 2024) that learn underlying biological patterns from vast DNA datasets across species to support various downstream genomic tasks.

Time series prediction represents another extensively studied field with universal applications across domains including meteorology, finance, public administration, business, and numerous scientific disciplines. We define time series data as any real-valued data collected in temporal sequence. However, LLM-based foundation models struggle with effectively modeling real-valued time series (Tan et al., 2024) due to their tokenization mechanisms and the absence of time series data in their training datasets.

Drawing inspiration from the foundation model concept, researchers have begun developing specialized time series foundation models (Liu et al., 2024a; Woo et al., 2024b; Ansari et al., 2024a). The rationale behind these specialized models is threefold: (1) although time series from different domains are generated by distinct underlying processes, they share universal modeling characteristics such as trend patterns and various seasonality types; (2) many domains face significant data scarcity issues, for instance, a newly established retail location lacks historical data for immediate customer traffic prediction, making zero-shot prediction capabilities invaluable; and (3) pre-training on extensive, multi-domain time series data exposes models to rare patterns, enhancing their representational learning capabilities.

Recent advancements have produced several notable time series foundation models. Chronos begins with the T5 (Raffel et al., 2020) encoder-decoder language model architecture and is pre-trained from scratch on substantial volumes of synthetic and real-world time series data. It conceptualizes time series as a specialized language by quantizing real values at each time step into a fixed-size dictionary during tokenization. MOIRAI employs an encoder-only transformer architecture with a multi-frequency patch encoding tokenization method, mapping non-overlapping segments of input time series to fixed-length embeddings. Additionally, MOIRAI directly models multivariate time series through self-attention within each variable and cross-attention across variables. Pretrained on approximately 350GB of real-world time series data from diverse domains, both Chronos and MOIRAI demonstrate state-of-the-art zero-shot forecasting performance on previously unseen benchmark datasets, significantly outperforming task-specific models trained specifically on those datasets.

However, there are limitations regarding the current attempts of time series foundation models. The limitations are two-fold: i) **For better TSFMs:** The optimal pipeline for developing and pre-training time series foundation models is still under-explored. Existing works are still centering around the previous ”one-model-one-task” paradigm. Besides, existing time series foundation models have limited abilities such as they lack the support to heterogeneous covariates. Our study address this limitation by developing TSFM technologies from various aspects. ii) **For applications of TSFMs:** Existing time series foundation models have yet to demonstrate transferability to scientific domains. Our study addresses this critical gap by exploring their potential in astrophysics, specifically analyzing the brightness observations (light curves) of celestial objects using Zwicky Transient Facility (ZTF) (Masci et al., 2018) data. This dataset presents unique challenges: (1) light curves are irregularly sampled with significant gaps due to observation constraints, including a recurring six-month gap resulting from earth-sun positioning; (2) the extreme scale and growth rate of astrophysical data make traditional machine learning approaches impractical due to high computational costs, limited model capacity, and vulnerability to distribution shifts.

For the rest of the thesis, I talk about my work in improving the pre-training efficiency for LLM-based foundation models in Chapter 2. Chapter 3 is about my work in studying different aspects pipeline-wised and model-wised that contribute to better time series foundation models. Chapter 4 is my work in studying the applications of time series foundation models to accelerate scientific discoveries in astrophysics.

1.2. Preliminaries

LLM-based foundation models. In summary, their success hinges on three ingredients: *self-supervision training*, *scale*, and *efficient distributed training*. **self-supervision training:** A modern Large Language Model (LLM) couples the *transformer* architecture (Vaswani et al., 2017) with *self-supervised next-token prediction.* Concretely, a text sequence $\mathbf{x}_{1:T} = (x_1, \ldots, x_T)$, consisting of tokens, x_i, is mapped to a stream of contextual embeddings via stacked multi-head self-attention and feed-forward layers. The model is trained to maximize $\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})$ (autoregressive) or, in the masked-language variant, to predict randomly masked tokens. **scale:** Scaling laws show that lowering the cross-entropy loss, and hence improving downstream performance, requires increasing *simultaneously* (i) parameter count N, (ii) training tokens D, and (iii) compute C roughly as $C \propto N^{1.3}$ and $D \propto N$ (Kaplan et al., 2020; Hoffmann et al., 2022). State-of-the-art LLMs are pre-trained on *hundreds of billions to trillions* of tokens drawn from filtered Common Crawl (Common Crawl, 2024) snapshots, Wikipedia, books corpora, news, forum data, GitHub code, patents, and mathematical proofs. For instance, GPT-3 (175 B) used ≈ 300 B tokens; PaLM-2 and Gemini report $\mathcal{O}(10^{12})$ tokens; LLaMA-2 (70 B) trains on 2 T tokens with a 90/5/5 split of web, code, and "refined" data (Touvron et al., 2023a). Data curation pipelines aggressively deduplicate, remove low-quality text, and apply heuristic safe-guard filters. Training costs scale super-linearly with size: GPT-3 consumed $\sim 3 \times 10^{23}$ floating-point operations (FLOPs) or $\sim 3{,}640$ A100-GPU-days; LLaMA-2-70B requires $\sim 5 \times 10^{23}$ FLOPs; state-of-the-art frontier models surpass 10^{24} FLOPs, often realized on clusters of 2048–6144 NVIDIA A100/H100 or TPU v4 chips interconnected by high-bandwidth networks. Energy budgets exceed 10^7 kWh, motivating research on efficient training schedules, quantization, and sparsity. Because a single accelerator cannot host billion-parameter models or the massive activation footprint of sequence lengths $L \geq 2,048$, practitioners combine *data parallelism* (sharding data, replicating weight matrices across devices), *tensor/model parallelism* (sharding weight matrices), and *pipeline parallelism.* **efficient distributed training:** This is non-trivial: memory and time constraints force practitioners to combine data, tensor, and pipeline parallelism while choosing batch sizes that maximize hardware utilization without degrading statistical efficiency. Choosing the *global batch size* is delicate: very large batches boost hardware utilization but reduce gradient noise, sometimes harming generalization. Our **adaptive batch-size schedule** (chapter 2) formalises this trade-off by monitoring optimisation

signals and resizing the batch on-the-fly, yielding faster convergence and provable non-convex guarantees in fully-sharded data-parallel (FSDP) training.

Time-series prediction & time-series foundation models. Time-series (TS) data consist of real-valued observations indexed by time and permeate climate science, finance, healthcare, and astronomy. Classical forecasting pipelines train one specialized model per dataset, an approach that struggles when labelled data are scarce or new patterns emerge. Inspired by LLMs, recent time-series foundation models (TSFMs) such as Chronos (Ansari et al., 2024a) and MOIRAI (Liu et al., 2024a) pre-train Transformers on hundreds of gigabytes of heterogeneous time series and achieve state-of-the-art zero-shot and few-shot performance on specialized benchmark datasets.

Astrophysics: light curves and variable-star classification. Time-domain astrophysics studies celestial objects whose brightness varies on timescales from minutes to decades. The *Zwicky Transient Facility* (ZTF) operates a 47 deg^2 camera at Palomar Observatory, California, surveying the Northern sky in g, r, and i bands with cadences ranging from $\sim$3 nights (public survey) to $\lesssim$1h (high-cadence programs) (Bellm et al., 2018; Bellm, 2014; Graham et al., 2019; Masci et al., 2018). Each night ZTF produces $\sim$1TB of raw images that are processed into *light curves*: irregular time series $\{(t_i, m_i, \sigma_i)\}$ of observation times, brightness (magnitudes), and uncertainties for every detected source.

A key scientific population in these data are *variable stars*, whose luminosity changes due to intrinsic stellar processes (e.g. pulsation) or extrinsic effects (eclipsing binaries). A large subset are *periodic variables* such as RR Lyrae, Cepheids, δ Scuti, eclipsing binaries whose periods encode stellar mass, radius, and evolution stage and serve as "standard candles" in cosmology. Reliable classification of variables therefore underpins distance-ladder calibration, Galactic structure mapping, and time-critical follow-up of rare transients. Traditionally, astronomers extract hand-crafted features (periodograms, amplitude ratios, color indices) and train task-specific classifiers on survey-dependent data. The explosive scale and heterogeneity of modern surveys now motivate foundation-model approaches: pre-train on millions of unlabeled light curves, then transfer compact embeddings to downstream tasks such as variable-star taxonomy, distance estimation, or anomaly detection. Chapter 4 operationalizes this agenda by introducing **StarEmbed**, a ZTF-based benchmark study that proposes and evaluate a pipeline to utilize general-purpose time-series foundation models to achieve state-of-the-art zero-shot periodic variable star classification performance, which demonstrates the enormous potentials of adopting the foundation-model paradigm in this field to accelerate scientific discoveries.

Pre-training, fine-tuning, few-shot learning and zero-shot inference. Both LLM and time-series foundation-model pipelines follow a two-stage protocol. In the pre-training stage a model θ_{pre} is trained in a self-supervised fashion on vast unlabeled corpora by next-token prediction for text, masked-value or segment prediction for time series so that it learns general representations of linguistic or temporal structure. Fine-tuning then adapts θ_{pre} to a specific downstream task T by minimising the training loss on a much smaller, labeled or unlabeled dataset $\mathcal{D}_T$, producing θ_{ft}. **Few-shot** learning means the model is able to learn the task by a limited amount of labeled data. In the era of LLMs and foundation models, few-shot learning could include (fine-tuning) or not include (in-context learning) the updates of the model weights since the model also supports learning from some examples in the prompt during inference without updating any model weights. **Zero-shot** inference means that foundation models perform T without any weights updates and examples in the prompt. In time-series foundation models, zero-shot inference appears as immediate forecasting or classification on an unseen dataset without any model weights updates.

CHAPTER 2

Improving Pre-Training Efficiency for Large Foundation Models

LLM-based (Large Language Model) foundation models derive their power from scale: billions of parameters, trillions of training tokens, and petaflop-days of compute. Yet realizing this scale in practice hinges on non-trivial optimization engineering. Foremost is the batch-size dilemma, larger batches increase memory utilization but empirically harms generalization performance, whereas smaller batches do the opposite. Besides, in consideration of the enormous data scale in the time series domain and inspired by the scaling law (Kaplan et al., 2020), TSFMs of a much larger size in the future are foreseeable. It is important to study methodologies to achieve good pre-training efficiency. This chapter tackles this problem through **our work** (Lau et al., 2024a) on existing LLMs, Adaptive Batch Size Schedules for Distributed Training of Language Models. We formalize batch size as a dynamic control variable, develop a noise–efficiency principle for adjusting it on-the-fly, and embed the algorithm in PyTorch Fully-Sharded Data Parallel so that data, tensor, and pipeline parallelism co-exist seamlessly. Extensive experiments on models up to three billion parameters in the LLaMA-2 family show consistent loss–compute gains over fixed or warm-up schedules, while our non-convex convergence analysis provides the first theoretical guarantee for adaptive batching with Adam.

2.1. Adaptive Batch Size Schedules for Distributed Training of Language Models with Data and Model Parallelism

2.1.1. Introduction

Large-batch training (i.e., using large batch sizes) is arguably the current *de facto* training paradigm for large language models, driven by recent advances and the availability of computational hardware for deep learning. For instance, the open-weight model, Llama 3 405B (Llama Team, AI @ Meta, 2024), utilizes a batch size of 1024 sequences of length 4096, resulting in 4M tokens per batch. Despite the efficient utilization of available hardware through parallelization, a major drawback of large-batch training is the issue of "generalization gap" (see e.g., (LeCun et al., 2002))—where model generalization performance deteriorates compared to small-batch training without heavy tuning of other hyperparameters. See Figure 2.1 for a graphical illustration of the existence of generalization gaps with different batch sizes when training a vanilla transformer with 61M parameters. Keskar et al. (2017) argued that small-batch methods tend to converge to flat minima, leading to better generalization. To close this generalization gap, several works (Hoffer et al., 2017; Smith et al., 2018; Smith and Le, 2018) have proposed using large learning rates to offset the effect of large batch sizes, recovering the generalization performance of using small batches. However, the training of language (and vision) models based on the attention mechanism (Vaswani et al., 2017) and the transformer architecture is notoriously unstable. Reducing training instability, including unwanted loss spikes (see e.g., (Zhai et al., 2023; Wortsman et al., 2024)), demands significant tuning and cautious hyperparameter selections, like using a small learning rate.

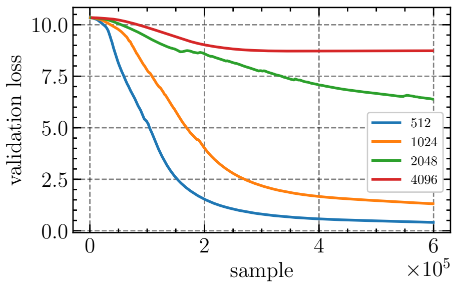


Figure 2.1. Generalization gap in transformer pretraining.

Beyond using a large learning rate to balance the intrinsic trade-off between training efficiency and generalization performance of large-batch training, Keskar et al. (2017) also suggested the use of *adaptive sampling methods* (Byrd et al., 2012; Friedlander and Schmidt, 2012). These methods are essentially adaptive batch size schemes that progressively improve the accuracy of the batch gradient approximation by gradually increasing batch sizes throughout the model training process. This concept has been explored by De et al. (2016, 2017) and Lau et al. (2024c), but their implementations are limited to the single-device setting, where all data samples are implicitly assumed to reside on the same device. This limitation makes them unfit for data-parallel distributed training wherein data is spread across various workers in a parallel system, potentially encompassing several network-connected nodes, thereby preventing the scaling necessary to train large models. Beyond the single-device setting, Lau et al. (2024b) have also extended such adaptive batch size schemes to local gradient methods for local batch sizes, where model synchronization is performed every several gradient steps rather than every step.

Data parallelism (Krizhevsky et al., 2012), such as `DistributedDataParallel` (`DDP`) in PyTorch (Li et al., 2020b) and counterparts in TensorFlow (Abadi et al., 2015) and JAX (Bradbury et al., 2018; Frostig et al., 2018), is arguably the most popular paradigm for distributed training in deep learning. In data parallelism (alone), each worker holds a local copy of the model parameters (as well as gradient and optimizer states). The global input batch is divided into multiple minibatches for each training step, so each worker performs forward and backward computations with a different minibatch. After each training step, all GPUs perform an *all-reduce* collective communication to synchronize gradients, followed by a global model parameter update. This ensures that all local copies of the model remain identical after the parameter update steps. Adaptive batch size schemes can be developed based on the approaches in (Byrd et al., 2012; Friedlander and Schmidt, 2012; Bollapragada et al., 2018) for data-parallel settings, providing practical adaptive batch size schedules in PyTorch `DDP` for training large-scale deep neural networks, which require data parallelism.

While these practical schemes open up the possibility of distributed training of larger models with GPUs of lower memory, they are constrained by the inherent design of `DDP`—the need to maintain a model replica at each worker. State-of-the-art large language models (LLMs) now consist of billions or even hundreds of billions of parameters (e.g., Llama 3 405B (Llama Team, AI @ Meta, 2024)). Distributed training with only data parallelism thus unfortunately fails, as the memory required to store such models well exceeds the available memory of a single GPU. Even worse, access to expensive workstation-level GPUs with more memory is often limited to industrial labs, whereas academic researchers and end-users often have to resort to less powerful consumer-level GPUs or workstation-level GPUs with less memory.

To alleviate this limitation inherent to data parallelism, more memory-efficient paradigms of parallelism, such as model parallelism (Shoeybi et al., 2019), have been proposed. In model parallelism, model parameters are sharded into various components and distributed to different workers. In particular, PyTorch Fully Sharded Data Parallel (`FSDP`) (Zhao et al., 2023) is an implementation of model parallelism in PyTorch (Paszke et al., 2019), marking the first native feature in PyTorch that can support models with up to trillions of parameters without relying on more sophisticated third-party libraries for model parallelism such as DeepSpeed (Rasley et al., 2020), Megatron-LM (Shoeybi et al., 2019; Narayanan et al., 2021; Korthikanti et al., 2023), and their combinations (Smith et al., 2022), which could be overwhelming to get started with and too technical to modify for users' specific needs. Moreover, PyTorch `FSDP` has been widely adopted in the pretraining of various open-source language models such as OPT (Zhang et al., 2022a), TinyLlama (Zhang et al., 2024a), OLMo (Groeneveld et al., 2024; Team OLMo et al., 2025), and DRBX (The Mosaic Research Team, 2024).

However, even with data parallelism and model parallelism, LLM pretraining involving models with up to hundreds of billions of parameters and trillions of tokens (e.g., Llama 3 405B (Llama Team, AI @ Meta, 2024)), still incurs extensive costs (more than millions of US dollars per model) and imposes a significant carbon footprint. Consequently, there is a pressing need for developing proper and well-crafted training strategies. In this work, we focus on choosing dynamic batch size schedules, which deserve more attention than they have, since, unlike other optimizer hyperparameters, batch sizes also control training efficiency via memory utilization of GPUs, in addition to affecting model generalization performance and training stability. The current practice of choosing batch sizes in LLM pretraining, however, remains heuristic, in the sense that it usually involves either constant large batch sizes or prespecified heuristic warmup schedules which could be very hard to design.

Contributions. In this work, we propose theoretically principled adaptive batch size schedules based on the adaptive sampling method (Byrd et al., 2012) for pretraining large language models, which are also generally applicable to training other deep neural networks. On the theoretical front, we establish a convergence guarantee for the proposed adaptive batch size schedules for Adam, the *de facto* optimizer for pretraining language models. Various recent works have shown, both empirically and theoretically, that Adam outperforms SGD in training attention-based language models (Pan and Li, 2023; Kunstner et al., 2023, 2024; Zhang et al., 2024b). Our convergence guarantee complements the existing results of adaptive batch size schedules for SGD (De et al., 2016, 2017) and AdaGrad (Lau et al., 2024c). From a practical perspective, we develop a solution of adaptive batch size schedules based on PyTorch `FSDP`, which are tailor-made for pretraining LLMs with more than billions of parameters.

2.1.2. Related Work

Large-batch training of language models. Large-batch training has proven to be very successful for different deep learning applications including computer vision (Goyal et al., 2017; Akiba et al., 2017) and natural language processing (You et al., 2020; Liu et al., 2019; Puri et al., 2018). From an empirical perspective, many open-source or open-weights models, such as OPT (Zhang et al., 2022a), BLOOM (BigScience Workshop et al., 2022), Mistral 7B (Jiang et al., 2023), Baichuan 2 (Yang et al., 2023), Qwen (Bai et al., 2023; Yang et al., 2024), OLMo (Groeneveld et al., 2024; Team OLMo et al., 2025), Gemma (Gemma Team, 2024; Gemma Team et al., 2024), Llama (Touvron et al., 2023b; Llama Team, AI @ Meta, 2024) and DeepSeek (DeepSeek-AI et al., 2024a,b), revealed that they were pretrained with large numbers of GPUs or TPUs (i.e., data-parallel sizes), hence naturally making use of large-batch training. While using large batch sizes is now standard, the rationale for choosing the magnitude of such large batch sizes is mostly based on hardware availability. Only recently in the training of Stable LM 2 1.6B, Bellagente et al. (2024) clarified the selection of global batch sizes, aiming to strike an optimal balance between minimizing training time and the extra training tokens needed to reach the desired final training loss. Shallue et al. (2019) study the effects of data parallelism by performing ablation studies on different batch sizes by training different models on different datasets using different optimizers, finding no evidence that large batch sizes degrade generalization performance with careful hyperparameter search. From a more theoretical perspective, McCandlish et al. (2018) develop a model for understanding the *critical batch size* that determines the tradeoff between speed and efficiency of large-batch training. Kaplan et al. (2020) further study the *scaling law* of the critical batch size as a power of the training loss only for language models. However, in most of these works, benchmarking was performed with different magnitudes of constant batch sizes, with the notable exception of McCandlish et al. (2018) which provided a case study of dynamically varying the batch size with an adaptive batch size schedule, but only using a simple model (CNN) and dataset (SVHN). The effect of adaptive batch sizes for pretraining language models, to the best of our knowledge, remains elusive to the community.

Batch size schedules. Adaptive sampling methods (Byrd et al., 2012; Friedlander and Schmidt, 2012; Bollapragada et al., 2018), which adjust batch sizes based on gradient noise or gradient approximation quality, are further explored in deep learning (De et al., 2016, 2017; Lau et al., 2024c; Ostroukhov et al., 2024) but have not been applied to data parallelism with distributed samplers. The development of adaptive batch size schedules for deep learning is not a novel concept, featuring methodologies such as Big Batch SGD (De et al., 2016, 2017), CABS (Balles et al., 2017), AdaBatch (Devarakonda et al., 2017), SimiGrad (Qin et al., 2021) and AdaScale SGD (Johnson et al., 2020). Our work is also closely related to and motivated by the heuristic technique of batch size warmup/batch ramp, which has been widely adopted in pretraining LLMs and even in reinforcement learning (Hilton et al., 2023). Batch size warmup usually involves prespecified schedules of multiple batch size stages starting from training with multiple increasing smaller batch sizes for small portions of the total training tokens, followed by training with the remaining tokens using a large batch size. For instance, GPT-3 (Brown et al., 2020) was pretrained by gradually increasing the batch size linearly from a small value (32k tokens) to the full value (3.2M tokens) over the first 4–12 billion tokens of training. Nemotron-4 (Parmar et al., 2024) was pretrained with a batch size schedule of batch sizes 384–768–1152 sequences for 2.5%–2.5%–95% of the total number of training tokens. Llama 3 405B (Llama Team, AI @ Meta, 2024) was trained using the following batch size schedule:

an initial batch size of 4M tokens with a sequence length 4096 tokens for 252M tokens; a batch size of 8M tokens with a sequence length of 8192 tokens for 2.87T tokens; a batch size of 16M tokens for the remainder of a total of about 15T training tokens. Such a batch size recipe is found to be able to stabilize training—few loss spikes were observed and it did not require interventions to correct for model training divergence. Despite potentially improving training efficiency or data parallelism, batch size warmup schedules remain heuristic and their impact on training is difficult to grasp. Another related yet seemingly orthogonal technique is sequence length warmup (Li et al., 2022; Jin et al., 2023), which progressively grows the sequence length throughout the pretraining process. Note that the pretraining of Llama 3 405B employs both batch size warmup and sequence length warmup.

2.1.3. Adaptive Batch Size Schedules with 2D Parallelism

We present the adaptive batch size schedules for data and model parallelism (termed *2D parallelism*), facilitating the scaling of pretraining for models with billions of parameters.

Notation. We define $[\![n]\!] := \{1, \ldots, n\}$ for $n \in \mathbb{N}^* := \mathbb{N} \setminus \{0\}$. We denote the inner product in $\mathbb{R}^d$ by $\langle \cdot, \cdot \rangle$ and its induced L_2-norm by $\| \cdot \|$, and $\|\cdot\|_1$ stands for the L_1-norm. For a vector $x \in \mathbb{R}^d$, $[x]_j$ denotes its jth coordinate ($j \in [\![d]\!]$). For a function $f \colon \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$, $\partial_j f$ denotes its partial derivative with respect to its jth coordinate for $j \in [\![d]\!]$. The ceiling function is denoted by $\lceil \cdot \rceil$. The disjoint union of sets $\mathcal{S}_1, \ldots, \mathcal{S}_J$ is denoted by $\bigsqcup_{j \in \{J\}} \mathcal{S}_j$.

2.1.3.1. Vanilla Adaptive Batch Size Schedules. We consider the empirical risk minimization problem in which we want to minimize the loss function $\mathrm{L} \colon \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$ in the form of a finite-sum objective:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \ \mathrm{L}(w) := \frac{1}{n} \sum_{i=1}^{n} \ell(w; z_i), \tag{2.1.1}$$

where $\ell \colon \mathbb{R}^d \times \mathcal{Z} \to \mathbb{R} \cup \{\pm\infty\}$ is the individual loss function, and $\mathcal{D}_n := \{z_i\}_{i=1}^n$ is the set of n training samples. If $\ell(\cdot; z)$ is continuously differentiable for any $z \in \mathcal{Z}$, then the gradient of the loss function and its batch counterpart (i.e., the batch gradient) are given by

$$\nabla \mathrm{L}(w) \coloneqq \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(w; z_i) \quad \text{and} \quad \nabla \mathrm{L}_{\mathcal{B}}(w) \coloneqq \frac{1}{b}\sum_{i\in\mathcal{B}} \nabla \ell(w; z_i),$$

where the batch $\mathcal{B} \subseteq [\![n]\!]$ is a subset of indices of data points sampled uniformly without replacement, and $b \coloneqq |\mathcal{B}|$ is the corresponding batch size. We write $\ell_i(w) \coloneqq \ell(w; z_i)$ and $\nabla\ell_i(w) \coloneqq \nabla\ell(w; z_i)$. The batch gradient $\nabla \mathrm{L}_{\mathcal{B}}$ is used to approximate the full gradient $\nabla \mathrm{L}$ as the number of samples n is prohibitively large.

Norm test. Falling into the family of adaptive sampling methods, the *norm test* (Byrd et al., 2012) is motivated by measuring the quality of the approximation of the full gradient $\nabla \mathrm{L}$ by the batch gradient $\nabla \mathrm{L}_{\mathcal{B}}$ through the lens of approximation of a descent direction for the loss L. If $\ell(\cdot; z)$ is also convex, then $-\nabla \mathrm{L}_{\mathcal{B}}$ is a descent direction for L at $w \in \mathbb{R}^d$ if and only if $\langle \mathrm{L}_{\mathcal{B}}(w), \mathrm{L}(w)\rangle \geqslant 0$, that is, $\mathrm{L}_{\mathcal{B}}$ and L share the same direction at w. It can be shown that the above inner product condition is equivalent to the norm condition: $\|\nabla \mathrm{L}_{\mathcal{B}}(w) - \nabla \mathrm{L}(w)\| \leqslant \eta\|\nabla \mathrm{L}(w)\|$ for any $\eta \in [0, 1)$. This condition cannot be checked directly, since the number of samples n is in billions for LLMs and the full gradient $\nabla \mathrm{L}$ is unavailable. Instead, we have to resort to a batch approximation:

$$\frac{\|\mathrm{Var}_{i\in\mathcal{B}}(\nabla\ell_i(w))\|_1}{b} \cdot \frac{n-b}{n-1} \leqslant \eta^2\|\nabla \mathrm{L}_{\mathcal{B}}(w)\|^2, \tag{2.1.2}$$

where

$$\mathrm{Var}_{i\in\mathcal{B}}(\nabla\ell_i(w)) \coloneqq \frac{1}{b-1}\sum_{i\in\mathcal{B}}(\nabla\ell_i(w) - \nabla \mathrm{L}_{\mathcal{B}}(w))^2.$$

The adjustment factor $(n-b)/(n-1)$ is approximated by 1 as we take $n \to \infty$. Consequently, to ensure that the batch gradient approximates the descent direction of the full objective L well, the *(approximate) norm test* checks the following condition at each iteration $k \in \mathbb{N}^*$:

$$\frac{\|\mathrm{Var}_{i\in\mathcal{B}_k}(\nabla\ell_i(w_k))\|_1}{b_k} = \frac{1}{b_k(b_k-1)}\sum_{i\in\mathcal{B}_k}\left[\|\nabla\ell_i(w_k) - \nabla \mathrm{L}_{\mathcal{B}_k}(w_k)\|^2\right] \leqslant \eta^2\|\nabla \mathrm{L}_{\mathcal{B}_k}(w_k)\|^2, \tag{2.1.3}$$

and increases the next batch size b_{k+1} if the above inequality is not satisfied, using

$$b_{k+1} = \left\lceil \frac{\|\mathrm{Var}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k))\|_1}{\eta^2 \|\nabla \mathrm{L}_{\mathcal{B}_k}(w_k)\|^2} \right\rceil.$$

The condition can be viewed as an approximation of the following *exact variance norm test* in the stochastic setting:

$$\mathbb{E}_k\Big[\|\nabla \mathrm{L}_{\mathcal{B}_k}(w_k) - \nabla \mathrm{L}(w_k)\|^2\Big] \leqslant \eta^2 \|\nabla \mathrm{L}(w_k)\|^2, \tag{2.1.4}$$

i.e., the motivating norm condition holds in expectation. Here $\mathbb{E}_k := \mathbb{E}[\cdot \mid \mathcal{F}_k]$ denotes the conditional expectation with respect to the σ-algebra up to the current batch at iteration k, i.e., $\mathcal{F}_k := \sigma(\{w_0, \mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_{k-1}\})$. After the next batch size is determined, the training loop continues with an optimizer step. The test implicitly makes a heuristic assumption that the next batch of size b_{k+1} will satisfy the approximate norm test at the current iterate w_k, but this is never checked to streamline the training loop.

2.1.3.2. Adaptive Batch Size Schedules with Data Parallelism. To allow training with large batch sizes with parallelized computations, a data-parallel extension of the *norm test*, which is referred to as DDP-Norm, can be developed and can be implemented based on PyTorch `DDP`. A special treatment of the norm test with data parallelism is necessary since data samples now reside in different workers, but we need to compute the mean and the variance of all the per-sample gradients in the norm test.

Specifically, at each iteration k, the global batch $\mathcal{B}_k$ is split across J workers with minibatches $(\mathcal{B}_{k,j})_{j \in [\![J]\!]}$ of equal size $b_{k,J}$ such that the global batch is the disjoint union of all minibatches, i.e., $\mathcal{B}_k = \bigsqcup_{j \in [\![J]\!]} \mathcal{B}_{k,j}$. Notice that at each worker $j \in [\![J]\!]$, the minibatch gradient can be computed by $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) = {}^1\!/_{b_{k,J}} \sum_{i \in \mathcal{B}_{k,j}} \nabla \ell_i(w_k)$. Since the minibatches have equal size and are disjoint, applying the law of total expectation, the global batch gradient is equal to $\nabla \mathrm{L}_{\mathcal{B}_k}(w_k) = {}^1\!/_J \sum_{j=1}^{J} \nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k)$. Note that the averages across workers are computed using *all-reduce* operations in PyTorch `DDP`. When minibatch sizes exceed the maximum memory of the workers, the technique of gradient accumulation is applied to simulate larger global batch sizes.

It is worth noting that efficiently implementing the approximate norm test (2.1.3) in deep learning libraries such as PyTorch (Paszke et al., 2019) is highly nontrivial, since per-sample gradients $\nabla \ell_i(w_k)$ are unavailable in the backward step of a standard training loop, but only the batch gradient $\nabla \mathrm{L}_{\mathcal{B}_k}(w_k)$ under a single-device setting or the minibatch gradient $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k)$ at each worker j under PyTorch `DDP`. If we were to implement the native approximate norm test (2.1.3), we would have had to compute per-sample gradients in parallel using vectorized mappings and based on a deep copy of the model, leading to undesirable memory and computational overheads. Thus, in practical implementation under data parallelism, instead of the approximate norm test (2.1.3), we propose to make use of the minibatch gradients of the workers to construct an estimator for the gradient variance

$$\widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k)) \coloneqq \frac{1}{J} \sum_{j \in [\![J]\!]} \Big(\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) - \nabla \mathrm{L}_{\mathcal{B}_k}(w_k) \Big)^2,$$

leading to the following more efficient implementation:

$$\frac{1}{b_k} \cdot \frac{1}{J} \sum_{j \in [\![J]\!]} \left[\left\| \nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) - \nabla \mathrm{L}_{\mathcal{B}_k}(w_k) \right\|^2 \right] \leqslant \eta^2 \| \nabla \mathrm{L}_{\mathcal{B}_k}(w_k) \|^2. \tag{2.1.5}$$

From now on, we refer the above alternative test as DDP-Norm. This implementation is much more computationally efficient since the minibatch gradients $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k)$ are already available at each worker and the global batch gradient $\nabla \mathrm{L}_{\mathcal{B}_k}(w_k)$ can be computed using *all-reduce* operations. Note however that this implementation requires an additional *all-reduce* operation every time to compute the quantity on the left hand side of (2.1.5) and additional memory to store it.

2.1.3.3. Adaptive Batch Size Schedules with 2D Parallelism via PyTorch `FSDP`. To enable the training of models with more than billions of parameters, model-parallel training presents a more sophisticated paradigm of parallelism. It shards the parameters of models and allocates different shards to different workers. In essence, PyTorch `FSDP` (Zhao et al., 2023), which shares similarities with `ZeRO-3` (Rajbhandari et al., 2020; Ren et al., 2021) in DeepSpeed (Rasley et al., 2020), operates by substituting the *all-reduce* operation in PyTorch `DDP` with *all-gather* and *reduce-scatter* operations.

For the purpose of mathematical illustration, we focus particularly on the tensor parallelism aspect of model parallelism. Coupled with data parallelism, it is established that each worker j possesses its own set of sharded parameters $\mathcal{W}_j$, $j \in \{\, J \,\}$, such that all the model parameters are denoted by $w_k = (w_{k,j})_{j\in\{\, J \,\}}$. Here, the sharded parameters on worker j are represented by $w_{k,j} \in \mathcal{W}_j$. Consequently, to compute the microbatch gradient at worker j, the gradients of all parameter shards must be resharded to obtain $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) = (\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_{k,1}), \dots, \nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_{k,J}))$, which can be efficiently implemented using the API of PyTorch `FSDP`. The implementation of DDP-Norm based on PyTorch `FSDP` is referred to as FSDP-Norm.

2.1.4. Convergence Analysis

Complementary to the convergence results of the norm test for SGD (De et al., 2016, 2017) and AdaGrad (Lau et al., 2024c), we derive convergence guarantees for Adam, acknowledging its prevalence in training deep neural networks for both computer vision and, more recently, language models. Adam (Kingma and Ba, 2015) employs the following update formula (with bias corrections for m_k and v_k dropped):

(2.1.6)

$$(\forall k \in \mathbb{N}^*) \quad m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k, \quad v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2, \quad w_{k+1} = w_k - \alpha m_k \odot v_k^{-1/2},$$

where $g_k := \nabla \mathrm{L}_{\mathcal{B}_k}(w_k)$, $\alpha > 0$ is a constant learning rate, $(m_k)_{k\in\mathbb{N}^*}$ and $(v_k)_{k\in\mathbb{N}^*}$ are the sequences of exponential weighted moving averages of the first two moments of the batch gradients respectively, $(\beta_1, \beta_2) \in (0, \infty)^2$ are weighting parameters, $\odot$ denotes the Hadamard product, and the power operations are performed coordinate-wise. We omit the bias corrections of m_k and v_k to simplify the analysis, but note that it can be easily extended to incorporate bias corrections. We also consider the more challenging scenario where $v_k^{-1/2}$ instead of $(v_k^{1/2} + \varepsilon)^{-1}$ is used in the update, since the denominator of the adaptive step sizes is no longer lower bounded away from 0. In our analysis, we invoke the following assumptions.

Assumption 2.1.1 (L-Lipschitz smoothness)**.** The loss function L is L-Lipschitz smooth ($L > 0$): for any $(u, v) \in \mathbb{R}^d \times \mathbb{R}^d$, we have $\|\nabla \mathrm{L}(u) - \nabla \mathrm{L}(v)\| \leqslant L\|u - v\|$.

Similarly to the analysis for AdaGrad (Lau et al., 2024c), we also require a coordinate-wise version of the (exact variance) norm test to hold due to the use of adaptive step size.

Proposition 2.1.1. The *coordinate-wise* (exact variance) norm test with constant $\eta \in (0, 1)$ ensures that, for every iteration $k \in [\![K]\!]$, the coordinate-wise batch gradient $\partial_i \mathrm{L}_{\mathcal{B}_k}(w_k)$ satisfies the following *coordinate-wise expected strong growth* (E-SG) condition: for all $i \in [\![d]\!]$, we have

$$\mathbb{E}_k[(\partial_i \mathrm{L}_{\mathcal{B}_k}(w_k))^2] \leqslant (1 + \eta^2)(\partial_i \mathrm{L}(w_k))^2.$$

Following closely a similar analysis to that in (Wang et al., 2023), we provide the following convergence results of the norm test for Adam.

Theorem 2.1.1. Suppose that Assumption 2.1.1 holds. Let $(w_k)_{k \in \mathbb{N}^*}$ be the Adam iterates generated by (2.1.6), where the batch size $b_k := |\mathcal{B}_k|$ is chosen such that the coordinate-wise (exact variance) norm test with constant $\eta \in (0, 1)$ is satisfied at each iteration $k \in \mathbb{N}^*$. Then, if $0 < \beta_1 \leqslant \sqrt{\beta_2} - 8(1 + \eta^2)(1 - \beta_2)/\beta_2^2$ and $\beta_2 \in (0, 1)$, we have $\sum_{k=1}^{K} \mathbb{E}[\|\nabla \mathrm{L}(w_k)\|] \leqslant \widetilde{\mathrm{O}}(K)$, where $\widetilde{\mathrm{O}}$ hides any logarithmic factors.

The full statement of this theorem and its proofs, as well as more in-depth related discussions, are deferred to Appendix A.1.2. The convergence results presented do not account for the decoupled weight decay in AdamW (Loshchilov and Hutter, 2019), which is more commonly used as an optimizer for language model pretraining. Furthermore, considerations such as learning rate schedules and gradient clipping are not included in these findings. Extending the above convergence guarantees to these settings is highly challenging and nontrivial and is left for future work.

2.1.5. Experiments

To showcase the versatility and scalability of FSDP-Norm, we conduct experiments with various families of decoder-only autoregressive language models at with different sizes and pretraining datasets. These include MicroLlama 300M (Wang, 2024), TinyLlama 1.1B (Zhang et al., 2024a) and OpenLlama 3B (Geng and Liu, 2023) on the C4 dataset (Raffel et al., 2020). The C4 dataset are tokenized using the Llama 2 tokenizer (Touvron et al., 2023b) with a vocabulary size of 32,000. Experiments are conducted on workstations equipped with 4 NVIDIA L40S GPUs (MicroLlama) and 4 NVIDIA A100-SXM 80GB GPUs (TinyLlama and OpenLlama). The training of the latter two models only feasible with PyTorch `FSDP` but not with PyTorch `DDP` using such hardware configurations, even with mixed-precision training (`bfloat16` is used). Our implementation utilizes the PyTorch `FSDP` API in PyTorch 2.6.1 and is simplified through Lightning Fabric of Lightning 2.4 (Falcon and The PyTorch Lightning team, 2019). For the ease of training language models, we also use LitGPT 0.5.3 (Lightning AI, 2023). Open-source implementation of DDP-Norm and FSDP-Norm is available at `https://github.com/timlautk/adaptive-batch-fsdp`.

Training Specifications. Adhering to the pretraining configurations of open-source LLMs such as TinyLlama (Zhang et al., 2024a) and OLMo (Groeneveld et al., 2024), our training specifications include a linear warmup followed by a cosine decay learning rate schedule, and the AdamW optimizer with weight decay and gradient clipping. The adaptive batch size schedule is set to a maximum global batch size, above which the norm test is no longer performed, opting for fixed interval testing over step-by-step (a test interval 1 is used, but longer interval entails reduced overheads brought by the test). Efficiency dictates using the test in its original form rather than its coordinate-wise variant, despite convergence guarantees. Given that batch sizes increase to the maximum possible values in the early stages, we only pretrain our models for a number of samples that are sufficient to display the behavior of our method, treating these experiments mainly as proofs of concept. Detailed configurations are provided in Appendix A.1.3.

2.1.5.1. MicroLlama 300M. We first pretrain MicroLlama with 300M trainable parameters on the C4 dataset (Raffel et al., 2020) under the same sets of other hyperparameters in order to better understand the effect of adaptive batch sizes. We compare with various constant batch size baselines $b_k \in \{2048, 4096, 8192\}$ and a stagewise batch size schedule 2048-4096-8192 for 2.5-2.5-95% of training tokens mimicking a popular batch size warmup for pretraining LLMs, and plot the results in Figure 2.2. We apply DDP-Norm for this relatively small model to demonstrate the applicability of the proposed schedules with PyTorch `DDP`. In Table 2.1, we report the total number of gradient steps (step), average batch size (bsz.), wall-clock time (time; in hours), best training loss (loss) and best validation loss (val loss; estimated by 100 iterations).

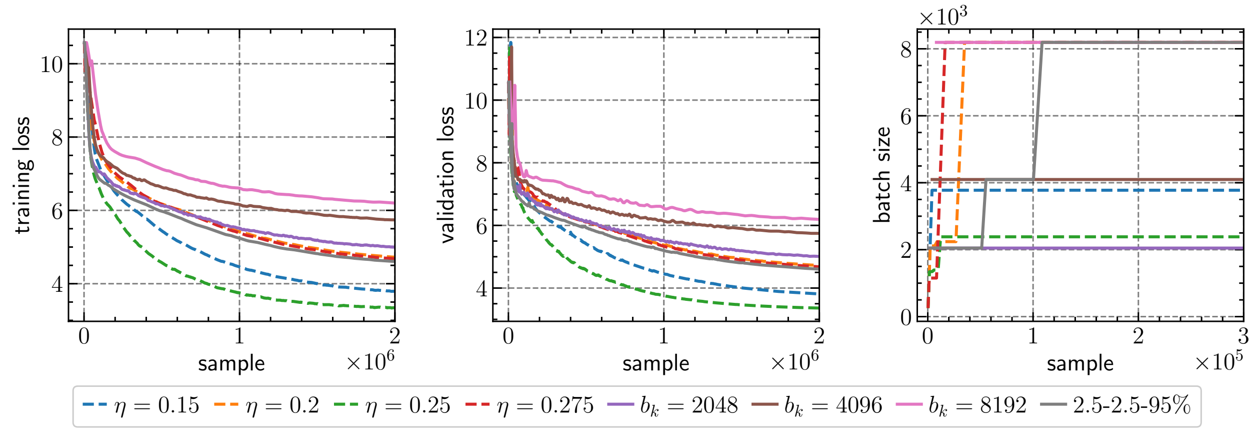


Figure 2.2. Training loss, validation loss and batch size schedule for MicroLlama 300M

We observe from Figure 2.2 that with $\eta = 0.2$ or $\eta = 0.275$, our proposed DDP-Norm outperforms the constant batch size baselines by a large margin in terms of validation loss. Specifically, using the same number of training samples, from Table 2.1, our method achieves lower validation losses when using similar number of steps ($\eta = 0.2$ versus $b_k = 8192$), when we use the number of steps as the criterion of measuring training efficiency. Our proposed schedule with $\eta = 0.2$ performs slightly worse than the stagewise batch size schedule, but it is expected since the latter has a smaller averaged batch size and takes a larger number of training steps. It is also worth noting that the design of the stagewise schedule is completely heuristic and might require lots of tuning, e.g., the number of stages, values of batch sizes and their ratios.

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.15$	531	3770	9.31	3.764	3.811
$\eta = 0.2$	254	7878	8.85	4.699	4.720
$\eta = 0.25$	843	2373	10.30	3.313	3.361
$\eta = 0.275$	252	7965	8.84	4.669	4.677
$b_k = 2048$	977	2048	11.18	4.976	5.005
$b_k = 4096$	489	4096	9.66	5.722	5.741
$b_k = 8192$	245	8192	8.48	6.183	6.192
2.5-2.5-95%	269	7439	8.78	4.594	4.604

Table 2.1. Results of MicroLlama 300M

We also observe that our method uses smaller batches at early stages and larger batches at later stages of training (e.g., $\eta \in \{0.2, 0.275\}$). This behavior has greater benefits regarding training efficiency because a larger batch size at each step means fewer number of required steps for the whole training process. On the other hand, our method greatly mitigates the side-effect of large-batch training—higher validation loss at the end of training—by starting from a small batch size and adaptively increasing it. Thus, our method enjoys both the good generalization performance of small batches and the high training efficiency of large batches. More importantly, our method is able to automatically increase batch sizes whenever it is necessary, to values that are completely adaptive to the training dynamics. Taking the adaptive batch size schedules in Figure 2.2 as an example, it is almost impossible to hand-craft similar schemes.

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.05$	261	7676	32.53	5.663	5.671
$\eta = 0.075$	267	7521	32.67	5.705	5.704
$\eta = 0.08$	270	7415	32.61	5.109	5.113
$\eta = 0.085$	274	7312	32.83	4.257	4.256
$b_k = 4096$	489	4096	34.48	3.814	3.817
$b_k = 8192$	245	8192	32.41	4.895	4.893
2.5-2.5-95%	269	7439	32.80	4.368	4.367

Table 2.2. Results of TinyLlama 1.1B

2.1.5.2. TinyLlama 1.1B. We also pretrain TinyLlama 1.1B on the C4 dataset, which necessitates the use of PyTorch `FSDP` and FSDP-Norm. From Figure 2.3 and table 2.2, similar conclusions can be made. We observe that our proposed FSDP-Norm effectively narrows the generalization gap between large and small batches, compared with constant batch sizes and with stagewise batch size schedule baselines. Specifically, our method facilitates the adoption of larger batch sizes of 8192 during the later stages of training. For instance, our method with $\eta = 0.085$ achieves an averaged batch size of 7312, yet it achieves validation loss closer to that of $b_k = 4096$, compared to $b_k = 8192$. Our proposed method is also able to reduce the magnitude of potential loss spikes which are obvious in using constant batch sizes.

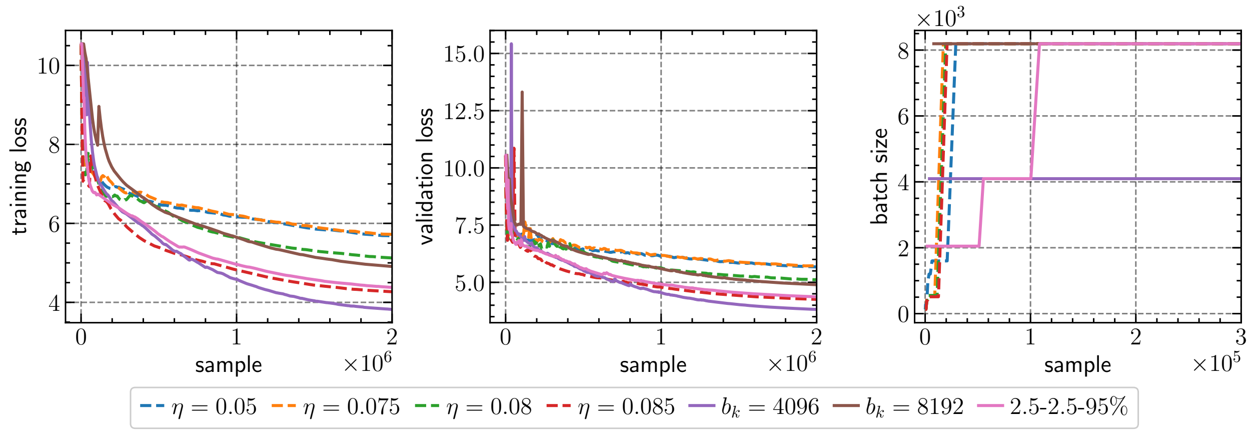


Figure 2.3. Training loss, validation loss and batch size schedule for TinyLlama 1.1B

2.1.5.3. OpenLlama 3B. We finally pretrain OpenLlama 3B on the C4 dataset, where a shorter sequence length of 512 instead of 2048 is used due to constraint on compute resources. Again, we observe similar phenomena to those of the smaller models, as revealed in Figure 2.4 and table 2.3. Specifically, with $\eta = 0.15$, the proposed approach requires slightly longer training time and larger number of training steps than the constant batch size 8192, while achieving a lower validation loss. While using a constant batch size 4096 achieves an even lower validation loss, it requires substantially more training steps and more than one hour of additional training time.

2.1.5.4. Further Discussions of Experimental Results.

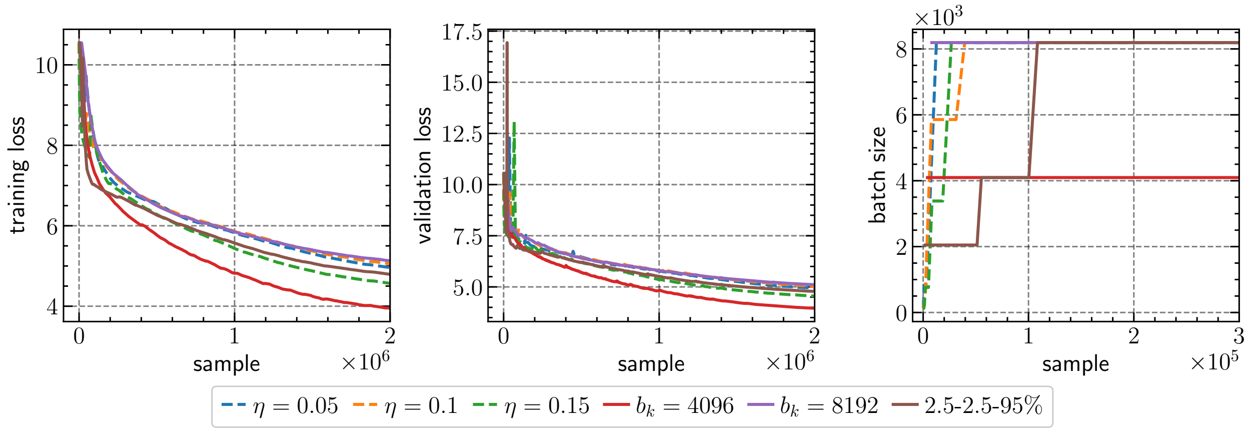


Figure 2.4. Training loss, validation loss and batch size schedule for OpenLlama 3B

scheme	steps	bsz.	time	loss	val loss
$\eta = 0.05$	249	8045	19.54	4.943	4.935
$\eta = 0.1$	253	7926	19.73	5.026	5.031
$\eta = 0.15$	259	7726	19.59	4.549	4.554
$b_k = 4096$	489	4096	20.75	3.934	3.956
$b_k = 8192$	245	8192	19.53	5.113	5.104
2.5-2.5-95%	269	7439	19.59	4.776	4.781

Table 2.3. Results of OpenLlama 3B

The effect of η. The hyperparameter η in the adaptive batch size schedules has the effect of controlling the probability of *obtaining a descent direction* and hence *increasing the batch size*. Obviously, choosing a right value of η is vital for our method to succeed. Across all three sets of experiments of different model scales, we found that larger values of η generally lead to more gradual batch size increments, but smaller values would allow full utilization of available compute resources at earlier stages of training but might defeat the prupose of adaptive batch sizes. Note that η also varies with the base learning rate α and the quality of the training datasets. In the series of works of adaptive sampling methods (Byrd et al., 2012; Bollapragada et al., 2018; Bollapragada and Wild, 2023), there are in-depth discussions on choosing the learning rate via some line-search procedures, which are however usually infeasible when training large deep neural networks.

Scaling law of critical batch size. We conjecture that there are more general scaling laws of the critical batch size (see e.g., (Kaplan et al., 2020; Su et al., 2024; Sclocchi and Wyart, 2024; Zhang et al., 2025a)) in relation to η which controls gradient approximation quality and the scale of gradient noise. For most choices of η in the three sets of experiments, we choose η small enough so that global batch sizes increase rapidly and reach the maximum possible values. However, in Figure 2.2, when $\eta = 0.15$, the final batch size is around 3800, which might be the critical batch size at this value of η. It is thus crucial to understand the notion of critical batch sizes through the lens of gradient approximation quality and we leave this for future work.

2.1.6. Conclusion

We create an efficient PyTorch `FSDP` implementation of the norm test for large-scale distributed training, focusing on hardware use and ease of development. Our implementation shows that adaptive batch size schedules can pretrain Llama 2 language models with up to 3 billion parameters using few GPUs (Touvron et al., 2023b). Furthermore, we provide convergence guarantees of the norm test for Adam, suggesting that our proposed adaptive batch size schedules are not only practically feasible, but also theoretically principled. Due to its generality, versatility, and scalability, we foresee extensive use of the adaptive batch size schedules in pretraining large transformer models like vision transformers (ViT) (Dosovitskiy et al., 2021) and autoregressive image models (Aim) (El-Nouby et al., 2024). We emphasize our attention on a PyTorch `FSDP` approach due to its integration with PyTorch. However, a more advanced implementation of the adaptive batch size schedules, using a new version of PyTorch `FSDP` (`FSDP2`) and tensor parallelism via PyTorch `DTensor` (Distributed Tensor), as well as availing of stronger computational hardware, will significantly enhance the scalability of the method for training models exceeding 7B parameters with 2D, 3D or even 4D parallelism. For further exploration, we refer readers to the `torchtitan` (Liang et al., 2025) and `lingua` (Videau et al., 2024) repositories. Furthermore, while our current implementation is based on PyTorch `FSDP`, but is readily extendable to other deep learning frameworks such as JAX (Bradbury et al., 2018) with `FSDP` and/or GShard (Lepikhin et al., 2021).

Limitations. In this work, we are primarily concerned with model generalization performance measured by validation loss without any evaluation on downstream benchmarks. The main reason for this is that we did not fully pretrain the models for sufficient number of tokens, implying that these models will not be competitive on downstream benchmarks. However, we expect that models fully pretrained with our proposed schedules will achieve very competitive performance on the evaluation of downstream benchmarks. We also remark that we can also incorporate other paradigms of parallelism such as pipeline and context parallelism with our proposed scheme, leading to 4D parallelism (data, tensor, pipeline, context parallel) for large-scale pretraining. While not supported in our current implementation, this can be achieved using the recent library `picotron` (Zhao and Mom, 2025) or the more sophisticated library Megatron-LM (Shoeybi et al., 2019). We leave this implementation for future work.

CHAPTER 3

Improving Time Series Modeling Pipeline & Time Series Foundation Models

Turning torrents of time series into scientific insights demands more than a model architecture. It requires an end-to-end pipeline that spans data ingestion, data pre-processing, hyperparameter optimization, modeling, and downstream task applications. This chapter is about our contributions in various aspects of this pipeline. RDAS (Li and Liu, 2025), a C++ raw-data engine, streams and pre-processes unstructured event sequences up to nanosecond resolution into structured time series of any resolution, making large-scale time-series corpora practical for time series foundation-model pre-training. For time series modeling itself, from the foundation model perspective, we explore the technological components to equip time series models with long-range memory and the ability to react to sudden events withtout cumbersome model training on carefully collected and processed data. Our work, STanHop-Net (Wu et al., 2023c), injects sparse Hopfield memory and plug-and-play external storage to handle long-range dependencies and abrupt regime shift. After the advent of LLMs, researchers have made first attempts of developing time series foundation models such as MOIRAI and Chronos. However, they are pure time series forecaster without proper supports to static features or covariates, which have demonstrated their effectiveness in the pre-LLM era. Our work (Li et al., 2025b) in proposing a neural architecture to strengthen existing time series models' modeling ability on covariates further demonstrates the benefits from proper modeling of covariates. Further extending from this direction, in our work, Hopformer: Homogeneity-Pursuit Transformer for Time Series Forecasting (Zhang et al., 2025b), we develop a framework that integrates a cross-sectional covariates regression mechanism with time series foundation models. Finally, we also look for a better hyperparameter optimization methodology for time series prediction models with the work, SWGA: A Distributed Hyperparameter Search Method for Time Series Prediction Models (Li et al., 2025a), by proposing a sliding-window genetic algorithm that couples Bayesian warm-starts with distributed evolution to find robust configurations that reduce the out-of-sample loss by about 56.1%.

3.1. RDAS: A Low Latency and High Throughput Raw Data Engine for Machine Learning Systems

3.1.1. Introduction

Recent advancements in machine learning (ML) have seen large pretrained models, such as those used in natural language processing (NLP), computer vision (CV), and multi-modality fields, achieve unparalleled performance. A key characteristic of these models is their substantial data requirements. For example, ChatGPT, a model renowned for its capabilities, was trained on an extensive dataset comprising 570GB of text data from the Internet. The effectiveness of these models is further enhanced by Transformer-based architectures, which are known for scaling efficiently with increased data size (Kaplan et al., 2020). Given the ongoing trend towards larger models, it is reasonable to anticipate that current data scales will continue to grow to meet these evolving requirements.

While the success of large pretrained models in NLP and CV is noteworthy, their expansion into other domains like time series prediction (Ma et al., 2024) and DNA understanding (Ji et al., 2021; Zhou et al., 2023) presents new challenges. The key issue lies in the limited quantity and granularity of training data available in these fields. For example, popular datasets in time series prediction, such as ETTh1, ETTh2, ETTm1, ETTm2, ILI, Traffic, and ECL, are relatively small, typically under 500MB, with only a few reaching between 1 to 10GB. These datasets, predominantly normalized and simplified from their original, more complex forms, result in significant information loss. ETTh1, for instance, is an hourly electricity dataset derived from higher-frequency sensor data. This loss is a major drawback, as raw datasets, often unstructured like event messages, contain richer details than their normalized counterparts.

To address this, we propose an integrated approach encompassing end-to-end data acquisition and processing. This method differs from traditional practices by processing raw data into structured form dynamically during model training. Unlike the static nature of preprocessed, normalized data, this dynamic approach allows for flexible and adaptive data transformation. This could include batch-specific normalization adjustments based on prior training results or on-the-fly data augmentations. Ultimately, this end-to-end process aims to harness the full potential of raw data, preserving its fine-grained nature for more effective model training.

However, the advantage of raw data's granularity comes with its own set of challenges, notably its unstructured nature and inherent heterogeneity. This heterogeneity significantly complicates the data acquisition and processing. Take autonomous driving systems as an example: they rely on a diverse array of sensors, including laser, image, inertial measurement units (IMUs), and odometry sensors (Nitta et al., 2018). Each sensor type generates data streams that vary in format, type, and resolution, adding layers of complexity (Kim et al., 2020) (Liu et al., 2020) (Queralta and Westerlund, 2019) (Manjanna et al., 2018) (Yurtsever et al., 2020).

To effectively manage the complexities of raw data acquisition and processing, thereby enabling ML systems to access the most detailed information in raw data, we propose abstracting diverse systems into a unified framework. This model conceptualizes the process as a competition among heterogeneous entities for a variety of resources, governed by multiple constraints like time of arrival, importance, ranking, cost, and gain. An entity could be, for example, a robot awaiting a task or a request for computational resources in a cluster. We envision this as N connected queues, where N represents the number of constraints, and refer to it as a 'message book' for clarity.

The message book abstraction offers several advantages for complex systems management. First, its intuitive nature facilitates easy understanding and implementation. Second, it is a versatile abstraction, applicable across various data types and systems in different domains. For example, in cloud computing, the message book can represent tasks awaiting execution, with computational power and time as the contested resources and submission time and task priority as constraints. The primary goal in such a system is maximizing the allocation of computational resources over time. This concept extends beyond cloud computing to encompass IoT systems, single-robot systems, and large-scale decentralized robotics systems. Third, the message book's construction is driven purely by data, relying on the most fundamental and unprocessed message data, thereby eliminating the need for complex data preprocessing. These attributes render the message book abstraction particularly well-suited for creating efficient, low-latency data visualization tools and data access interfaces in ML systems.

The implementation of a message book data structure is crucial for efficiently managing complex systems. Firstly, without such a model, it becomes challenging to determine the order in which participants access resources, as this decision relies heavily on various constraint functions. Secondly, given the heterogeneity, volume, and dispersed nature of the data, directly analyzing every minute activity of each participant is impractical. The message book addresses this by offering a unified representation that consolidates this diverse, granular data in real-time. This occurs concurrently with the data loading and model training processes in a ML system. As a result, the message book not only simplifies the understanding of the system's real-time dynamics but also serves as an effective intermediary for subsequent tasks. These tasks can range from visualizing system statistics to aiding in model training, thereby providing a foundational tool for various downstream applications.

Constructing the message book requires the system to adeptly route, filter, process, and aggregate all the activity information generated during its operation. We conceptualize each activity within the system as an event, with each occurring event represented as a message initiation and transmission process. Given that the message book's state is continuously evolving with the system's operation, it is crucial to model both the message book and the event message manipulations in a streaming fashion. The core research challenge we address involves developing an event-based data engine. This engine is designed to intelligently aggregate the most detailed event messages into a dynamic, general-purpose message book data structure, operating in a streaming manner. The proposed data engine is tailored to facilitate both real-time and historical data retrieval, offering efficient space and time complexity.

We organize the rest of the paper as follows. Section 2 mentioned prior work related to data processing systems, event processing systems, etc. Section 3 demonstrates the system architecture and implementation of RDAS. Section 4 shows experiments and results. Section 5 is the conclusion.

3.1.2. Related Work

Data loading frameworks Common data loading frameworks such as PyTorch Datasets & DataLoaders [1] and Tensorflow tf.data [2] are in a different position in the ML pipeline. They are for loading structured data from permanent storage such as local disk and remote storage such as S3. RDAS is to address the first-mile problem of how to acquire and aggregate raw data from data sources into a structured representation. These data loading frameworks can be naturally the next component that connects to RDAS in the ML pipeline.

Data processing system. General-purpose data processing systems include Pandas (McKinney et al., 2011), Dask (Rocklin, 2015), Numpy (Oliphant, 2006). However, they are for structured data that is generated by processing the raw unstructured data. There are also some big data frameworks that can process raw data such as Hadoop (White, 2012), Flink (Carbone et al., 2015), Spark (Zaharia et al., 2010), Storm (Evans, 2015) and Hive (Thusoo et al., 2009). But, they all serve as basic infrastructures and building blocks to construct data processing pipelines. They lack the higher level design to meaningfully process the raw data. This is where RDAS steps in.

Event processing system. In an IoT system, information sharing and communication are often modeled as the distribution of real-time event messages. (Rakkesh et al., 2016) proposes an event processing solution to detect vehicle speed violations. (Nawaz et al., 2019) proposes an event processing engine to process events from data streams for supply chain management purposes. (Syafrudin et al., 2018) proposes an event processing architecture to monitor the automotive manufacturing process. However, all these solutions are domain-specific. They are not a generic framework that is suitable for various kinds of domains in ML.

[1] https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
[2] https://www.tensorflow.org/guide/data

3.1.3. Methodology

3.1.3.1. Message Book. To maintain a simple and efficient data structure, we use queues to implement the message book data structure. If there are N constraints, there are N different queues. Each constraint corresponds to one queue and we store the specific entity objects in the queue whose constraint has the lowest priority. The entire message book data structure is a hierarchical structure. Each element of a queue with a specific constraint is a queue whose constraint is of a lower priority. For example, Figure 3.1 shows a minimal example of the message book. In this example, there are two constraints and constraint 1 has a higher priority than constraint 2, meaning that constraint 1 has to be met first. To be more intuitive, taking game matching as an example, constraint 1 could be the players' skill levels and constraint 2 could be the players' geographical distance. Each entity object stored in the message book represents a player in this case. Game matching for a player needs to first find other players with the same skill levels then among which find the player with the smallest distance. In this case, each element of the constraint 1 queue is a Struct that includes some basic information for that level such as the total number of players that are in that level, and a pointer to the queue of players in that level. In the queue of players, each element is a Struct representing a player and they are sorted according to constraint 2, which is the geographical distance that is of a lower priority. We use C++ standard library to implement the message book. The details are in B.1.2.

Traverse. The traversing of all entities (players) follows the priority of the constraints that are level first and entity second. This process starts from the index of the first level. For each level, it starts from the first entity and traverses all entities following the pointer to the next entity in each entity's entry. Then, it follows the pointer to the next level in the current level's entry and traverses all entities of the next level. This process goes on until it traverses all levels' entities. The traversing has $O(n)$ time complexity.

Random access to existing entities. Each time a new level or a new entity is added to the message book, their id-to-index mapping is saved in the hash map. Thus, for the random access to any specific entity or level, it is $O(1)$ time complexity using the hash map.

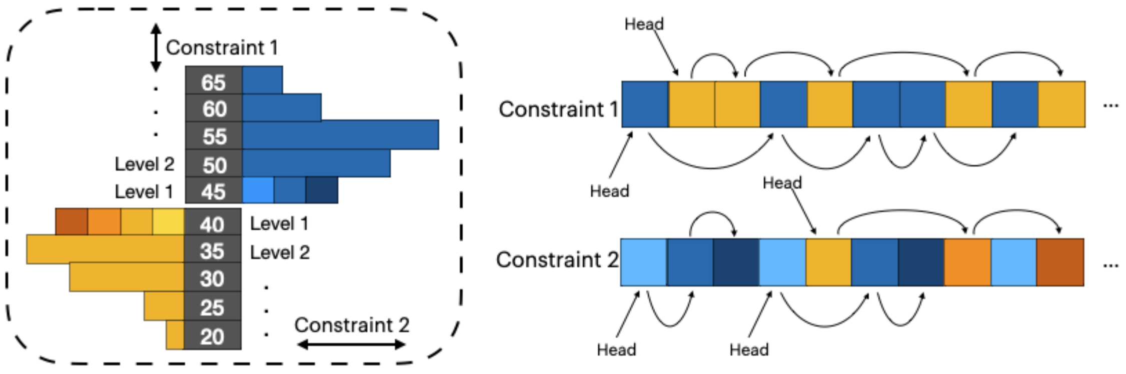


Figure 3.1. A Minimal example of the message book implementation. Left: a visualization of the message book with two constraints. Constraint 1 has a higher priority than constraint 2. Right: the implementation of the message book using two queues. The top corresponds to constraint 1 and the bottom corresponds to constraint 2. Only the queue of the least prioritized constraint(bottom queue) stores the entity objects. The top queue only stores level information.

Add entities or levels. To add a new entity to the queue of Constraint 1 or add a new level to the queue of Constraint 2, the algorithm first gets the index of free entry in the queue by popping an element from the corresponding FreeVector. Then, it creates a Struct representing the new entity or new level in the location indicated by the index in the corresponding queue. This is $O(1)$ time complexity.

Delete entities or levels. To delete an entity or a level, RDAS gets its index in the corresponding queue using the corresponding hash map. Then, it pushes that index into the corresponding FreeVector. This is $O(1)$ time complexity.

3.1.3.2. Data Transmission Format and Channels. The domains of machine learning systems that our system is for are those whose raw data are event data. For instance, in transportation systems, all raw data are event data that depict vehicles', road sensors', and traffic lights' status at each time point. In cloud systems, all the raw data are event data about different machines, different components, or different software of the system. In IoT or robotics, all the raw data are the events generated by different sensors, devices, or robots. Because of the blooming of edge device computation power and high-speed, low-latency communication technologies, highly decentralized IoT systems and robotics systems are becoming a reality. We choose a data format that is suitable to the current and future trends of heterogeneous event data transmission scenarios in various domains. In this case, we need a suitable data storage and transmission format that is capable of providing the properties of large volume, high frequency, low latency and high compatibility with network transmission of the event messages data because all the sensors and other system participants transmit data over the network (Huang and Savkin, 2018) (Pereira et al., 2019) (Winfield, 2000) (Nestmeyer et al., 2017). In RDAS, we store all the raw event messages in *pcap* (Packet Capture) format. Pcap is a direct capture of the data packet (event messages) that the senders send over the network (Sikos, 2020). It provides a single truth of source and an unbeatable granularity of up to nanosecond regarding the timestamp (Girela-Lopez et al., 2021). These features ensure that it has high credibility and enough capacity for any highly heterogeneous raw data source requiring various degrees of transmission latency, transmission frequency, and communication credibility.

Besides the foundational data storage format for the raw event message data, we also define a concept called data transmission channel in RDAS. There are four different channels as shown by Figure 3.2, which are actually four parallel data feeds when the system is operating. They are Alpha channel 1, Alpha channel 2, Beta channel 1 and Beta channel 2. Alpha channel only includes incremental information so that the transmission can have extreme compactness and high frequency. The Alpha channel is necessary when the frequency of new updates is very high because it is impossible to store the snapshot of the entire message book's status for every new update regarding the scale of the storage that is needed and the large percentage of redundant data in each snapshot. Beta channel only includes snapshot information the sender sends at a slightly lower frequency so that the transmission data volume is not too large. In the meantime, any device listening to this feed at any time can get a relatively new snapshot of the sender's state and start evolving the state from this snapshot using information from the Alpha channel. To ensure a high degree of robustness of the data transmission, both Alpha channel and Beta channel is a set of two parallel channels. Senders send the same packets simultaneously on channel 1 and channel 2. Receivers can use channel 1 and channel 2 to do cross verification or if some packets are corrupted on one channel, the system can recover them from the other channel. It is also noteworthy that the use of Alpha channel and Beta channel is flexible. For senders that only need to send stateless information, the use of Beta channel is optional. For senders that have a very limited amount of state data and have no requirement on high-frequency transmission, they can choose to only use the Beta channel to broadcast their full state periodically or whenever there is an update. Such a high degree of flexibility is an essential property that makes RDAS suitable to highly heterogeneous systems.

3.1.3.3. Core Data Processing Pipeline. The data processing pipeline of RDAS consists of four phases: channel merging, channel routing, channel decoding, knowledge distilling, and verification. The first three phases support the message layer. The last phase supports the feature layer and the verification layer. All these phases are piped to each other in a streaming manner, which means that it is not the case where a phase first processes all the data and then hands them over all at once to the next phase. It is the alternative case where when each phase processes a very small piece of the data, it immediately hands it over to the next phase and then starts processing the next piece. It is easy to see that in this streaming manner, all the events can be processed in the same chronological order as in that they took place and the large data size induced along the temporal dimension does not impair the processing speed.

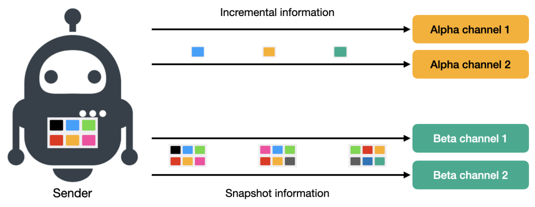


Figure 3.2. Two types of data channels: Alpha for high-frequency incremental information; Beta for low-frequency snapshot information. Each channel has a backup channel for robustness.

In the first phase, channel merging, the system merges all channels from all senders together into one monolithic stream. This process makes sure all the network packets in this stream are in chronological order using their timestamp of nanosecond granularity. In the second phase, channel routing, the system routes each packet to a specialized channel decoder that is responsible for decoding the data of a specific channel. The routing is based on the unique IP address of the sender. The third phase, channel decoding, as shown by Figure 3.3 is a major part of the pipeline and handled

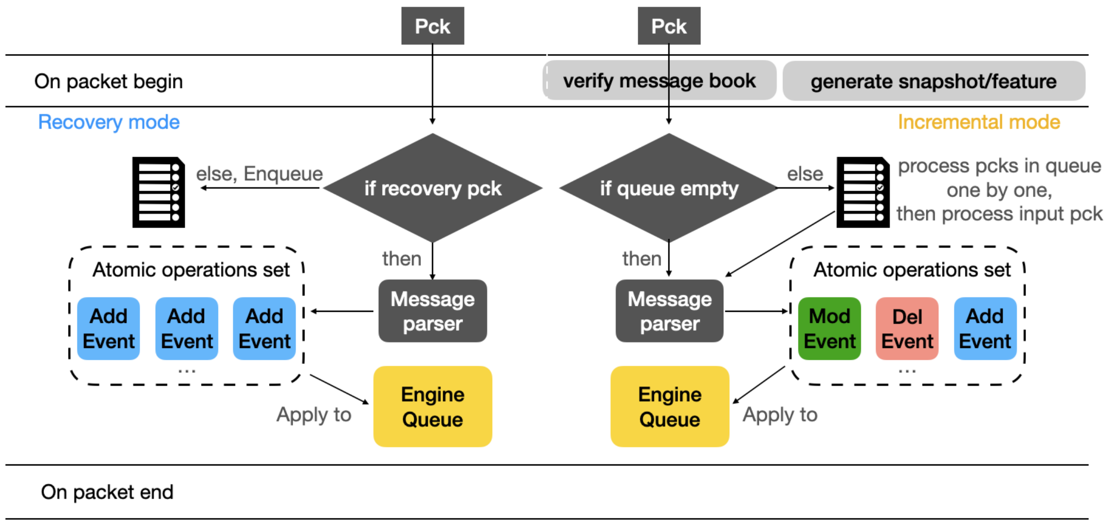


Figure 3.3. The feed decoding process: (1) two modes, recovery mode, and incremental mode. In recovery mode, the system uses the messages from the Beta channel to recover a snapshot of the message book from scratch. In the incremental model, the system applies incremental messages to the snapshot to evolve the message book. (2) The system preserves two user APIs on-packet-begin and on-packet-end, to generate useful information on the fly such as snapshots and features and do verification at any timestamp.

by a channel decoder. There is a recovery mode and an incremental mode of the channel decoder and these two modes can switch back and forth into each other. It is usually in the recovery mode when the decoder is at the start of the decoding process or the decoded stream has corruptions for some reason in the middle of the decoding process and the decoder needs to be re-calibrate it to a correct checkpoint of the states of the sender. The incremental mode is to apply incremental changes to a base snapshot of states so that the system can maintain an evolving and always on-sync message book. There are two sub-phases in the decoder. We call the one before the decoder starts processing the packet on-packet-start, which is an API allowing the user to define and conduct any task before the decoder processes the packet. RDAS generates message book snapshots through this API. We call another one after the decoder processes the packet on-packet-end. It is also an API to allow users to define and conduct any task before the channel decoding phase ends for this specific packet.

When the channel decoder is in recovery mode and a packet comes in, after the on-packet-start phase, the decoder checks the packet if it is from an Alpha channel or a Beta channel. If it is from an Alpha channel, the decoder would push it into the buffer queue without processing it. If it is from a Beta channel, the decoder hands it to a message parser that parses the content of the packet and produces a set of atomic operations. An atomic operation is a standardized and generic operation whose definition is a function. For example, some of the atomic operations are `Add Event`, `Modify Event`, and `Cancel Event`. The data engine uses a single set of available atomic operations of limited size for all kinds of data. Then, the decoder applies the set of atomic operations to the message book data structure so that the state of the message book incorporates the new information from the newcomer packet. Usually, in the recovery mode, because the message book is either in an empty state or corrupted state, the atomic operations are just `Add Events` to fill the empty message book or overwrite the existing message book.

When the channel decoder is in incremental mode, the process is almost the same as the recovery mode. The decoder first checks the buffer queue. If it is not empty, the decoder processes packets in the queue one by one until the queue is empty. After emptying the buffer queue, the decoder starts to handle the input packet. If the input packet is from a Beta channel, the decoder discards it. If it is from an Alpha channel, the message parser parses it to a set of atomic operations then the decoder applies them to the message book to incorporate new information from the packet.

The last phase in the pipeline is knowledge distillation and verification. It supports both the feature layer and the verification layer. This phase usually does not happen at the end of decoding the current packet. It happens at the start of decoding the next packet to guarantee the captured snapshot is the closest timestamp to the boundary timestamp. The feature generation and message book verification happen through the on-packet-begin API so that users have the flexibility to tailor these two processes. This phase is an essential bridge users can use to connect the message book to various sorts of downstream tasks such as data visualization terminals, analytical models, logging devices, etc.

3.1.3.4. Time Traveling. The time-traveling feature is a representative feature of RDAS based on its core data processing pipeline. It allows the query of a snapshot of the data stream at any timestamp with a constant maximum loading time no matter how far the snapshot's timestamp is from the start of the stream. The mechanism behind the time traveler is that the system first generates and caches a bunch of snapshots in the permanent storage at a frequency much lower than the original raw data granularity, for instance, every 10 minutes. Then, when the user jumps to any specific timestamp, RDAS first loads the snapshot from the cache that is the closest to that timestamp and then applies all incremental messages between the snapshot timestamp and the target timestamp to the message book to generate the target snapshot the query requests for.

3.1.4. Experiments

3.1.4.1. Case Study. We demonstrate the efficacy of RDAS by building a simulated high-frequency robotics logistics system and using it to acquire and parse the unstructured raw event data into the message book representation that is ready to be connected with the later data loading and model training pipeline. The reason that we choose to simulate a scenario in such a domain is that its raw data has high heterogeneity and high granularity, which provides enough complexities to test our system. Besides, the logistics industry is one of the kinds of complex systems that foresees to eventually transition from a labor-intensive industry into an automation industry. In 2021, there are about 1.3 million delivery drivers and about 40 million packages are in delivery every day in the US. In China, those numbers are about 4 million and 3 billion. However, this industry receives many critiques from both workers about its intense working schedule and clients about its far-from-satisfaction efficiency. Many delivery companies are considering or attempting to automate the delivery process by adopting delivery robots.

In an efficient logistics system with all delivery drivers as delivery robots, millions of robots should be accurately and optimally dispatched to deliver tens of millions of packages based on their capacity and package pick-up distance and deliver them to the client. It is essentially a message book model with very high precision in timestamps and large enough capacity to handle millions of events sent from different robots in both real-time and offline analysis.

In this scenario, there are two types of participants in the system. The first one is the local package dispatch centers that receive packages from higher-level dispatch centers from time to time. Whenever a batch of packages arrives at a local dispatch center, it sends out a Ready-to-Give (RtG) request to the system. The second one is the delivery robots that send a Ready-to-Take (RtT) request associated with the local dispatch center that is the nearest to them anytime they have capacity and are ready to fetch and deliver some new packages from the local dispatch center. During the operation, the system maintains a message book for each local dispatch center in parallel. In the message book, the constraints that decide the order of events are the distance and the time of arrival (ToA) of the request. The distance is discrete and represents the straight line distance to that specific dispatch center. Thus, there will be two types of distances. The first type is the distance between the destination of the delivery and the dispatch center. The second is the distance between the delivery robot and the dispatch center. There are different zones for different distance ranges. For instance, `Zone 1` is within [0km, 3km), `Zone 2` is within [3km, 6km), `Zone 3` is within [6km, 9km), etc.

During the operation of the system, all the delivery robots keep sending location/distance information periodically (for example, every 10 min) to the local dispatch center that they belong to. They are all in the message book no matter whether they are on their way to deliver packages or to the dispatch center to fetch new packages. Besides, all the packages are in the message book until the delivery is complete. They also have two states in the message book, either `being-delivered` or `to-be-delivered`. Once the dispatch center assigns a package to a delivery robot, the package's state changes from `to-be-delivered` to `being-delivered` and it leaves the message book once the delivery is complete.

3.1.4.2. Experiment Setup. The experiment is twofold. One is to demonstrate RDAS's large throughput, the other is to show the low latency and the low space cost of RDAS's time-traveling feature. We use ROS to simulate the dynamics of the aforementioned logistics system and generate the event message data. We assume there are always 2000 delivery robots and 20000 packages on the message book and the robots and the dispatch center send messages at some preset frequency.

We built a separate ROS program to uniformly sample and generate these messages from all the possible message types to simulate reality. There are in total six types of event messages that affect the message book status. Three are for delivery robots and the other three are for packages. For each data stream, the system generates a snapshot every 10 minutes and caches it to the disk.

For delivery robots, the first type of message is `AddRobot`. When a new robot becomes online, the robot sends this message. Its fields include `distance`, `capacity`, `state`, `last_update_time`. The second type is `ModifyRobot`. When a robot's status changes between `Occupied` and `Empty`, it sends out this message. The third type of message is `RemoveRobot`, when a robot becomes offline, it sends this message. For packages, when a new package arrives at the dispatch center and waits for delivery, the dispatch center sends out an `AddPackage` message. When a package's status changes from `to-be-delivered` to `being-delivered`, the dispatch center sends an `ModifyPackage` message. Lastly, when a package delivery is complete, the dispatch center sends a `RemovePackage` message to remove the delivered package from the queue.

For the throughput experiment, the baseline system is the same as RDAS except that the baseline system does not have the incremental message aggregating mechanism and message book representation. Instead, it assumes each message includes the full picture. We assume there are two message data streams for RDAS and the baseline system respectively. In each time unit, two streams have the same number of messages and the same amount of information such as the number of robots and packages and their status. The only difference is that one stream uses incremental messages and the other uses full-picture messages. We measure the total size of the messages each system processes within one second. Under the same network bandwidth, a system processes more information (higher throughput) at each time unit if it requires a smaller total message size to recover the same amount of information. Thus, we represent the throughput by taking the inverse of the total message size per second with a scale.

The time-traveling feature is a representative feature that can demonstrate the superb performance of RDAS. We conduct experiments to measure the latency and the space cost of the time-traveling mechanism. To have a low latency for this mechanism, there are two types of overhead to consider. The first is the time cost to generate the snapshot cache. The second is the latency to apply the incremental messages between the loaded snapshot time and the target timestamp to generate a new snapshot on the fly. We measure both of them. Furthermore, we also measure the size of disk space to store the snapshot cache. The experiment demonstrates that the random access time traveler enables an O(1) latency with respect to the number of minutes away from the initialization point of the data stream and the size of the disk space to save the snapshot cache is within a reasonable range.

3.1.4.3. Results. We can see from Table 3.1 and Figure 3.4 that with the incremental message aggregation and the message book representation, RDAS's throughput is several magnitudes higher than the baseline system. When the data stream's frequency is at 10^3hz, RDAS's throughput is about 6 times larger than the baseline. The discrepancy keeps increasing when the stream frequency increases. When the frequency is 10^7hz, RDAS's throughput is about 400 times higher than the baseline.

Table 3.1. Throughput of RDAS and the baseline system for data stream of different frequencies. It is a scaled value of the inverse of the total message size per second.

	Data stream frequency				
System	**10^3hz**	**10^4hz**	**10^5hz**	**10^6hz**	**10^7hz**
RDAS	1	163.68	1850.35	5989.47	10366.29
Baseline	0.15	1.47	14.87	23.66	26.18

To demonstrate RDAS's low latency, we first run the system several times to repeatedly measure the time it takes to process 10000 packets and take an average. The final result is that it takes about 0.01 seconds to process 10000 network packets. If the data simulation program generates the data stream at a 100hz frequency, a day(24 hours) of the streamed Pcap data has 8.64 million messages and the whole snapshot cache generation for them only takes 8.64 seconds. Even the stream is of 100000hz frequency, meaning on average, there are 100000 messages every second, which is already very unlikely in real-world robotics and IoT systems or systems in other domains, RDAS only takes 2.4 hours to cache a day of snapshot data. Considering the snapshot cache generation is an offline

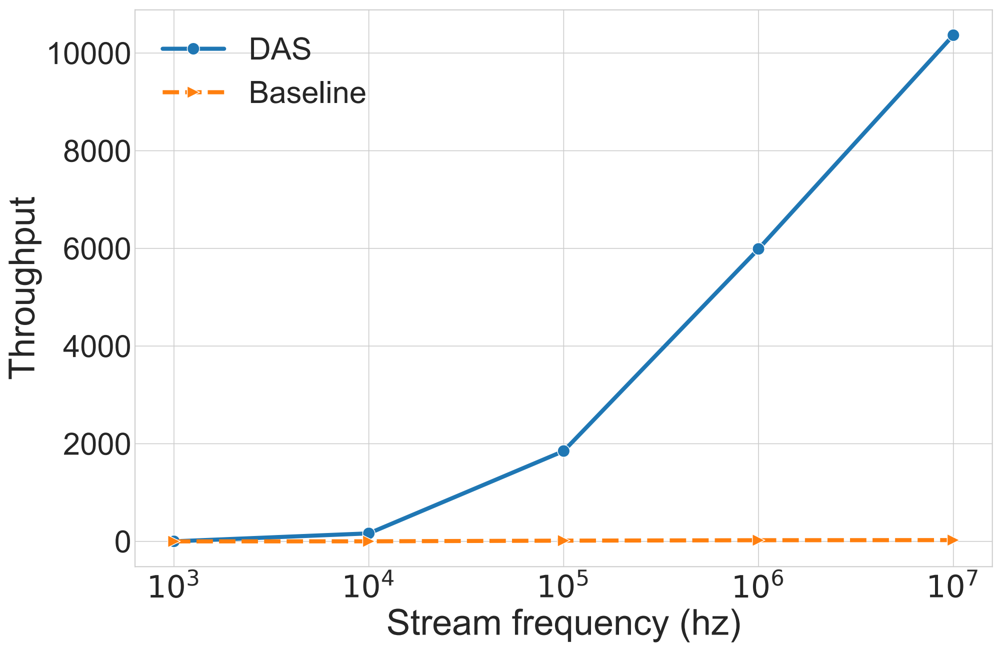


Figure 3.4. Throughput (the larger the better): starting from the stream frequency of 10^3hz, the throughput of RDAS is about 6 times larger than that of the baseline system. When the frequency increases to 10^7hz, the RDAS is about 400 times larger than the baseline. RDAS's throughput increases much faster than the baseline's when the frequency increases.

task, such a level of time complexity is already sufficient. It is also noteworthy that in these settings, the latency for the time traveler to generate a requested snapshot on the fly has a reasonably short upper limit which is 1 second.

We also measure the random access waiting time in querying snapshots of the data stream at any timestamp to demonstrate RDAS's low latency. We construct a baseline system using the same codebase as RDAS except that the baseline does not have the random access message aggregating feature and it can only start building the target snapshot from the initialization point of the data stream. By comparing their random access waiting time, we can clearly see from Table 3.2 and Figure 3.5 that RDAS has a constant bound but the baseline keeps increasing, causing long latency when the target timestamp is far away from the initialization point.

Lastly, we examine the disk space consumption of the snapshot cache. Although the snapshot size varies in different systems in different domains, our logistics robotics system example, with

Table 3.2. Random access waiting time (milliseconds) for RDAS is a piece-wise linear function with the peaks always at a similar constant number. However, it is an increasing linear function for baseline with no bounds, which indicates a much higher and eventually unacceptably long latency for the user to obtain a snapshot.

	Time distance range (min)					
System	**0**	**5**	**10**	**15**	**20**	**25**
RDAS	22.0	2886.7	23.6	2983.5	22.4	3011.8
Baseline	23.4	2874.3	6023.7	8800.9	12084.4	14858.9

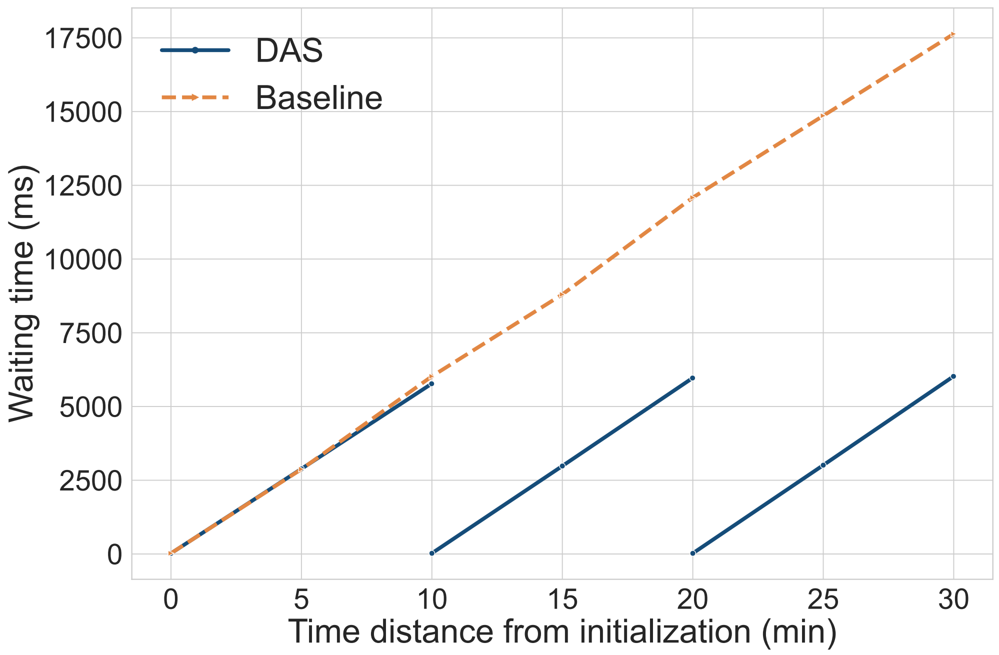


Figure 3.5. Random access waiting time: On the data stream of 10000hz frequency, RDAS's random access waiting time always has a nearly constant bound at around 6000ms. However, the baseline model has a linearly increasing waiting time with respect to the further and further timestamps.

2000 delivery robots and 20000 packages on the message book for each dispatch center at any time, is already ambitious and thus, representative enough. We assume that without compromising the user experience, a 1-second latency (waiting time) is a reasonable upper limit when the user jumps around to query snapshots at random timestamps. It means we need to at least cache 1 snapshot for every 1 million messages because the processing speed is 1 million messages per second. Hence, disk consumption is a meaningful and critical metric. We generate a day of snapshot cache by generating one snapshot for every 1 million messages. The average size of each snapshot file is

100KB. Following the same deduction as above, if the simulated data stream is of 100hz frequency, the snapshot cache of one day is only about 860KB and it is about 860MB if the frequency is 100000hz. This shows that the space cost of RDAS is very low.

3.1.5. Conclusion

We propose RDAS, a general-purpose raw data acquisition and processing system for machine learning systems in various domains. It provides a unified data interface to bridge the unstructured, high-resolution and heterogeneous raw data of up to nanosecond granularity to common data loading and model training pipeline. It features a novel data representation mechanism, the message book, with the incremental message aggregation mechanism and a low-latency random access time traveler. RDAS allows users to quickly query a snapshot of the message book at an arbitrary timestamp. The experiments demonstrate RDAS has a high throughput, low latency, and consumes reasonably small disk space with full support to high-resolution raw event data. Future works could provide a programming language agnostic interface that enables quick integration into any existing machine learning frameworks.

3.2. SWGA: A Distributed Hyperparameter Search Method for Time Series Prediction Models

3.2.1. Introduction

In the realms of machine learning and deep learning, hyperparameter tuning stands as a cornerstone to effective model training. It is important to tune the hyperparameters of the model on a validation dataset. First, this adjustment helps minimize the risk of model overfitting to the training data, which often severely degrades out-of-sample performance. Second, by fine-tuning hyperparameters on a validation set with a distribution similar to the training data, the model can achieve better performance on out-of-sample data with a matching distribution. Lastly, since many hyperparameters pertain to the model architecture and computational efficiency, optimal configurations can enhance model efficiency. Consequently, researchers widely adopt hyperparameter tuning across various machine learning and deep learning domains (Vaswani et al., 2017; Dosovitskiy et al., 2020; Zhang and Yan, 2022; Liu et al., 2021a; Zhou et al., 2021; Devlin et al., 2018; Arik and Pfister, 2021; Huang et al., 2020).

Time series prediction remains a crucial endeavor in various sectors. Domains such as energy (Hong et al., 2020; Nti et al., 2020b; Reneau et al., 2023), finance (Fischer and Krauss, 2018), house pricing (Xu and Zhang, 2021), and medical treatment (Prakarsha and Sharma, 2022) heavily rely on predicting future time series values based on historical data. With the swift advances in machine learning, optimizing performance on unseen data demands rigorous hyperparameter search. Time series data, characterized by temporal dependencies and non-stationarity, poses unique challenges. Temporal dependencies mandate models to discern patterns evolving with time, while non-stationarity implies fluctuating statistical properties, leading to potential distribution shifts (Kim et al., 2021; Fan et al., 2023) and possible model overfitting (Roelofs et al., 2019).

Traditional general-purpose hyperparameter search algorithms do not take into account the domain knowledge of the time series prediction problem by design. Specifically, many time series prediction models suffer from non-stationary time series and the temporal distribution shift in the dataset is a long-lasting problem (Du et al., 2021). In this work, we propose a hyperparameter search process that caters for temporal distribution shifts in time series data.

Addressing these challenges, we present the Sliding Window Genetic Algorithm (SWGA), a pioneering method tailored for hyperparameter search in time series prediction models. SWGA offers three innovations: a sliding window technique to mitigate overfitting due to time series distribution shifts, a warm-up phase that utilizes Bayesian optimization for crafting a solid initial population, and inherent compatibility with distributed computation across multi-node clusters.

This paper delves into SWGA's underlying methodology and assesses its efficacy across diverse time series datasets, consistently demonstrating its edge over conventional genetic algorithms in identifying optimal hyperparameters for out-of-sample time series predictions.

There are four major contributions of this paper:

- We introduce a warm-up stage using a lightweight TPE method, enhancing the initialization of the initial population. Compared to the random initialization in traditional genetic algorithms, this approach offers a more promising starting point for subsequent iterations, ultimately guiding the algorithm towards optimal convergence.
- We unveil a configurable sliding window mechanism for hyperparameter search tailored for time series datasets, bolstering the search's resilience against distribution shifts in time series data.
- We demonstrate an effective way to incorporate the consideration of the distribution shift in time series into the hyperparameter search process to create a domain-knowledge-enhanced hyperparameter search method that is better than its general-purpose counterpart. Using genetic algorithm (GA) as an example in the experiments, we demonstrated that our proposed way (warm-up and sliding window) can greatly enhance the base method, GA, into SWGA, a method that gives much better out-of-sample results for time series prediction models.
- Our algorithm seamlessly integrates with the Ray distributed computation framework (Moritz et al., 2018), making it adaptable to a wide range of parallelism scenarios.

We structure the rest of the paper as follows: Section 3.2.2 reviews related works. Section 3.2.3 provides the necessary background. Section 3.2.4 elaborates on the SWGA methodology. Section 3.2.5 outlines our experimental design, datasets, and results. Section 3.2.6 is the conclusion of the paper.

3.2.2. Related Work

In this section, we discuss the relevant literature on hyperparameter tuning methods for time series prediction, covering traditional optimization techniques, distributed computing approaches, and evolutionary algorithms.

Researchers widely use traditional optimization techniques, such as grid search and random search (Bergstra and Bengio, 2012), for hyperparameter tuning in time series prediction models. Although these methods are conceptually simple, they come with high computational costs and inefficient exploration of large hyperparameter search spaces. Bayesian optimization methods, which gained popularity due to their ability to model the performance landscape and guide the search towards promising regions of the hyperparameter space (Snoek et al., 2012), still demand substantial computational resources for large-scale time series prediction problems.

To tackle the computational challenges associated with hyperparameter tuning, researchers propose distributed computing approaches. Some examples include Population-based Training (PBT) (Jaderberg et al., 2017), Asynchronous Successive Halving Algorithm (ASHA) (Li et al., 2020a), and Hyperband (Li et al., 2017). These methods exploit parallelism to accelerate the search and find success in various machine learning tasks. But, their full applicability to time series prediction problems requires further study, and adaptations may be necessary to handle the unique challenges of time series data, such as non-stationarity and temporal dependencies.

Researchers employ evolutionary algorithms, such as Genetic Algorithms (GAs), for hyperparameter optimization in various machine learning tasks (Alibrahim and Ludwig, 2021) (Elgeldawi et al., 2021). GAs exhibit several attractive properties, such as global search capabilities and robustness to local optima, making them suitable for complex optimization problems. The literature contains several distributed GA variants, including Distributed Genetic Algorithm (DGA) (Belding, 1995), Island Model Genetic Algorithm (Whitley et al., 1999), and Master-Slave Genetic Algorithm (Cantu-Paz and Goldberg, 2000). While these methods apply to a wide range of optimization problems, their application to time series prediction tasks remains limited.

K-fold cross-validation (Kohavi et al., 1995) is a popular technique used for model evaluation and hyperparameter tuning in machine learning. This method involves partitioning the dataset into K equally sized folds, where each fold serves as a validation set exactly once, while the remaining K-1 folds are used for training the model. By averaging the performance metrics across all K iterations, K-fold cross-validation provides a more robust and reliable estimate of the model's generalization performance compared to a single train-test split. This approach is particularly useful in scenarios where the dataset size is limited, as it maximizes the usage of available data for both training and evaluation. Moreover, K-fold cross-validation effectively reduces the risk of overfitting and helps to identify a model that generalizes well to new, unseen data. Our algorithm may look similar to K-fold cross-validation, but they are very different.

Ray (Moritz et al., 2018) is a distributed computing framework that supports various distributed computing infrastructures. We integrate it into our algorithm implementation to enable the distributed hyperparameter search capability.

3.2.3. Background

Time series prediction. Suppose that we have a multivariate time series with N variates. It is also a set of N univariate time series $\{z^i_{1:T_0}\}^N_{i=1}$. There are in total T_0 time steps. The prediction target is the next τ time steps $\{z^i_{T_0+1:T_0+\tau}\}^N_{i=1}$. We are trying to model:

$$p(z^i_{T_0+1:T_0+\tau}|\{z^i_{1:T_0}\}^N_{i=1};\Phi) = \prod_{i=1}^{\tau} p(z_{T_i}|\{z^i_{1:T_0}\}^N_{i=1};\Phi)$$

In this conditional distribution, Φ is the parameter of the prediction model.

Hyperparameter search. Consider a machine learning model M characterized by a set of hyperparameters $H = \{h_1, h_2, ..., h_n\}$. Each hyperparameter h_i has a domain $d(h_i)$ from which a value can be selected. The goal of hyperparameter search is to find a configuration $C = \{c_1, c_2, ..., c_n\}$, where each $c_i \in d(h_i)$, that optimizes the performance of the model M on a given dataset. This can be mathematically formulated as:

$$C^* = \arg\min_{C\in d(H)} L(M(H = C), D) \tag{3.2.1}$$

Here, L represents a loss function that quantifies the discrepancy between the predictions of the model M with hyperparameters set to C and the true values in the dataset D. The aim is to find the hyperparameter configuration C^* that minimizes this loss.

Distribution shift. Time series prediction models often suffer from non-stationarity from the time series data. The distribution in these data shifts along the time direction. To mitigate distribution shift, people usually use domain generalization ((Li et al., 2018; Muandet et al., 2013; Wang et al., 2022)) and domain adaptation ((Tzeng et al., 2017; Ganin et al., 2016; Wang et al., 2018)). Domain generalization focuses on learning from the source domain and hopes to generalize well on the target domain while domain adaptation is to reduce the distribution distance between the source and target domain. They both have the goal to bridge the distributions of source and target domains. However, our method is different from these methods in the sense that we address the distribution shift from the hyperparameter search perspective.

Tree-structured Parzen Estimator. The Tree-structured Parzen Estimator (TPE) is a prominent method for hyperparameter optimization. TPE models the joint distribution $p(x, y)$ of the hyperparameters x and the objective function y. In contrast to other optimization techniques that model $p(y|x)$ and then invert this relationship, TPE models $p(x|y)$ and $p(y)$ directly. TPE divides the hyperparameters into two sets depending on the observed y values and then generates new candidate hyperparameters from a distribution that favors the promising set. In doing so, TPE provides a more flexible way of exploring the hyperparameter space, especially when the distribution of hyperparameters is non-uniform. However, TPE can be computationally expensive as the number of hyperparameters grows and sensitive to the choice of the threshold that separates the two sets of hyperparameters. Besides, TPE runs in a sequential manner. It is hard to run in parallel and utilize the modern multi-node distributed computing clusters.

3.2.4. Methodology

To initialize the first population, rather than using random generation, we use a Tree-structured Parzen Estimator (TPE) to repeatedly run a small number of trials to generate the initial population. We call this process the warm-up stage and it provides a better starting population for the genetic algorithm. To make the hyperparameter search more robust to the time series's distribution shift problem and prevent the algorithm from overfitting the validation set, we create this configurable sliding window mechanism when conducting the genetic algorithm. We first split a dataset into the training set, validation set, and testing set according to a fixed ratio. Then, we evenly split the training set into multiple chunks of the same size. Then, we split the validation set into the same number of chunks. The hyperparameter search process goes as follows. First, we define a window of a length of a fixed number of chunks. The window starts from the earliest chunk and slides one chunk after each iteration of SWGA along the time direction. Starting from the population of the first generation, at each iteration, SWGA trains the model with each individual config in the population only on the data within the fixed-length window and does the model validation on the data chunk right after the window. Then, the window slides along the time dimension using a fixed stride of one data chunk. The size of each data chunk and the size of the window are both configurable. Figure 3.6 shows an example of how SWGA works on a 3-year time series dataset.

In detail, The entire process of SWGA, also shown by Algorithm 1 - 5, is as follows. First, it splits the datasets into the training set, validation set, and testing set. In the warm-up stage, the TPE algorithm runs a small number of trials on the training set and the validations set to produce one hyperparameter config. This process repeats several times until it generates enough individuals for the initial population. Then, the genetic algorithm process starts. In each iteration, it first evaluates each individual in the population and sorts them according to the ascending fitness value. The fitness value is the trained model's Root Mean Square Error (RMSE) or Mean Absolute Error (MAE) on the validation data. Then, based on the ranking of each individual in the population regarding the fitness value, the algorithm generates a new population for the next iteration. The

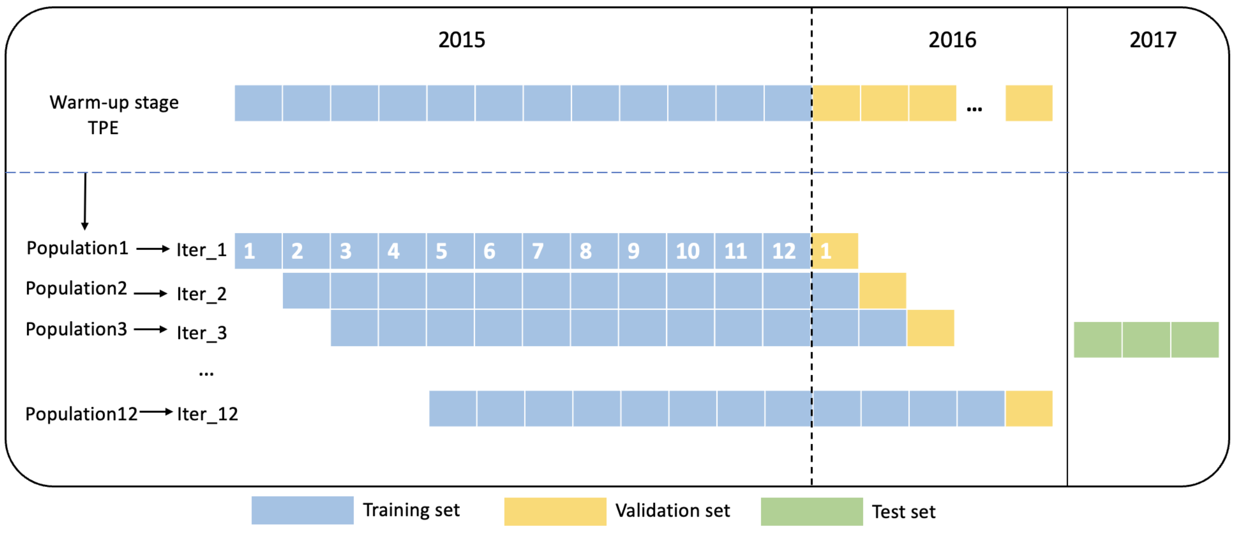


Figure 3.6. A demonstration of SWGA. From top to bottom, they are the different stages of the tuning process. In this specific example, the training set is the history time series for 2015 and the validation set is the history time series for 2016. They are both divided into 12 chunks respectively. The test set is the history time series for 2017. In each iteration, there is a sliding window including in total 13 chunks with 12 as training set and 1 as validation set. It generates the population on this window for the next iteration. The best configuration from the final generation is used to obtain the evaluation metrics on the test set.

population generation process is as follows. It first creates a set of parents from the top k individuals (low fitness values) and from the tail 2 individuals of the population. Then, it applies the crossover operator and mutation operator to the parents to generate the offsprings. The offsprings and the top k individuals together become the next generation of the population. Lastly, after the final iteration, the top individual in the population becomes the final winner. SWGA then uses this configuration to train a model on the original training set and report the RMSE or MAE on the testing set.

SWGA variant. We also consider a variant of the SWGA. In each iteration, rather than only using one data chunk right after the training window to validate the model, we use all the data chunks in the validation set that is not in the window. However, this variant is much slower and the experiment results show that it does not provide a better performance. Thus, we do not use this variant to run the experiments in the experiment section.

Since genetic algorithms are natively parallelization-friendly, we also integrate SWGA with Ray compute framework to support parallelized hyperparameter search on various computation infrastructures including single-node multi-core and multi-node multi-core.

Algorithm 1 SWGA

```
1:  Raw dataset separates into $trn\_set, val\_set, tst\_set$
2:  Initialize $population$ as an empty list
3:  Initialize $fitnesses$ as an empty list
4:  for $i = 1, 2, \ldots, K$ do
5:      if $i == 1$ then
6:          $population_0 =$
                        WarmUpStage($trn\_set, val\_set$)
7:          $trn\_set$ splits into $N$ chunks
8:          $val\_set$ splits into $N$ chunks
9:      end if
10:     $trn\_set_i$, $val\_set_i$ = GetDataset($i$, $trn\_set$, $val\_set$)
11:     $population_i =$
                GenNextPop($population_{i-1}, fitnesses_{i-1}$)
12:     for each individual configuration $h$ in $population_i$ do
13:         Evaluate the fitness of $h$ and add to $fitnesses_i$
14:     end for
15: end for
    return the config with the best fitness in $fitnesses_K$
```

Algorithm 2 GetDataset(i, trn_set, val_set)

```
1:  $trn =$
        concatenate($trn\_set[chunk_i, \ldots, chunk_{i+N-1}]$)
2:  $val = val\_set[chunk_{i+N}]$
    return $trn, val$
```

Algorithm 3 CrossoverMutate($parent1$, $parent2$)

```
1:  $crossover\_rate = 0.7$
2:  $mutate\_rate = 0.2$
3:  for each hyperparameter key $c$ in the config space do
4:      $child[c]$ = RandomChoice($parent1[c]$, $parent2[c]$)
5:      $child[c]$ = RandomChoice($child[c]$, $default\_config[c]$) # mutation
6:  end for
    return $child$
```

Algorithm 4 WarmUpStage(trn_set, val_set)

```
1:  Initialize $init\_pop$ as an empty list
2:  while size_of($init\_pop$) $<$ POPULATION_SIZE do
3:      best_config = TPE(num_trials=10)
4:      Add best_config to $init\_pop$
5:  end while
    return $init\_pop$
```

Algorithm 5 GenNextPop($population_{i-1}$, $fitnesses_{i-1}$)

```
1:  Get $topk\_selection$ from $population_{i-1}$ according to fitness in $fitnesses_{i-1}$
2:  Get $tail2\_selection$ from $population_{i-1}$ according to fitness in $fitnesses_{i-1}$
3:  Initialize $parent\_pairs$ as empty list
4:  Initialize $next\_pop$ as empty list
5:  for $i = 1, 3, 5, \ldots$, k-1 (two elements each time) do
6:      Append ($topk\_selection[i]$, $topk\_selection[i+1]$) to $parent\_pairs$
7:  end for
8:  Append ($tail2\_selection[0]$, $tail2\_selection[1]$) to $parent\_pairs$
9:  for each $parent\_pair$ in $parent\_pairs$ do
10:     $child$ = CrossoverMutate($parent\_pair$)
11:     Add $child$ to $next\_pop$
12: end for
    return $next\_pop \cup topk\_selection$
```

3.2.5. Experiments

To demonstrate the efficacy of our methodology, we conduct two tasks in our experiments.

Task 1. In this task, we focus on showcasing the effectiveness of our proposed methodology by ablation studies. We use SWGA, GA and SWGA* (SWGA without the warm-up stage) to search for hyperparameters for 5 common time series prediction model architectures respectively. Then, we compare the out-of-sample prediction performance of these models. Through this task, we show that both our proposed warm-up stage and the sliding window mechanism are effective and our proposed SWGA method indeed has a performance gain compared to the base GA method. The results are in Table 3.3 and 3.4.

Task 2. In this task, we demonstrate SWGA's values in real applications. We use SWGA to search for hyperparameters for three latest SOTA time series prediction models in the literature including iTransformer (Liu et al., 2023), DLinear (Zeng et al., 2023) and PatchTST (Nie et al., 2022). We show these models' immediate improvements regarding the out-of-sample prediction performance on the same long-term forecasting task, training, and testing dataset as the original setups in the literature.

Experiment Configurations. We conduct all the experiments on a Ray cluster node with 48 CPU cores and 8 Nvidia RTX 2080Ti GPUs. For Task 1, we use SWGA to do a hyperparameter search on 4 different prediction models on 10 multivariate time series datasets from different domains. For each dataset, we first split the dataset using the 8:1:1 ratio into the training set, validation set, and testing set. Then, we split the training set and validation set respectively into 12 trunks. We use seven historical timesteps to predict one timestep ahead. Each experiment repeats 5 times and we report the mean RMSE and mean MAE. Since SWGA has the sliding window mechanism that increases the number of trials on different hyperparameter configurations, to ensure that there is a fair comparison, we make sure all the compared methods including the baseline have the same total number of trials in the hyperparameter tuning process. To obtain the RMSE and the MAE, we first use the hyperparameters that the hyperparameter search method finds to initialize the model. Then, we train the model on the training set and test the model on the test set. We report the model's RMSE and MAE on the test set. For Task 2, we ensure that all models have the same settings as that in Table 1 of the iTransformer (Liu et al., 2023) paper. The only difference is that we use SWGA to do hyperparameter search. The hyperparameter search space we use is in B.2.1

Datasets. In the experiments, we use ten real-world datasets: (i) Beijing PM2.5: This is an hourly multivariate time series dataset ranging from 2010 to 2014. It has (ii) SML2010: A month of home monitoring multivariate time series data of resolution of 15 minutes. (iii) Appliance Energy: Four months of energy use multivariate time series dataset of 10-minute resolution. (iv) Individual household electricity: Four years electricity use multivariate time series dataset of 1-minute resolution. (v) Exchange: It is a multivariate dataset including daily exchange rates in eight different countries from 1990 to 2010. (vi) ETT (Electricity Transformer Temperature) datasets are multivariate time series. There are two collection sources of them with labels 1 and 2. There are

Table 3.3. Comparison of RMSEs between SWGA and GA for different models and datasets. SWGA achieves the best results on most of the models and datasets. SWGA* represents the version of SWGA that does not use TPE to generate the initial population. Instead, it uses the random sampling method.

Model	Method	Dataset									
		Beijing PM2.5	SML2010	Appliance Energy	Individual Household Electricity	Exchange	ETTh1	ETTh2	ETTm1	ETTm2	Traffic
	SWGA	**0.071**	**0.055**	**0.069**	0.032	**0.015**	**0.034**	**0.056**	**0.027**	**0.015**	**0.015**
Catboost	SWGA*	0.073	0.056	0.074	**0.022**	**0.015**	0.073	**0.056**	0.074	0.022	0.039
	GA	0.081	0.137	0.070	0.078	0.080	0.062	0.117	0.038	0.065	0.067
	SWGA	**0.079**	0.159	**0.068**	**0.062**	**0.051**	0.072	0.095	**0.051**	**0.052**	**0.051**
LightGBM	SWGA*	0.088	0.164	0.078	0.065	**0.051**	**0.071**	**0.093**	0.070	0.085	0.054
	GA	0.080	**0.137**	0.078	0.078	0.078	0.082	0.108	0.073	0.096	0.067
	SWGA	**0.087**	0.136	**0.108**	0.070	**0.049**	**0.080**	**0.100**	0.108	**0.050**	0.078
XGBoost	SWGA*	0.091	0.159	0.147	**0.052**	0.052	0.091	0.159	0.147	0.052	**0.052**
	GA	0.414	**0.121**	0.440	0.426	0.208	0.081	0.145	**0.075**	0.132	0.376
	SWGA	**0.087**	**0.265**	**0.092**	**0.065**	**0.013**	**0.087**	**0.265**	**0.092**	**0.065**	0.024
LSTM	SWGA*	0.198	0.349	0.180	0.568	0.017	0.198	0.369	0.180	0.568	**0.023**
	GA	0.168	0.636	0.197	0.176	0.260	0.737	0.649	0.593	0.656	0.110
	SWGA	**0.071**	**0.121**	**0.071**	**0.058**	0.109	**0.056**	**0.040**	**0.078**	**0.109**	**0.042**
Transformer	SWGA*	0.072	0.197	0.089	0.084	**0.095**	0.118	0.181	0.102	0.143	0.084
	GA	0.608	0.879	0.707	0.911	0.730	1.312	0.990	0.732	1.142	0.578

Table 3.4. Comparison of MAEs between SWGA and GA for different models and datasets. SWGA achieves the best results on most of the models and datasets.

Model	Method	Dataset									
		Beijing PM2.5	SML2010	Appliance Energy	Individual Household Electricity	Exchange	ETTh1	ETTh2	ETTm1	ETTm2	Traffic
	SWGA	**0.018**	**0.039**	**0.026**	**0.015**	0.045	**0.058**	**0.023**	0.053	**0.023**	0.016
Catboost	SWGA*	0.034	0.116	0.027	0.016	0.054	0.143	0.180	**0.024**	0.132	**0.014**
	GA	0.062	0.117	0.038	0.065	**0.028**	0.070	0.092	0.060	0.082	0.058
	SWGA	**0.067**	0.120	0.050	0.063	**0.051**	0.068	**0.077**	**0.045**	**0.057**	**0.051**
LightGBM	SWGA*	0.072	0.114	0.051	**0.052**	0.052	**0.067**	0.082	0.047	0.058	0.053
	GA	0.077	**0.112**	**0.045**	0.068	0.074	0.071	0.092	0.056	0.089	0.057
	SWGA	0.089	0.106	0.313	**0.022**	**0.042**	**0.048**	**0.045**	**0.045**	**0.042**	**0.034**
XGBoost	SWGA*	**0.080**	0.110	**0.108**	0.275	0.059	0.049	0.063	0.049	0.060	0.199
	GA	0.410	**0.103**	0.418	0.416	0.074	0.067	0.114	0.067	0.108	0.371
	SWGA	**0.018**	**0.040**	0.029	**0.016**	**0.014**	**0.048**	**0.021**	**0.038**	**0.015**	**0.016**
LSTM	SWGA*	0.036	0.059	**0.025**	0.017	**0.014**	0.197	0.080	0.128	0.119	0.024
	GA	0.236	0.665	0.217	0.089	0.288	0.491	0.569	0.719	0.583	0.152
	SWGA	**0.069**	**0.109**	0.056	**0.053**	**0.093**	**0.083**	**0.105**	**0.061**	**0.072**	**0.060**
Transformer	SWGA*	0.076	0.170	**0.051**	0.065	0.125	0.163	0.121	0.082	0.084	0.156
	GA	0.532	0.612	0.790	0.434	0.770	0.790	0.811	1.210	1.229	0.993

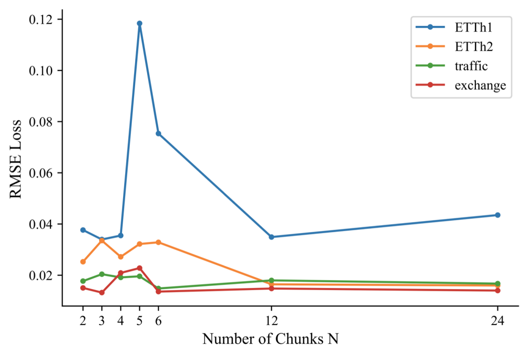
two collection resolutions that are 1 hour and 15 minutes. So, there are four specific datasets in this category: ETTh1, ETTh2, ETTm1, and ETTm2. (vii) Traffic: A multivariate dataset recording the hourly road occupancy rates from various sensors on freeways in San Francisco from 2016 to 2018.

Out-of-sample performance (task 1). As we can see from Table 3.3, SWGA consistently outperforms the traditional genetic algorithm on most of the datasets and different models. On average, SWGA reduces the RMSE on the out-of-sample testing set by 54.6% compared to GA. SWGA* is the SWGA without the warm-up stage. Instead, SWGA* uses the random sampling method to generate the initial population as the traditional genetic algorithm. On average, SWGA* reduces the RMSE on the out-of-sample testing set by 34.0% compared to GA. By comparing the results of SWGA* and the results of GA, we can know that the configurable sliding window mechanism indeed brings a significant reduction to the out-of-sample RMSE. By comparing the results of SWGA and SWGA*, we can see that the warm-up stage contributes additional reduction to the RMSE on top of the sliding window's contribution in most cases.

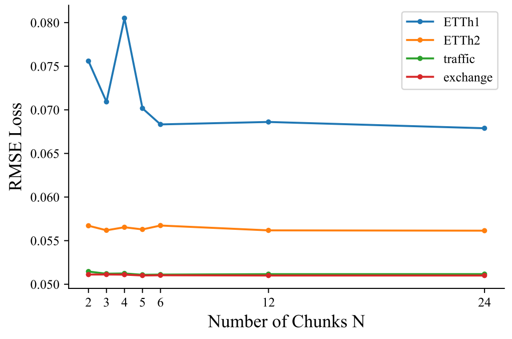
Table 3.4's results are consistent with Table 3.3. SWGA has the lowest MAE on most of the datasets. On average, SWGA has about a 57.6% reduction compared to the MAE of GA. SWGA* reduces the MAE by about 42.6% compared to GA. In both the MAE and RMSE metrics, SWGA yields a significant improvement over GA.

Besides, SWGA shows a consistent advantage in both Table 3.3 3.4 across various kinds of popular time series prediction model architectures including tree models (Catboost, LightGBM, XGBoost), recurrent models (LSTM), and attention-based models (Transformer). This further demonstrates SWGA's advantage and application value.

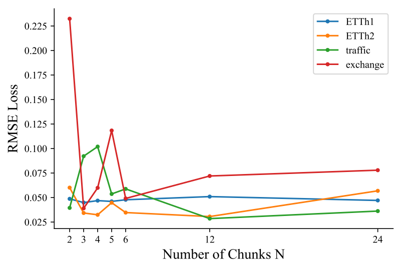
Improvement on latest SOTA time series forecasting models (task 2). As we can see from Table 3.5, by using SWGA to do hyperparameter search on three SOTA time series forecasting models, without other additional modifications, we immediately get an improvement as much as 6.46% of average reduction of MSE. This demonstrates that a considerable amount of additional testing performance of time series predictions models is achievable by using a good set of hyperparameters.



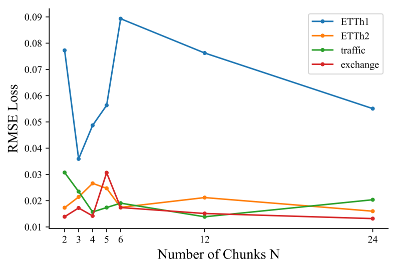
(a) Catboost



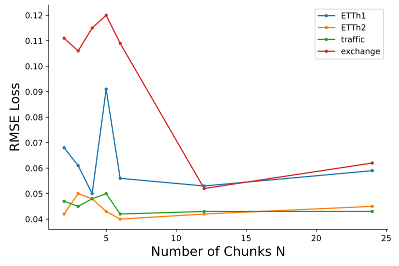
(b) LightGBM



(c) XGBoost



(d) LSTM



(e) Transformer

Figure 3.7. SWGA testing loss (RMSE) for different numbers of chunks (N).

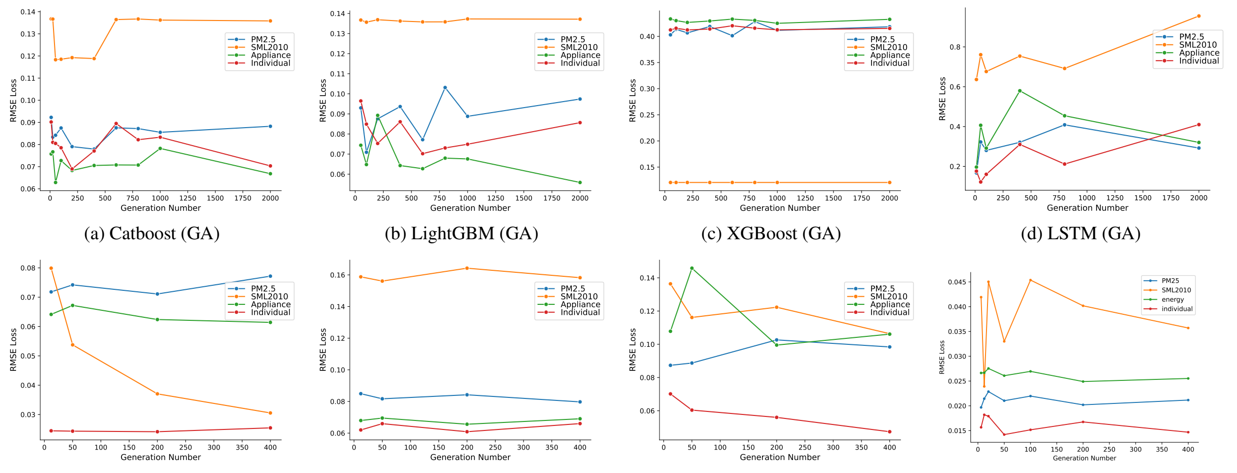
Besides, it demonstrates our SWGA method's capability of gaining such addition testing performance on a wide range of existing SOTA models in an easy plug-and-play manner.

Number of chunks. The above experiments set the number of training chunks N to 12 and it already produces a much better performance than the baseline GA. To investigate the effect of different Ns on the out-of-sample testing loss, we conduct experiments adjusting the value of N in SWGA. Figure 3.7 shows that different models and datasets have their own optimal N values. For instance, for XGBoost, SWGA with $N = 6$ exhibits the best out-of-sample RMSE for all four of

Table 3.5. The improvement of results on testing dataset of three SOTA time series forecasting models by using SWGA to search for a better set of hyperparameters. We calculate and show the average percentage of the reduction of the mean square error (MSE) after using SWGA to do hyperparameter search. We can see that Each of them has a considerable amount of free improvement without any change to their dataset and model architecture.

Model	Dataset					
	ETTh1	Weather	ETTm1	Exchange	ETTh2	ETTm2
iTransformer	1.92%	1.90%	2.20%	1.14%	0.80%	4.00%
DLinear	0.80%	2.84%	5.50%	1.30%	4.25%	6.05%
PatchTST	6.46%	1.14%	3.30%	6.13%	1.60%	3.39%

those datasets. The different effects from N further prove that our sliding window mechanism is meaningful and necessary for time series data.



(a) Catboost (GA) (b) LightGBM (GA) (c) XGBoost (GA) (d) LSTM (GA)

(e) Catboost (SWGA) (f) LightGBM (SWGA) (g) XGBoost (SWGA) (h) LSTM (SWGA)

Figure 3.9. Comparison of hyperparameter search dynamics using GA (top group: a-d) and SWGA (bottom group: e-h). RMSE loss of the final model on the out-of-sample testing set is shown for different numbers of generations for each algorithm.

Scalability. To examine how varying the number of distributed computer nodes impacts optimisation time. We conduct experiments by adjusting the number of nodes in SWGA. All experiments in this section are conducted using the ETTh1 dataset. As depicted in Figure 3.8, we observe a reduction in optimization time with an increase in the number of nodes, and this relationship appeared nearly linear. The results demonstrate the good scalability and efficiency of our proposed framework.

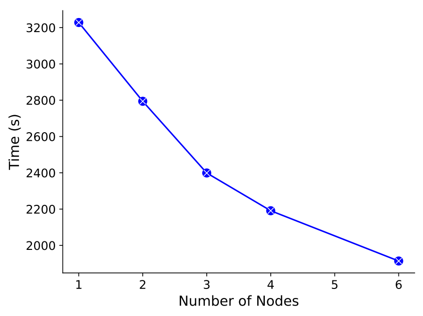


Figure 3.8. When the number of computation nodes increases, the optimization time decreases nearly linearly.

Optimization Dynamics. We conduct experiments to show the optimization dynamics of the baseline, GA algorithm on the four models on those four different datasets. Figure 3.9 have three major takeaways: (i) For all four models, on most of the datasets, SWGA is able to reach a much lower out-of-sample testing loss compared to the baseline GA. (ii) SWGA's out-of-sample testing loss decreases in a smoother way while the baseline GA's loss optimization process is much more volatile bouncing up and down. This indicates that it is safer to use the tuned hyperparameter configuration from SWGA compared to that from the baseline GA where there is a higher chance that the tuned configuration is on the out-of-sample testing loss peak that bounces up from a previous local minimum. (iii) In some cases such as the XGBoost case, the out-of-sample loss from the base GA fails to decrease properly while the SWGA is able to.

3.2.6. Conclusion

We propose SWGA, a distributed genetic algorithm for hyperparameter search for time series data. Compared to a regular genetic-based algorithm that uses random initialization to initialize the initial population, we propose a warm-up stage that uses TPE with a small number of trials to generate the initial population to provide a better starting point. To combat the distribution shift challenge on time series datasets, we propose a configurable sliding window mechanism. Besides, SWGA natively supports parallelized hyperparameter search on a Ray cluster. The experiment results on various models and time series datasets from different domains show that SWGA has a huge performance gain over the vanilla genetic algorithm. On average, there is a decrease of roughly

57.6% in the MAE and 54.6% in the RMSE when using SWGA in comparison to GA. Additionally, we also demonstrate the good scalability of SWGA.

3.3. STanHop: Sparse Tandem Hopfield Model for Memory-Enhanced Time Series Prediction

3.3.1. Introduction

In this work, we aim to enhance multivariate time series prediction by incorporating relevant additional information specific to the inference task at hand. This problem holds practical importance due to its wide range of real-world applications. On one hand, multivariate time series prediction itself poses a unique challenge given its multi-dimensional sequential structure and noise-sensitivity (Masini et al., 2023; Reneau et al., 2023; Nie et al., 2022; Fawaz et al., 2019). A proficient model should robustly not only discern the correlations between series within each time step, but also grasp the intricate dynamics of each series over time. On the other hand, in many real-world prediction tasks, one significant challenge with existing time series models is their slow responsiveness to sudden or rare events. For instance, events like the 2008 financial crisis and the pandemic-induced market turmoil in 2021 (Laborda and Olmo, 2021; Bond and Dow, 2021; Sevim et al., 2014; Bussiere and Fratzscher, 2006), or extreme climate changes in weather forecasting (Le et al., 2023; Sheshadri et al., 2021) often lead to compromised model performance. To combat these challenges, we present **STanHop-Net** (**S**parse **Tan**dem **Hop**field **Net**work), a novel Hopfield-based deep learning model, for multivariate time series prediction, equipped with optional memory-enhanced capabilities.

Our motivation comes from the connection between associative memory models of human brain (specifically, the modern Hopfield models) and the attention mechanism (Hu et al., 2023; Ramsauer et al., 2020). Based on this link, we propose to enhance time series models with external information (e.g., real-time or relevant auxiliary data) via the memory retrieval mechanism of Hopfield models. In its core, we utilize and extend the deep-learning-compatible Hopfield layers (Hu et al., 2023; Ramsauer et al., 2020). Differing from typical transformer-based architectures, these layers not only replace the attention mechanisms (Ramsauer et al., 2020; Widrich et al., 2020) but also serve as differentiable memory modules, enabling integration of external stimuli for enhanced predictions.

In this regard, we first introduce a set of generalized sparse Hopfield layers, as an extension of the sparse modern Hopfield model (Hu et al., 2023). Based on these layers, we propose a structure termed the **STanHop** (**S**parse **Tan**dem **Hop**field layers) block. In STanHop, there are two sequentially joined sub-blocks of generalized sparse Hopfield layers, hence tandem. This tandem design sparsely learn and store temporal and cross-series representations in a sequential manner.

Furthermore, we introduce **STanHop-Net** (**S**parse **Tan**dem **Hop**field **Net**work) for time series, consisting of multiple layers of STanHop blocks to cater for multi-resolution representation learning. To be more specific, rather than relying only on the input sequence for predictions, each stacked StanHop block is capable of incorporating additional information through the Hopfield models' memory retrieval mechanism from a pre-specified external memory set. This capability facilitates the injection of external memory at every resolution level when necessary. Consequently, STanHop-Net not only excels at making accurate predictions but also allows users to integrate additional information they consider valuable for their specific downstream inference tasks with minimal effort.

We provide visual overviews of STanHop-Net in fig. 3.10 and STanHop block in fig. 3.11.

Contributions. We summarize our contributions as follows:

- Theoretically, we introduce a unified sparsity-aware modern Hopfield model, termed the generalized sparse Hopfield model. We show that it not only offer a tighter memory retrieval error bound compared to the dense modern Hopfield model (Ramsauer et al., 2020), but also retains the robust theoretical properties of the dense model, such as fast fixed point convergence and exponential memory capacity. Moreover, it serves as a unified model that encompasses both the sparse (Hu et al., 2023) and dense (Ramsauer et al., 2020) models as its special cases.
- Computationally, we show the one-step approximation of the retrieval dynamics of the generalized sparse Hopfield model is connected to sparse attention mechanisms, akin to

(Hu et al., 2023; Ramsauer et al., 2020). This connection allows us to introduce the `GSH` layers featuring learnable sparsity, for time series representation learning. As a result, these layers achieve faster memory-retrieval convergence and greater noise-robustness compared to the dense model.

- Methodologically, with `GSH` layer, we present **STanHop** (**S**parse **Tan**dem **Hop**field layers) block, a hierarchical tandem Hopfield model design to capture the intrinsic multi-resolution structure of both temporal and cross-series dimensions of time series with resolution-specific sparsity at each level. In addition, we introduce the idea of pseudo-label retrieval, and debut two external memory plugin schemes — Plug-and-Play and Tune-and-Play memory plugin modules — for memory-enhanced predictions.
- Experimentally, we validate STanHop-Net in multivariate time series predictions, considering both with and without the incorporation of external memory. When external memory isn't utilized, STanHop-Net consistently matches or surpasses many popular baselines, across diverse real-world datasets. When external memory is utilized, STanHop-Net demonstrates further performance boosts in many settings, benefiting from both proposed external memory schemes.

Notations. We write $\langle \boldsymbol{a}, \boldsymbol{b} \rangle := \boldsymbol{a}^{\mathbb{T}} \boldsymbol{b}$ as the inner product for vectors $\boldsymbol{a}, \boldsymbol{b}$. The index set $\{1, \cdots, I\}$ is denoted by $[I]$, where $I \in \mathbb{N}_+$. The spectral norm is denoted by $\|\cdot\|$, which is equivalent to the l_2-norm when applied to a vector. Throughout this paper, we denote the memory patterns (keys) by $\boldsymbol{\xi} \in \mathbb{R}^d$ and the state/configuration/query pattern by $\boldsymbol{x} \in \mathbb{R}^d$ with $n := \|\boldsymbol{x}\|$, and $\Xi := (\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M) \in \mathbb{R}^{d \times M}$ as shorthand for stored memoery (key) patterns $\{\boldsymbol{\xi}_\mu\}_{\mu \in [M]}$. Moreover, we set $m := \mathrm{Max}_{\mu \in [M]} \|\boldsymbol{\xi}_\mu\|$ be the largest norm of memory patterns.

3.3.2. Background: Modern Hopfield Models

Let $\boldsymbol{x} \in \mathbb{R}^d$ be the query pattern and $\Xi = (\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_M) \in \mathbb{R}^{d \times M}$ be the M memory patterns.

Hopfield Models. Hopfield models are associative models that store a set of memory patterns Ξ in such a way that a stored pattern $\boldsymbol{\xi}_\mu$ can be retrieved based on a partially known or contaminated version, a query $\boldsymbol{x}$. The models achieve this by embedding the memories Ξ in the *energy landscape* $E(\boldsymbol{x})$ of a physical system (e.g., the Ising model in (Hopfield, 1982) or its higher-order generalizations (Lee et al., 1986; Peretto and Niez, 1986; Newman, 1988)), where each memory $\boldsymbol{\xi}_\mu$ corresponds to a local minimum. When a query $\boldsymbol{x}$ is introduced, the model initiates energy-minimizing *retrieval dynamics* $\mathcal{T}$ at the query's location. This process then navigate the energy landscape to locate the nearest local minimum $\boldsymbol{\xi}_\mu$, effectively retrieving the memory most similar to the query $\boldsymbol{x}$.

Constructing the energy function, $E(\boldsymbol{x})$, is straightforward. As outlined in (Krotov and Hopfield, 2016), memories get encoded into $E(\boldsymbol{x})$ using the *overlap-construction*: $E(\boldsymbol{x}) = F(\Xi^{\mathsf{T}}\boldsymbol{x})$, where $F : \mathbb{R}^M \to \mathbb{R}$ is a smooth function. This ensures that the memories $\{\boldsymbol{\xi}_\mu\}_{\mu\in[M]}$ sit at the stationary points of $E(\boldsymbol{x})$, given $\boldsymbol{\nabla}_{\boldsymbol{x}} F(\Xi^{\mathsf{T}}\boldsymbol{x})|_{\boldsymbol{\xi}_\mu} = 0$ for all $\mu \in [M]$. The choice of F results in different Hopfield model types, as demonstrated in (Krotov and Hopfield, 2016; Demircigil et al., 2017; Ramsauer et al., 2020; Krotov and Hopfield, 2020). However, determining a suitable retrieval dynamics, $\mathcal{T}$, for a given energy $E(\boldsymbol{x})$ is more challenging. For effective memory retrieval, $\mathcal{T}$ must:

(T1) Monotonically reduce $E(\boldsymbol{x})$ when applied iteratively.

(T2) Ensure its fixed points coincide with the stationary points of $E(\boldsymbol{x})$ for precise retrieval.

Modern Hopfield Models. Ramsauer et al. (2020) propose the modern Hopfield model with a specific set of E and $\mathcal{T}$ satisfying above requirements, and integrate it into deep learning architectures via its strong connection with attention mechanism, offering enhanced performance, and theoretically guaranteed exponential memory capacity. Specifically, they introduce

(3.3.1)
$$E(\boldsymbol{x}) = -\operatorname{lse}(\beta, \Xi^{\mathsf{T}}\boldsymbol{x}) + \frac{1}{2}\langle \boldsymbol{x}, \boldsymbol{x}\rangle + \text{Const.}, \quad \text{and} \quad \mathcal{T}_{\text{Dense}}(\boldsymbol{x}) = \Xi \operatorname{Softmax}(\beta \Xi^{\mathsf{T}}\boldsymbol{x}) = \boldsymbol{x}^{\text{new}},$$

where $\Xi^{\mathsf{T}}\boldsymbol{x} = (\langle \boldsymbol{\xi}_1, \boldsymbol{x}\rangle, \ldots, \langle \boldsymbol{\xi}_M, \boldsymbol{x}\rangle) \in \mathbb{R}^M$, $\operatorname{lse}(\beta, \boldsymbol{z}) := \log\Big(\sum_{m=1}^M \exp\{\beta z_\mu\}\Big)/\beta$ is the log-sum-exponential for any given vector $\boldsymbol{z} \in \mathbb{R}^M$ and $\beta > 0$. Their analysis concludes that:

- $\mathcal{T}_{\text{Dense}}$ converges well (Ramsauer et al., 2020, Theorem 1,2) and can retrieve patterns accurately in just one step (Ramsauer et al., 2020, Theorem 4), i.e. (T1) and (T2) are satisfied.
- The modern Hopfield model (3.3.1) possesses an exponential memory capacity in pattern size d (Ramsauer et al., 2020, Theorem 3).
- Notably, the one-step approximation of $\mathcal{T}_{\text{Dense}}$ mirrors the attention mechanism in transformers, leading to a novel deep architecture design: the Hopfield layers.

In a related vein, Hu et al. (2023) introduce a principled approach to constructing modern Hopfield models using the convex conjugate of the entropy regularizer. Unlike the original modern Hopfield model (Ramsauer et al., 2020), the key insight of (Hu et al., 2023) is that the convex conjugate of various entropic regularizers can yield distributions with varying degrees of sparsity. Leveraging this understanding, we introduce the generalized sparse Hopfield model in the next section.

3.3.3. Generalized Sparse Hopfield Model

In this section, we extend the entropic regularizer construction of the sparse modern Hopfield model (Hu et al., 2023) by replacing the Gini entropic regularizer with the Tsallis α-entropy (Tsallis, 1988),

$$\Psi_\alpha(\boldsymbol{p}) \coloneqq \begin{cases} \frac{1}{\alpha(\alpha-1)} \sum_{\mu=1}^{M} \Big(p_\mu - p_\mu^\alpha\Big), & \alpha \neq 1, \\ -\sum_{m=1}^{M} p_\mu \ln p_\mu, & \alpha = 1, \end{cases}, \quad \text{for } \alpha \geqslant 1, \tag{3.3.2}$$

thereby introducing the generalized sparse Hopfield model. Subsequently, we verify the connection between the memory retrieval dynamics of the generalized sparse Hopfield model and attention mechanism. This leads to the Generalized Sparse Hopfield (`GSH`) layers for deep learning.

3.3.3.1. Energy Function, Retrieval Dynamics and Fundamental Limits. Let $\boldsymbol{z}, \boldsymbol{p} \in \mathbb{R}^M$, and $\Delta^M \coloneqq \{\boldsymbol{p} \in \mathbb{R}_+^M \mid \sum_\mu^M p_\mu = 1\}$ be the $(M-1)$-dimensional unit simplex. **Energy Function.** We introduce the generalized sparse Hopfield energy function[3]:

$$\mathcal{H}(\boldsymbol{x}) = -\Psi_\alpha^\star\Big(\beta\Xi^\mathbb{T}\boldsymbol{x}\Big) + \frac{1}{2}\langle\boldsymbol{x},\boldsymbol{x}\rangle + \text{Const.}, \quad \text{with } \Psi_\alpha^\star(\boldsymbol{z}) \coloneqq \int \mathrm{d}\boldsymbol{z}\,\alpha\text{-EntMax}(\boldsymbol{z}), \tag{3.3.3}$$

where $\alpha\text{-EntMax}(\cdot) : \mathbb{R}^M \to \Delta^M$ is a finite-domain distribution map defined as follows.

Definition 3.3.1. The variational form of α-EntMax is defined by the optimization problem

$$\alpha\text{-EntMax}(\boldsymbol{z}) \coloneqq \underset{\boldsymbol{p}\in\Delta^M}{\operatorname{argmax}}[\langle\boldsymbol{p},\boldsymbol{z}\rangle - \Psi_\alpha(\boldsymbol{p})], \tag{3.3.4}$$

where $\Psi_\alpha(\cdot)$ is the Tsallis entropic regularizer given by (3.3.2). See Remark B.3.1 for a closed form.

$\Psi_\alpha^\star(\boldsymbol{p})$ is the convex conjugate of the Tsallis entropic regularizer $\Psi_\alpha(\boldsymbol{p})$ (Definition B.3.1) and hence

Lemma 3.3.1. $\nabla\Psi_\alpha^\star(\boldsymbol{z}) = \operatorname{argmax}_{\boldsymbol{p}\in\Delta^M}[\langle\boldsymbol{p},\boldsymbol{z}\rangle - \Psi_\alpha(\boldsymbol{p})] = \alpha\text{-EntMax}(\boldsymbol{z})$.

Proof. See appendix B.3.2.1 for a detailed proof. □

Retrieval Dynamics. With Lemma 3.3.1, it is clear to see that the energy function (3.3.3) aligns with the overlap-function construction of Hopfield models, as in (Hu et al., 2023; Ramsauer et al., 2020). Next, we introduce the corresponding retrieval dynamics satisfying the monotinicity property (T1).

Lemma 3.3.2 (Generalized Sparse Hopfield Retrieval Dynamics)**.** Let t be the iteration number. The retrieval dynamics of the generalized sparse Hopfield model is a 1-step update of the form

$$\mathcal{T}(\boldsymbol{x}_t) \coloneqq \nabla_{\boldsymbol{x}}\Psi_\alpha^\star\Big(\beta\Xi^\mathbb{T}\boldsymbol{x}_t\Big) = \Xi\,\alpha\text{-EntMax}\Big(\beta\Xi^\mathbb{T}\boldsymbol{x}_t\Big) = \boldsymbol{x}_{t+1}, \tag{3.3.5}$$

that minimizes the energy function (3.3.3) monotonically over t.

Proof. See appendix B.3.2.2 for a detailed proof. □

To see how this model store and retrieve memory patterns, we first introduce the following definition.

[3]This energy function (3.3.3) is equivalent up to an additive constant.

Definition 3.3.2 (Stored and Retrieved)**.** Assuming that every pattern $\boldsymbol{\xi}_\mu$ surrounded by a sphere S_μ with finite radius $R \coloneqq \frac{1}{2}\operatorname{Min}_{\mu,\nu\neq\mu\in[M]}\|\boldsymbol{\xi}_\mu - \boldsymbol{\xi}_\nu\|$, we say $\boldsymbol{\xi}_\mu$ is *stored* if there exists a generalized fixed point of $\mathcal{T}$, $\boldsymbol{x}^\star_\mu \in S_\mu$, to which all limit points $\boldsymbol{x} \in S_\mu$ converge to, and $S_\mu \cap S_\nu = \emptyset$ for $\mu \neq \nu$. We say $\boldsymbol{\xi}_\mu$ is ϵ*-retrieved* by $\mathcal{T}$ with $\boldsymbol{x}$ for an error ϵ, if $\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant \epsilon$.

To ensure the convergence property (T2) of retrieval dynamics (3.3.5), we have the next lemma.

Lemma 3.3.3 (Convergence of Retrieval Dynamics $\mathcal{T}$)**.** Given the energy function (3.3.3) and retrieval dynamics $\mathcal{T}(\boldsymbol{x})$ (3.3.5), respectively. For any sequence $\{\mathrm{x}_t\}_{t=0}^\infty$ generated by the iteration $\mathrm{x}_{t'+1} = \mathcal{T}(\mathrm{x}_{t'})$, all limit points of this sequence are stationary points of $\mathcal{H}$.

Proof. See appendix B.3.2.3 for a detailed proof. □

Intuitively, Lemma 3.3.3 suggests that for any query $\boldsymbol{x}$, $\mathcal{T}$ (given by (3.3.5)) monotonically and iteratively approaches stationary points of $\mathcal{H}$ (given by (3.3.3)), where the memory patterns $\{\boldsymbol{\xi}_\mu\}_{\mu\in[M]}$ are stored. This completes the construction of a well-defined modern Hopfield model.

Fundamental Limits. To highlight the computational benefits of the generalized sparse Hopfield model, we analyze the fundamental limits of the memory retrieval error and memory capacity.

Theorem 3.3.1 (Retrieval Error)**.** Let $\mathcal{T}_{\text{Dense}}$ be the retrieval dynamics of the dense modern Hopfield model (Ramsauer et al., 2020). Let $\boldsymbol{z} \in \mathbb{R}^M$, $z_{(\nu)}$ be the ν'th element in a sorted descending z-sequence $\boldsymbol{z}_{\text{sorted}} \coloneqq z_{(1)} \geqslant \ldots \geqslant z_{(M)}$, and $\kappa(\boldsymbol{z}) \coloneqq \operatorname{Max}\{k \in [M] \mid 1 + kz_{(k)} > \sum_{\nu\leqslant k} z_{(\nu)}\}$. For all $\boldsymbol{x} \in S_\mu$, it holds $\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$, and

$$\text{for } 2 \geqslant \alpha \geqslant 1, \quad \|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant 2m(M-1)\exp\left\{-\beta\left(\langle\boldsymbol{\xi}_\mu, \boldsymbol{x}\rangle - \operatorname*{Max}_{\nu\in[M]}\langle\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\nu\rangle\right)\right\}, \tag{3.3.6}$$

$$\text{for } \alpha \geqslant 2, \quad \|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant m + d^{1/2}m\beta\left[\kappa\left(\operatorname*{Max}_{\nu\in[M]}\langle\boldsymbol{\xi}_\nu, \boldsymbol{x}\rangle - \left[\Xi^{\mathbb{T}}\boldsymbol{x}\right]_{(\kappa)}\right) + \frac{1}{\beta}\right]. \tag{3.3.7}$$

Corollary 3.3.1.1 (Noise-Robustness)**.** In cases of noisy patterns with noise $\boldsymbol{\eta}$, i.e. $\widetilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta}$ (noise in query) or $\widetilde{\boldsymbol{\xi}}_\mu = \boldsymbol{\xi}_\mu + \boldsymbol{\eta}$ (noise in memory), the impact of noise $\boldsymbol{\eta}$ on the sparse retrieval error $\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$ is linear for $\alpha \geqslant 2$, while its effect on the dense retrieval error $\|\mathcal{T}_{\text{Dense}}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$ (or $\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$ with $2 \geqslant \alpha \geqslant 1$) is exponential.

Proof. See appendix B.3.2.4 for a detailed proof. □

Intuitively, Theorem 3.3.1 implies the sparse model converge faster to memory patterns than the dense model (Ramsauer et al., 2020), and the larger sparsity leads the lower retrieval error.

Lemma 3.3.4 (Memory Capacity Lower Bound)**.** Suppose the probability of successfully storing and retrieving memory pattern is given by $1-p$. The number of memory patterns sampled from a sphere of radius m that the sparse Hopfield model can store and retrieve has a lower bound: $M \geqslant \sqrt{p}C^{\frac{d-1}{4}}$, where C is the solution for $C = {}^{b}\!/_{W_0(\exp\{a+\ln b\})}$ with $W_0(\cdot)$ being the principal branch of Lambert W function (Olver et al., 2010), $a := {}^{4}\!/_{d-1}\Big\{\ln\big[{}^{2m(\sqrt{p}-1)}\!/_{(R+\delta)}\big]+1\Big\}$ and $b := {}^{4m^2\beta}\!/_{5(d-1)}$. For sufficiently large β, the sparse Hopfield model has a larger lower bound on the exponential-in-d memory capacity compared to that of dense counterpart (Ramsauer et al., 2020): $M \geqslant M_{\text{Dense}}$.

Proof. See appendix B.3.2.5 for a detailed proof. $\square$

Lemma 3.3.4 offers a lower bound on the count of patterns effectively stored and retrievable by $\mathcal{T}$ with a minimum precision of R, as defined in Definition 3.3.2. Essentially, the capacity of the generalized sparse Hopfield model to store and retrieve patterns grows exponentially with pattern size d. This mirrors findings in (Hu et al., 2023; Ramsauer et al., 2020). Notably, when $\alpha = 2$, the results of Theorem 3.3.1 and Lemma 3.3.4 reduce to those of (Hu et al., 2023).

3.3.3.2. Generalized Sparse Hopfield (GSH) Layers for Deep Learning. Now we introduce the Generalized Sparse Hopfield (GSH) layers for deep learning, by drawing the connection between the generalized sparse Hopfield model and attention mechanism.

Generalized Sparse Hopfield (`GSH`) Layer. Following (Hu et al., 2023), we extend (3.3.5) to multiple queries $\boldsymbol{X} := \{\boldsymbol{x}_i\}_{i\in[T]}$. From previous section, we say that the Hopfield model, as defined by (3.3.3) and (3.3.5), functions within the associative spaces $\boldsymbol{X}$ and Ξ. Given any *raw* query $\boldsymbol{R}$ and memory $\boldsymbol{Y}$ that are input into the Hopfield model[4], we compute $\boldsymbol{X}$ and Ξ as $\boldsymbol{X}^{\mathbb{T}} = \boldsymbol{R}\boldsymbol{W}_Q := \boldsymbol{Q}$ and $\Xi^{\mathbb{T}} = \boldsymbol{Y}\boldsymbol{W}_K := \boldsymbol{K}$, using matrices $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$. Therefore, we rewrite $\mathcal{T}$ in (3.3.5) as

[4]The *raw* query $\boldsymbol{R}$ and memory $\boldsymbol{Y}$ may originate from data, external sets, or hidden representations throughout a given deep learning pipeline. They are not necessarily usable as $\boldsymbol{X}$ and Ξ. Therefore, to use (3.3.5), they must be mapped into d-dimensional associative spaces.

$(\boldsymbol{Q}^{\text{new}})^{\mathbb{T}} = \boldsymbol{K}^{\mathbb{T}}\,\alpha\text{-EntMax}\Big(\beta\boldsymbol{K}\boldsymbol{Q}^{\mathbb{T}}\Big)$. Taking transpose and projecting $\boldsymbol{K}$ to $\boldsymbol{V}$ with $\boldsymbol{W}_V$, we have

$$\boldsymbol{Z} \coloneqq \boldsymbol{Q}^{\text{new}}\boldsymbol{W}_V = \alpha\text{-EntMax}\Big(\beta\boldsymbol{Q}\boldsymbol{K}^{\mathbb{T}}\Big)\boldsymbol{K}\boldsymbol{W}_V = \alpha\text{-EntMax}\Big(\beta\boldsymbol{Q}\boldsymbol{K}^{\mathbb{T}}\Big)\boldsymbol{V}, \tag{3.3.8}$$

which leads to the attention mechanism with $\alpha\text{-EntMax}$ activation function. Plugging back the raw patterns $\boldsymbol{R}$ and $\boldsymbol{Y}$, we arrive the foundation of the Generalized Sparse Hopfield (GSH) layer,

$$\texttt{GSH}(\boldsymbol{R}, \boldsymbol{Y}) = \boldsymbol{Z} = \alpha\text{-EntMax}\Big(\beta\boldsymbol{R}\boldsymbol{W}_Q\boldsymbol{W}_K^{\mathbb{T}}\boldsymbol{Y}^{\mathbb{T}}\Big)\boldsymbol{Y}\boldsymbol{W}_K\boldsymbol{W}_V. \tag{3.3.9}$$

By (3.3.6), $\mathcal{T}$ retrieves memory patterns with high accuracy after a single activation. This allows (3.3.9) to integrate with deep learning architectures just like (Hu et al., 2023; Ramsauer et al., 2020).

Remark 3.3.1. α is a learnable parameter (Correia et al., 2019), enabling GSH to learn input sparsity.

GSHPooling and GSHLayer Layers. Following (Hu et al., 2023), we introduce two more variants: the GSHPooling and GSHLayer layers. They are similar to the GSH, and only differ in how to obtain the associative sets $\boldsymbol{Q}, \boldsymbol{Y}$. For GHSPooling$(\boldsymbol{Y})$, $\boldsymbol{K} = \boldsymbol{Y}\boldsymbol{W}_K, \boldsymbol{V} = \boldsymbol{K}\boldsymbol{W}_V$, and $\boldsymbol{Q}$ is a learnable variable independent from any input. For GSHLayer$(\boldsymbol{R}, \boldsymbol{Y})$, we have $\boldsymbol{K} = \boldsymbol{V} = \boldsymbol{Y}$, and $\boldsymbol{Q} = \boldsymbol{R}$. Note that GSHLayer can have $\boldsymbol{Q}$ as learnable parameter or as an input. Where if $\boldsymbol{Q}$ was served as an input, the whole GSHLayer has no learnable parameters and can be used as a lookup table. We provide an example of memory retrieval for image completion using GSHLayer in appendix B.3.3.3.

3.3.4. Methodology

In this section, we introduce a Hopfield-based deep architecture (STanHop-Net) tailored for memory-enhanced learning of noisy multivariate time series. These additional memory-enhanced functionalities enable STanHop-Net to effectively handle the problem of slow response to sudden or rare events (e.g, 2021 pandemic meltdown in financial market) by making predictions using both

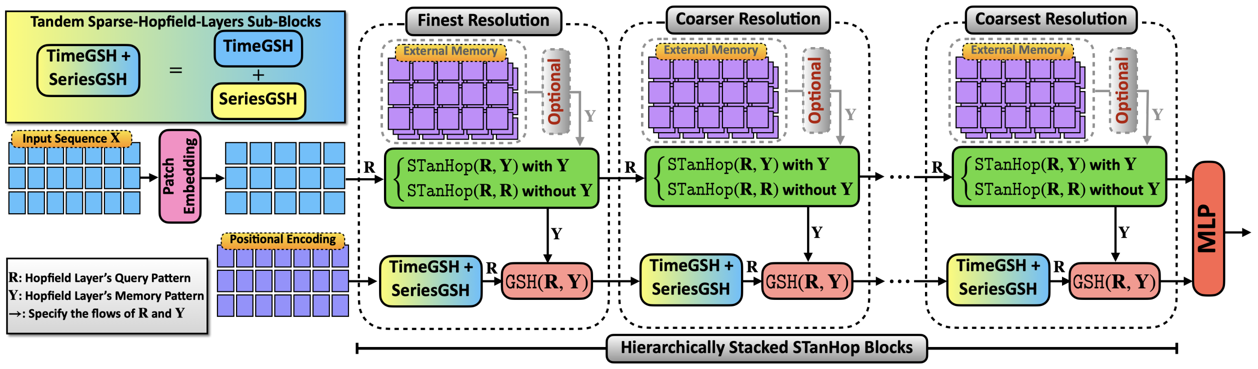

Tandem Sparse-Hopfield-Layers Sub-Blocks
TimeGSH + SeriesGSH = TimeGSH + SeriesGSH
Input Sequence X
Patch Embedding
Positional Encoding
R: Hopfield Layer's Query Pattern
Y: Hopfield Layer's Memory Pattern
→: Specify the flows of R and Y
Finest Resolution
Coarser Resolution
Coarsest Resolution
External Memory
Optional
STanHop(R, Y) with Y
STanHop(R, R) without Y
GSH(R, Y)
MLP
Hierarchically Stacked STanHop Blocks

Figure 3.10. **STanHop-Net Overview. Patch Embedding:** Given an input multivariate time series $\boldsymbol{X} \in \mathbb{R}^{C\times T\times d}$ consisting C univariate series, T time steps and d features, the patch embedding aggregates temporal information for each univariate series, subsequently reducing temporal dimensionality from T to $P = T/P$ for all d features. **STanHop Block:** The STanHop block leverages the Generalized Sparse Hopfield (GSH) model (section 3.3.3). It captures time series representations from its input through two tandem sparse-Hopfield-layers sub-blocks (i.e. TimeGSH and SeriesGSH, see fig. 3.11), catering to both temporal and cross-series dimensions. **STanHop-Net:** Using a stacked encoder-decoder structure, STanHop-Net facilitates hierarchical multi-resolution learning. This design allows STanHop-Net to extract distill representations from both temporal and cross-series dimensions across multiple scales (multi-resolution in a hardwired fashion via coarse-graining layers, see section 3.3.4.4). Moreover, each stacked block has optional external memory plugin functionalities for enhanced predictions (section 3.3.4.3). These representations from all resolutions are then merged, providing a holistic representation learning for downstream predictions specially tailored for time series data.

in-context inputs (e.g., historical data) and external stimuli (e.g., real-time or relevant past data). In the following, we consider multivariate time series $\boldsymbol{X} \in \mathbb{R}^{C\times T\times d}$ comprised of C univariate series. Each univariate series has T time steps and d features.

3.3.4.1. Patched Embedding. Motivated by (Zhang and Yan, 2023), we use a patching technique on model input that groups adjacent time steps into subseries patches. This method extends the input time horizon without altering token length, enabling us to capture local semantics and critical information more effectively, which is often missed at the point-level. We define the multivariate input sequence as $\boldsymbol{X} \in \mathbb{R}^{C\times T\times d}$, where C, T, d denotes the number of variates, number of time steps and the number of dimensions of each variate. Given a time series sequence $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_T\}$ and a patch size P, the patching operation divides X into $\boldsymbol{S} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_{T/P}\}$. For each patched sequence $\boldsymbol{s}_i \in \mathbb{R}^{C\times P\times d}$ for $i \in [T/P]$, we define the patched embedding as $\text{EMB}(\boldsymbol{s}_i) = \mathbf{E}^{\text{emb}}\boldsymbol{s}_i + \mathbf{E}^{\text{pos}}(i) \in \mathbb{R}^{D_{\text{emb}}}$, where D_{emb} is the embedding dimension, $\mathbf{E}^{\text{emb}} \in \mathbb{R}^{D_{\text{emb}}\times P}$, and $\mathbf{E}^{\text{pos}} \in \mathbb{R}^{T/P\times D_{\text{emb}}}$ is the positional encoding.

When T is not divisible to P, assuming $T = P \times C_n + c$ with $C_n, c \in \mathbb{N}_+$, we pad the sequence by repeating the first c elements in the sequence. Consequently, this patching embedding significantly improves computational efficiency and memory usage.

3.3.4.2. STanHop: Sparse Tandem Hopfield Block. We introduce STanHop (**S**parse **Tan**dem **Hop**field) block which comprises one `GSHLayer`-based external memory plugin module, and two tandem sub-blocks of `GSH` layers to process both time and series dimensions, i.e. TimeGSH and SeriesGSH sub-blocks in fig. 3.11. In essence, STanHop not only sequentially extracts temporal and cross-series information of multivariate time series with (learnable) data-dependent sparsity, but also utilizes both acquired (in-context) representations and external stimulus through the memory plugin modules for the downstream prediction tasks.

Given a hidden vector, $\boldsymbol{R} \in \mathbb{R}^{C\times T\times D_{\text{hidden}}}$, and its corresponding external memory set $\boldsymbol{Y} \in \mathbb{R}^{M\times C\times T\times D_{\text{hidden}}}$, where C denotes the channel number and T denotes the number of time segments (patched time steps), To clarify, the `GSH` layer only operates on the last two dimensions, i.e., $t \in T$ and $d \in D_{\text{hidden}}$. Thus, the operation $\texttt{GSH}(\boldsymbol{Z}, \boldsymbol{Z})$ extracts information of the temporal dynamics of $\boldsymbol{Z}$ from the segmented time series. Here we define the dimensional transpose operation T. For a given tensor $\boldsymbol{X} \in \mathbb{R}^{a\times b\times c}$, we have $\mathsf{T}^{acb}_{abc}(\boldsymbol{X}) \coloneqq \boldsymbol{X}' \in \mathbb{R}^{a\times c\times b}$, i.e. this operation rearranges the dimensions of the original tensor $\boldsymbol{X}$ from (a, b, c) to a new order (a, c, b). Given a set of query pattern $\boldsymbol{Q} \in \mathbb{R}^{\text{len}_Q \times D_{\text{hidden}}}$, we define a single block of STanHop as

(Memory Plugin Module, see section 3.3.4.3)

$$\boldsymbol{Z} = \texttt{Memory}(\boldsymbol{R}, \boldsymbol{Y}),$$

(Temporal GSH) $$\boldsymbol{Z}^t = \texttt{T}^{cth}_{tch}(\text{LayerNorm}(\text{Z} + \text{FF}(\texttt{GSH}(\text{Z}, \text{Z})))) \in \mathbb{R}^{T \times C \times D_{\text{hidden}}},$$

($\boldsymbol{R}^\star$ is learnable and randomly initialized)

$$\boldsymbol{Z}^p = \texttt{GSHPooling}\Big(\boldsymbol{R}^\star, \boldsymbol{Z}^t\Big) \in \mathbb{R}^{T \times \text{len}_Q \times D_{\text{hidden}}},$$

(Cross-series GSH) $$\boldsymbol{Z}^c = \texttt{GSH}\Big(\boldsymbol{Z}^t, \boldsymbol{Z}^p\Big) \in \mathbb{R}^{T \times C \times D_{\text{hidden}}},$$

$$\boldsymbol{Z}^* = \text{LayerNorm}\Big(\text{Z}^{\text{t}} + \text{FF}(\text{Z}^{\text{c}})\Big) \in \mathbb{R}^{\text{T} \times \text{C} \times \text{D}_{\text{hidden}}},$$

$$\boldsymbol{Z}_{\text{out}} = \text{LayerNorm}(\text{Z}^* + \text{FF}(\text{Z}^*)) \in \mathbb{R}^{\text{T} \times \text{C} \times \text{D}_{\text{hidden}}},$$

where $\texttt{Memory}(\cdot, \cdot)$ is the external memory plugin module introduced in the next section. Note that, if we choose to turn off the external memory functionalities (or external memory is not available) during training, we set $\boldsymbol{Y} = \boldsymbol{R}$ such that $\texttt{Memory}(\boldsymbol{R}, \boldsymbol{R}) = \boldsymbol{R}$ (see section 3.3.4.3 for details). Here $\texttt{GSHPooling}(\boldsymbol{R}^\star, \boldsymbol{Z}^t)$ takes $\boldsymbol{Z}^t$ and a randomly initialized query $\boldsymbol{R}^\star$ as input. Importantly, $\boldsymbol{R}^\star$ not only acts as learnable prototype patterns learned by pooling over $\boldsymbol{Z}^t$, but also as a knob to control the computational complexity by picking the hidden dimension of $\boldsymbol{R}^\star$. We summarize the STanHop block as $\boldsymbol{Z}_{\text{out}} = \texttt{STanHop}(\boldsymbol{R}, \boldsymbol{Y}) \in \mathbb{R}^{T \times C \times D_{\text{hidden}}}$.

3.3.4.3. External Memory Plugin Module and Pseudo-Label Retrieval. Here we introduce the external memory modules (i.e., $\texttt{Memory}(\cdot, \cdot)$ in section 3.3.4.2 or Memory Plugin blocks in fig. 3.11) for external memory functionalities. These modules are tailored for time series modeling by incorporating task-specific supplemental information (such as relevant historical data for sudden or rare events predictions) for subsequent inference. To this end, we introduce two memory plugin modules: **Plug-and-Play Memory Plugin** and **Tune-and-Play Memory Plugin**. For query $\boldsymbol{R}$ and memory $\boldsymbol{Y}$, we denote them by $\texttt{PlugMemory}(\boldsymbol{R}, \boldsymbol{Y})$ and $\texttt{TuneMemory}(\boldsymbol{R}, \boldsymbol{Y})$.

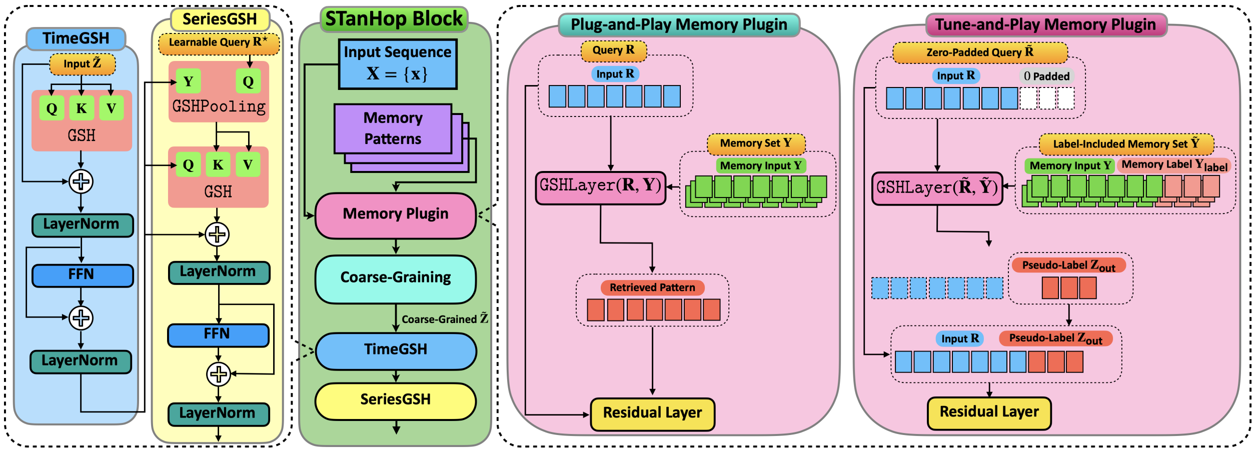


Figure 3.11. **STanHop Block.** **(Left)** Tandem Hopfield-Layer Blocks: TimeGSH and SeriesGSH. Notably, in the `GSHPooling` block of SeriesGSH, the learnable query $\boldsymbol{R}^{\star}$ is initialized randomly and employed to store learned prototype patterns from temporal representations extracted during training. **(Right)** Plug-and-Play and Tune-and-Play Memory Plugins.

Plug-and-Play Memory Plugin. This module enables performance enhancement utilizing external memory without any fine-tuning. Given a trained STanHop-Net (without external memory), we use a parameter fixed `GSHLayer` for memory retrieval. Explicitly, given an input sequence $\boldsymbol{R} \in \mathbb{R}^{|R| \times D_{\text{hidden}}}$ and a corresponding external memory set $\boldsymbol{Y} \in \mathbb{R}^{M \times |R| \times d}$, where $|\boldsymbol{R}|$ and D_{hidden} are the sequence length and hidden dimension of $\boldsymbol{R}$ respectively. We define the memory retrieval operation as $\boldsymbol{Z} = \texttt{PlugMemory}(\boldsymbol{R}, \boldsymbol{Y}) = \text{LayerNorm}(\boldsymbol{R} + \texttt{GSHLayer}(\boldsymbol{R}, \boldsymbol{Y}))$ with all parameters fixed.

Tune-and-Play Memory Plugin. Here we propose the idea of "pseudo-label retrieval" using `GSHLayer` for time series prediction. Specifically, we use modern Hopfield models' memory retrieval mechanism to generate pseudo-labels for a given $\boldsymbol{R}$ from a *label-included* memory set $\widetilde{\boldsymbol{Y}}$, thereby enhancing predictions. Intuitively, this method supplements predictions by learning from demonstrations and we use the retrieved pseudo-labels (i.e., learned *pseudo*-predictions) as additional features. An illustration of this mechanism is shown in fig. 3.11. Firstly, we prepare the *label-included* external memory as $\widetilde{\boldsymbol{Y}} = \boldsymbol{Y} \oplus \boldsymbol{Y}_{\text{label}}$, where $\widetilde{\boldsymbol{Y}}$ is the concatenation of memory sequences and their corresponding labels. Next, we denote the padded $\boldsymbol{R}$ as $\widetilde{\boldsymbol{R}}$, where $\widetilde{\boldsymbol{R}} \in \mathbb{R}^{|\widetilde{\boldsymbol{Y}}| \times d}$. And we utilize the `GSHLayer` to retrieve the pseudo-label from the memory sequences as $\boldsymbol{Z}_{\text{out}}$. Then we concatenate $\boldsymbol{R}$

and the pseudo-label $\boldsymbol{Z}_{\text{out}}$ and send it to a feed forward layer to encode the pseudo-label information: $\boldsymbol{Z}_{\text{out}} = \texttt{GSHLayer}(\widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{Y}})$, $\boldsymbol{Z}_{\text{pseudo}} = \boldsymbol{R} \oplus \boldsymbol{Z}_{\text{out}}$ and then $\widetilde{\boldsymbol{Z}} = \text{LayerNorm}(\text{FF}(\boldsymbol{Z}_{\text{pseudo}}) + \boldsymbol{Z}_{\text{pseudo}})$. In other words, we first obtain a weight matrix from the association between $\widetilde{\boldsymbol{R}}$ and $\widetilde{\boldsymbol{Y}}$, and then multiply this weight matrix with $\boldsymbol{Y}_{\text{label}}$ to obtain $\boldsymbol{Z}_{\text{out}}$. We summarize the Tune-and-Play memory plugin as $\widetilde{\boldsymbol{Z}} = \texttt{TuneMemory}(\boldsymbol{R}, \boldsymbol{Y})$.

3.3.4.4. Coarse-Graining. To cope with the intrinsic multi-resolution inductive bias of time series, we introduce a coarse-graining layer in each STanHop block. Given an hidden vector output, $\boldsymbol{Z} \in \mathbb{R}^{C\times T\times D_{\text{hidden}}}$, grain level Δ, and a weight matrix $\mathbf{W} \in \mathbb{R}^{D_{\text{hidden}}\times 2D_{\text{hidden}}}$, and $\oplus$ denotes the concatenation operation. We denote $\boldsymbol{Z}_{c,t,d}$ with $c \in [C], t \in [T], d \in [D_{\text{hidden}}]$ as the element representing the c-th series, t-th time segment, and d-th dimension. The coarse-graining layer consists a vector concatenation and a matrix multiplication: $\widehat{\boldsymbol{Z}}_{c,t,:} = \boldsymbol{Z}_{c,t,:} \oplus \boldsymbol{Z}_{c,t+\Delta,:} \in \mathbb{R}^{2D_{\text{hidden}}}$ and then $\widetilde{\boldsymbol{Z}}_{c,t,:} = \mathbf{W}\widehat{\boldsymbol{Z}}_{c,t,:} \in \mathbb{R}^{D_{\text{hidden}}}$, such that $\widehat{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times 2D_{\text{hidden}}}$ and $\widetilde{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times D_{\text{hidden}}}$, similar to (Liu et al., 2021b; Zhang and Yan, 2023). Operationally, it first obtains the representation of smaller resolution, and then distills information via a linear transformation. We express this course-graining layer as $\text{CoarseGrain}(\text{Z}, \Delta) = \widetilde{\text{Z}}$.

3.3.4.5. Multi-Layer STanHop for Multi-Resolution Learning. Finally, we construct the STanHop-Net by stacking STanHop blocks in a hierarchical fashion, enabling multi-resolution feature extraction with resolution-specific sparsity. Given a prediction window size $P \in \mathbb{R}$, number of layer $L \in \mathbb{R}$, and a learnable positional embedding for the decoder $\boldsymbol{E}_{\text{dec}}$, we construct our multi-layer STanHop as an autoencoder structure. The encoder structure consists of a course-graining operation first, following by an `STanHop` layer. The decoder follows the similar structure as the standard transformer decoder (Vaswani et al., 2017), but we replace the cross-attention mechanism to a `GSH` layer, and self-attention layer as `STanHop` layer. We summarize the STanHop-Net network structure in fig. 3.10, and in algorithm 8 in appendix.

3.3.5. Experiments

We demonstrate the validity of STanHop-Net and external memory modules by testing them on various experimental settings with both synthetic and real-world datasets.

3.3.5.1. Multivariate Time Series Prediction without external memory. table 3.6 includes the experiment results of the multivariate time series predictions using STanHop-Net without external memory. We implement three variants of STanHop-Net: **StanHop-Net**, **StanHop-Net (D)** and **StanHop-Net (S)**, with `GSH`, `Hopfield` (Ramsauer et al., 2020) and `SparseHopfield` (Hu et al., 2023) layers respectively. Our results show that in 47 out of 58 cases, STanHop-Nets rank in the top two, delivering top-tier performance compared to all baselines.

Data. Following (Zhang and Yan, 2023; Zhou et al., 2022; Wu et al., 2021), we use 6 realistic datasets: ETTh1 (Electricity Transformer Temperature-hourly), ETTm1 (Electricity Transformer Temperature-minutely), WTH (Weather), ECL (Electricity Consuming Load), ILI (Influenza-Like Illness), Traffic. The first four datasets are split into train/val/test ratios of 14/5/5, and the last two are split into 7/1/2. **Metrics.** We use Mean Square Error (MSE) and Mean Absolute Error (MAE) as accuracy metrics. **Setup.** Here we use the same setting as in (Zhang and Yan, 2022): multivariate time series predictions tasks on 6 real-world datasets. For each dataset, we evaluate our models with several different prediction horizons. For all experiments, we report the mean MSE, MAE over 10 runs. **Baselines.** We benchmark our method against 5 leading methods listed in table 3.6. Baseline results are quoted from competing papers when possible and reproduced otherwise. **Hyperparameters.** For each experiment, we optimize the hyperparameters using the "sweep" function from Weights and Biases (Biewald et al., 2020). We conduct 100 random search iterations for each setting, selecting the best set based on the validation performance.

For datasets, hyperparameter tuning, implementations and training details, please see appendix B.3.5.

3.3.5.2. Memory-Enhanced Prediction: Memory Plugin via Hopfield Layer. In table 3.7 and fig. 3.12, we showcase STanHop-Net with external memory enhancements delivers performance

Table 3.6. **Accuracy Comparison for Multivariate Time Series Predictions without External Memory.** We implement 3 STanHop variants, **STanHop-Net (D)** with **D**ense `Hopfield` layer (Ramsauer et al., 2020), **STanHop-Net (S)** with **S**parse `SparseHopfield` layer (Hu et al., 2023) and **STanHop-Net** with our `GSH` layer respectively. We report the average Mean Square Error (MSE) and Mean Absolute Error (MAE) metrics of 10 runs, with variance omitted as they are all $\leqslant 0.44\%$. We benchmark our method against leading transformer-based methods (FEDformer (Zhou et al., 2022), Informer (Zhou et al., 2021) and Autoformer (Wu et al., 2021), Crossformer (Zhang and Yan, 2022)) and a linear model with seasonal-trend decomposition (DLinear (Zeng et al., 2023)). We evaluate each dataset with different prediction horizons (showed in the second column). We have the best results **bolded** and the second best results underlined. In 47 out of 58 settings, STanHop-Nets rank either first or second. Our results indicate that our proposed STanHop-Net delivers consistent top-tier performance compared to all the baselines, even without external memory.

Models		FEDFormer		DLinear		Informer		Autoformer		Crossformer		**STanHop-Net (D)**		**STanHop-Net (S)**		**STanHop-Net**	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.318	0.384	0.312	0.355	0.577	0.549	0.439	0.440	0.305	0.367	0.301	0.363	0.298	0.360	**0.294**	**0.351**
	48	0.342	0.396	0.352	**0.383**	0.685	0.625	0.429	0.442	0.352	0.394	0.356	0.406	0.355	0.399	**0.340**	0.387
	168	0.412	0.449	0.416	**0.430**	0.931	0.752	0.493	0.479	0.410	0.441	**0.398**	0.440	0.419	0.458	**0.398**	0.437
	336	0.456	0.474	0.450	**0.452**	1.128	0.873	0.509	0.492	**0.440**	0.461	0.458	0.472	0.484	0.484	0.450	0.472
	720	0.521	0.515	**0.484**	**0.501**	1.215	0.896	0.539	0.537	0.519	0.524	0.516	0.522	0.541	0.533	0.512	0.511
ETTm1	24	0.290	0.364	0.217	0.289	0.323	0.369	0.410	0.428	0.211	0.293	0.205	0.278	**0.191**	**0.270**	0.195	0.273
	48	0.342	0.396	0.278	**0.330**	0.494	0.503	0.483	0.464	0.300	0.352	0.303	0.340	0.293	0.341	**0.270**	0.333
	96	0.366	0.412	0.310	0.354	0.678	0.614	0.502	0.476	0.320	0.373	0.325	0.377	0.322	0.362	**0.286**	**0.352**
	288	0.398	0.433	**0.369**	**0.386**	1.056	0.786	0.604	0.522	0.404	0.427	0.410	0.429	0.395	0.413	0.366	0.399
	672	0.455	0.464	0.416	**0.417**	1.192	0.926	0.607	0.530	0.569	0.528	0.574	0.516	0.556	0.510	**0.400**	0.431
ECL	48	0.229	0.338	0.155	0.258	0.344	0.393	0.241	0.351	0.156	0.255	0.159	0.264	0.170	0.273	**0.152**	**0.252**
	168	0.263	0.361	**0.195**	**0.287**	0.368	0.424	0.299	0.387	0.231	0.309	0.296	0.368	0.288	0.373	0.227	0.304
	336	0.305	0.386	**0.238**	**0.316**	0.381	0.431	0.375	0.428	0.323	0.369	0.326	0.374	0.317	0.375	0.317	0.361
	720	0.372	0.434	**0.272**	**0.346**	0.406	0.443	0.377	0.434	0.404	0.423	0.412	0.428	0.440	0.450	0.405	0.416
	960	0.393	0.449	**0.299**	**0.367**	0.460	0.548	0.366	0.426	0.433	0.438	0.446	0.447	0.467	0.463	0.430	0.431
WTH	24	0.357	0.412	0.357	0.391	0.335	0.381	0.363	0.396	0.294	0.343	0.304	0.351	0.303	0.352	**0.292**	**0.341**
	48	0.428	0.458	0.425	0.444	0.395	0.459	0.456	0.462	0.370	0.411	0.374	0.411	0.372	0.411	**0.363**	**0.402**
	168	0.564	0.541	0.516	0.516	0.608	0.567	0.574	0.548	0.473	0.494	0.480	0.501	0.496	0.511	**0.332**	**0.393**
	336	0.533	0.536	0.536	0.537	0.702	0.620	0.600	0.571	**0.495**	**0.515**	0.507	0.526	0.514	0.530	0.499	**0.515**
	720	0.562	0.557	0.582	0.571	0.831	0.731	0.587	0.570	**0.526**	**0.542**	0.545	0.557	0.548	0.556	0.533	0.546
ILI	24	**2.687**	1.147	2.940	1.205	4.588	1.462	3.101	1.238	3.041	1.186	3.305	1.241	3.194	1.176	3.121	**1.139**
	36	2.887	**1.160**	**2.826**	1.184	4.845	1.496	3.397	1.270	3.406	1.232	3.542	1.314	3.193	1.169	3.288	1.182
	48	2.797	1.155	**2.677**	1.155	4.865	1.516	2.947	1.203	3.459	1.221	3.409	1.208	3.15	1.142	3.122	**1.120**
	60	**2.809**	**1.163**	3.011	1.245	5.212	1.576	3.019	1.202	3.640	1.305	3.668	1.269	3.43	1.196	3.416	1.180
Traffic	24	0.562	0.375	**0.351**	0.261	0.608	0.334	0.550	0.363	0.491	0.271	0.484	0.266	0.499	0.277	0.452	**0.247**
	48	0.567	0.374	0.370	0.270	0.644	0.359	0.595	0.376	0.519	0.295	0.516	0.293	0.516	0.290	**0.315**	**0.261**
	168	0.607	0.385	**0.395**	0.277	0.660	0.391	0.649	0.407	0.513	0.289	0.511	0.301	0.517	0.289	0.501	**0.276**
	336	0.624	0.389	**0.415**	0.289	0.747	0.405	0.624	0.388	0.530	0.300	0.531	0.316	0.544	0.303	0.506	**0.288**
	720	0.623	0.378	**0.455**	0.313	0.792	0.430	0.674	0.417	0.573	0.313	0.569	0.303	0.563	0.311	0.539	**0.300**

boosts in many scenarios. The external memory enhancements support two plugin schemes, **Plug-and-Play** and **Tune-and-Play**. They focus on different benefits. `TuneMemory` is especially useful for task-relevant knowledge incorporation by fine-tuning on an external task-relevant memory set[5]. On the other hand, `PlugMemory` provides a more robust representation of inputs with high uncertainty by doing a retrieval (fig. 3.11) on an external task-relevant memory set, without the work of any training or fine-tuning. Below we provide 4 practical scenarios to showcase the aforementioned benefits of `TuneMemory` and `PlugMemory` external memory modules. The detailed setups of each case can be found in the appendix.

Case 1 (`TuneMemory`). We take the single variate, *Number of Influenza incidence in a week* (denoted as ILI OT), from the **ILI** dataset as a straightforward example. In this dataset, we are aware of the existence of recurring annual patterns, which can be readily identified through visualizations in fig. B.16. Notably, the signal patterns around the spring of 2014 closely resemble past springs. Thus, in predictions tasks with input located in the yearly recurring period, we collect similar patterns from the past to form a task-relevant external memory set.

Case 2 (`TuneMemory`). In many sociological studies (Wang et al., 2021a,b), electricity usage exhibits consistent patterns across different regions, influenced by the daily and weekly routines of residents and local businesses. Thus, we collect sequences that match the length of the input sequence but are from 1 to 20 weeks prior, obtaining a task-relevant external memory set of size 20.

In addition, we also include analysis of **"bad" external memory sets**, to verify the effectiveness of incorporating informative external memory sets. We construct the "bad" external memory sets by randomly selecting from dataset without any task-relevant preference, see appendix B.3.5.2 for more details about such selection. The results indicate that, by properly selecting external memory sets, we further improve the models' performance. On the contrary, randomly chosen external memory sets can negatively impact performance. We report the results of Case 1 and Case 2 in table 3.7.

Case 3 (`PlugMemory`). Through `PlugMemory`, informative patterns can be extracted from a memory set for the given noisy input. To verify this ability, we construct the external memory sets based on the weekly pattern spotted in ETTh1 and ETTm1, and add noise of different scales into the

[5]Task-relevant means the relevance to the inputs of the time series forecasting. A task-relevant memory set could be a set of some history time series segments that are relevant to the inputs of the prediction.

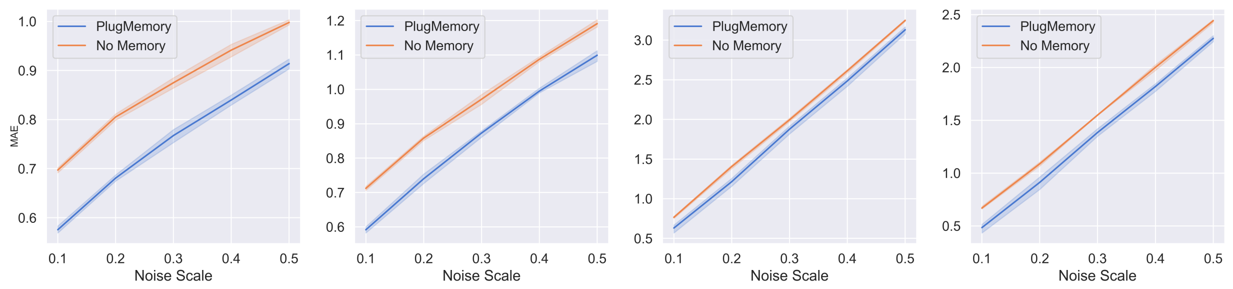


Figure 3.12. **Visualization of Memory Plugin Scenarios Case 3 & 4. From Left to Right:** MAE against different noise levels with (1) ETTh1 + prediction horizon 336; (2) ETTh1 + prediction horizon 168; (3) ETTm1 + prediction horizon 288; and (4) ETTm1 + prediction horizon 96. The results show the robustness of `PlugMemory` against different level of noise.

input sequence. We add the noise following $x \leftarrow x + \text{scale} \cdot \text{std}(x)$. For Case 3, we use the ETTh1 dataset. **Case 4 (`PlugMemory`).** For Case 4, we evaluate `PlugMemory` on the ETTm1 dataset.

Table 3.7. **Performance Comparison of the StanHop Model with `TuneMemory` and Ablation Using Bad External Memory Sets (`TuneMemory`-(b)).** We report the mean MSE and MAE over 10 runs with variances omitted as they are $\leqslant 0.79\%$. For ILI OT, we consider prediction horizons of 12, 24, and 60. For ETTh1, we choose prediction horizons of 24, 48, and 720, covering both short and long durations. The results indicate that using dataset insights and `TuneMemory` enhances our model's performance.

Case 1 (ILI OT)							**Case 2** (ETTh1)						
	Default		`TuneMemory`		`TuneMemory`-(b)			Default		`TuneMemory`		`TuneMemory`-(b)	
	MSE	MAE	MSE	MAE	MSE	MAE		MSE	MAE	MSE	MAE	MSE	MAE
12	4.011	1.701	3.975 (**-0.9%**)	1.693 (**-0.5%**)	4.340 (+8.2%)	1.789 (+5.1%)	24	0.294	0.351	0.284 (**-3.4%**)	0.351 (±0%)	0.300 (+2%)	0.361 (+2.8%)
24	4.254	1.771	3.960 (**-6.9%**)	1.690 (**-4.6%**)	4.271 (+0.4%)	1.776 (+0.3%)	48	0.340	0.387	0.328 (**-3.5%**)	0.379 (**-2.1%**)	0.342 (+0.6%)	0.388 (+0.3%)
60	3.613	1.685	3.572 (**-1.1%**)	1.528 (**-9.3%**)	3.821 (+5.8%)	1.725 (+2.4%)	720	0.512	0.511	0.504 (**-1.6%**)	0.512 (**-0.2%**)	0.514 (+0.4%)	0.521 (+2.0%)

3.3.6. Conclusion

We propose the generalized sparse modern Hopfield model and present STanHop-Net, a Hopfield-based time series prediction model with external memory functionalities. Our design improves time series forecasting performance, quickly reacts to unexpected or rare events, and offers both strong theoretical guarantees and empirical results. Empirically, STanHop-Nets rank in the top two in 47 out of our 58 experiment settings compared to the baselines. Furthermore, with `PlugMemory` and `TuneMemory` modules, it showcases average performance boosts of $\sim$12% and $\sim$3% for each.

3.4. A Benchmark Study For Limit Order Book (LOB) Models and Time Series Forecasting Models on LOB Data

3.4.1. Introduction

The Limit Order Book (LOB) serves as the order-matching engine for all exchanges, providing the most granular market time series data for analysis (Weber, 1999; Ntakaris et al., 2018). As a universal data format across markets and assets (e.g., stocks and futures), LOB contains high-resolution macroeconomic information crucial for asset price predictions (Chan et al., 2005; Harris and Panchapagesan, 2005; Large, 2007; Avellaneda and Stoikov, 2008; Roşu, 2009; Eisler et al., 2012).

Two primary research tracks in LOB data analysis are **M**id-**P**rice **T**rend **P**rediction (**MPTP**) and **M**id-**P**rice **R**eturn **F**orecasting (**MPRF**). Inspired by the success of deep learning in domains like natural language processing (Vaswani et al., 2023) and computer vision (He et al., 2016), researchers have proposed deep learning models for LOB analysis and general time series predictions (Zeng et al., 2023; Liu et al., 2023; Nie et al., 2022; Wang et al., 2024a).

Despite these advancements, the field lacks comprehensive benchmark studies comparing model performance. While Prata et al. (2024) conducted a benchmark study for MPTP, it was limited to stock datasets, excluding other asset types. For MPRF, which encompasses both LOB modeling and time series prediction, no benchmark exists that evaluates deep learning-based LOB models and state-of-the-art time series forecasting models on LOB data. This work aims to address these gaps in both MPTP and MPRF tasks.

To address these limitations, we conduct a comprehensive study on both MPTP and MPRF tasks. For MPTP, we evaluate state-of-the-art LOB models using the open-source FI-2010 stock dataset and our proprietary futures dataset, CHF-2023, measuring each model's F1 scores across five different prediction horizons. We also conduct an ablation study to assess the predictive power of various LOB feature types, including basic LOB features, time-insensitive features, and time-sensitive features.

For the MPRF task, we evaluate both LOB-specific models and state-of-the-art time series forecasting models on FI-2010. We assess their forecasting performance using three metrics: Mean-Square Error (MSE), Coefficient of Determination (R^2), and Pearson Correlation (Corr). Additionally, we propose a novel neural architecture featuring cross-variate mixing layers, called **CVML**, designed to enhance the forecasting performance of existing time series models on LOB data.

Our experiments reveal several key findings. For MPTP, we observe inconsistent model performance rankings between the stock and futures LOB datasets, with models generally performing worse on the futures data. This suggests limited generalizability of current LOB model architectures and highlights the distinct underlying characteristics of LOB data from different assets. Our ablation study confirms that each feature subset contributes unique predictive power, underscoring the importance of comprehensive feature selection in LOB modeling. For MPRF, our results demonstrate that LOB models incorporating LOB-specific inductive bias significantly outperform general-purpose time series forecasting models. The latter, lacking LOB-specific design considerations, show minimal forecasting power when directly applied to LOB datasets. Our proposed CVML module significantly improves the forecasting capabilities of all benchmarked time series models, extending their predictive power to the more complex and noisy LOB time series data.

Contributions. We summarize our contributions as follows. **1)** We evaluate existing LOB models on the MPTP task using a proprietary futures LOB dataset, CHF-2023, to assess the transferability of models designed for stock LOB data across asset classes. **2)** We evaluate existing LOB models on the MPTP task using a proprietary futures LOB dataset, CHF-2023, to assess the transferability of models designed for stock LOB data across asset classes. **3)** We present the first benchmark on the MPRF task in the literature. **4)** We pioneer benchmarking state-of-the-art time series forecasting models on the MPRF task, bridging the gap between general-purpose and LOB-specific time series forecasting. **5)** We propose a novel Cross-Variate Mixing Layer (CVML) as an add-on to existing time series models, enhancing their MPRF performance significantly. These contributions advance LOB modeling across different assets and tasks while providing a new tool to enhance time series model performance on LOB data.

Organization. section 3.4.2 includes related work. section 3.4.3 discusses background information on the MPTP and MPRF tasks. section 3.4.4 details our benchmark studies and our proposed CVML architecture. section 3.4.5 discusses the prediction performance gap between the stock and futures datasets in MPTP. section 3.4.6 includes concluding remarks.

3.4.2. Related Works

Price Trend Prediction Surveys. Several comprehensive benchmark surveys have examined deep learning applications in price trend prediction (Ozbayoglu et al., 2020; Sezer et al., 2020; Jiang, 2021), each with a distinct focus. Jiang (2021) emphasize reproducibility, analyzing model architectures, evaluation metrics, and implementations in stock price and market index prediction studies from 2017 to 2019. A follow-up study by Kumbure et al. (2022) extend this analysis to datasets and input variables commonly used in stock market predictions. Hu et al. (2021) review 86 papers on stock and foreign exchange price prediction, while other surveys (Rundo et al., 2019; Mintarya et al., 2023) compare machine learning and deep learning methods in stock market prediction, concluding that deep learning approaches generally offer superior accuracy. Nti et al. (2020a) broaden the scope beyond technical analysis, reviewing 122 papers from 2007 to 2018 covering technical, fundamental, and combined analyses. Additionally, Shah et al. (2019) evaluate the real-world applicability of models through backtesting performance. Notably, these surveys do not include benchmarks for prediction models on Limit Order Book (LOB) data. The most relevant work is a benchmark study by Prata et al. (2024) on mid-price trend prediction models using LOB data. However, our work addresses three key limitations of their study: We use a proprietary dataset with a time range 200 times larger than the open-source dataset they employed. We benchmark models on both stock and futures datasets, whereas they focused solely on stocks. We extend our analysis to include the mid-price forecasting problem (a regression task), benchmarking both

LOB models and state-of-the-art time series forecasting models, in addition to the mid-price trend prediction task (a classification problem) they addressed. These enhancements allow our study to provide a more comprehensive and diverse evaluation of LOB-based prediction models across different assets and problem types.

Time Series Forecasting Models. The success of deep learning in natural language processing and computer vision has significantly influenced time series forecasting, with deep learning models becoming predominant in this field. Transformer-based architectures, in particular, have emerged as the leading approach for multivariate time series forecasting (Nie et al., 2022; Liu et al., 2023). However, recent developments have shown that models based on linear layers (Zeng et al., 2023; Wang et al., 2024a; Chen et al., 2023a; Oreshkin et al., 2020; Challu et al., 2023; Zhang et al., 2022b) can achieve comparable performance to transformer-based models. While convolutional neural networks (O'shea and Nash, 2015; Wu et al., 2023a; Franceschi et al., 2019) and recurrent networks (Hochreiter and Schmidhuber, 1997; Lai et al., 2018; Franceschi et al., 2019) have also been applied to time series forecasting, their performance generally lags behind that of transformers and linear-based architectures. Notably, there has been limited overlap between general time series forecasting and Limit Order Book (LOB) time series analysis. To date, no comprehensive benchmarking of state-of-the-art time series forecasting models on the complex LOB time series data has been conducted. To address this gap, our paper selects four state-of-the-art time series forecasting models, encompassing both transformer-based and linear architectures. Our aim is to bridge the divide between general-purpose time series forecasting and the more specialized field of LOB time series forecasting, providing insights into the applicability and performance of these models on LOB data.

3.4.3. Problem Definition

We model the LOB as a time series $\mathbb{L} \in \mathbb{R}^{4\times L\times T}$, where $\mathbb{L}(t) \in \mathbb{R}^{4\times L}$ represents the LOB at time step $t \in [0, T]$. Specifically, $\mathbb{L}(t) = \{P_i^{\text{bid}}(t), Q_i^{\text{bid}}(t), P_i^{\text{ask}}(t), Q_i^{\text{ask}}(t)\}_{i\in[0,L]}$, with T observed time steps and L levels on each side of the order book. $P^{\text{bid/ask}}i(t)$ and $Q_i^{\text{bid/ask}}(t)$ denote the price and quantity at level i at time t, respectively. Levels are ordered by price, representing order priority in matching. The best bid price is the highest bid, while the best ask is the lowest ask. The mid-price, $\mathrm{mp}(t)$, is the average of these best prices. Execution of more bid (ask) orders decreases (increases) the mid-price.

Mid-Price Return Forecasting (MPRF). To address mid-price volatility and non-stationarity, we model the problem as forecasting the mid-price *return*. At time t, our target is:

$$\text{target}_h(t) = \mathrm{mp}(t+h)/\mathrm{mp}(t) - 1,$$

where h is the forecasting horizon.

Mid-Price Trend Prediction (MPTP). An alternative approach models mid-price prediction as a classification problem, categorizing trends into three classes: **U** (upward), **D** (downward), and **S** (stable). Following (Ntakaris et al., 2018), we generate labels from raw LOB data by comparing the current mid-price to the average future mid-price:

$$\begin{cases} \mathbf{U}, & \text{if} \quad \mathrm{avg\_mp}(k,t) > \mathrm{mp}(t) \times (1+\alpha) \\ \mathbf{D}, & \text{if} \quad \mathrm{avg\_mp}(k,t) < \mathrm{mp}(t) \times (1-\alpha) \\ \mathbf{S}, & \text{if} \quad \text{Otherwise}, \end{cases}$$

where $\mathrm{avg\_mp}(k,t) = (\Sigma_{i=1}^{k}\mathrm{mp}(t+i))/k$. Using $\alpha = 0.002\%$ yields approximately equal distribution (33%) for each label.

3.4.4. Experiments

We conduct all experiments using 3 seeds to minimize the effect of random initialization.

Datasets: **1) FI-2010** (Ntakaris et al., 2018)[6]: This Limit Order Book (LOB) dataset includes 10 trading days of data from five Finnish companies on the NASDAQ Nordic stock market. It was designed to evaluate machine learning models' performance on stock price trend prediction. **2) CHF-2023**: To address FI-2010's limitations (single asset type and short time span), we use this proprietary futures LOB dataset. It covers 5 years of LOB data for SC (Crude Oil), one of China's most liquid futures contracts. The raw data[7] is the most granular LOB dataset available for the Chinese futures market, offering 500ms resolution snapshots with five-level bid and ask information. We use both datasets in MPTP to study LOB models' generalizability to different assets. We only use FI-2010 in MPRF for efficient experimentation.

Evaluation Metrics: For MPTP, we use the F1 scores. For MPRF, we use Mean Squared Error (MSE), Pearson Correlation Coefficient (Corr), and Coefficient of Determination (R^2). MSE quantifies the average squared difference between predicted and actual returns, providing a measure of prediction accuracy. The Pearson Correlation Coefficient assesses the linear relationship between predicted and actual returns, indicating the direction and strength of their association. Lastly, R^2 represents the proportion of variance in the target variable explained by our model. The implementation of MSE and R^2 are from Scikit-learn (Pedregosa et al., 2011). The implementation of Pearson Correlation Coefficient is from SciPy (Virtanen et al., 2020).

Hyperparameter Search. For MPTP, we use the hyperparameters from the original paper of the models. For MPRF, we perform a grid search. Details are in appendix B.4.5.

Table 3.8. **Input lookback size and number of features for MPTP Models.** (Tsantekidis et al., 2017b,b,a; Tran et al., 2018; Zhang et al., 2019; Passalis et al., 2019; Tsantekidis et al., 2020; Wallbridge, 2020; Passalis et al., 2020; Zhang and Zohren, 2021; Guo and Chen, 2022) The number of features is formatted as [basic]/[basic + time-insensitive]/[basic + time-insensitive + time-sensitive]

	MLP	LSTM	CNN1	CTABL	DEEPLOB	DAIN	CNNLSTM	CNN2	TRANSLOB	TLONBoF	BINCTABL	DEEPLOBATT	DLA
Lookback size	100	100	100	10	100	15	300	300	100	15	10	50	5
Features (FI-2010)	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144	40/86/144
Features (CHF-2023)	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66	20/46/66

[6]License: Creative Commons Attribution 4.0 International (CC BY 4.0)

[7]`http://www.cffex.com.cn/u/cms/www/202201/20211342wucd.pdf`

Table 3.9. **Input lookback size and number of features for MPRF Models.** (Nie et al., 2022; Zeng et al., 2023; Liu et al., 2023; Wang et al., 2024a) The number of features is formatted as [basic]/[basic + time-insensitive]/[basic + time-insensitive + time-sensitive]

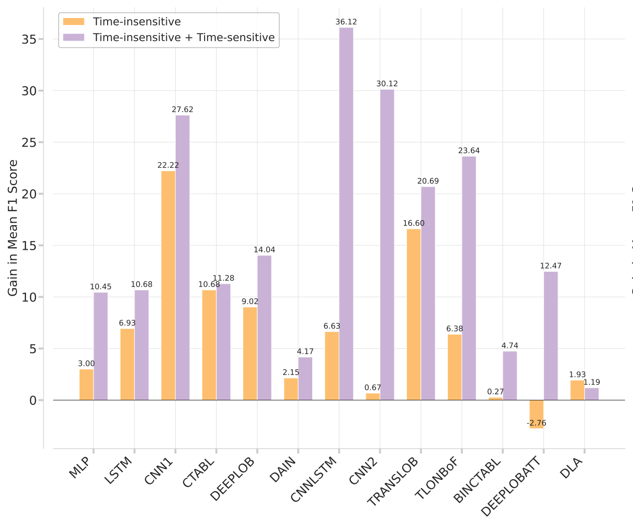
	MLP	LSTM	CNN1	BINCTABL	DAIN	TRANSLOB	PatchTST	DLinear	iTransformer	TimeMixer
Lookback size	100	100	100	10	15	100	100	100	100	100
Features (FI-2010)	41/86/144	41/86/144	41/86/144	41/86/144	41/86/144	41/87/145	41/87/145	41/87/145	41/87/145	41/87/145

3.4.4.1. Models. Our benchmark includes models for two tasks: MPTP and MPRF. For MPTP, we select 13 state-of-the-art models. The input and output of each mid-price trend prediction model follow the same protocol in their original paper. For MPRF, we choose 10 models with two goals in mind: benchmarking diverse neural architectures (MLP, CNN, LSTM, and Transformer) and evaluating the importance of domain-specific inductive bias for LOB data. We include some models from the MPTP list that have LOB-specific adaptations, as well as high-performing general-purpose time series forecasters. This mix allows us to compare specialized LOB models against successful general-purpose forecasters. For time series forecasting models including PatchTST, DLinear, iTransformer and TimeMixer, the LOB data as well as the history mid-price return are input as a multivariate time series. The output for all MPRF models is a scaler representing the return of horizon h. Each input is $\mathbb{R}^{T\times 4L}$, output is $\mathbb{R}$. Detailed model and input information is in appendix B.4.4. table 3.8 and table 3.9 include the input lookback size and feature dimension for each model.

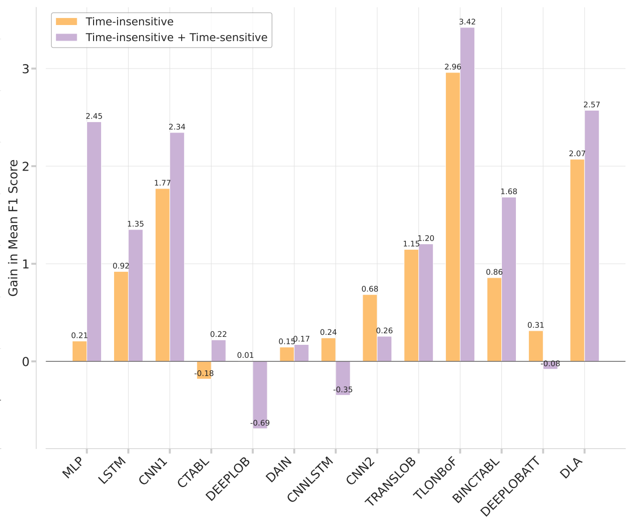
3.4.4.2. Mid-price Trend Prediction Results. table 3.10 reveals inconsistencies in the top-performing models between the stock FI-2010 and futures CHF-2023 datasets. While BINCTABL, DAIN, and DEEPLOB perform significantly better than other models on the FI-2010 dataset, this advantage is not present in the CHF dataset. This discrepancy suggests that models that exhibit strong performance on the stock LOB dataset are not robust to the futures LOB datasets. To investigate the predictive power of different feature types (basic, time-insensitive, time-sensitive), we evaluate the models on two feature subsets: one with only basic features, and another with basic and time-insensitive features. fig. 3.13 illustrates the gains in prediction performance based on average F1 scores across 5 horizons for each feature set (full results are available in appendix B.4.8).

Table 3.10. **Mid-price Trend Prediction F1 Scores (Mean&Standard Deviation) on Basic LOB data + time-insensitive features + time-sensitive features**. We provide the F1 scores on mid-price trend predictions across horizons {1,2,3,5,10} on for the FI-2010 and CHF-2023 datasets. The model performance ranking is not consistent between two datasets, indicating that models' prediction power for one asset is not automatically transferable to another asset.

	FI-2010						CHF-2023					
Model	K=1	K=2	K=3	K=5	K=10	avg	K=1	K=2	K=3	K=5	K=10	avg
MLP	54.0 (3.7)	52.6 (2.0)	55.6 (0.3)	55.0 (0.4)	52.3 (2.8)	53.9	42.4 (1.3)	47.0 (0.5)	47.8 (0.7)	46.3 (0.3)	41.3 (1.0)	45.0
LSTM	76.3 (0.3)	75.4 (0.1)	76.9 (0.9)	75.6 (1.1)	62.9 (1.3)	73.4	49.7 (0.4)	<u>50.9 (0.4)</u>	<u>51.4 (0.3)</u>	**49.2 (0.3)**	45.8 (0.3)	49.4
CNN1	73.6 (0.3)	72.0 (0.8)	75.3 (0.5)	78.4 (0.8)	78.9 (1.1)	75.6	48.5 (0.9)	49.7 (1.0)	50.6 (0.6)	48.2 (0.5)	45.5 (0.8)	48.5
CTABL	76.6 (0.2)	72.6 (0.4)	77.9 (0.2)	82.1 (0.5)	83.3 (0.5)	78.5	45.2 (2.1)	46.4 (1.0)	46.7 (0.9)	45.6 (0.7)	43.8 (0.5)	45.5
DEEPLOB	**81.2 (0.3)**	<u>82.4 (0.7)</u>	<u>86.1 (0.5)</u>	88.1 (0.2)	88.7 (0.2)	85.3	43.2 (5.3)	47.9 (2.4)	49.5 (1.8)	46.6 (1.3)	41.4 (1.3)	45.7
DAIN	<u>80.8 (0.1)</u>	79.7 (0.1)	85.8 (0.1)	<u>90.0 (0.0)</u>	<u>93.2 (0.0)</u>	85.9	45.7 (0.3)	48.4 (0.5)	48.7 (0.5)	47.1 (0.2)	41.8 (0.8)	46.3
CNNLSTM	74.1 (0.4)	68.1 (0.3)	73.7 (0.7)	77.3 (1.1)	77.8 (0.9)	74.2	45.4 (0.6)	47.0 (1.3)	48.1 (0.9)	46.7 (1.0)	46.0 (0.6)	46.6
CNN2	73.1 (0.2)	67.5 (1.0)	70.8 (4.2)	74.4 (3.4)	65.7 (1.2)	70.3	45.5 (0.6)	46.2 (0.6)	47.9 (0.5)	46.4 (0.5)	<u>46.2 (0.3)</u>	46.4
TRANSLOB	74.5 (1.1)	70.5 (0.3)	75.2 (0.3)	78.4 (0.2)	68.2 (3.2)	73.4	**52.2 (0.4)**	**52.1 (0.3)**	**51.4 (0.4)**	<u>49.0 (0.4)</u>	**46.2 (0.4)**	50.2
TLONBoF	61.3 (6.1)	66.9 (0.3)	72.6 (0.7)	75.2 (4.0)	71.2 (2.7)	69.4	<u>50.4 (0.4)</u>	50.3 (0.7)	50.3 (0.3)	47.1 (1.2)	45.0 (0.3)	48.6
BINCTABL	80.3 (0.1)	**83.4 (0.3)**	**87.9 (0.2)**	**91.3 (0.1)**	**93.2 (0.5)**	87.2	47.2 (0.8)	47.8 (1.1)	48.2 (0.6)	47.0 (0.6)	44.9 (0.4)	47.0
DEEPLOBATT	77.9 (0.0)	78.5 (0.0)	82.5 (0.0)	82.9 (0.0)	82.1 (0.0)	80.8	47.4 (0.8)	49.9 (0.6)	50.6 (0.4)	48.8 (0.2)	44.8 (0.7)	48.3
DLA	71.8 (0.0)	71.1 (0.0)	76.4 (0.0)	85.0 (0.0)	59.3 (0.0)	72.7	48.4 (2.1)	50.5 (0.6)	50.4 (0.5)	47.8 (0.6)	41.1 (0.8)	47.6
avg	73.5	72.4	76.7	79.5	75.1	75.4	47.0	48.8	49.4	47.4	44.1	47.3



(a) Gain on FI-2010



(b) Gain on CHF-2023

Figure 3.13. **Gains in mean F1 scores.** These plots illustrate the incremental prediction power gained from time-insensitive and time-sensitive features on FI-2010 (a) and CHF-2023 (b) datasets. Yellow bars indicate the improvement in F1 scores when adding time-insensitive features to basic LOB features for most models. Purple bars, compared to yellow, demonstrate the further enhancement in F1 scores when incorporating time-sensitive features alongside basic and time-insensitive features.

We define feature set gains as the difference between its average F1 scores and those of the basic features. Consequently, basic features have zero gain, serving as the baseline. Positive gains for time-insensitive and time-sensitive features indicate additional predictive power beyond raw LOB readings. fig. 3.13 demonstrates that both time-insensitive and time-sensitive features contribute positive gains on top of basic LOB features for both FI-2010 and CHF-2023 datasets. This finding affirms the universal effectiveness of these feature sets across stock and futures datasets, underscoring their value in enhancing model performance regardless of the asset type.

3.4.4.3. Mid-price Return Forecasting Results. table 3.11 shows that LOB models with LOB-specific inductive bias significantly outperform general-purpose time series prediction models. This highlights the importance of incorporating LOB-related inductive bias in model design. Additionally, the results indicate that general-purpose models lack the generalization needed for strong forecasting performance on LOB datasets. This performance gap underscores the specialized nature of LOB data and the need for tailored models in financial forecasting.

Prediction Performance Gap Between LOB Models and Time Series Models. In the MPRF task, a significant performance gap exists between LOB-specific models and general time series models. Notably, complex Transformer-based models including PatchTST and iTransformer underperform compared to simpler LOB-specific architectures based-on MLPs or LSTMs. This discrepancy suggests that without LOB-aware architectural design, conventional time series models struggle to generate accurate predictions on LOB data due to its low signal-to-noise ratio. This observation indicates that the sophisticated temporal modeling capabilities of state-of-the-art time series models may be impeded by the noisy temporal dynamics and intricate cross-variate correlations inherent in LOB data. Based on this hypothesis, we propose a novel module called Cross-Variate Mixing Layers (CVML) to enhance the signal-to-noise ratio of LOB time series. CVML serves as an add-on layer preceding the time series modeling layers in standard time series prediction models. It accepts raw LOB data, mixes the features (or different variates from a time series perspective), and outputs an intermediate multivariate time series as input for subsequent modeling layers. CVML integrates seamlessly with existing time series models without requiring further modifications and can be trained end-to-end. CVML has five Conv1D layers, each with a kernel size of 2 and $\lceil N/2 \rceil$ output channels, where N is the number of input features. This design leverages convolution kernels

Table 3.11. **Mid-price Return Forecasting Results (Mean) on Basic LOB data**. 10 LOB models and time series forecasting models are benchmarked to compare their Mean Square Error (MSE), Pearson correlation (Corr), and Coefficient of Determination (R^2) on mid-price return forecasting across 5 horizons $\{1,2,3,5,10\}$ on the FI-2010 dataset. LOB models perform much better than general-purpose time series models, indicating that it is essential to include LOB-relevant inductive bias into the model design to achieve good forecasting power on LOB datasets. For each horizon, the best model is bolded, and the next best model is underlined.

		FI-2010				
Model	Metric	K=1	K=2	K=3	K=5	K=10
MLP	MSE	0.659 (0.005)	1.096 (0.021)	1.431 (0.020)	1.972 (0.008)	2.963 (0.162)
	Corr	0.084 (0.002)	0.101 (0.001)	0.102 (0.011)	0.106 (0.006)	0.125 (0.019)
	R^2	-0.004 (0.007)	-0.016 (0.020)	-0.016 (0.014)	-0.008 (0.004)	-0.060 (0.058)
LSTM	MSE	**0.638 (0.005)**	**1.035 (0.009)**	**1.328 (0.007)**	**1.824 (0.013)**	**2.571 (0.024)**
	Corr	**0.173 (0.020)**	**0.211 (0.024)**	**0.244 (0.007)**	**0.274 (0.014)**	0.298 (0.009)
	R^2	**0.027 (0.007)**	**0.041 (0.008)**	**0.057 (0.005)**	**0.067 (0.006)**	**0.081 (0.009)**
CNN1	MSE	0.665 (0.008)	1.058 (0.007)	1.379 (0.015)	1.879 (0.023)	2.681 (0.017)
	Corr	0.129 (0.006)	0.185 (0.013)	0.210 (0.004)	0.248 (0.016)	**0.298 (0.007)**
	R^2	-0.013 (0.012)	0.020 (0.007)	0.021 (0.011)	0.039 (0.012)	0.041 (0.006)
BINCTABL	MSE	0.650 (0.000)	1.047 (0.001)	1.347 (0.008)	1.838 (0.016)	2.612 (0.012)
	Corr	0.106 (0.004)	0.176 (0.003)	0.215 (0.015)	0.249 (0.016)	0.278 (0.003)
	R^2	0.010 (0.000)	0.029 (0.001)	0.044 (0.006)	0.061 (0.008)	0.069 (0.004)
DAIN	MSE	0.693 (0.011)	1.114 (0.014)	1.436 (0.004)	1.977 (0.004)	2.824 (0.014)
	Corr	0.038 (0.004)	0.068 (0.007)	0.085 (0.002)	0.107 (0.003)	0.127 (0.001)
	R^2	-0.057 (0.016)	-0.032 (0.013)	-0.019 (0.003)	-0.011 (0.002)	-0.007 (0.005)
TRANSLOB	MSE	0.659 (0.005)	1.088 (0.002)	1.395 (0.003)	1.904 (0.015)	2.704 (0.029)
	Corr	0.079 (0.010)	0.090 (0.019)	0.150 (0.018)	0.210 (0.004)	0.267 (0.009)
	R^2	-0.005 (0.008)	-0.008 (0.002)	0.009 (0.002)	0.027 (0.008)	0.033 (0.010)
PatchTST	MSE	0.654 (0.000)	1.077 (0.000)	1.406 (0.001)	1.949 (0.001)	2.795 (0.002)
	Corr	0.081 (0.002)	0.082 (0.001)	0.079 (0.004)	0.092 (0.003)	0.085 (0.003)
	R^2	0.003 (0.001)	0.002 (0.000)	0.002 (0.001)	0.004 (0.000)	0.001 (0.001)
DLinear	MSE	0.652 (0.000)	1.073 (0.000)	1.402 (0.000)	1.945 (0.001)	2.782 (0.000)
	Corr	0.080 (0.002)	0.081 (0.001)	0.074 (0.001)	0.083 (0.002)	0.084 (0.002)
	R^2	0.006 (0.000)	0.006 (0.000)	0.005 (0.000)	0.006 (0.001)	0.005 (0.000)
iTransformer	MSE	0.683 (0.008)	1.183 (0.031)	1.582 (0.016)	2.279 (0.095)	3.401 (0.076)
	Corr	0.045 (0.004)	0.045 (0.007)	0.033 (0.005)	0.063 (0.004)	0.056 (0.004)
	R^2	-0.041 (0.012)	-0.096 (0.028)	-0.123 (0.012)	-0.165 (0.048)	-0.216 (0.027)
TimeMixer	MSE	0.657 (0.001)	1.075 (0.002)	1.394 (0.002)	1.888 (0.018)	2.643 (0.017)
	Corr	0.083 (0.005)	0.110 (0.001)	0.135 (0.005)	0.201 (0.025)	0.271 (0.014)
	R^2	-0.001 (0.002)	0.004 (0.001)	0.011 (0.001)	0.035 (0.009)	0.055 (0.006)

to extract cross-variate features and capture correlations. Additionally, we implement increasing dilation in the kernel for each successive layer to enhance temporal signal extraction.

To demonstrate CVML's efficacy, we prepend it to four time series models without altering their core architectures. Results show significant improvements in forecast performance across

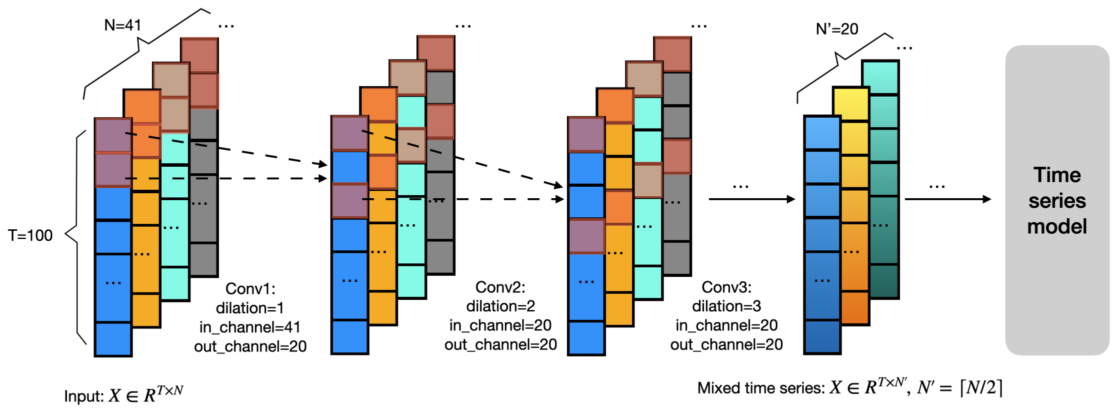


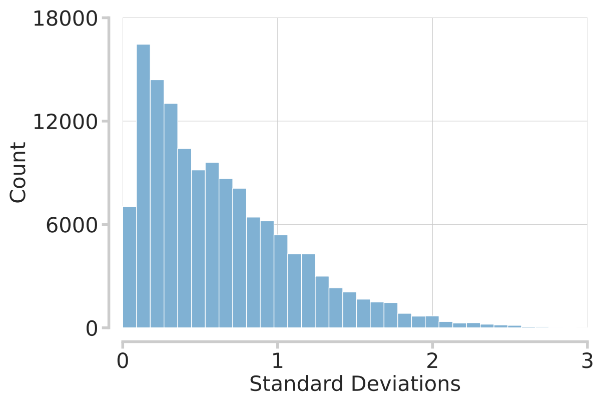
Figure 3.14. **Cross-Variate Mixing Layers (CVML) Architecture.** CVML consists of 5 Conv1D layers, with the first 3 illustrated here. Each Conv1D layer employs a kernel size of 2 and matches its input channels to the number of variates in the input time series. The architecture features increasing dilation across successive layers to expand the receptive field. Input $X \in \mathbb{R}^{T \times N}$ is transformed into a mixed time series $X' \in \mathbb{R}^{T \times N'}$, $N' = \lceil N/2 \rceil$, before feeding into the subsequent time series model.

Table 3.12. **Time Series Model Performance with and without CVML** on the FI-2010 Dataset using basic LOB features. The % column indicates the percentage improvement from adding CVML.

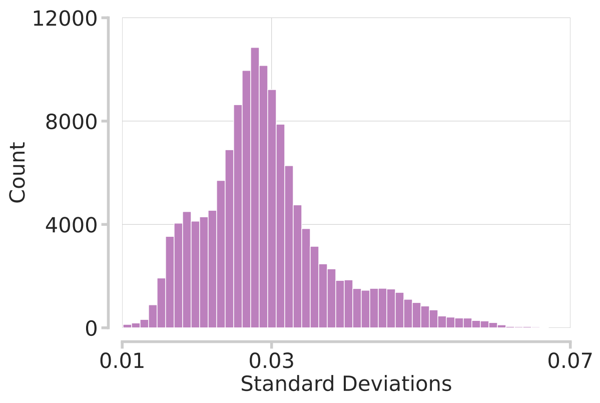
	MSE (↓)						Corr (↑)						R^2 (↑)					
Model	K=1	K=2	K=3	K=5	K=10	%	K=1	K=2	K=3	K=5	K=10	%	K=1	K=2	K=3	K=5	K=10	%
PatchTST-CVML	**0.653**	**1.071**	**1.370**	**1.893**	**2.646**	3.1	**0.070**	**0.113**	**0.165**	**0.191**	**0.241**	86.2	**0.005**	**0.007**	**0.028**	**0.033**	**0.054**	958.3
PatchTST	0.654	1.077	1.406	1.949	2.795		0.081	0.082	0.079	0.092	0.085		0.003	0.002	0.002	0.004	0.001	
DLinear-CVML	**0.650**	**1.042**	**1.352**	**1.796**	**2.548**	5.9	**0.104**	**0.192**	**0.205**	**0.291**	**0.313**	174.9	**0.010**	**0.035**	**0.040**	**0.082**	**0.089**	814.3
DLinear	0.652	1.073	1.402	1.945	2.782		0.080	0.081	0.074	0.083	0.084		0.006	0.006	0.005	0.006	0.005	
iTransformer-CVML	**0.654**	1.084	**1.402**	**2.002**	**2.649**	14.6	**0.054**	**0.070**	**0.088**	**0.121**	**0.249**	140.5	**0.002**	**-0.005**	**0.005**	**-0.024**	**0.053**	104.8
iTransformer	0.683	**1.183**	1.582	2.279	3.401		0.045	0.045	0.033	0.063	0.056		-0.041	-0.096	-0.123	-0.165	-0.216	
TimeMixer-CVML	**0.642**	**1.033**	**1.329**	**1.807**	**2.494**	4.6	**0.160**	**0.221**	**0.257**	**0.298**	**0.353**	61.1	**0.022**	**0.043**	**0.056**	**0.076**	**0.109**	194.2
TimeMixer	0.657	1.075	1.394	1.888	2.643		0.083	0.110	0.135	0.201	0.271		-0.001	0.004	0.011	0.035	0.055	

all metrics (MSE, Corr, and R^2), as shown in table 3.12. Averaging across four models and five horizons, CVML achieves a 7.4% improvement in MSE, 101.6% in Pearson Correlation (Corr), and 244.9% in R^2. This highlights CVML's potential as a powerful add-on for enhancing general time series models on complex LOB data, bridging the gap between specialized LOB models and general-purpose forecasting approaches.

To illustrate CVML's effects, we analyze the standard deviation (std) of the time steps in each mid-price return input, then plot the histogram of these std values across all inputs. fig. 3.15 demonstrates that CVML significantly reduces std values of the LOB time series, indicating its effective smoothing effect. Furthermore, the distribution in fig. 3.15b more closely resembles a



(a) Raw Inputs' Std.



(b) CVML outputs' Std.

Figure 3.15. **Standard Deviation Distribution (Std.) of Inputs Before and After CVML Processing on the FI-2010 test set.** This histogram is from the TimeMixer with CVML add-on. The CVML outputs show lower std values and a distribution closer to normal, suggesting noise reduction.

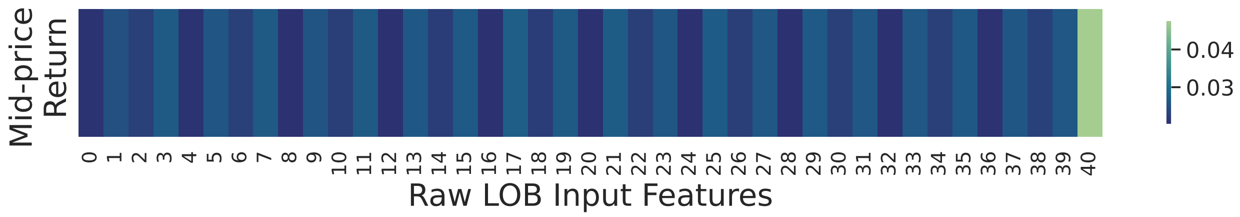
normal distribution. These transformations collectively enhance the time series models' ability to capture temporal signals by presenting them with more structured and less noisy data. This visualization provides evidence of CVML's role in improving the signal-to-noise ratio of LOB data, thereby facilitating more accurate predictions by subsequent time series models.

We demonstrate that CVML enhances the ability of time series models to capture cross-variate and temporal correlations, using iTransformer and PatchTST as examples. We chose them for their interpretable attention mechanisms and their representative modeling on different correlation types: iTransformer focuses on cross-variate correlations through self-attention on the variate dimension, while PatchTST emphasizes temporal correlations via self-attention on time dimension patches. We train both models on FI-2010 and analyze the average attention scores from their final attention layers across the test set. For iTransformer, we visualize attention scores from the mid-price return (target variate, id: 40) to all other variates and itself. For PatchTST, we examine average attention scores from the last time step patch (id: 10) to all other patches and itself.

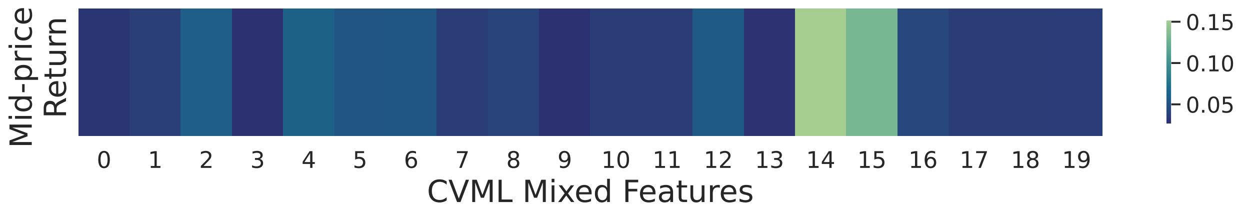
Cross-Variate Correlations: fig. 3.16a reveals that without CVML, iTransformer's mid-price return attention is predominantly self-focused, with a uniform pattern corresponding to the LOB input feature layout ($\{V_i^{\text{bid}}, P_i^{\text{bid}}, V_i^{\text{ask}}, P_i^{\text{ask}}\}_{i=1}^{10}$). This uniform attention suggests only surface-level capture of cross-variate correlations, failing to differentiate between LOB levels. Notably, it doesn't reflect the expected stronger correlation of mid-price to the best bid and ask prices. In contrast, fig. 3.16b shows that with CVML, iTransformer exhibits varied attention across variates, indicating a more nuanced modeling of LOB-level correlations. **Temporal Correlations:** fig. 3.16c demonstrates PatchTST's attention w/o CVML, showing strong self-attention for the latest time patch but no discernible temporal pattern with other patches. Conversely, fig. 3.16d demonstrates that with CVML, PatchTST captures a clear decaying temporal pattern and stronger attention to immediate past neighbors, indicating improved modeling of temporal dependencies.

3.4.4.4. Ablation Study. CVML is designed to capture two types of correlations: cross-variate and temporal. To demonstrate the efficacy of this design, we conduct an ablation study using two modified versions of CVML. The first variant (CVML-abla1) reduces the kernel size to 1, focusing solely on cross-variate correlations while eliminating temporal correlations. The second variant (CVML-abla2) maintains the original kernel size of 2 but employs depthwise convolution (Chollet, 2017; Pandey, 2024), which processes each variate independently without cross-variate information aggregation. We replicate the experiments from table 3.12 using these ablated versions and compare their average forecast performance across the five prediction horizons. fig. 3.17 shows that both ablated versions underperform the original CVML. These results underscore the importance of CVML's dual-correlation design, highlighting its ability to effectively capture both cross-variate and temporal dependencies in LOB data. The full results of the two ablated CVMLs including MSE, Corr, and R^2 are in table B.20.

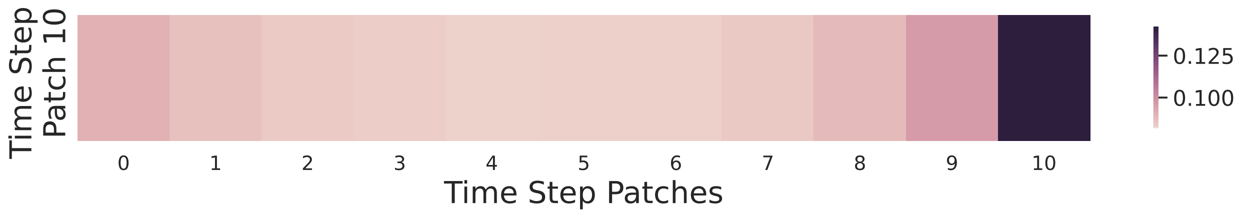
To verify that CVML's performance gain does not come model size increase, we examine the model size of each model before and after adding CVML. table 3.13 shows the number of learnable parameters. The percentage indicates the size of the vanilla model compared to the counterpart with CVML. Except TimeMixer, all other models are of more than 90% size of the counterpart with CVML. Thus, we increase TimeMixer's number of layers to increase its learnable parameters to 14156, about 109% of the CVML version and test its performance.



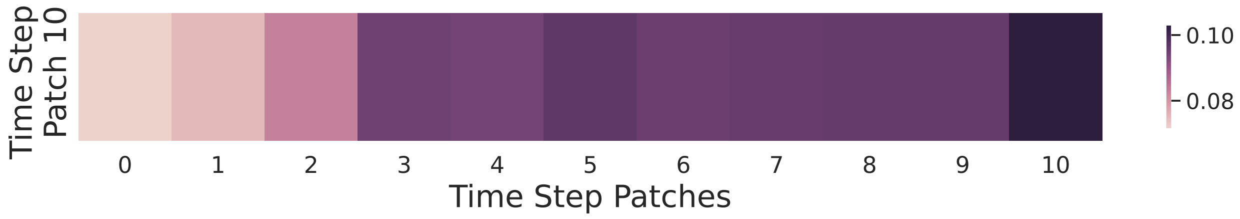
(a) iTransformer Cross-Variate Attentions (w/o CVML)



(b) iTransformer Cross-Variate Attentions (w/ CVML)



(c) PatchTST Temporal Attentions (w/o CVML)



(d) PatchTST Temporal Attentions (w/ CVML)

Figure 3.16. **Impact of CVML on Cross-Variate and Temporal Attention Scores.** (a, b) Cross-variate attention from mid-price return (id: 40) to all variates. (c, d) Temporal attention from the latest time step patch (id: 10) to all historical patches. iTransformer w/ CVML (b) captures more nuanced cross-variate correlations compared to (a). PatchTST w/ CVML (d) reveals clearer temporal dependencies than (c).

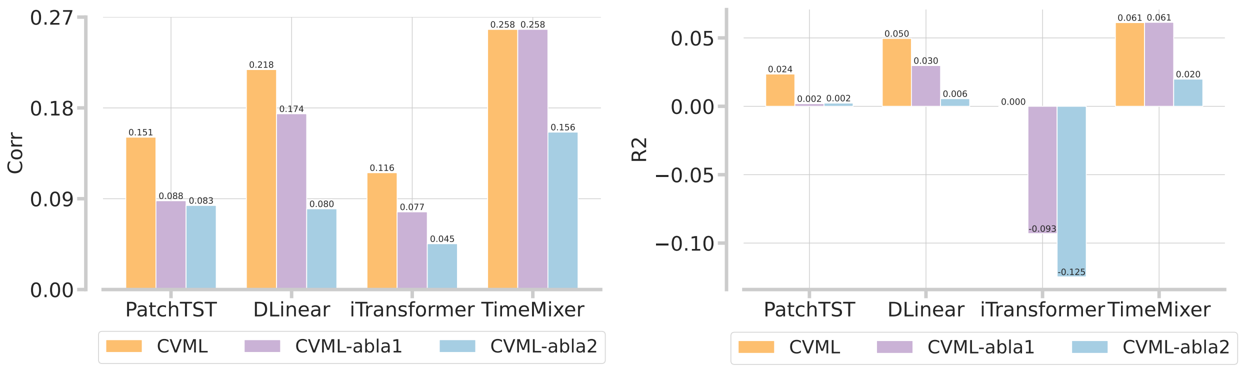
To ensure that CVML's performance gains are not solely attributable to increased model complexity, we compare the model sizes before and after incorporating CVML. table 3.13 presents the number of learnable parameters for each model, with percentages indicating the size of the vanilla model relative to its CVML-enhanced counterpart. All models except TimeMixer is over 90% of the size of the version with CVML integration. We augment TimeMixer's architecture by increasing its number of layers, resulting in 14,156 learnable parameters, approximately 109% of its CVML version's size. We then evaluate this enlarged TimeMixer and it still significantly underperforms TimeMixer-CVML, proving that CVML's gains are not solely attributable to increased model complexity.

Table 3.13. Model Size Comparison

	PatchTST	DLinear	iTransformer	TimeMixer
Vanilla	33766 (92.65%)	8282 (148%)	6358017 (99.96%)	9471 (72.71%)
w/ CVML	36444	5614	6360803	13025

Table 3.14. Enlarged TimeMixer (109% of TimeMixer+CVML size)

	K=1	K=2	K=3	K=5	K=10
Corr	0.072	0.110	0.146	0.206	0.268
R^2	-0.010	-0.001	0.011	0.030	0.053

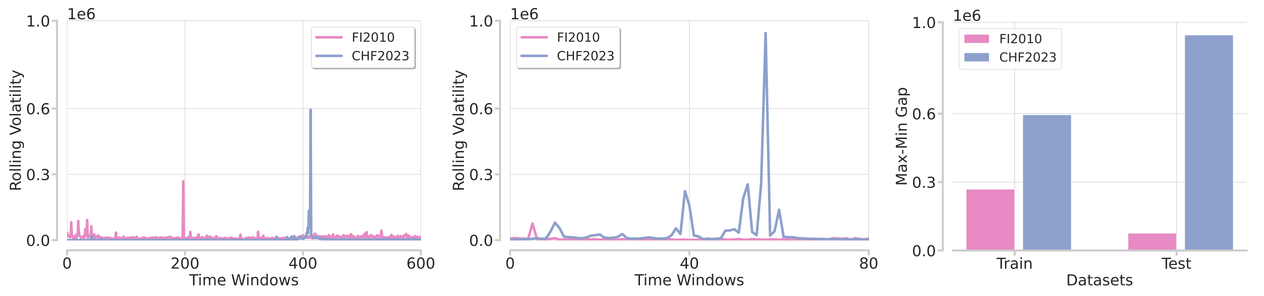


(a) Correlation (b) R^2

Figure 3.17. **Average Correlation and R^2 Scores Across Five Prediction Horizons.** This figure compares the performance of CVML and its two ablated versions (CVML-abla1 and CVML-abla2) across four time series models. The CVML consistently outperforms its ablated counterparts. Notably, CVML-abla2, which lacks cross-variate information aggregation, performs the worst, highlighting the critical importance of cross-variate mixing in CVML's effectiveness.

3.4.5. Discussions

Prediction Performance Gap Between FI-2010 and CHF-2023 in MPTP. Overall, models demonstrate superior performance on the FI-2010 dataset but not on the CHF-2023 dataset. To



(a) FI-2010/CHF-2023 Training Data (b) FI-2010/CHF-2023 Testing Data (c) FI-2010/CHF-2023 Max-Min Gap in Rolling Volatility

Figure 3.18. **Rolling Volatility Comparison: FI-2010 vs. CHF-2023.** The CHF-2023 dataset exhibits sharp volatility spikes compared to the more stable FI-2010 dataset. (a) and (b) display the maximum volatility point for every 500 time steps. (c) shows CHF-2023 has a larger max-min gap.

understand this discrepancy, we analyze the rolling volatility of both datasets using the formula: $\sigma_S = \mathrm{std}(S) \times \sqrt{\mathrm{annualized}}$ where σ_S is the standard deviation of mid-price returns in a history window of size S, and $\sqrt{\mathrm{annualized}}$ is a normalization factor. The $\mathrm{annualized}$ term equals the number of time steps in a 252-day trading year at the dataset's resolution. We calculate rolling volatility using a sliding window with $S = 20$ and a step size of 1. fig. 3.18 reveals that FI-2010's rolling volatility is relatively consistent across the entire time series. In contrast, CHF-2023 exhibits several extreme volatility spikes, presenting significant challenges for model fitting and prediction. Two notable spike periods are identified: September 2021 to January 2022: Coinciding with crude oil price increases due to supply disruptions and production constraints (United States International Trade Commission, 2021). May 2023 to June 2023: Corresponding to crude oil price fluctuations following the EU's import ban on Russian crude oil and products (French, 2024). This analysis highlights the unique characteristics and challenges inherent in the futures dataset compared to FI-2010. The inclusion of futures data in our benchmark provides a more comprehensive evaluation, capturing market dynamics not present in stock-only datasets. This diversity in data sources enhances the robustness and applicability of our benchmark study to real-world financial forecasting scenarios.

3.4.6. Conclusion

We present a comprehensive benchmark of neural network architectures on two critical LOB prediction tasks: mid-price trend prediction (MPTP) and mid-price return forecasting (MPRF), using both an open-source stock LOB dataset and a proprietary futures LOB dataset. We compare specialized LOB models against state-of-the-art general-purpose time series forecasting models and propose an add-on architecture to improve general-purpose time series forecasting models' forecasting performance on LOB data. Our research yields **four** conclusions.

- **Feature Importance.** All three sets of LOB features, basic, time-insensitive, and time-sensitive, demonstrate significant predictive power for the MPTP task.
- **Asset-Specific Challenges.** The futures LOB dataset exhibits unique characteristics that degrade the predictive performance of LOB models initially designed on stock data.
- **Model Specialization.** LOB-specific models significantly outperform general-purpose state-of-the-art time series models on LOB data for the MPRF task.
- **CVML Add-on.** Our proposed add-on, cross-variate mixing layers (CVML), substantially improve the predictive performance of general-purpose time series models.

3.5. Hopformer: Homogeneity-Pursuit Transformer for Time Series Forecasting

3.5.1. Introduction

Time series forecasting is a fundamental problem in deep learning with applications in finance, healthcare, energy, and beyond. Traditional forecasting approaches such as ARIMA, Exponential Smoothing (ETS), and Gaussian Processes have long provided interpretable and statistically robust models, but they often struggle with non-stationary, long-range dependencies. Deep learning methods, particularly Transformers, have demonstrated superior performance in capturing complex temporal dependencies, but at the cost of computational overhead and reduced interpretability (Zhou et al., 2021; Li et al., 2019; Zhou et al., 2022; Zhang and Yan, 2023; Wu et al., 2021).

Recent work on universal time series forecasting has sought to create models that generalize across multiple datasets with cross-frequency structure, heterogeneous dimensionality, and distributional variation (Woo et al., 2024a; Liu et al., 2024a; Ansari et al., 2024b; Liu et al., 2024c; Das et al., 2024; Nie et al., 2022). These large pretrained time series models have demonstrated strong zero-shot generalizability on time series forecasting tasks across different domains. However, a central challenge in this setting is the presence of high-dimensional or sparsely observed external covariates, which can vary across datasets. It is still an open question on developing universal time series forecasters that integrate high dimensional covariates. Some models, such as Chronos (Ansari et al., 2024b), avoid the use of covariates altogether, instead, focusing on univariate time series modeling alone. In contrast, MOIRAI (Woo et al., 2024a), a Transformer-based pretrained multivariate time series model, supports a subset of types of covariates, that is, those that form a multivariate time series. Besides, in MOIRAI's experiments, many datasets involve few or no correlated covariates , suggesting that its generalization capability is not well studied with covariates involved. Motivated by the observation of this gap, we seek to step further towards the goal of universal forecasters. We develop a methodology that integrates covariates modeling into a typical time series forecasting framework to achieve better time series forecasting performance on top of the existing SOTA large pretrained time series models.

In this paper, we propose Hopformer (**Ho**mogeneity-**P**ursuit Trans**former**), a novel and flexible two-stage forecasting framework designed to address the aforementioned challenges. In the first stage, Hopformer leverages a pool of cross-sectional regression models—including linear, tree-based, and neural regressors—to extract deterministic trend components from time series and covariates. These models are combined via a sparsity pattern aggregation (SPA)(Rigollet and Tsybakov, 2011) scheme, which assigns exponentially weighted convex combinations to experts based on their empirical performance and complexity. This stage serves as a universal interface that aligns heterogeneous series through trend-level homogeneity.

The second stage focuses on modeling residual dynamics. We apply LoRA (Low-Rank Adaptation) (Hu et al., 2022) to fine-tune a pre-trained Transformer on the residuals. This allows us to efficiently capture nonlinear and long-range temporal dependencies with minimal computational cost. By tuning only a small number of low-rank parameters, LoRA preserves the expressiveness of the Transformer while enabling scalability and fast adaptation.

The key contributions of this work are as follows:

- We propose Hopformer, a hybrid two-stage framework that combines expert-based cross-sectional trend extraction with efficient residual modeling via a LoRA-fine-tuned Transformer. Our method performs homogeneity pursuit through a learnable trend interface that unifies heterogeneous covariates and time series into a shared representation space.
- We design and analyze a sparsity pattern aggregation scheme that adaptively combines regression experts, and we establish an oracle inequality that guarantees near-optimal performance compared to the best sparse model in the expert pool.
- For the residual modeling stage, we apply LoRA fine-tuning and provide generalization bounds under dependent data, offering theoretical justification for its robust performance.
- Empirical results show that Hopformer outperforms SOTA forecasters across domains.

3.5.2. Related Work

Universal Time Series Forecasting. Traditional time series forecasting methods often adopt a one-model-per-dataset approach(Wu et al., 2023b; Nie et al., 2023; Liu et al., 2024b), limiting their ability to generalize across diverse datasets. Some models have explored generative pre-training transformers specifically for time series forecasting(Cao et al., 2024; Xue and Salim, 2023; Ekambaram et al., 2024; Jin et al., 2024). These approaches require users to design and train new modules for each task or limit their application to a single type of tasks. To address this limitation, recent research has focused on developing universal forecasting models capable of handling a wide range of time series data. SimpleTS (Yao et al., 2023) introduces an adaptive model selection framework that efficiently classifies time series types and selects optimal prediction models, improving generalization across diverse datasets. MOIRAI(Woo et al., 2024a), a Masked Encoder-based Universal Time Series Forecasting Transformer, introduces enhancements to the conventional Transformer architecture to tackle challenges such as cross-frequency learning and accommodating varying numbers of variates in multivariate time series. FlexTSF (Xiao et al., 2024) proposes a universal forecasting model that supports both regular and irregular time series, enhancing generalization across datasets with variable regularities. UniTS(Gao et al., 2024) integrates predictive and generative tasks into a single framework, utilizing a modified transformer to capture universal time series representations. These advancements signify a shift towards more adaptable and generalized time series forecasting models.

LoRA and Efficient Fine-Tuning for Forecasting. Low-Rank Adaptation (LoRA) has emerged as a prominent technique for the efficient fine-tuning of large-scale pre-trained models (Hu et al., 2022). By introducing low-rank updates to the model's weights, LoRA significantly reduces the number of trainable parameters, thereby lowering computational costs and mitigating the risk of overfitting . In the context of time series forecasting, LoRA has been applied to foundational models like Lag-Llama(Rasul et al., 2024) and Chronos(Ansari et al., 2024b), demonstrating its potential for adapting to out-of-domain modalities(Gupta et al., 2024a). Furthermore, research exploring Parameter-Efficient Fine-Tuning (PEFT) techniques, including LoRA, has shown improved forecasting performance in healthcare applications, particularly in predicting vital signs of sepsis patients(Gupta et al., 2024b). These studies highlight the effectiveness of LoRA and similar PEFT methods in enhancing the adaptability and efficiency of time series forecasting models.

3.5.3. Methodology

Problem Formulation. We consider a dataset of N time series $\mathcal{D} = \{\mathbf{X}^{(i)}\}_{i=1}^{N}$, where each time series $\mathbf{X}^{(i)} = (X_1^{(i)}, X_2^{(i)}, \ldots, X_{T_i}^{(i)}) \in \mathbb{R}^{T_i \times D}$ consists of T_i time steps. Each time series shares a common set of covariates $\mathbf{Z}^{(i)} = (Z_1^{(i)}, Z_2^{(i)}, \ldots, Z_{T_i}^{(i)}) \in \mathbb{R}^{T_i \times d}$, where each column represents the same set of features including lagged values, seasonal terms, or trend indicators across all series at time t. While the target values $\mathbf{X}^{(i)}$ vary across time series, the associated predictors are constructed using a shared feature design, enabling consistent modeling and aggregation across heterogeneous time series. The objective of this problem is to develop a forecaster (regressor) F to have a prediction $\mathbf{X}^{(i)}_{t:t+h} = F(\mathbf{Z}^{(i)}_{t-l:t+h})$ at decision time step t. In this paper, we approach this formula with a two-stage cross-sectional regression framework. In the first stage, we use multiple expert models $\{g_j\}_{j=1}^{M}$ to extract trend component $X^{(i),trend}_{t+h}$ by solving a cross-sectional problem, $X^{(i),trend}_{t} = \sum_{j=1}^{M} \omega_j g_j(\mathbf{Z}^{(i)}_t) + \epsilon^{(i)}_t$. In the second stage, we use a LoRA-fine-tuned Transformer model to further approximate the residuals $\epsilon^{(i)}_t$ with $\widehat{R}^{(i)}_{t:t+h} = f^{\text{LoRA}}_{\mathbf{W}}(R^{(i)}_{t-l:t})$. We finally aggregate the prediction as $\widehat{X}^{(i)}_{t:t+h} = \widehat{X}^{(i),\text{trend}}_{t:t+h} + \widehat{R}^{(i)}_{t:t+h}$.

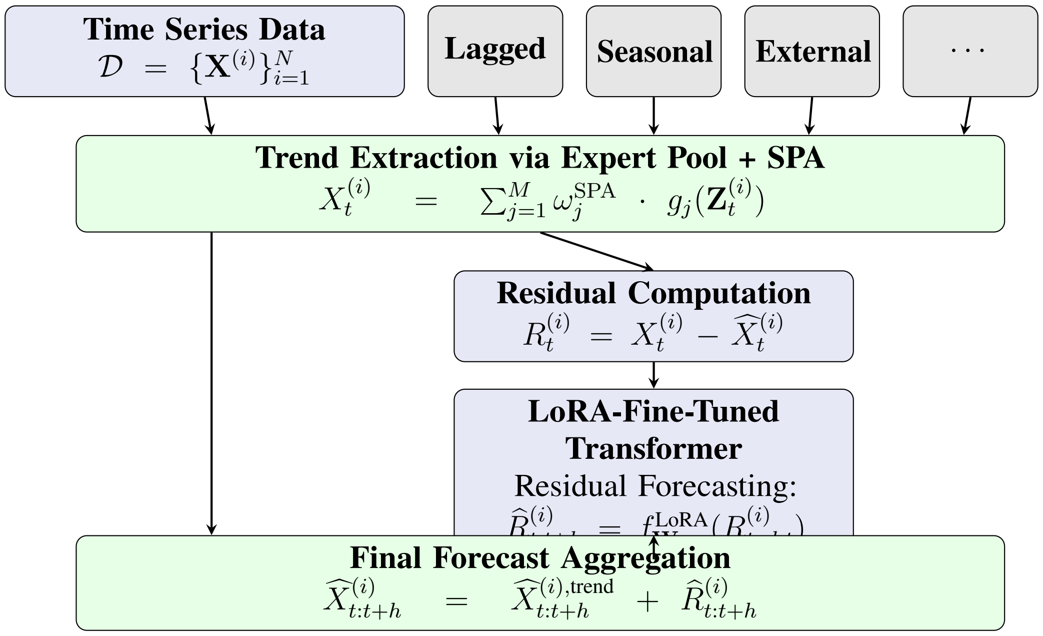

Time Series Data
$\mathcal{D} = \{\mathbf{X}^{(i)}\}_{i=1}^{N}$
Lagged
Seasonal
External
$\cdots$
Trend Extraction via Expert Pool + SPA
$X^{(i)}_t = \sum_{j=1}^{M} \omega^{\text{SPA}}_j \cdot g_j(\mathbf{Z}^{(i)}_t)$
Residual Computation
$R^{(i)}_t = X^{(i)}_t - \widehat{X}^{(i)}_t$
LoRA-Fine-Tuned Transformer
Residual Forecasting:
$\widehat{R}^{(i)}_{t:t+h} = f^{\text{LoRA}}_{\mathbf{W}}(R^{(i)}_{t-l:t})$
Final Forecast Aggregation
$\widehat{X}^{(i)}_{t:t+h} = \widehat{X}^{(i),\text{trend}}_{t:t+h} + \widehat{R}^{(i)}_{t:t+h}$

Figure 3.19. Workflow of the Hopformer framework.

Universal time series forecasting presents significant challenges due to the high heterogeneity across datasets. Time series often exhibit varying temporal patterns, distributional properties, and covariate structures, making it difficult for a single model to generalize effectively. While prior approaches have attempted to mitigate this by explicitly modeling external covariates, empirical results suggest that their practical contribution to forecasting accuracy may be limited—especially when such covariates are sparse, noisy, or inconsistently available across domains.

To address this, we propose a unified trend extraction strategy based on aggregating a diverse pool of regression models, each capturing different structural aspects of the time series. Rather than relying on a fixed parametric form, we apply a sparsity pattern aggregation scheme that combines multiple candidate models—such as those incorporating trend, seasonality, and autoregressive dynamics—into a single, adaptive forecast. This step serves as a homogeneity pursuit mechanism: by systematically absorbing the effect of available covariates and deterministic patterns, we project heterogeneous time series into a shared residual space, where distributional and structural variation is substantially reduced.

This decomposition offers several advantages:

- It captures a wide range of interpretable trends and temporal effects using a flexible, data-driven combination of expert models.
- It avoids the need for explicit covariate modeling, alignment, or preprocessing—improving robustness and generalization in settings with missing or non-standard metadata.
- It enables the second-stage model to focus on learning residual, nonlinear dependencies in a more homogeneous representation space, facilitating more efficient and stable deep forecasting.

3.5.3.1. Cross-Sectional Regression Model for Trend Extraction. To extract the trend, we first construct an ensemble of cross-sectional regression models $\{g_j\}_{j=1}^{M}$, each trained to capture different aspects of the temporal structure using a shared predictor design. The final trend estimate is obtained by aggregating these models via a weighted combination:

$$X_t^{(i)} = \sum_{j=1}^{M} \omega_j^{\mathrm{SPA}} \cdot g_j(\mathbf{Z}_t^{(i)}) \tag{3.5.1}$$

where $\mathbf{Z}_t^{(i)} \in \mathbb{R}^d$ denotes the covariates for time series

i at time t, constructed using common features such as lagged values, seasonal indicators, trend terms, and optionally, external covariates.

For example, consider X_t as the daily sales for a retail store at time t.

The feature set $\boldsymbol{Z}_t$ may include: the linear trend (t) to model long-term growth or decline; the seasonality indicators, $\sin(2\pi t/7)$, $\cos(2\pi t/7)$, to capture weekly periodicity; the external covariate such as promotion indicator $\text{Promo}_t \in \{0,1\}$; the lagged sales value X_{t-7} to model weekly autocorrelation. The *sparsity pattern aggregate* (SPA) weights $\{\omega_j^{\text{SPA}}\}$ are adaptively assigned to balance empirical performance and model complexity, enabling the ensemble to robustly capture dominant deterministic patterns across diverse time series. While the SPA weights $\omega^{\text{SPA}} = (\omega_1^{\text{SPA}}, \ldots, \omega_M^{\text{SPA}})$ formally depend on both the time index t and the series index i, we simplify notation by omitting these subscripts throughout this section and the following theoretical results. These weights are constructed as following:

We define a sparsity pattern as a binary vector $\boldsymbol{p} \in \mathcal{P} := \{0,1\}^M$, where each coordinate $p_j \in \{0,1\}$ indicates the inclusion ($p_j = 1$) or exclusion ($p_j = 0$) of the j-th expert model in the aggregation. For any $\boldsymbol{p} \in \mathcal{P}$, we define the restricted parameter space:

$$\mathbb{R}^{\boldsymbol{p}} = \{\omega \cdot \boldsymbol{p} : \omega \in \mathbb{R}^M\} \subseteq \mathbb{R}^M,$$

where $\cdot$ denotes the Hadamard (elementwise) product. The dimensionality of this subspace is denoted by $|\boldsymbol{p}|$, the number of nonzero components in the pattern $\boldsymbol{p}$. To formalize the SPA construction, we consider the sample version of the workflow illustrated in Figure 3.19. For each time series i and time point t, we assume access to an i.i.d. sample of size n drawn from the underlying conditional distribution. Specifically, let $\{(X_{t,k}^{(i)}, \mathbf{Z}_{t,k}^{(i)})\}_{k=1}^n$ denote the observed realizations of the response and corresponding predictor vectors, where $\mathbf{Z}_{t,k}^{(i)} \in \mathbb{R}^d$ is the feature vector and $X_{t,k}^{(i)} \in \mathbb{R}$ is the target time series. These samples serve as the training input for constructing the ensemble of candidate estimators and evaluating their empirical performance under the aggregation scheme. Denote the sample version of target time series and the matrix of features as $\mathbf{X}$ and $\mathbf{Z}$. For any $\boldsymbol{p} \in \mathcal{P}$, let $\widehat{\omega}_{\boldsymbol{p}}$ be the least squares estimator restricted to $\mathbb{R}^{\boldsymbol{p}}$, defined by

$$\widehat{\omega}_{\boldsymbol{p}} \in \arg\min_{\pi \in \mathbb{R}^{\boldsymbol{p}}} \|\mathbf{X} - \mathbf{Z}\pi\|_2^2. \tag{3.5.2}$$

Let $\pi = (\pi_{\boldsymbol{p}})_{\boldsymbol{p} \in \mathcal{P}}$ be a probability measure over the collection of sparsity patterns $\mathcal{P}$, which we refer to as a prior over expert selections.

The SPA weights are defined as:

$$\omega^{\text{SPA}} := \frac{\sum\limits_{\boldsymbol{p} \in \mathcal{P}} \widehat{\omega}_{\boldsymbol{p}} \exp\left(-\frac{1}{4\sigma^2} \sum\limits_{k=1}^{n} \left(X_{t,k}^{(i)} - g_{\widehat{\omega}_{\boldsymbol{p}}}(\mathbf{Z}_{t,k}^{(i)})\right)^2 - \frac{|\boldsymbol{p}|}{2}\right) \pi_{\boldsymbol{p}}}{\sum\limits_{\boldsymbol{p} \in \mathcal{P}} \exp\left(-\frac{1}{4\sigma^2} \sum\limits_{k=1}^{n} \left(X_{t,k}^{(i)} - g_{\widehat{\omega}_{\boldsymbol{p}}}(\mathbf{Z}_{t,k}^{(i)})\right)^2 - \frac{|\boldsymbol{p}|}{2}\right) \pi_{\boldsymbol{p}}}. \tag{3.5.3}$$

To instantiate the SPA weights, we adopt the following prior over the space of patterns $\boldsymbol{p} \in \mathcal{P}$ as in Rigollet and Tsybakov (2011):

$$\pi_{\boldsymbol{p}} := \begin{cases} \frac{1}{H}\left(\frac{|\boldsymbol{p}|}{2eM}\right)^{|\boldsymbol{p}|}, & \text{if } |\boldsymbol{p}| < R, \\ \frac{1}{2}, & \text{if } |\boldsymbol{p}| = M, \\ 0, & \text{otherwise}, \end{cases}$$

where $R = \text{rk}(\mathbf{Z})$ is the rank of the design matrix, and $H = 2\sum_{m=0}^{R} \binom{M}{m} \left(\frac{m}{2eM}\right)^m$ is a normalization constant. This prior favors sparse patterns while ensuring that the full model retains sufficient mass for statistical guarantees. Following Rigollet and Tsybakov (2011), the SPA estimator under this prior can be efficiently implemented using a Metropolis approximation.

This formulation allows Hopformer to adaptively emphasize simpler, performant trend models, serving as a learnable interface for residual modeling in the second stage. The residual sequence is then computed as:

$$R_t^{(i)} = X_t^{(i)} - \widehat{X}_t^{(i)}, \tag{3.5.4}$$

which isolates the idiosyncratic, nonlinear, and long-range dependencies that will be modeled by a fine-tuned Transformer in the next stage.

3.5.3.2. LoRA-Fine-Tuned Transformer for Residual Forecasting. To efficiently model the residual sequence $R_t^{(i)}$ obtained in the first stage, we leverage a Transformer-based architecture, fine-tuning it using LoRA. Instead of updating the full set of parameters, LoRA injects trainable low-rank matrices into the attention layers, allowing for efficient adaptation while keeping most of the original model frozen.

For a given residual input $R_{t-l:t}^{(i)}$, we employ a Transformer to capture temporal dependencies via self-attention. The standard self-attention mechanism operates on a set of query, key, and value projections:

$$\text{Attn}(\mathbf{x}\mathbf{W}_q, \mathbf{C}\mathbf{W}_k, \mathbf{C}\mathbf{W}_v) = \text{softmax}\left(\frac{\mathbf{x}\mathbf{W}_q(\mathbf{W}_k)^\top \mathbf{C}^\top}{\sqrt{d_k}}\right)\mathbf{C}\mathbf{W}_v, \tag{3.5.5}$$

where $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i \in \mathbb{R}^{d_{in} \times d_{out}}$.

The residual forecasting model is thus defined as:

$$\widehat{R}_{t:t+h}^{(i)} = f_{\mathbf{W}}^{\text{LoRA}}(R_{t-l:t}^{(i)}) \tag{3.5.6}$$

where $f_{\mathbf{W}}^{\text{LoRA}}$ is the LoRA-enhanced Transformer model. The core adaptation mechanism follows a low-rank decomposition of the attention weight update:

$$\mathbf{W} + \Delta\mathbf{W} = \mathbf{W} + \mathbf{B}\mathbf{A} \tag{3.5.7}$$

where $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is the original pre-trained weight matrix, which remains frozen during fine-tuning. $\mathbf{B} \in \mathbb{R}^{d_{\text{in}} \times r}$ and $\mathbf{A} \in \mathbb{R}^{r \times d_{\text{out}}}$ are trainable low-rank matrices with rank $r \ll \min(d_{\text{in}}, d_{\text{out}})$. The residual update is applied as:

$$\mathbf{h} \leftarrow \mathbf{h} + s\Delta\mathbf{h}, \quad \Delta h := \mathbf{B}\mathbf{A}\mathbf{x} \tag{3.5.8}$$

where $s \geqslant 1$ is a tunable scaling hyperparameter controlling the adaptation strength.

By leveraging LoRA, the proposed method reduces computational costs while preserving the Transformer's expressive power. This enables the model to focus on capturing high-order residual dependencies without overfitting to dataset-specific noise. Finally, the predicted residuals are combined with the baseline regression forecast to obtain the final time series prediction:

$$\widehat{X}^{(i)}_{t:t+h} = \widehat{X}^{(i),\text{trend}}_{t:t+h} + \widehat{R}^{(i)}_{t:t+h} \tag{3.5.9}$$

This two-stage strategy enables efficient and scalable forecasting, capturing both deterministic trends and complex residual patterns while preserving generalization across diverse datasets.

3.5.4. Theoretical Guarantees

3.5.4.1. Oracle Inequality for Trend Aggregation. We analyze the performance of the trend aggregation procedure in the first stage of Hopformer, where the goal is to estimate an unknown regression function $\eta : \mathbb{R}^d \to \mathbb{R}^D$ based on observed data $\{(\mathbf{X}_t^{(i)}, \mathbf{Z}_t^{(i)})\}_{i=1}^N$. We assume the data is generated as

$$\mathbf{X}_t^{(i)} = \eta(\mathbf{Z}_t^{(i)}) + \xi_i, \quad i = 1, \dots, N,$$

where $\{\mathbf{Z}_t^{(i)}\}_{i=1}^N \subset \mathbb{R}^d$ are the covariates at time t and $\xi_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2 I_D)$, where I_D is identity matrix, are independent Gaussian noise variables. Let $\widehat{g}_\omega$ be the aggregated predictor using any weights ω, and define the SPA estimator as

$$\widehat{g}_{\text{SPA}}(x) = \sum_{j=1}^{M} \omega_j^{\text{SPA}} \cdot g_j(x),$$

where g_j is the j-th expert model. Then the following oracle inequality holds:

Theorem 3.5.1. Under the model defined above, the SPA estimator $\widehat{g}_{\mathrm{SPA}}$ satisfies the following inequality:

$$\mathbb{E}\|\widehat{g}_{\mathrm{SPA}} - \eta\|^2 \leqslant \min_{\boldsymbol{p}\in\mathcal{P}:\pi_{\boldsymbol{p}}\neq 0}\left\{\mathbb{E}\left\|\widehat{g}_{\widehat{\omega}_{\boldsymbol{p}}} - \eta\right\|^2 + \frac{4\sigma^2\log\left(\pi_{\boldsymbol{p}}^{-1}\right)}{n}\right\}, \tag{3.5.10}$$

where π is the prior distribution over sparsity patterns, $\widehat{\omega}_{\boldsymbol{p}}$ is the least-squares solution restricted to pattern $\boldsymbol{p}$, and $\widehat{g}_{\widehat{\omega}_{\boldsymbol{p}}}(x) = \sum_j (\widehat{\omega}_{\boldsymbol{p}})_j \cdot g_j(x)$.

The proof of this result is provided in Appendix B.5.1.1.

This result provides a theoretical guarantee that the SPA estimator performs nearly as well as the best sparse aggregation of experts in hindsight, up to a complexity penalty determined by the prior mass $\pi_{\boldsymbol{p}}$ and the sample size n. Specifically, the right-hand side of the inequality consists of two terms:

- The first term, $\mathbb{E}\left\|\widehat{g}_{\widehat{\omega}_{\boldsymbol{p}}} - \eta\right\|^2$, reflects the squared prediction error of the best estimator restricted to the support pattern $\boldsymbol{p}$.
- The second term, $\frac{4\sigma^2\log\left(\pi_{\boldsymbol{p}}^{-1}\right)}{n}$, acts as a regularization penalty that grows with the model complexity (as encoded by the prior) and shrinks with more data.

The inequality formalizes a bias-variance trade-off: sparse models (i.e., those with small $|\boldsymbol{p}|$) tend to receive higher prior probability $\pi_{\boldsymbol{p}}$ and thus incur smaller penalty terms, encouraging parsimony in the aggregated predictor. At the same time, the aggregation procedure is data-driven and adapts to patterns with low empirical error, ensuring strong approximation capability.

Together, this analysis justifies our homogeneity pursuit strategy: by leveraging a flexible prior and adaptive aggregation, we can distill the dominant shared structure across heterogeneous time series datasets into a unified trend representation that is both statistically efficient and interpretable. This learned trend interface serves as a robust input to the second-stage residual modeling.

3.5.4.2. Generalization Bound for LoRA Fine-Tuning. LoRA fine-tuning enables efficient adaptation of Transformer models to new time series data while keeping the number of trainable parameters low. To analyze the performance of the fine-tuned residual model across different time series distributions, we derive a generalization error bound.

In the context of time series learning, let $\mathcal{Z}$ denote the space of residuals obtained from cross-sectional regression. Suppose we observe a training set $\boldsymbol{R}_T = (R_1, \dots, R_T) \in \mathcal{Z}^T$ generated from a process $\mathcal{P}$. The hypothesis space is $\mathcal{W}$, and the learning algorithm takes $\boldsymbol{R}_T$ as input and outputs $\mathbf{W} \in \mathcal{W}$ according to a conditional distribution $P_{\mathbf{W}|\boldsymbol{R}_T}$. The loss function is defined as $\ell : \mathcal{W} \times \mathcal{Z} \to \mathbb{R}^+$. The population risk is given by

$$L_{\mathcal{P}}(w) = \mathbb{E}_{\mathcal{P}}[\ell(w, R)]$$

and the empirical risk is

$$L_{\boldsymbol{R}_T}(w) = \frac{1}{T-d} \sum_{i=d+1}^{N} \ell(w, R_i)$$

where $d \geqslant 0$, serves as an offset or burn-in period, ensuring that only valid samples contribute to the training loss computation. For example, in an AR(2) model, predictions depend on the past two time steps, meaning that at least two observations are required before a valid loss can be computed.

Given a fine-tuning training set $\boldsymbol{R}_T$, let r be the lora-rank, and assume each tuned parameter is quantized to q bits. Then we have the following theorem to bound the generalization error $\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})]$ using the information-theoretic generalization framework.

Theorem 3.5.2. Assuming the loss function $\ell(w, R)$ is σ-subgaussian. If the target time series is stationary and β-mixing, then there exists a constant $a > 0$ such that $2am \leqslant T$ ensuring

$$\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})] \leqslant \sqrt{\frac{2\sigma^2}{m} I(\boldsymbol{R}_T; \mathbf{W})}, \tag{3.5.11}$$

where $I(\boldsymbol{R}_T; \mathbf{W}) = D_{KL}(P_{\mathbf{W},\boldsymbol{R}_T} || P_{\mathbf{W}} \otimes P_{\boldsymbol{R}_T})$ is the mutual information and D_{KL} is the KL divergence.

We extend Theorem 1 in (Xu and Raginsky, 2017) into non-i.i.d time series data. The definitions of stationarity and mixing can be found in Appendix B.5.1.2. With the similar techniques in (Yao et al., 2024), we can bound this mutual information under LoRA-finetuning as in the following corollary:

Corollary 3.5.2.1 ((Yao et al., 2024)). With the same assumptions in Theorem 3.5.2,

$$\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})] \leqslant \sqrt{\frac{6\sigma^2}{m} qr \sum_{i \in \mathcal{I}} (d_{in} + d_{out})},$$

where $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i \in \mathbb{R}^{d_{in} \times d_{out}}$.

This bound shows that the expected generalization error depends on the number of fine-tuned parameters in LoRA, which is significantly smaller than in full fine-tuning. The low-rank adaptation introduces fewer degrees of freedom, implicitly reducing the mutual information between the training data and the learned weights. From an information-theoretic perspective, this low mutual information ensures that the model is less sensitive to any specific training instance, promoting better generalization. Although stronger bounds (e.g., high-probability concentration inequalities) can be derived under additional assumptions, the stability guarantee already provided by LoRA suffices in our setting. This result highlights the theoretical benefit of LoRA as a parameter-efficient and generalization-friendly fine-tuning method in time series forecasting.

3.5.5. Experiments

In this section, we evaluate the performance of Hopformer on a diverse set of time series forecasting benchmarks. Specifically, we address the following research questions: (i) Accuracy: How does Hopformer compare to state-of-the-art methods across a range of forecasting tasks? (ii) Ablation: What is the individual contribution of each key component of Hopformer to its overall performance? (iii) Robustness: How does Hopformer perform when varying context lengths and forecast horizons?

Dataset and Baselines: We evaluate Hopformer's performance on 6 datasets, including the Illness, EPF, and M5 benchmarks, along with three synthetic datasets (Sales1, Sales2, and Electricity). Dataset statistics are summarized in Table 3.15, where the total number of time steps is expressed as 'time steps per series × number of series'. The motivation for using these synthetic datasets is to illustrate the impact of covariates on time series forecasting while minimizing data leakage when comparing Hopformer with pre-trained foundational models for time series prediction. We benchmark Hopformer against Chronos-bolt, PatchTST, TemporalFusionTransformer, and two traditional statistical models, ARIMA and ETS. Please refer to the appendix for more details on the synthetic data.

Table 3.15. Statistics of Datasets used in Experiments.

Dataset	Illness	EPF	M5	Sale1	Sale2	Electricity
Covariates	7	2	12	4	8	7
Timesteps	966	52 416 x 5	414 x 30 490	730 x 200	730 x 200	184 800 x 5

Implementation and Evaluation Metrics: We implement Hopformer using the AutoGluon library (Shchur et al., 2023). In the cross-sectional stage, the expert pool includes regression models such as XGBoost, LightGBM, Linear Regression, Random Forest, and CatBoost for processing future covariates, while a SimpleFeedForward network is used for past covariates. We use default hyperparameter for these regressors, and use ARIMA and ETS to capture lag and seasonal patterns, with max_ts_steps = 1000. For aggregation, we apply SPA (Equation 3.5.3) to Hopformer, and implement Linear Regression (Equation 3.5.2), Equal Weighting, and Single Best strategies for our ablation study. In the second stage, we use the Chronos-bolt-small (Chronos) model to implement three models: a zero-shot model without fine-tuning, a fully fine-tuned model, and a model fine-tuned using LoRA. For LoRA, we employ a low-rank adaptation with rank = 8, a scaling factor of $\alpha = 16$, and a dropout rate of 5%, applied to the query, key, value, and output projections. We fine-tune models for 100 gradient steps.

We evaluate Hopformer using 20 rolling windows, each with a context length of 512 and a forecast horizon of 24. We partition the dataset so that each test set consists of 20 consecutive prediction periods, with one prediction period reserved for validation, and the remaining data used for training. The latest 1,000 time steps serve to train the regressors and compute the SPA weights. Model performance is assessed by computing the Mean Absolute Scaled Error (MASE) and Mean

Absolute Percentage Error (MAPE), averaged over all rolling windows. See Table 3.16 for our results.

Table 3.16. Forecasting performance of Hopformer, Chornos-bolt-small (Chronos), their hybrid variants, and PatchTST (PTST), TemporalFusionTranformer (TFT), ARIMA, and ETS. Each cell reports MASE, and MASE (lower is better). The bold numbers indicate the two lowest metric values in each row.

Models		Hopformer			Cross-Sectional				Chronos			DL and Stats			
Variants		0-shot	Full	LoRA	SPA	Lasso	Best	Equal	0-shot	Full	LoRA	PTST	TFT	Arima	Ets
Illness	MASE	3.805	**3.297**	**3.289**	3.686	3.791	4.940	4.537	3.568	3.982	3.985	4.231	4.940	5.153	5.091
	MAPE	0.133	**0.115**	**0.115**	0.125	0.130	0.167	0.159	0.123	0.135	0.134	0.146	0.167	0.190	0.188
EPF	MASE	**0.654**	0.664	0.667	1.279	1.760	0.813	17.17	**0.662**	0.674	0.720	1.466	0.862	0.895	1.091
	MAPE	**1.114**	1.191	1.181	2.554	3.027	1.804	11.67	1.180	1.252	1.265	3.252	1.689	1.383	**1.124**
M5	MASE	1.006	0.988	0.989	1.189	1.188	2.189	2.194	1.006	**0.987**	0.987	1.022	**0.980**	1.185	1.238
	MAPE	0.703	0.678	0.680	**0.591**	0.596	1.438	1.883	0.704	0.683	0.679	0.667	0.671	0.594	**0.592**
Sale1	MASE	0.946	**0.761**	**0.819**	1.592	1.598	1.646	1.610	1.095	0.927	0.971	0.911	0.883	1.191	1.136
	MAPE	0.915	**0.631**	0.686	1.482	1.483	1.510	1.523	0.951	**0.679**	0.707	0.881	0.813	1.029	1.380
Sale2	MASE	0.340	**0.270**	**0.264**	0.538	0.539	0.539	0.561	0.542	0.307	0.301	0.447	0.428	0.552	0.849
	MAPE	0.423	**0.306**	**0.301**	0.729	0.740	0.736	0.839	0.518	0.315	0.309	0.486	0.485	635	0.912
Elec.	MASE	0.765	**0.737**	**0.730**	2.612	2.648	2.633	2.652	0.817	0.770	0.756	1.205	1.391	0.940	1.449
	MAPE	0.078	**0.075**	**0.075**	0.263	0.266	0.257	0.259	0.083	0.079	0.078	0.143	0.164	0.106	0.171

Overall Performance: Table 3.16 summarizes the performance of the forecasting models across six datasets. We highlight three key findings: (i) The best of the Hopformer variants outperforms baseline models, with an average relative improvement of 6.56% in MASE across all datasets. Performance gains are particularly notable on synthetic datasets with future covariates, achieving a 4.45% improvement in MAPE compared to the best baseline. (ii) In the zero-shot scenario, Hopformer attains comparable or superior results to Chronos in five out of six datasets, illustrating the efficacy of the SPA. Specifically, SPA achieves a 8.73% average reduction in MASE, highlighting its general compatibility with foundational models (visualized in Figure 3.20). (iii) Fine-tuning the residual module with LoRA achieves nearly identical performance to full fine-tuning (within 1% difference in MAPE), underscoring LoRA's efficiency by updating only targeted attention parameters.

Ablation Study: To isolate and quantify the contributions of Hopformer's aggregation strategies, we conduct an ablation analysis. As reported in cross-sectional column of Table 3.16, SPA delivers

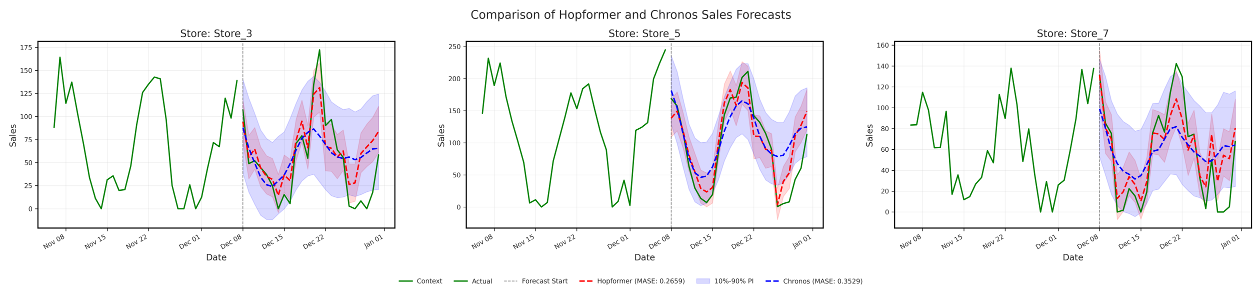


Figure 3.20. Zero-shot forecast comparison between Hopformer and Chronos on four representative stores from the Sales1 dataset. Blue lines show the ground-truth sales, while yellow and red lines depict model predictions.

the lowest MASE and MAPE on every dataset, cutting MASE by an average of 7.08% and MAPE by 6.99% relative to the next-best strategy. We further stress-test the methods by varying the expert-pool size from 2 to 20 regressors (seen in appendix). SPA's gain widens as the pool grows.

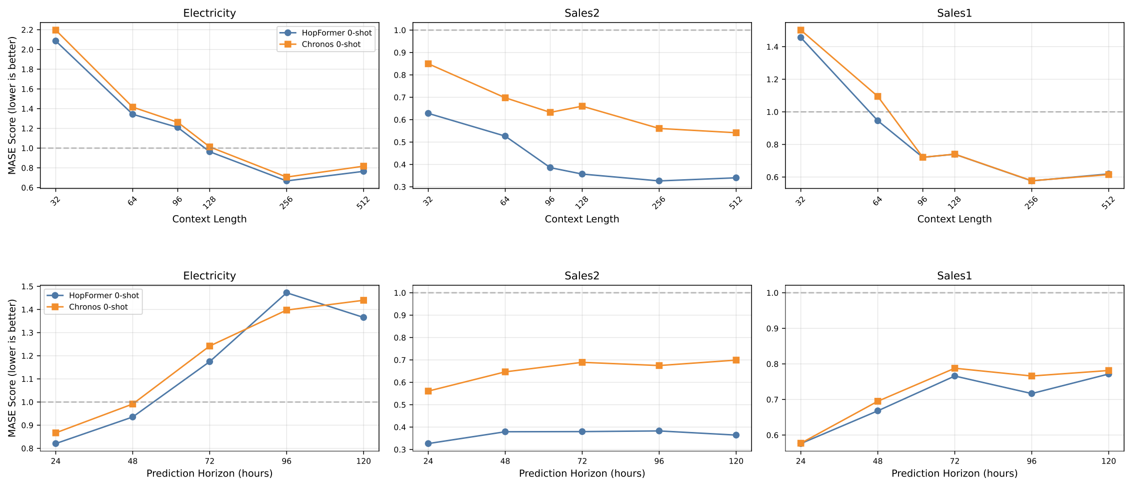


Figure 3.21. (Top) Effect of Context Length on Model Performance Across Datasets (prediction length = 24). (Bottom) Effect of Prediction Horizon on Model Performance Across Datasets (context length = 256).

Robustness Analysis: Figure 3.21 probes Hopformer's robustness on zero-shot setting by sweeping (top) the context window and (bottom) the forecast horizon on all six benchmarks. The result yields two main take-aways. (i) As the context window contracts, Hopformer retains most of its SPA-based advantage over Chronos. On Sales2, shrinking the context from 512 to 32 steps raises Chronos' MASE by 0.50, but Hopformer's by only 0.30. (ii) Both models deteriorate as the horizon grows, yet Hopformer remains ahead across all ranges because it starts from a lower error.

On Sales2, the jump from 24 to 120-step forecasts adds 0.12 MASE to Hopformer and 0.05 to Chronos, but Hopformer's absolute error is still lower at every horizon. These patterns confirm that the SPA aggregation helps Hopformer preserve its advantage under both limited-context and long-range-forecast settings.

3.5.6. Discussion and Conclusion

In this work, we introduced Hopformer, a novel two-stage forecasting framework designed for time series with high-dimensional covariates. In the first stage, Hopformer performs trend extraction via a sparse aggregation over a pool of expert regressors, allowing it to adaptively absorb the effects of covariates without requiring them as explicit inputs. This aggregation is guided by a sparsity pattern prior and implemented using an exponentially weighted scheme with provable oracle guarantees. In the second stage, a LoRA-fine-tuned Transformer models the residual sequence, capturing long-range and nonlinear dependencies with minimal parameter overhead. Our theoretical analysis establishes generalization bounds for both trend aggregation and LoRA fine-tuning, demonstrating that our method remains stable under dependent data and achieves near-optimal error with controlled complexity. Empirically, Hopformer delivers state-of-the-art accuracy across a diverse set of time series benchmarks, validating the effectiveness of combining sparsity-driven regression with parameter-efficient residual modeling.

Several directions remain open for exploration. First, exploring alternative parameter-efficient fine-tuning approaches beyond LoRA both theoretically and empirically could enhance flexibility in various scenarios. Second, incorporating task-aware pretraining or meta-learning strategies into the residual modeling stage may further improve robustness and cross-task generalization.

CHAPTER 4

Accelerating Scientific Discoveries in Astrophysics with Time Series Foundation Models

In this chapter, we study the applications of time series foundation models on one of the most data-intensive frontiers in modern astronomy: time-domain surveys. The Zwicky Transient Facility (ZTF) nightly records millions of irregular, multi-band light-curves whose variability encodes stellar structure, binary dynamics, and explosive transients. Manual feature engineering and survey-specific models can no longer keep pace with this deluge. We introduce StarEmbed(Li et al., 2025c), the first benchmark study that develops an effective mechanism to utilize time series foundation models' abilities to help with periodic-variable-star classification, an important scientific task in the astrophysics domain. It also comprehensively evaluates pre-trained time-series foundation models on this task. We show that generic TSFMs, without domain retraining, approach the accuracy of astronomy-specific models and hand-crafted features, while offering far greater scalability. These results demonstrate the great potentials of utilizing time series foundation models to accelerate discoveries in scientific domains.

4.1. StarEmbed: A Benchmark for Evaluating Pre-trained Light-Curve Embeddings in Variable Star Classification

4.1.1. Introduction

Dynamic sources in the night sky have driven many paradigm shifting discoveries over the history of astronomy. Stars that exhibit brightness variations over regular, periodic intervals are especially valuable as they provide unique probes of stellar interiors and evolution, galactic structure, and can be used to measure the distance to nearby galaxies (e.g., Feast and Walker, 1987; Clementini et al., 2003; Genovali et al., 2014; Catelan and Smith, 2015; Ripepi et al., 2017). Dozens of different types of variable stars exist, and modern astronomical surveys, like the Zwicky Transient Facility (ZTF; Bellm et al., 2019), have produced an avalanche of multi-variate time-series data ("light curves;" $\sim 10^9$ stars each with $\sim 10^3$ observations over 7 yr from ZTF alone). These light curves are multi-variate because observations are conducted with a filter (or "passband") placed along the focal path of the telescope, limiting the image to only light from a specific wavelength range. Thus, light curves contain both brightness and "color" information (i.e. relative brightnesses across passbands), enabling inference of physical properties of the source. Less than 1% of the observed stars are periodic variables, but they provide outsize value relative to their non-variable counterparts, making it imperative that these stars be identified and analyzed via automated methods. The challenges associated with finding and sorting these variables will be exacerbated by the upcoming Vera C. Rubin Observatory, which will begin the Legacy Survey of Space and Time (LSST; Ivezić et al., 2019) in 2025 and discover $>10^8$ variable stars while monitoring $>10^{10}$ stars over the course of a decade.

Astronomical observations provide unique challenges within the domain of time-series analysis. In particular, there are frequent gaps of variable intervals in the observations (see Figure 4.1), and the presence of clouds, which change day-to-day and hour-by-hour, yields heteroskedastic uncertainties for the individual observations. Traditionally, classification has relied on extracting domain-specific features (periods, amplitudes, etc.) and using tree-based classifiers [e.g., Richards et al. 2011; Nun et al. 2015; Malanchev et al. 2021; Sánchez-Sáez et al. 2021]. However, the rise of deep learning and self-supervised learning has led to pre-trained embedding models that can encode light curves into rich feature vectors without manual feature engineering (Donoso-Oliva et al., 2023; Donoso-Oliva et al., 2025; Zuo et al., 2025). No standardized benchmarks for evaluating time-series embeddings exist in the astrophysics domain. This absence of benchmarks hinders objective evaluation of specialized models (Pan et al., 2024). With no benchmarks, it is impossible to assess the true benefits of domain-specific pre-training. We aim to establish a comprehensive evaluation of multiple

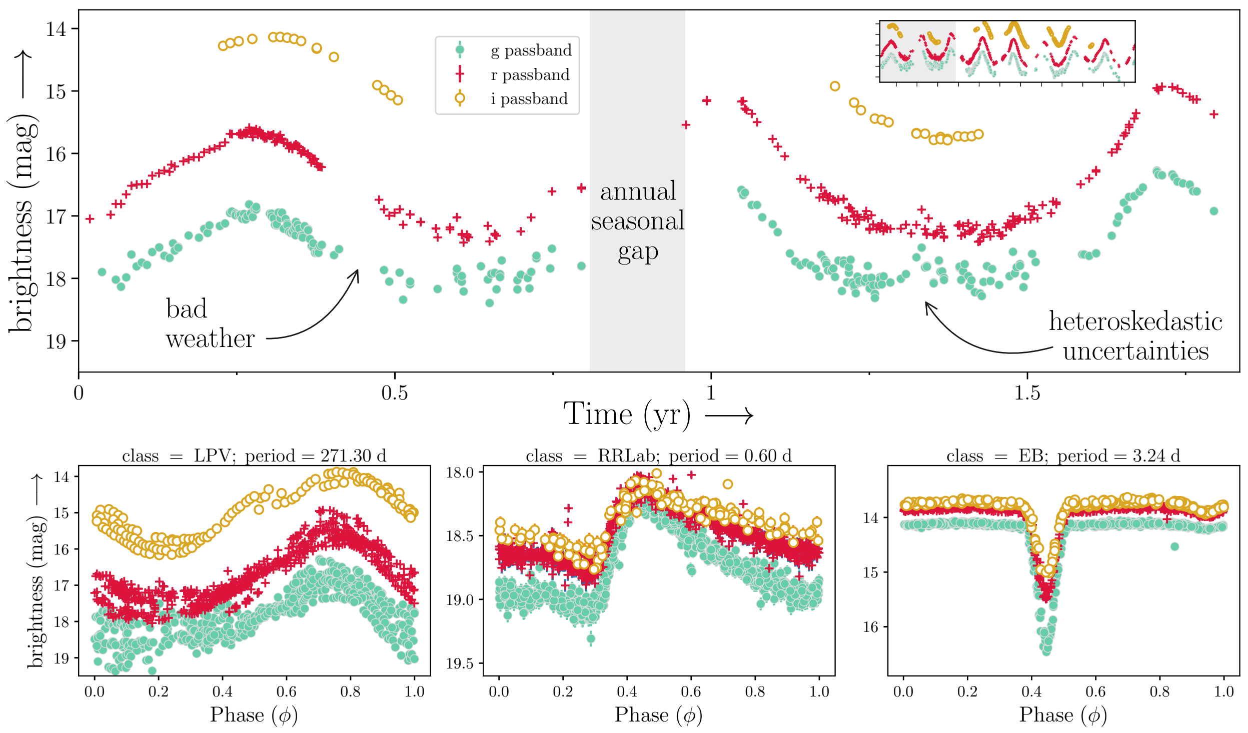


Figure 4.1. Example ZTF light curves illustrating unique characteristics of astronomical time series, including multiple passbands, large observational gaps, and heteroskedastic uncertainties. *Top panel*: Observed light curve of a periodic variable exhibiting typical characteristics of the observations. The inset shows the full ~6.5 yr duration of ZTF observations. *Lower panels*: Phase-folded light curves highlighting the differing periodic patterns in three different classes. Note that most stars have few i passband observations so we exclude these data from our analysis (see text for further details).

embedding models on consistent tasks and datasets. To do so, we: (1) introduce the first benchmark for pre-trained light curve embeddings for periodic variable star classification, (2) curate and standardize a set of ZTF variable stars with reliable labels, (3) evaluate three pre-trained embedding models (including SOTA domain-specific models and two time-series foundation models) on two complementary tasks: zero-shot clustering of embeddings and downstream supervised classification using simple models trained on the embeddings, and, finally, (4) compare learned embeddings against reasonable baselines (e.g., random embeddings, recurrent neural networks, and extracted features designed by domain experts) to quantify the value added by learned representations.

StarEmbed establishes a community resource by publicly releasing these benchmark datasets alongside standard splits, documentation, code, detailed data sheets, and guidelines for responsible use.

4.1.2. Related Work and Models

Major recent investments in time-domain astronomy have generated incredibly large datasets that naturally lend themselves to machine learning methods. The classification of periodic variable stars has been a problem of significant interest for literally centuries, as these sources provide direct insight into the pulsations, rotation, and orbits of distant stars. As such, both pre- (e.g., Debosscher et al., 2007) and post-deep learning models (e.g., Moreno-Cartagena et al., 2025) have been applied to this problem. We summarize the embedding models that we benchmark below. We do not fine-tune the pre-trained models in order to assess their zero-shot generalization capabilities.

4.1.2.1. Supervised Classifiers. The first machine learning models to classify variable stars used manually engineered features combined with classical machine learning models such as support vector machines (Debosscher et al., 2007) or gradient boosted decision trees (Boone, 2019). (Richards et al., 2011) achieved SOTA performance with 52 extracted features (including Fourier coefficients, variability amplitude, skewness, etc.) and a random forest (RF) classifier. Feature extraction varies from study to study, though some have attempted to standardize this step, including the `FATS` (Nun et al., 2015) and `light_curve` (Malanchev et al., 2021) packages. While very effective, hand-crafted features rely heavily on domain knowledge, can be brittle, and they are expensive to compute.

More recent efforts have introduced deep-learning methods to alleviate the need for explicit feature engineering. These models have used a wide range of architectures including recurrent neural networks (RNNs) (Muthukrishna et al., 2019; Becker et al., 2020; Shah et al., 2025), and (vision) transformers (Cabrera-Vives et al., 2024; Moreno-Cartagena et al., 2025). The challenge for these supervised methods is that they require large labeled training sets, and they rarely generalize in a way that allows them to be applied to data from more than one telescope. As a baseline, we build models using handcrafted features and a RNN to compare results with self-supervised pre-trained embedding models.

Handcrafted Features: For this baseline, we extract features using the `FATS` and `light_curve` software packages. Example features include: the best-fit period (using a Lomb-Scargle periodogram Lomb 1976; Scargle 1982), the amplitude of variations, the skewness, the kurtosis, and other metrics. In total, we define 69 features per passband, making the total embedding size 138 (see Appendix **??** for a full feature list with explanations).

RNN: Building on the work of (Muthukrishna et al., 2019), (Shah et al., 2025) achieve SOTA light curve classification performance using an RNN with gated recurrent units (GRU). We adopt a similar architecture with a standard non-hierarchical cross entropy. We note that achieving SOTA performance with RNNs requires, among other things, a balanced training set, which we do not perform here given our goals to benchmark the performance of different methods on the same training/test/validation set. While we report the metrics of our baseline RNN in Table 4.4, we note that better performance can be achieved with an alternate training approach.

4.1.2.2. Astrophysics Embedding models. With a high cost to obtain labels for astronomical sources, there has been a growing interest in using semi-supervised approaches to learn general representations of the data to aid in performing downstream tasks such as classification, anomaly detection, and physical parameter estimation. Recent approaches include variational autoencoders (Villar et al., 2020), sparse autoencoders (Dillmann et al., 2025), and contrastive learning (Zhang et al., 2024), but they are typically limited to a single class (e.g., supernovae). A few foundation models for astronomy attempt to produce useful representations of light curves, such as `FALCO` (Zuo et al., 2025) and `Astromer` (Donoso-Oliva et al., 2023; Donoso-Oliva et al., 2025). In particular, `Astromer` and `Astromer-2` are designed to apply to observations from any observatory (Donoso-Oliva et al., 2023; Donoso-Oliva et al., 2025), and thus, we adopt the `Astromer` models as an astronomy-specific foundation model.

`Astromer` (Donoso-Oliva et al., 2023) is a transformer-based model purpose-built to generate informative embedded representations of astronomical light curves. `Astromer` was pre-trained using self-supervised learning on 1.5 million single-band light curves from the MACHO survey (Alcock et al., 2000). The model's output is a fixed-length embedding (we use the recommended 256-dimensional embedding from the final attention layer). `Astromer`'s training objective was to reconstruct masked portions of the input sequence (a form of masked time series modeling), enabling it to learn general variability patterns. Prior results have shown that `Astromer`'s embeddings can boost classifier performance on small labeled sets. We use the publicly released `Astromer` weights to generate embeddings for each light curve. `Astromer` represents the SOTA domain-specific model and serves as a prime candidate to test whether astronomy-specific pre-training yields discernible benefits relative to more general models.

4.1.2.3. Time series foundation models. Inspired by the success of unsupervised representation learning in natural language processing and vision, recent work has developed self-supervised models for time-series data. In the general time-series domain, methods like `TS2Vec` aims to be a unversal framework to learn representations via contrastive learning, yielding SOTA results on diverse tasks (Yue et al., 2022). `TS2Vec`, in particular, performs hierarchical contrastive learning on augmented time-series and significantly outperforms prior unsupervised methods across 150+ benchmark datasets.

`Moirai` (Woo et al., 2024b) is designed to be a single foundation model that can forecast virtually any time-series, regardless of sampling frequency, dimensionality, or distribution. It pairs a multi-patch-size projection scheme (i.e., handling minute- to year-scale data), an any-variate attention mechanism that scales to arbitrary numbers of variables, and a flexible mixture-distribution output head for calibrated probabilistic forecasts. Trained on LOTSA (Woo et al., 2024b), an open archive of 27 billion observations spanning nine domains, `Moirai`'s Small/Base/Large variants deliver SOTA accuracy in both in-distribution and zero-shot settings, often outperforming models that are fully fine-tuned for a particular dataset. By unifying these capabilities in one openly released model and toolkit, `Moirai` marks a substantial step toward "universal" forecasting systems that need minimal task-specific engineering.

`Chronos` (Ansari et al., 2024a) is another pre-trained time series model showing comparable or even better results than `Moirai`. It treats forecasting as a language-modeling problem: it scales and quantizes real-valued time-series into a fixed vocabulary, then trains off-the-shelf Transformer LMs (T5-style models from 20M to 710M parameters) with an ordinary cross-entropy loss. Augmented by TSMixup and Gaussian-process–generated synthetic data, Chronos is pre-trained on a large collection of public datasets and evaluated on 42 benchmarks. The resulting models deliver strong probabilistic forecasts—significantly ahead of classical and deep-learning baselines on in-domain data and in zero-shot settings, showing that "language of time-series" tokenization alone is enough to build a competitive universal forecaster.

4.1.2.4. Random Embeddings As a Sanity Check Baseline. We generate random valued vectors and treat them as light curve embeddings to establish a performance floor. Random vectors carry no information about the data, so they should yield very poor clustering and classification results. This baseline allows confirmation that any alternative models with superior performance capture useful information in the embeddings. We expect the Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI) near 0 and classification near random guess accuracy per class. The random values are sampled from a Gaussian distribution with the same dimension as the other models.

4.1.3. Dataset

We present a dataset of real, multi-variate time-series observations of periodic variable stars. The flux is presented in magnitudes (an astronomy specitic unit), while the time is recorded as the modified Julian date. The observations are from ZTF, which repeatedly scans all stars visible from the Northern hemisphere every few days. ZTF observes in three different passbands, g, r, and i[1] (see 4.1) corresponding to visible green, visible red, and infrared light, respectively (Dekany et al.,

[1]Most ZTF sources have very few or no observations in i band and we therefore exclude it from our analysis.

2020). We us observations from ZTF data release 23 (DR23) which spans a duration of ∼6.5 yr. In total, ZTF DR23 contains billions of light curves.

Careful selection of light curves from the ZTF DR23 data set is necessary as the periodic variable stars which we hope to exploit for astrophysical inferences are rare. There are many periodic variable star catalogs with classifications in the literature, however, most rely on low-capacity ML models to produce their labels. We opt to avoid such catalogs as their labels are subject to noise from the imperfect classifier and the decision boundaries between classes are likely simplified representations learned by the classifier.

For training, we adopt the Catalina Surveys Periodic Variable Star Catalog (CSPVS; Drake et al., 2014a), a human-labeled catalog of periodic variables discovered by the Catalina Real-Time Transient Survey (Drake et al., 2009). We extract ZTF light curves for each star in the CSPVS catalog. Stars that lack a ZTF light curve are omitted; we also remove (i) bad observations indicated by the image quality flag in ZTF DR23; (ii) light curves with less than 32 total observations; (iii) light curves that have no observations in both the g and r ZTF passbands; and (iv) light curves from classes with fewer than 400 total examples. The resulting dataset contains ∼40,000 ZTF light curves of hand-labeled periodic variable stars across seven classes.

Nature naturally produces an imbalance between the number of periodic variables in different classes, and this is further exacerbated by each class having a different detection efficiency (e.g., LPVs have much larger amplitude variations than any other class making them easier to identify). To ensure that each split gets a representative number of examples from each class, we sample each class into the train, validation, and test splits independently in a 7:1:2 ratio. Table 4.1 shows the counts of examples per class in each split. We release our train-validation-test dataset splits and the generated embeddings as a public dataset on HuggingFace (`https://huggingface.co/datasets/wormyu/StarEmbed`).

4.1.3.1. ZTF Dataset in Context: Relevance to Upcoming Observatories. Variable star science has an expansive scope that extends beyond ZTF and periodic variables, the `StarEmbed` benchmark

Table 4.1. Number of periodic variable stars in our dataset across each class and in each split

Class	EW	EA	RRab	RRc	RRd	RS CVn	LPV
Train	18998	2889	1386	3233	298	942	255
Validation	2690	410	194	463	42	134	35
Test	5387	818	397	926	83	276	70
Total after cuts	27075	4117	1977	4622	423	1352	360

is designed to allow for the addition of new datasets and metrics in future expansions. This flexibility is crucial to the long-term health of this benchmark as numerous new astronomical time-domain surveys will begin in the coming years, including: LSST, the Nancy Gracy Roman Space Telescope (Akeson et al., 2019), and the Ultraviolet Explorer (Kulkarni et al., 2021). Each of these surveys has unique observations that will affect the resulting embeddings and downstream task performance is not yet known. As the largest astronomical time-domain experiment to date, ZTF is an apt choice for building a preparatory benchmark dataset that has already bee used to explore emerging areas like multi-modality (e.g., Duev and van der Walt, 2021; Carrasco-Davis et al., 2021; Gagliano et al., 2023; Rehemtulla et al., 2024) and transformers (e.g., Allam et al., 2023; Zhang et al., 2024).

All current and future datasets associated with the `StarEmbed` benchmark are public or will be made public. The ZTF DR23 can be accessed through the Caltech Infrared Processing and Analysis Center[2]; the CSPVS catalog is available via VizieR[3] (Drake et al., 2014b); and our selection of ZTF light curves for CSPVS stars is available on Hugging Face[4]. Our publicly available dataset includes the necessary metadata and a permissive license for reuse. No personal or sensitive information is present in these datasets (they consist of astronomical observations).

4.1.4. Evaluation Methodology

We assess the quality of embeddings for both: (1) unsupervised clustering, and (2) supervised classification using the embeddings as features. Together, these give a comprehensive view of both

[2]https://irsa.ipac.caltech.edu/Missions/ztf.html
[3]https://cdsarc.cds.unistra.fr/viz-bin/cat/J/ApJS/213/9
[4]https://huggingface.co/datasets/wormyu/StarEmbed

the intrinsic structure captured by the embeddings and their usefulness for downstream tasks. Below we detail the experimental settings, including data splits, training procedure for classifiers, and metrics used to evaluate the embeddings, and we release our code to reproduce our benchmark experiments (`https://tinyurl.com/jwew993p`).

Hyperparameter Tuning: To fairly compare the different embeddings, we conduct a hyperparameter search on each model when training the downstream MLP classifier. We search over batch size $B \in \{128, 256, 512, 1024\}$, learning rate $lr \in \{0.01, 0.001, 0.0001\}$ and dropout rate $\in \{0.0, 0.1\}$. Every hyperparameter triple runs once on 4 NVIDIA H100 GPUs. The training process is at most 50 epochs, and stops early if the validation loss fails to improve for 3 epochs. In practice this training takes less than 30 epochs for all models before the early stopping is triggered.

Embedding Hyperparameters: To maintain consistency throughout the comparisons in the benchmark, we use an identical embedding size across different models if available (e.g., there exists a pre-trained version of the model using the same embedding size). Since `ASTROMER` uses an embedding size of 256, we use the same embedding size for other models, specifically, `Chronos-Bolt-Tiny`, `Chronos-Tiny`, and Random Embeddings. The smallest pre-trained version of `MOIRAI`, `Moirai-small` uses an embedding size of 384, which we adpot here.

4.1.4.1. Clustering-Based Evaluation (Zero-Shot). In this setting, we treat the embeddings of each model as points in a feature space and apply clustering algorithms to see if natural groupings correspond to known variable star classes. Specifically, we use k-means clustering with k set to the number of true classes in the dataset (for a given dataset's evaluation, e.g., $k = 7$ for Catalina, which has 7 classes in our subset). We run k-means on the entire set of embeddings for that dataset (usually using the test split or the full dataset if not doing classification). To mitigate randomness, we repeat k-means with 10 different initializations and choose the clustering with the lowest intra-cluster variance (the standard approach). We then evaluate clustering quality using the following metrics:

Normalized Mutual Information (NMI): NMI measures the mutual information between the cluster assignments and the true class labels, normalized to range [0,1]. An NMI of 1 indicates perfect correlation (clusters exactly match classes), while NMI near 0 indicates no better than random assignment. NMI is invariant to label permutations, which is suitable since cluster labels are arbitrary.

Adjusted Rand Index (ARI): ARI evaluates pairwise clustering agreements, adjusted for chance. It considers how often pairs of light curves are in the same cluster vs. the same true class. ARI = 1 for a perfect clustering, and ARI $\approx$ 0 for random clustering (negative values are worse than chance).

By comparing NMI/ARI across embeddings, we can tell which has more separable class structure without any supervised training. A high NMI/ARI suggests that the embedding has essentially learned to differentiate variable types on its own (perhaps by capturing differences in light curve morphology). Models that yield poor clustering scores, close to zero, indicate that the embedding might not encode class information (it could still encode other factors like light curve amplitude or survey-specific features). We expect domain-specific models like `ASTROMER` to potentially excel here if their unsupervised objective correlates with class patterns. Baseline features have been used for clustering in some studies (e.g., to group variable stars by type), so they might show reasonable performance. Random embeddings should give NMI and ARI close to 0 as a reference point.

We also visualize the embedding spaces using dimensionality reduction, with `t-SNE`, to provide an intuitive picture of how well the clusters correspond to individual classes. For example, a 2D `t-SNE` plot for `ASTROMER` embeddings might show distinct clusters for RR Lyrae vs. Cepheids, etc. These visualizations will be included in the appendix and help qualitatively verify the clustering metrics.

4.1.4.2. Downstream Classification (Linear Evaluation). To directly measure how useful the embeddings are for classifying variable stars, we perform a "linear evaluation" experiment: we train a simple classifier on the embeddings to predict the variable star class labels, while holding the embeddings fixed. This simulates a scenario where one uses the pretrained embeddings as feature inputs for a classification task but does not fine-tune the embedding model (hence "zero-shot" in

Table 4.2. K-means clustering results with metrics including normalized mutual information (NMI) and adjusted rand index (ARI). Each experiment repeats three times with random seed= $42, 100, 200$ respectively and the numbers show the mean (std). The general-purpose time series pretrained models demonstrate the best general performance.

Methods	NMI	ARI	F1
`ASTROMER`	0.1402(0.0374)	**0.1762(0.0269)**	0.2642(0.0457)
`Moirai-small`	0.1772(0.0080)	0.0310(0.0221)	0.2200(0.0237)
`Chronos-tiny`	<u>0.2250(0.0004)</u>	<u>0.1627(0.0010)</u>	<u>0.3043(0.0021)</u>
`Chronos-Bolt-tiny`	**0.2256(0.0007)**	0.1358(0.0006)	**0.3164(0.0009)**
Random Embeddings	0.0015 (0.0001)	0.0003 (0.0008)	0.1049(0.0013)
Hand-crafted Features	0.0720(0.0160)	0.0370(0.0430)	0.1662(0.0093)

terms of the embedding model). The classification models we use are intentionally lightweight to ensure that any performance differences come from the quality of the embeddings rather than a powerful classifier compensating for a weak embedding. We use a multilayer perceptron (MLP) with three hidden layers of size 1024, 512 and 256 respectively, and an output layer for class predictions. Detailed information on the classifiers and their hyperparameters for each embedding model are in the Appendix.

Classification Metrics: We report standard multi-class classification metrics, including: *Accuracy*: the overall fraction of correctly classified examples. This gives a high-level sense of performance that nevertheless can be misleading for our imbalanced label set. *Macro-averaged F1 Score*: The F1 score (harmonic mean of precision and recall) computed per class and then averaged equally across all classes. Macro-F1 is sensitive to performance on minority classes, treating each class equally. This is important because some classes (e.g., Cepheids) have many fewer labels than others (e.g., eclipsing binaries). A model that only does well on majority classes will have a low macro-F1 even if the accuracy is high. *Precision/Recall*: We also include the overall precision and recall to provide a more comprehensive evaluation on the classification performance of the the MLP trained on different embeddings.

4.1.5. Benchmark Results

Table 4.3. Hierarchical clustering (Ward's Linkage) results with metrics including normalized mutual information (NMI) and adjusted rand index (ARI). Since it is a deterministic method, the results from three random seeds are identical. The observation is consistent with K-means clustering's results that general-purpose time series pretrained models have the best general performance.

Methods	NMI	ARI	F1
`ASTROMER`	0.1383	**0.1449**	<u>0.2816</u>
`Moirai-small`	0.1361	-0.0049	0.1656
`Chronos-tiny`	<u>0.2121</u>	0.0999	0.2215
`Chronos-Bolt-tiny`	**0.2275**	<u>0.1399</u>	**0.3317**
Random Embeddings	0.0012	0.0001	0.1119
Hand-crafted Features	0.0603	-0.0001	0.1600

Table 4.4. Classification results with metrics including accuracy, precision, recall and F1 score. Each metric is averaged across different classes for each model. Each experiment repeats three times with random seed=42,100,200 respectively and the numbers show the mean (std)

Methods	Accuracy	Precision	Recall	F1
Astromer	0.4586 (0.1621)	0.1434 (0.0075)	0.1621 (0.0028)	0.1339 (0.0156)
Baseline RNN	0.6900 (0.0383)	0.5000 (0.0002)	0.6600 (0.0089)	0.5500 (0.0057)
Moirai-small	0.711 (0.0446)	0.5652 (0.0185)	0.6939 (0.0140)	0.5921 (0.0134)
Chronos-tiny	**0.7745 (0.0168)**	**0.5943 (0.0014)**	**0.7603 (0.0055)**	**0.6449 (0.0080)**
Chronos-Bolt-tiny	<u>0.7397 (0.0129)</u>	<u>0.5687 (0.0095)</u>	<u>0.7155 (0.0037)</u>	<u>0.6123 (0.0089)</u>
Random Embeddings	0.3383 (0.1572)	0.1367 (0.0240)	0.1457 (0.0022)	0.0956 (0.0466)
Handcrafted Features	0.6794 (0.0148)	0.5221 (0.0208)	0.3701 (0.0347)	0.3802 (0.0379)

4.1.5.1. Clustering. Tables 4.2 and 4.3 show that (1) the earned embeddings from pre-trained models produce better clustering performance compared to hand-crafted features, and (2) domain-specific pretrained models do not produce significantly better embeddings compared to general-purpose time series pretrained models.

4.1.5.2. Classification Results. Classification performance. Table **??** and Figure **??** show that (1) general-purpose time series pret-rained models' embeddings produce the best MLP classifiers compared to domain-specific pretrained models' embeddings. (2) Compared to the clustering results from table 4.2 and table 4.3, almost all models' embeddings gain a better classification metric after further training a MLP classifier on the embeddings. (3) The fully-supervised SOTA classifier, RNN,

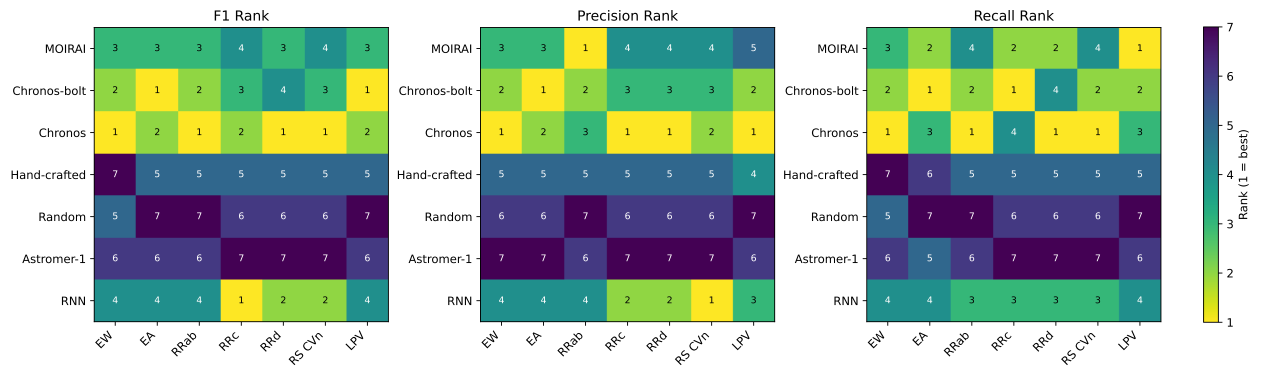


Figure 4.2. Ranking of classification performance on each class on each metric. The results show that time series pretrained models' embeddings achieve top ranking in most of the classes, demonstrating a comparable or better performance compared to hand-crafted features.

is the only domain-specific model that has comparable classification performance to a simple MLP classifier trained on the embeddings from general-purpose time series pretrained models. However, it comes with a significantly higher cost. The RNN takes ∼1,000 epochs to train and substantial effort to try different strategies to counter class imbalance while it takes no more than 15 epochs to train a MLP classifier on all time series pre-trained models' embeddings. Furthermore, tuning is not necessary. It already has very competitive classification results even we do not apply any strategy to counter the data imbalance issue.

Classification robustness. During the experiments, we also notice that data imbalance issues severely affect traditional fully-supervised classifier such as the strong baseline, an RNN, model. For instance, if without applying any method to counter the data imbalance issue, the RNN model would converge quickly in the training (the first several epochs) and always predict the same class in the test set, as shown by fig. 4.3, presenting a disastrous prediction performance (near zero recall) on all other classes. It shows the opposite for the results from general-purpose time series pretrained models' embeddings. They produce decent results even without any strategy conducted to counter

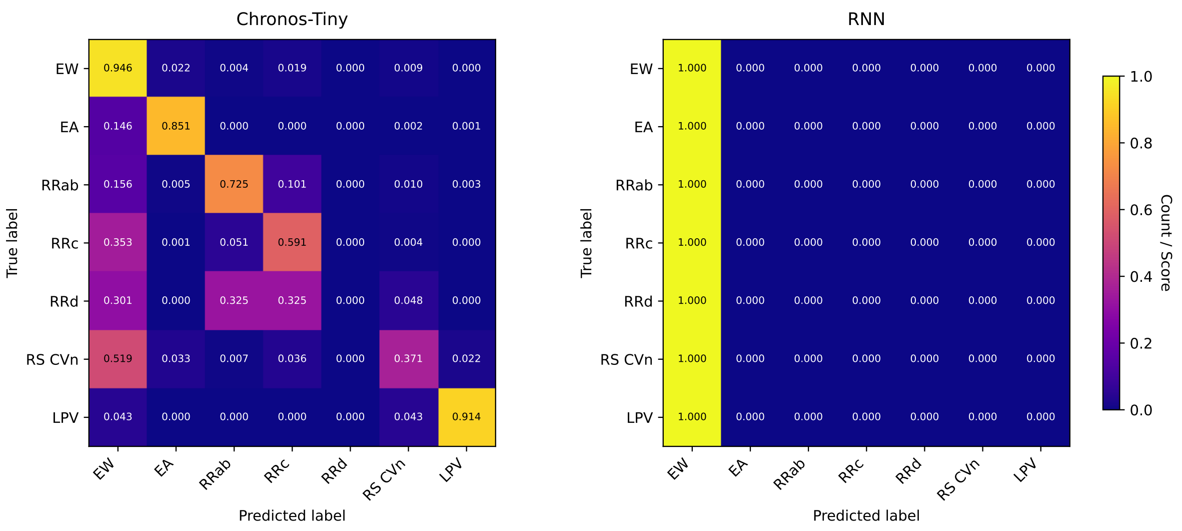


Figure 4.3. Confusion matrices on the test set for the Chronos-Tiny and RNN model without countering the data imbalance issue. RNN has a significant collapse while Chronos-Tiny still gives decent classification performance.

the data imbalance issue (more than 50% recall for all classes except one class), demonstrating a much better robustness.

4.1.6. Discussion and Conclusion

This study introduces `StarEmbed`, the first public, large-scale benchmark expressly designed to evaluate light-curve representations from different types of models. By harmonizing expert-vetted CSDR1 labels with ∼40,000 multi-band ZTF DR23 light curves, we deliver a rigorously curated, seven-class dataset together with standardized splits and Croissant-formatted metadata, lowering the barrier for reproducible research on astronomical time-series embeddings.

Using this benchmark, we perform a comprehensive comparison of (i) domain-specific embeddings (`ASTROMER`), (ii) embeddings from three state-of-the-art, general-purpose time-series foundation models (`Moirai` and `Chronos` families), and (iii) three strong baselines including hand-crafted features and a fully-supervised RNN. Two main findings emerge: **(1) Zero-shot power of foundation embeddings**: without any in-domain fine-tuning, general-purpose models consistently surpass both engineered hand-crafted features and the domain-specific `ASTROMER` embeddings in clustering and classification, while remaining remarkably robust to severe class imbalance. **(2) Cost-effectiveness and scalability**: a small MLP trained for less than 15 epochs on frozen foundation embeddings equals or exceeds the performance of a carefully tuned, imbalance-mitigated RNN that required orders of magnitude more compute and experimentation time ($\sim$1000 epochs).

Taken together, these results advocate a shift from bespoke, fully-supervised pipelines toward off-the-shelf foundation representations plus lightweight task heads for variable-star analysis—and, by extension, for other forthcoming petascale time-series challenges. By releasing all data, code, and model wrappers, we hope `StarEmbed` will become a focal point for benchmarking advances in self-supervised and foundation models, spur work on even broader time-series corpora, and catalyze progress on downstream tasks such as anomaly detection, period estimation, and real-time alert triage in next-generation sky surveys. The importance of this benchmark cannot be understated, there are no direct comparisons of performance for machine learning light curve classification in the astronomy literature because every study uses different observations and typically considers a different set of classes. `StarEmbed` provides the first well-curated dataset to address this shortcoming and will provide a foundation for new work on flagship experiments like LSST.

Limitations We conduct comprehensive experiments to show the hierarchy of quality of embeddings from different types of models and demonstrate the potential of a general-purpose pretrained embedding model for the astrophysics community. However, we leave a more thorough analysis on the reasons behind these findings to future work.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rachel Akeson, Lee Armus, Etienne Bachelet, Vanessa Bailey, Lisa Bartusek, Andrea Bellini, Dominic Benford, David Bennett, Aparna Bhattacharya, Ralph Bohlin, et al. The wide field infrared survey telescope: 100 hubbles for the 2020s. *arXiv preprint arXiv:1902.05569*, 2019.

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

Taha Aksu, Gerald Woo, Juncheng Liu, Xu Liu, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Gift-eval: A benchmark for general time series forecasting model evaluation. *arXiv preprint arXiv:2410.10393*, 2024.

Charles Alcock, RA Allsman, David R Alves, TS Axelrod, Andrew C Becker, DP Bennett, Kem H Cook, N Dalal, Andrew John Drake, KC Freeman, et al. The macho project: Microlensing results from 5.7 years of largemagellanic cloud observations. *The Astrophysical Journal*, 542(1):281, 2000.

Hussain Alibrahim and Simone A Ludwig. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1551–1559. IEEE, 2021.

Tarek Allam, Jr., Julien Peloton, and Jason D. McEwen. The Tiny Time-series Transformer: Low-latency High-throughput Classification of Astronomical Transients using Deep Model Compression. *arXiv e-prints*, art. arXiv:2303.08951, March 2023. doi: 10.48550/arXiv.2303.08951.

Donald WK Andrews. Cross-section regression with common shocks. *Econometrica*, 73(5): 1551–1585, 2005. URL https://www.jstor.org/stable/3598883.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. 2024a.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series, 2024b. URL https://arxiv.org/abs/2403.07815.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

Marco Avellaneda and Sasha Stoikov. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 2008.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

I. Becker, K. Pichara, M. Catelan, P. Protopapas, C. Aguirre, and F. Nikzat. Scalable end-to-end recurrent neural network for variable star classification. *mnras*, 493(2):2981–2995, April 2020. doi: 10.1093/mnras/staa350.

Theodore C Belding. The distributed genetic algorithm revisited. *arXiv preprint adap-org/9504007*, 1995.

Marco Bellagente, Jonathan Tow, Dakota Mahan, Duy Phung, Maksym Zhuravinskyi, Reshinth Adithyan, James Baicoianu, Ben Brooks, Nathan Cooper, Ashish Datta, Meng Lee, Emad Mostaque, Michael Pieler, Nikhil Pinnaparju, Paulo Rocha, Harry Saini, Hannah Teufel, Niccolo Zanichelli, and Carlos Riquelme. Stable LM 2 1.6B technical report. *arXiv preprint arXiv:2402.17834*, 2024.

Eric Bellm. The zwicky transient facility. In *The Third Hot-wiring the Transient Universe Workshop*, volume 27, 2014.

Eric C Bellm, Shrinivas R Kulkarni, Matthew J Graham, Richard Dekany, Roger M Smith, Reed Riddle, Frank J Masci, George Helou, Thomas A Prince, Scott M Adams, et al. The zwicky transient facility: system overview, performance, and first results. *Publications of the Astronomical Society of the Pacific*, 131(995):018002, 2018.

Eric C. Bellm, Shrinivas R. Kulkarni, Matthew J. Graham, Richard Dekany, Roger M. Smith, Reed Riddle, Frank J. Masci, George Helou, Thomas A. Prince, Scott M. Adams, C. Barbarino, Tom Barlow, James Bauer, Ron Beck, Justin Belicki, Rahul Biswas, Nadejda Blagorodnova, Dennis Bodewits, Bryce Bolin, Valery Brinnel, Tim Brooke, Brian Bue, Mattia Bulla, Rick Burruss, S. Bradley Cenko, Chan-Kao Chang, Andrew Connolly, Michael Coughlin, John Cromer, Virginia Cunningham, Kishalay De, Alex Delacroix, Vandana Desai, Dmitry A. Duev, Gwendolyn Eadie, Tony L. Farnham, Michael Feeney, Ulrich Feindt, David Flynn, Anna Franckowiak, S. Frederick, C. Fremling, Avishay Gal-Yam, Suvi Gezari, Matteo Giomi, Daniel A. Goldstein, V. Zach Golkhou, Ariel Goobar, Steven Groom, Eugean Hacopians, David Hale, John Henning, Anna Y. Q. Ho, David Hover, Justin Howell, Tiara Hung, Daniela Huppenkothen, David Imel, Wing-Huen Ip, Željko Ivezić, Edward Jackson, Lynne Jones, Mario Juric, Mansi M. Kasliwal, S. Kaspi, Stephen Kaye, Michael S. P. Kelley, Marek Kowalski, Emily Kramer, Thomas Kupfer, Walter Landry, Russ R. Laher, Chien-De Lee, Hsing Wen Lin, Zhong-Yi Lin, Ragnhild Lunnan, Matteo Giomi, Ashish Mahabal, Peter Mao, Adam A. Miller, Serge Monkewitz, Patrick Murphy, Chow-Choong Ngeow, Jakob Nordin, Peter Nugent, Eran Ofek, Maria T. Patterson, Bryan Penprase, Michael Porter, Ludwig Rauch, Umaa Rebbapragada, Dan Reiley, Mickael Rigault, Hector Rodriguez, Jan van Roestel, Ben Rusholme, Jakob van Santen, S. Schulze, David L. Shupe, Leo P. Singer, Maayane T. Soumagnac, Robert Stein, Jason Surace, Jesper Sollerman, Paula Szkody, F. Taddia, Scott Terek, Angela Van Sistine, Sjoert van Velzen, W. Thomas Vestrand, Richard Walters, Charlotte Ward, Quan-Zhi Ye, Po-Chieh Yu, Lin Yan, and Jeffry Zolkower. The Zwicky Transient Facility: System Overview, Performance, and First Results. *pasp*, 131(995):018002, January 2019. doi: 10.1088/1538-3873/aaecbe.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Dimitri P Bertsekas, W Hager, and O Mangasarian. Nonlinear programming. athena scientific belmont. *Massachusets, USA*, 1999. URL `http://athenasc.com/nlpsol1.pdf`.

Lukas Biewald et al. Experiment tracking with weights and biases. *Software available from wandb. com*, 2:233, 2020. URL `https://www.kaggle.com/code/ayuraj/experiment-tracking-with-weights-and-biases`.

BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa

Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ajibade, Bharat Saxena, Carlos Muñoz Ferrandis, Daniel McDuff, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang,

Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Raghu Bollapragada and Stefan M. Wild. Adaptive sampling quasi-Newton methods for zeroth-order stochastic optimization. *Mathematical Programming Computation*, 15(2):327–364, 2023.

Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, 2018.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Philip Bond and James Dow. Failing to forecast rare events. *Journal of Financial Economics*, 142(3):1001–1016, 2021. URL https://www.sciencedirect.com/science/article/abs/pii/S0304405X21002981.

Kyle Boone. Avocado: Photometric Classification of Astronomical Transients with Gaussian Process Augmentation. *aj*, 158(6):257, December 2019. doi: 10.3847/1538-3881/ab5182.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Johann S Brauchart, Alexander B Reznikov, Edward B Saff, Ian H Sloan, Yu Guang Wang, and Robert S Womersley. Random point sets on the sphere—hole radii, covering, and separation. *Experimental Mathematics*, 27(1):62–81, 2018. URL https://www.tandfonline.com/doi/abs/10.1080/10586458.2016.1226209.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Matthieu Bussiere and Marcel Fratzscher. Towards a new early warning system of financial crises. *journal of International Money and Finance*, 25(6):953–973, 2006. URL https://www.sciencedirect.com/science/article/abs/pii/S0261560606000532.

Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical Programming*, 134(1):127–155, 2012.

G Cabrera-Vives, D Moreno-Cartagena, N Astorga, I Reyes-Jainaga, F Förster, P Huijse, J Arredondo, AM Muñoz Arancibia, A Bayo, M Catelan, et al. Atat: Astronomical transformer for time series and tabular data. *Astronomy & Astrophysics*, 689:A289, 2024.

Erick Cantu-Paz and David E Goldberg. Efficient parallel genetic algorithms: theory and practice. *Computer methods in applied mechanics and engineering*, 186(2-4):221–238, 2000.

Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. Tempo: Prompt-based generative pre-trained transformer for time series forecasting, 2024. URL https://arxiv.org/abs/2310.04948.

Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

Rodrigo Carrasco-Davis, Esteban Reyes, Camilo Valenzuela, Francisco Förster, Pablo A Estévez, Giuliano Pignata, Franz E Bauer, Ignacio Reyes, Paula Sánchez-Sáez, Guillermo Cabrera-Vives, et al. Alert classification for the alerce broker system: The real-time stamp classifier. *The Astronomical Journal*, 162(6):231, 2021.

M. Catelan and H. A. Smith. *Pulsating Stars*. 2015.

Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. Nhits: neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6989–6997, 2023.

Soon Huat Chan, Kenneth A Kim, and S Ghon Rhee. Price limit performance: evidence from transactions data and the limit order book. *Journal of Empirical Finance*, 12(2):269–290, 2005.

SA Chen, CL Li, N Yoder, SO Arik, and T Pfister. Tsmixer: An all-mlp architecture for time series forecasting. arxiv 2023. *arXiv preprint arXiv:2303.06053*, 2023a.

Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O Arik, and Tomas Pfister. Tsmixer: An all-mlp architecture for time series forecasting. *arXiv preprint arXiv:2303.06053*, 2023b.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

Gisella Clementini, Raffaele Gratton, Angela Bragaglia, Eugenio Carretta, Luca Di Fabrizio, and Marcella Maio. Distance to the large magellanic cloud: The rr lyraestars. *The Astronomical Journal*, 125(3):1309, 2003.

Common Crawl. Common crawl. https://commoncrawl.org/, 2024. Accessed: 2025-07-11.

Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

John M Danskin. *The theory of max-min and its application to weapons allocation problems*, volume 5. Springer Science & Business Media, 2012. URL https://link.springer.com/book/10.1007/978-3-642-46092-0.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024.

Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Big batch SGD: Automated inference using adaptive batch sizes. *arXiv preprint arXiv:1610.05792*, 2016.

Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Automated inference with adaptive batches. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Jonas Debosscher, LM Sarro, Conny Aerts, J Cuypers, Bart Vandenbussche, R Garrido, and E Solano. Automated supervised classification of variable stars-i. methodology. *Astronomy & astrophysics*, 475(3):1159–1183, 2007.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. DeepSeek-V2: A strong, economical, and efficient Mixture-of-Experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

Richard Dekany, Roger M. Smith, Reed Riddle, Michael Feeney, Michael Porter, David Hale, Jeffry Zolkower, Justin Belicki, Stephen Kaye, John Henning, Richard Walters, John Cromer, Alex Delacroix, Hector Rodriguez, Daniel J. Reiley, Peter Mao, David Hover, Patrick Murphy, Rick Burruss, John Baker, Marek Kowalski, Klaus Reif, Phillip Mueller, Eric Bellm, Matthew Graham, and Shrinivas R. Kulkarni. The Zwicky Transient Facility: Observing System. *pasp*, 132(1009): 038001, March 2020. doi: 10.1088/1538-3873/ab4ca2.

Mete Demircigil, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168:288–299, 2017.

Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Steven Dillmann, Juan Rafael Martínez-Galarza, Roberto Soria, Rosanne Di Stefano, and Vinay L. Kashyap. Representation learning for time-domain high-energy astrophysics: Discovery of extragalactic fast X-ray transient XRT 200515. *mnras*, 537(2):931–955, February 2025. doi: 10.1093/mnras/stae2808.

C Donoso-Oliva, I Becker, Pavlos Protopapas, Guillermo Cabrera-Vives, M Vishnu, and Harsh Vardhan. Astromer-a transformer-based embedding for the representation of light curves. *Astronomy & Astrophysics*, 670:A54, 2023.

Cristobal Donoso-Oliva, Ignacio Becker, Pavlos Protopapas, Guillermo Cabrera-Vives, Martina Cádiz-Leyton, and Daniel Moreno-Cartagena. Astromer 2. *arXiv e-prints*, art. arXiv:2502.02717, February 2025. doi: 10.48550/arXiv.2502.02717.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

A. J. Drake, M. J. Graham, S. G. Djorgovski, M. Catelan, A. A. Mahabal, G. Torrealba, D. García-Álvarez, C. Donalek, J. L. Prieto, R. Williams, S. Larson, E. Christen sen, V. Belokurov, S. E. Koposov, E. Beshore, A. Boattini, A. Gibbs, R. Hill, R. Kowalski, J. Johnson, and F. Shelly. The Catalina Surveys Periodic Variable Star Catalog. *apjs*, 213(1):9, July 2014a. doi: 10.1088/0067-0049/213/1/9.

A. J. Drake, M. J. Graham, S. G. Djorgovski, M. Catelan, A. A. Mahabal, G. Torrealba, D. Garcia-Alvarez, C. Donalek, J. L. Prieto, R. Williams, S. Larson, E. Christensen, V. Belokurov, S. E. Koposov, E. Beshore, A. Boattini, A. Gibbs, R. Hill, R. Kowalski, J. Johnson, and F. Shelly. VizieR Online Data Catalog: Catalina Surveys periodic variable stars (Drake+, 2014). art. J/ApJS/213/9, September 2014b. doi: 10.26093/cds/vizier.22130009.

AJ Drake, SG Djorgovski, A Mahabal, E Beshore, S Larson, MJ Graham, R Williams, E Christensen, M Catelan, A Boattini, et al. First results from the catalina real-time transient survey. *The Astrophysical Journal*, 696(1):870, 2009.

Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, Renjun Xu, and Chongjun Wang. Adarnn: Adaptive learning and forecasting of time series. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 402–411, 2021.

Dmitry A. Duev and Stéfan J. van der Walt. Phenomenological classification of the Zwicky Transient Facility astronomical event alerts. *arXiv e-prints*, art. arXiv:2111.12142, November 2021. doi: 10.48550/arXiv.2111.12142.

Zoltan Eisler, Jean-Philippe Bouchaud, and Julien Kockelkoren. The price impact of order book events: market orders, limit orders and cancellations. *Quantitative Finance*, 12(9):1395–1419, 2012.

Vijay Ekambaram, Arindam Jati, Pankaj Dayama, Sumanta Mukherjee, Nam H. Nguyen, Wesley M. Gifford, Chandra Reddy, and Jayant Kalagnanam. Tiny time mixers (ttms): Fast pre-trained models for enhanced zero/few-shot forecasting of multivariate time series, 2024. URL `https://arxiv.org/abs/2401.03955`.

Alaaeldin El-Nouby, Michal Klein, Shuangfei Zhai, Miguel Angel Bautista, Alexander Toshev, Vaishaal Shankar, Joshua M. Susskind, and Armand Joulin. Scalable pre-training of large autoregressive image models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

Enas Elgeldawi, Awny Sayed, Ahmed R Galal, and Alaa M Zaki. Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis. In *Informatics*, volume 8, page 79. Multidisciplinary Digital Publishing Institute, 2021.

Robert Evans. Apache storm, a hands on tutorial. In *2015 IEEE International Conference on Cloud Engineering*, pages 2–2. IEEE, 2015.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. URL `https://github.com/Lightning-AI/lightning`. Version 2.0.8.

Eugene F Fama and Kenneth R French. Comparing cross-section and time-series factor models. *The Review of Financial Studies*, 33(5):1891–1926, 2020. URL `https://academic.oup.com/rfs/article/33/5/1891/5555879`.

Wei Fan, Pengyang Wang, Dongkun Wang, Dongjie Wang, Yuanchun Zhou, and Yanjie Fu. Dish-ts: A general paradigm for alleviating distribution shift in time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7522–7529, 2023.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019. URL `https://link.springer.com/article/10.1007/s10618-019-00619-1`.

MW Feast and AR Walker. Cepheids as distance indicators. *IN: Annual review of astronomy and astrophysics. Volume 25 (A88-13240 03-90). Palo Alto, CA, Annual Reviews, Inc., 1987, p. 345-375.*, 25:345–375, 1987.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European journal of operational research*, 270(2):654–669, 2018.

Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.

Matthew French. Brent crude oil prices averaged $19 per barrel less in 2023 than 2022. https://www.eia.gov/todayinenergy/detail.php?id=61142, 2024. Accessed: 08.09.2024.

Michael P. Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of Machine Learning and Systems (MLSys)*, 2018.

Andreas Fürst, Elisabeth Rumetshofer, Johannes Lehner, Viet T Tran, Fei Tang, Hubert Ramsauer, David Kreil, Michael Kopp, Günter Klambauer, Angela Bitto, et al. Cloob: Modern hopfield networks with infoloob outperform clip. *Advances in neural information processing systems*, 35: 20450–20468, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8078e76f913e31b8467e85b4c0f0d22b-Abstract-Conference.html.

Alexander Gagliano, Gabriella Contardo, Daniel Foreman-Mackey, Alex I Malz, and Patrick D Aleo. First impressions: early-time classification of supernovae using host-galaxy information and shallow learning. *The Astrophysical Journal*, 954(1):6, 2023.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

Shanghua Gao, Teddy Koker, Owen Queen, Thomas Hartvigsen, Theodoros Tsiligkaridis, and Marinka Zitnik. Units: A unified multi-task time series model, 2024. URL https://arxiv.org/abs/2403.00131.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia

Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Google DeepMind Gemma Team. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Xinyang Geng and Hao Liu. OpenLLaMA: An open reproduction of LLaMA, May 2023. URL `https://github.com/openlm-research/open_llama`.

K Genovali, B Lemasle, Guillaume Bono, M Romaniello, M Fabrizio, I Ferraro, G Iannicola, CD Laney, M Nonino, M Bergemann, et al. On the fine structure of the cepheid metallicity gradient in the galactic thin disk. *Astronomy & Astrophysics*, 566:A37, 2014.

Francisco Girela-Lopez, Eduardo Ros, and Javier Diaz. Precise network time monitoring: Picosecond-level packet timestamping for fintech networks. *IEEE Access*, 9:40274–40285, 2021.

Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training

ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Matthew J Graham, SR Kulkarni, Eric C Bellm, Scott M Adams, Cristina Barbarino, Nadejda Blagorodnova, Dennis Bodewits, Bryce Bolin, Patrick R Brady, S Bradley Cenko, et al. The zwicky transient facility: science objectives. *Publications of the Astronomical Society of the Pacific*, 131(1001):078001, 2019.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. URL https://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471–476, 2016. URL https://www.nature.com/articles/nature20101.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

Yanhong Guo and Xinxin Chen. Forecasting the mid-price movements with high-frequency lob: A dual-stage temporal attention-based deep learning architecture. *Arabian Journal for Science and Engineering*, 48(8):9597–9618, 2022.

Divij Gupta, Anubhav Bhatti, Suraj Parmar, Chen Dan, Yuwei Liu, Bingjie Shen, and San Lee. Low-rank adaptation of time series foundational models for out-of-domain modality forecasting, 2024a. URL https://arxiv.org/abs/2405.10216.

Divij Gupta, Anubhav Bhatti, and Surajsinh Parmar. Beyond loRA: Exploring efficient fine-tuning techniques for time series foundational models. In *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024b. URL https://openreview.net/forum?id=YZJ8Re0gQv.

Lu Han, Han-Jia Ye, and De-Chuan Zhan. The capacity and robustness trade-off: Revisiting the channel independent strategy for multivariate time series forecasting. *arXiv preprint arXiv:2304.05206*, 2023.

Lawrence E Harris and Venkatesh Panchapagesan. The information content of the limit order book: evidence from nyse specialist trading decisions. *Journal of Financial Markets*, 8(1):25–67, 2005.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Jacob Hilton, Jie Tang, and John Schulman. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*, 2023.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Tao Hong, Pierre Pinson, Yi Wang, Rafa l Weron, Dazhi Yang, and Hamidreza Zareipour. Energy forecasting: A review and outlook. *IEEE Open Access Journal of Power and Energy*, 7:376–388, 2020.

Benjamin Hoover, Yuchen Liang, Bao Pham, Rameswar Panda, Hendrik Strobelt, Duen Horng Chau, Mohammed J Zaki, and Dmitry Krotov. Energy transformer. *arXiv preprint arXiv:2302.07253*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/57a9b97477b67936298489e3c1417b0a-Abstract-Conference.html.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982. URL https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984. URL https://www.pnas.org/doi/abs/10.1073/pnas.81.10.3088.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jerry Yao-Chieh Hu, Donglin Yang, Dennis Wu, Chenwei Xu, Bo-Yu Chen, and Han Liu. On sparse modern hopfield model, 2023. URL https://arxiv.org/abs/2309.12673.

Jerry Yao-Chieh Hu, Pei-Hsuan Chang, Robin Luo, Hong-Yu Chen, Weijian Li, Wei-Po Wang, and Han Liu. Outlier-efficient hopfield layers for large transformer-based models. 2024a. URL https://arxiv.org/abs/2404.03828.

Jerry Yao-Chieh Hu, Bo-Yu Chen, Dennis Wu, Feng Ruan, and Han Liu. Nonparametric modern hopfield models. 2024b. URL https://arxiv.org/abs/2404.03900.

Jerry Yao-Chieh Hu, Thomas Lin, Zhao Song, and Han Liu. On computational limits of modern hopfield models: A fine-grained complexity analysis. *arXiv preprint arXiv:2402.04520*, 2024c. URL https://arxiv.org/abs/2402.04520.

Zexin Hu, Yiqi Zhao, and Matloob Khushi. A survey of forex and stock price prediction using deep learning. *Applied System Innovation*, 4(1):9, 2021.

Hailong Huang and Andrey V Savkin. Towards the internet of flying robots: A survey. *Sensors*, 18 (11):4038, 2018.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Maximilian Ilse, Jakub Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *International conference on machine learning*, pages 2127–2136. PMLR, 2018. URL https://proceedings.mlr.press/v80/ilse18a.html?ref=https://githubhelp.com.

Željko Ivezić, Steven M. Kahn, J. Anthony Tyson, Bob Abel, Emily Acosta, Robyn Allsman, David Alonso, Yusra AlSayyad, Scott F. Anderson, John Andrew, James Roger P. Angel, George Z. Angeli, Reza Ansari, Pierre Antilogus, Constanza Araujo, Robert Armstrong, Kirk T. Arndt, Pierre Astier, Éric Aubourg, Nicole Auza, Tim S. Axelrod, Deborah J. Bard, Jeff D. Barr, Aurelian Barrau, James G. Bartlett, Amanda E. Bauer, Brian J. Bauman, Sylvain Baumont, Ellen Bechtol, Keith Bechtol, Andrew C. Becker, Jacek Becla, Cristina Beldica, Steve Bellavia, Federica B. Bianco, Rahul Biswas, Guillaume Blanc, Jonathan Blazek, Roger D. Bland ford, Josh S. Bloom, Joanne Bogart, Tim W. Bond, Michael T. Booth, Anders W. Borgland, Kirk Borne, James F. Bosch, Dominique Boutigny, Craig A. Brackett, Andrew Bradshaw, William Nielsen Brand t, Michael E. Brown, James S. Bullock, Patricia Burchat, David L. Burke, Gianpietro Cagnoli, Daniel Calabrese, Shawn Callahan, Alice L. Callen, Jeffrey L. Carlin, Erin L. Carlson, Srinivasan Chand rasekharan, Glenaver Charles-Emerson, Steve Chesley, Elliott C. Cheu, Hsin-Fang Chiang, James Chiang, Carol Chirino, Derek Chow, David R. Ciardi, Charles F. Claver, Johann Cohen-Tanugi, Joseph J. Cockrum, Rebecca Coles, Andrew J. Connolly, Kem H. Cook, Asantha Cooray, Kevin R. Covey, Chris Cribbs, Wei Cui, Roc Cutri, Philip N. Daly, Scott F. Daniel, Felipe Daruich, Guillaume Daubard, Greg Daues, William Dawson, Francisco Delgado, Alfred Dellapenna, Robert de Peyster, Miguel de Val-Borro, Seth W. Digel, Peter Doherty, Richard Dubois, Gregory P. Dubois-Felsmann, Josef Durech, Frossie Economou, Tim Eifler, Michael Eracleous, Benjamin L. Emmons, Angelo Fausti Neto, Henry Ferguson, Enrique Figueroa, Merlin Fisher-Levine, Warren Focke, Michael D. Foss, James Frank, Michael D. Freemon, Emmanuel Gangler, Eric Gawiser, John C. Geary, Perry Gee, Marla Geha, Charles J. B. Gessner, Robert R. Gibson, D. Kirk Gilmore, Thomas Glanzman, William Glick, Tatiana Goldina, Daniel A. Goldstein, Iain Goodenow, Melissa L. Graham, William J. Gressler, Philippe Gris, Leanne P. Guy, Augustin Guyonnet, Gunther Haller, Ron Harris, Patrick A. Hascall, Justine Haupt, Fabio Hernand ez, Sven Herrmann, Edward Hileman, Joshua Hoblitt, John A. Hodgson, Craig Hogan, James D. Howard, Dajun Huang, Michael E. Huffer, Patrick Ingraham, Walter R. Innes, Suzanne H. Jacoby, Bhuvnesh Jain, Fabrice Jammes, M. James Jee, Tim Jenness, Garrett Jernigan, Darko Jevremović, Kenneth Johns, Anthony S. Johnson, Margaret W. G. Johnson, R. Lynne Jones, Claire Juramy-Gilles, Mario Jurić, Jason S. Kalirai, Nitya J. Kallivayalil, Bryce Kalmbach, Jeffrey P. Kantor, Pierre Karst, Mansi M. Kasliwal, Heather Kelly, Richard Kessler, Veronica Kinnison, David Kirkby, Lloyd Knox, Ivan V. Kotov, Victor L. Krabbendam, K. Simon Krughoff, Petr Kubánek, John Kuczewski, Shri Kulkarni, John Ku, Nadine R. Kurita, Craig S. Lage, Ron Lambert, Travis Lange, J. Brian Langton, Laurent Le Guillou, Deborah Levine, Ming Liang, Kian-Tat Lim, Chris J. Lintott, Kevin E. Long, Margaux Lopez, Paul J. Lotz, Robert H. Lupton, Nate B. Lust, Lauren A. MacArthur, Ashish Mahabal, Rachel Mand elbaum, Thomas W. Markiewicz, Darren S. Marsh, Philip J. Marshall, Stuart Marshall, Morgan May, Robert McKercher, Michelle McQueen, Joshua Meyers, Myriam Migliore, Michelle Miller, David J. Mills, Connor Miraval, Joachim Moeyens, Fred E. Moolekamp, David G. Monet, Marc Moniez, Serge Monkewitz, Christopher Montgomery, Christopher B. Morrison, Fritz Mueller, Gary P. Muller, Freddy Muñoz Arancibia, Douglas R. Neill, Scott P. Newbry, Jean-Yves Nief, Andrei Nomerotski, Martin Nordby, Paul O'Connor, John Oliver, Scot S. Olivier, Knut Olsen, William O'Mullane, Sandra Ortiz, Shawn Osier, Russell E. Owen, Reynald Pain, Paul E. Palecek, John K. Parejko, James B. Parsons, Nathan M. Pease, J. Matt Peterson, John R. Peterson, Donald L. Petravick, M. E. Libby Petrick, Cathy E. Petry, Francesco Pierfederici, Stephen Pietrowicz, Rob Pike, Philip A. Pinto, Raymond Plante, Stephen Plate, Joel P. Plutchak, Paul A. Price, Michael Prouza, Veljko Radeka, Jayadev Rajagopal, Andrew P. Rasmussen, Nicolas Regnault, Kevin A. Reil, David J. Reiss, Michael A. Reuter, Stephen T. Ridgway, Vincent J. Riot, Steve Ritz, Sean Robinson, William Roby, Aaron Roodman, Wayne Rosing, Cecille Roucelle, Matthew R. Rumore, Stefano Russo, Abhijit Saha, Benoit Sassolas, Terry L. Schalk, Pim Schellart, Rafe H. Schindler, Samuel Schmidt, Donald P. Schneider, Michael D. Schneider, William Schoening, German Schumacher, Megan E. Schwamb, Jacques Sebag, Brian Selvy, Glenn H. Sembroski, Lynn G. Seppala, Andrew Serio, Eduardo Serrano, Richard A. Shaw, Ian Shipsey, Jonathan Sick, Nicole Silvestri, Colin T. Slater, J. Allyn Smith, R. Chris Smith, Shahram Sobhani, Christine Soldahl, Lisa Storrie-Lombardi, Edward Stover, Michael A. Strauss, Rachel A. Street, Christopher W. Stubbs, Ian S. Sullivan, Donald Sweeney, John D. Swinbank, Alexander Szalay, Peter Takacs, Stephen A. Tether, Jon J. Thaler, John Gregg Thayer, Sandrine Thomas, Adam J. Thornton, Vaikunth Thukral, Jeffrey Tice, David E. Trilling, Max Turri, Richard Van Berg, Daniel Vanden Berk, Kurt Vetter, Francoise Virieux, Tomislav Vucina, William Wahl, Lucianne Walkowicz, Brian Walsh, Christopher W. Walter, Daniel L. Wang, Shin-Yawn Wang, Michael Warner, Oliver Wiecha, Beth Willman, Scott E. Winters, David Wittman, Sidney C. Wolff, W. Michael Wood-Vasey, Xiuqin Wu, Bo Xin, Peter Yoachim, and Hu Zhan. LSST: From Science Drivers to Reference Design and Anticipated Data Products. *apj*, 873(2):111, Mar 2019. doi: 10.3847/1538-4357/ab042c.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Weiwei Jiang. Applications of deep learning in stock market prediction: recent progress. *Expert Systems with Applications*, 184:115537, 2021.

Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Chia-Yuan Chang, and Xia Hu. GrowLength: Accelerating LLMs pretraining by progressively growing training length. *arXiv preprint arXiv:2310.00576*, 2023.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-llm: Time series forecasting by reprogramming large language models, 2024. URL https://arxiv.org/abs/2310.01728.

Tyler Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. AdaScale SGD: A user-friendly algorithm for distributed training. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017. URL https://arxiv.org/abs/1703.03129.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

Ahmed Khaled and Peter Richtárik. Better theory for SGD in the nonconvex world. *Transactions on Machine Learning Research*, 2023.

Taekyoung Kim, Sudong Lee, Taehwa Hong, Gyowook Shin, Taehwan Kim, and Yong-Lae Park. Heterogeneous sensing in a multifunctional soft sensor for human-robot interfaces. *Science robotics*, 5(49):eabc6878, 2020.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Diederik P. Kingma and Jimmy Lei Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. URL https://arxiv.org/abs/2001.04451.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

Leo Kozachkov, Ksenia V Kastanenka, and Dmitry Krotov. Building transformers from neurons and astrocytes. *Proceedings of the National Academy of Sciences*, 120(34):e2219150120, 2023. URL https://www.biorxiv.org/content/10.1101/2022.10.12.511910v1.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Dmitry Krotov. A new frontier for hopfield networks. *Nature Reviews Physics*, pages 1–2, 2023. URL https://www.nature.com/articles/s42254-023-00595-y.

Dmitry Krotov and John Hopfield. Large associative memory problem in neurobiology and machine learning. *arXiv preprint arXiv:2008.06996*, 2020. URL https://arxiv.org/abs/2008.06996.

Dmitry Krotov and John J. Hopfield. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

S. R. Kulkarni, Fiona A. Harrison, Brian W. Grefenstette, Hannah P. Earnshaw, Igor Andreoni, Danielle A. Berg, Joshua S. Bloom, S. Bradley Cenko, Ryan Chornock, Jessie L. Christiansen, Michael W. Coughlin, Alexander Wuollet Criswell, Behnam Darvish, Kaustav K. Das, Kishalay De, Luc Dessart, Don Dixon, Bas Dorsman, Kareem El-Badry, Christopher Evans, K. E. Saavik Ford, Christoffer Fremling, Boris T. Gansicke, Suvi Gezari, Y. Goetberg, Gregory M. Green, Matthew J. Graham, Marianne Heida, Anna Y. Q. Ho, Amruta D. Jaodand, Christopher M. Johns-Krull, Mansi M. Kasliwal, Margaret Lazzarini, Wenbin Lu, Raffaella Margutti, D. Christopher Martin, Daniel Charles Masters, Barry McKernan, Yael Naze, Samaya M. Nissanke, B. Parazin, Daniel A. Perley, E. Sterl Phinney, Anthony L. Piro, G. Raaijmakers, Gregor Rauw, Antonio C. Rodriguez, Hugues Sana, Peter Senchyna, Leo P. Singer, Jessica J. Spake, Keivan G. Stassun, Daniel Stern, Harry I. Teplitz, Daniel R. Weisz, and Yuhan Yao. Science with the Ultraviolet Explorer (UVEX). *arXiv e-prints*, art. arXiv:2111.15608, November 2021. doi: 10.48550/arXiv.2111.15608.

Mahinda Mailagaha Kumbure, Christoph Lohrmann, Pasi Luukka, and Jari Porras. Machine learning techniques and data for stock market forecasting: A literature review. *Expert Systems with Applications*, 197:116659, 2022.

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between SGD and Adam on transformers, but sign descent might be. In *International Conference on Learning Representations (ICLR)*, 2023.

Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why Adam outperforms gradient descent on language models. *arXiv preprint arXiv:2402.19449*, 2024.

Ricardo Laborda and Jose Olmo. Volatility spillover between economic sectors in financial crisis prediction: Evidence spanning the great financial crisis and covid-19 pandemic. *Research in International Business and Finance*, 57:101402, 2021. URL https://www.sciencedirect.com/science/article/abs/pii/S0275531921000234.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, 2018.

Jeremy Large. Measuring the resiliency of an electronic limit order book. *Journal of Financial Markets*, 10(1):1–25, 2007.

*Tim Tsz-Kit Lau, *Weijian Li, Chenwei Xu, Han Liu, and Mladen Kolar. Adaptive batch size schedules for distributed training of language models with data and model parallelism. *arXiv preprint arXiv:2412.21124*, 2024a. * These authors contributed equally to this work.

Tim Tsz-Kit Lau, Weijian Li, Chenwei Xu, Han Liu, and Mladen Kolar. Communication-efficient adaptive batch size strategies for distributed local gradient methods. *arXiv preprint arXiv:2406.13936*, 2024b.

Tim Tsz-Kit Lau, Han Liu, and Mladen Kolar. AdAdaGrad: Adaptive batch size schemes for adaptive gradient methods. *arXiv preprint arXiv:2402.11215*, 2024c.

Phong VV Le, James T Randerson, Rebecca Willett, Stephen Wright, Padhraic Smyth, Clement Guilloteau, Antonios Mamalakis, and Efi Foufoula-Georgiou. Climate-driven changes in the predictability of seasonal precipitation. *Nature communications*, 14(1):3822, 2023. URL https://www.nature.com/articles/s41467-023-39463-9.

Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer Berlin Heidelberg, 2002.

YC Lee, Gary Doolen, HH Chen, GZ Sun, Tom Maxwell, and HY Lee. Machine learning using a higher order correlation network. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States); Univ. of Maryland, College Park, MD (United States), 1986. URL https://www.osti.gov/biblio/5760643.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021.

G. Leung and A.R. Barron. Information theory and mixing least-squares regressions. *IEEE Transactions on Information Theory*, 52(8):3396–3410, 2006. doi: 10.1109/TIT.2006.878172.

Conglong Li, Minjia Zhang, and Yuxiong He. The stability-efficiency dilemma: Investigating sequence length warmup for training GPT models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. Domain generalization with adversarial feature learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5400–5409, 2018.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020a.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. PyTorch distributed: Experiences on accelerating data parallel training. In *Proceedings of the VLDB Endowment*, 2020b.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019. URL https://arxiv.org/abs/1907.00235.

*Weijian Li, *Haozheng Luo, Chenwei Xu, and Han Liu. Swga: A distributed hyperparameter search method for time series prediction modelsswga: A distributed hyperparameter search method for time series prediction models. under review. * These authors contributed equally to this work, 2025a.

Weijian Li and Han Liu. Rdas: A low latency and high throughput raw data engine for machine learning systems. under review, 2025.

Weijian Li, Stephen S Cheng, Lining Mao, Alex Pyo, Kawatra Mehak, Li Jialong, Jiayi Wang, Ammar Gilani, Jingya Xun, Jui-Hui Chung, Jerry Yao-Chieh Hu, and Han Liu. A benchmark study for limit order book (lob) models and time series forecasting models on lob data. under review, 2025b.

Weijian Li, Qinjie Lin, Hong-Yu Chen, Nabeel Rehemtulla, Ved G. Shah, Dennis Wu, Adam Miller, and Han Liu. Starembed: A benchmark for evaluating pre-trained light-curve embeddings in variable star classification. under review, 2025c.

Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. TorchTitan: One-stop PyTorch native solution for production ready LLM pre-training. In *International Conference on Learning Representations (ICLR)*, 2025.

Lightning AI. LitGPT, 2023. URL https://github.com/Lightning-AI/litgpt.

Boyi Liu, Lujia Wang, Ming Liu, and Cheng-Zhong Xu. Federated imitation learning: A novel framework for cloud robotic systems with heterogeneous sensor data. *IEEE Robotics and Automation Letters*, 5(2):3509–3516, 2020. doi: 10.1109/LRA.2020.2976321.

Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*, 2021a. URL https://openreview.net/forum?id=0EXmFzUn5I.

Xu Liu, Juncheng Liu, Gerald Woo, Taha Aksu, Yuxuan Liang, Roger Zimmermann, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Moirai-moe: Empowering time series foundation models with sparse mixture of experts. *arXiv preprint arXiv:2410.10469*, 2024a.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting, 2024b. URL https://arxiv.org/abs/2310.06625.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. *arXiv preprint arXiv:2402.02368*, 2024c.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021b. URL https://openaccess.thecvf.com/content/ICCV2021/html/Liu_Swin_Transformer_Hierarchical_Vision_Transformer_Using_Shifted_Windows_ICCV_2021_paper.

Llama Team, AI @ Meta. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Nicholas R Lomb. Least-squares frequency analysis of unequally spaced data. *Astrophysics and space science*, 39(2):447–462, 1976.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Donghao Luo and Xue Wang. Moderntcn: A modern pure convolution structure for general time series analysis. In *The Twelfth International Conference on Learning Representations*, 2024.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

Qianli Ma, Zhen Liu, Zhenjing Zheng, Ziyang Huang, Siying Zhu, Zhongzhong Yu, and James T Kwok. A survey on time-series pre-trained models. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Spyros Makridakis and Michele Hibon. The m3-competition: results, conclusions and implications. *International journal of forecasting*, 16(4):451–476, 2000.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of forecasting*, 34(4): 802–808, 2018.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022.

KL Malanchev, MV Pruzhinskaya, VS Korolev, PD Aleo, MV Kornilov, EEO Ishida, VV Krushinsky, F Mondon, S Sreejith, AA Volnova, et al. Anomaly detection in the zwicky transient facility dr3. *Monthly Notices of the Royal Astronomical Society*, 502(4):5147–5175, 2021.

Sandeep Manjanna, Alberto Quattrini Li, Ryan N. Smith, Ioannis Rekleitis, and Gregory Dudek. Heterogeneous multi-robot system for exploration and strategic water sampling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4873–4880, 2018. doi: 10.1109/ICRA.2018.8460759.

André F. Martins and Ramón F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Frank J Masci, Russ R Laher, Ben Rusholme, David L Shupe, Steven Groom, Jason Surace, Edward Jackson, Serge Monkewitz, Ron Beck, David Flynn, et al. The zwicky transient facility: Data processing, products, and archive. *Publications of the Astronomical Society of the Pacific*, 131 (995):018003, 2018.

Ricardo P Masini, Marcelo C Medeiros, and Eduardo F Mendes. Machine learning advances for time series forecasting. *Journal of economic surveys*, 37(1):76–111, 2023. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/joes.12429`.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

Beren Millidge, Tommaso Salvatori, Yuhang Song, Thomas Lukasiewicz, and Rafal Bogacz. Universal Hopfield networks: A general framework for single-shot associative memory models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

Latrisha N Mintarya, Jeta NM Halim, Callista Angie, Said Achmad, and Aditya Kurniawan. Machine learning approaches in stock market prediction: A systematic literature review. *Procedia Computer Science*, 216:96–102, 2023.

Daniel Moreno-Cartagena, Pavlos Protopapas, Guillermo Cabrera-Vives, Martina Cádiz-Leyton, Ignacio Becker, and Cristóbal Donoso-Oliva. Leveraging Pre-Trained Visual Transformers for

Multi-Band Photometric Light Curve Classification. *arXiv e-prints*, art. arXiv:2502.20479, February 2025. doi: 10.48550/arXiv.2502.20479.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International conference on machine learning*, pages 10–18. PMLR, 2013.

Daniel Muthukrishna, Gautham Narayan, Kaisey S. Mandel, Rahul Biswas, and Renée Hložek. RAPID: Early Classification of Explosive Transients Using Deep Learning. *pasp*, 131(1005): 118002, November 2019. doi: 10.1088/1538-3873/ab1609.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

Falak Nawaz, Naeem Khalid Janjua, and Omar Khadeer Hussain. Perceptus: Predictive complex event processing and reasoning for iot-enabled supply chain. *Knowledge-Based Systems*, 180: 133–146, 2019.

Thomas Nestmeyer, Paolo Robuffo Giordano, Heinrich H Bülthoff, and Antonio Franchi. Decentralized simultaneous multi-target exploration using a connected network of multiple robots. *Autonomous robots*, 41(4):989–1011, 2017.

Charles M Newman. Memory capacity in neural network models: Rigorous lower bounds. *Neural Networks*, 1(3):223–238, 1988. URL `https://www.sciencedirect.com/science/article/abs/pii/0893608088900287`.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022. URL `https://arxiv.org/abs/2211.14730`.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Jbdc0vTOcol`.

Yasuhiro Nitta, Sou Tamura, and Hideki Takase. A study on introducing fpga to ros based autonomous driving system. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 421–424. IEEE, 2018.

Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37(8):852–866, 2018.

Isaac Kofi Nti, Adebayo Felix Adekoya, and Benjamin Asubam Weyori. A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*, 53 (4):3007–3057, 2020a.

Isaac Kofi Nti, Moses Teimeh, Owusu Nyarko-Boateng, and Adebayo Felix Adekoya. Electricity load forecasting: a systematic review. *Journal of Electrical Systems and Information Technology*, 7(1):1–19, 2020b.

Isadora Nun, Pavlos Protopapas, Brandon Sim, Ming Zhu, Rahul Dave, Nicolas Castro, and Karim Pichara. FATS: Feature Analysis for Time Series. *arXiv e-prints*, art. arXiv:1506.00010, May 2015. doi: 10.48550/arXiv.1506.00010.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Frank WJ Olver, Daniel W Lozier, Ronald F Boisvert, and Charles W Clark. *NIST handbook of mathematical functions hardback and CD-ROM*. Cambridge university press, 2010. URL https://dlmf.nist.gov/.

OpenAI. Chatgpt. https://chat.openai.com/, 2022. Accessed: 2025-03-11.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020.

Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

Petr Ostroukhov, Aigerim Zhumabayeva, Chulu Xiang, Alexander Gasnikov, Martin Takáč, and Dmitry Kamzolov. AdaBatchGrad: Combining adaptive batch size and adaptive step size. *arXiv preprint arXiv:2402.05264*, 2024.

Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied soft computing*, 93:106384, 2020.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pages 17156–17185. PMLR, 2022. URL https://proceedings.mlr.press/v162/paischer22a.html.

Rui Pan, Tuan Dung Nguyen, Hardik Arora, Alberto Accomazzi, Tirthankar Ghosal, and Yuan-Sen Ting. Astromlab 2: Astrollama-2-70b model and benchmarking specialised llms for astronomy. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 87–96. IEEE, 2024.

Yan Pan and Yuanzhi Li. Toward understanding why Adam converges faster than SGD for transformers. *arXiv preprint arXiv:2306.00204*, 2023.

Atul Pandey. Depth-wise convolution and depth-wise separable convolution. https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec, 2024. Accessed: 08.09.2024.

Jupinder Parmar, Shrimai Prabhumoye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, Vibhu Jawa, Jiwei Liu, Ameya Mahabaleshwarkar, Osvald Nitski, Annika Brundyn, James Maki, Miguel Martinez, Jiaxuan You, John Kamalu, Patrick LeGresley, Denys Fridman, Jared Casper, Ashwath Aithal, Oleksii Kuchaiev, Mohammad Shoeybi, Jonathan Cohen, and Bryan Catanzaro. Nemotron-4 15B technical report. *arXiv preprint arXiv:2402.16819*, 2024.

Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Deep adaptive input normalization for time series forecasting. *IEEE transactions on neural networks and learning systems*, 31(9):3760–3765, 2019.

Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Temporal logistic neural bag-of-features for financial time series forecasting leveraging limit order book data. *Pattern Recognition Letters*, 136:183–189, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Adrián Arenal Pereira, Jordán Pascual Espada, Rubén González Crespo, and Sergio Ríos Aguilar. Platform for controlling and getting data from network connected drones in indoor environments. *Future Generation Computer Systems*, 92:656–662, 2019.

Pierre Peretto and Jean-Jacques Niez. Long term memory storage capacity of multiconnected neural networks. *Biological Cybernetics*, 54(1):53–63, 1986. URL https://link.springer.com/article/10.1007/BF00337115.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Kandukuri Ratna Prakarsha and Gaurav Sharma. Time series signal forecasting using artificial neural networks: An application on ecg signal. *Biomedical Signal Processing and Control*, 76: 103705, 2022.

Matteo Prata, Giuseppe Masi, Leonardo Berti, Viviana Arrigoni, Andrea Coletta, Irene Cannistraci, Svitlana Vyetrenko, Paola Velardi, and Novella Bartolini. Lob-based deep learning models for stock price trend prediction: a benchmark study. *Artificial Intelligence Review*, 57(5):1–45, 2024.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Raul Puri, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro. Large scale language modeling: Converging on 40GB of text in four hours. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018.

Heyang Qin, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. SimiGrad: Fine-grained adaptive batching for large scale training using gradient similarity measurement. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jorge Pena Queralta and Tomi Westerlund. Blockchain-powered collaboration in heterogeneous swarms of robots. *arXiv preprint arXiv:1912.01711*, 2019.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

S.T. Rakkesh, A.R. Weerasinghe, and R.A.C. Ranasinghe. Simulation of real-time vehicle speed violation detection using complex event processing. In *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, pages 1–6, 2016. doi: 10.1109/ICIAFS.2016.7946549.

Hubert Ramsauer, Bernhard Schafl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlovic, Geir Kjetil Sandve, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020. URL https://arxiv.org/abs/2008.02217.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. Lag-llama: Towards foundation models for probabilistic time series forecasting, 2024. URL https://arxiv.org/abs/2310.08278.

Nabeel Rehemtulla, Adam A Miller, Theophile Jegou Du Laz, Michael W Coughlin, Christoffer Fremling, Daniel A Perley, Yu-Jing Qin, Jesper Sollerman, Ashish A Mahabal, Russ R Laher, et al. The zwicky transient facility bright transient survey. iii. btsbot: automated identification and follow-up of bright transients with deep learning. *The Astrophysical Journal*, 972(1):7, 2024.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing billion-scale model training. In *USENIX Annual Technical Conference (USENIX ATC)*, 2021.

Alex Reneau, Jerry Yao-Chieh Hu, Chenwei Xu, Weijian Li, Ammar Gilani, and Han Liu. Feature programming for multivariate time series prediction. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 29009–29029. PMLR, 23–29 Jul 2023. URL https://arxiv.org/abs/2306.06252.

Joseph W. Richards, Dan L. Starr, Nathaniel R. Butler, Joshua S. Bloom, John M. Brewer, Arien Crellin-Quick, Justin Higgins, Rachel Kennedy, and Maxime Rischard. On Machine-learned Classification of Variable Stars with Sparse and Noisy Time-series Data. *apj*, 733(1):10, May 2011. doi: 10.1088/0004-637X/733/1/10.

Philippe Rigollet and Alexandre Tsybakov. Exponential screening and optimal rates of sparse estimation. *The Annals of Statistics*, 39(2):731–771, 2011.

Vincenzo Ripepi, Maria-Rosa L Cioni, Maria Ida Moretti, Marcella Marconi, Kenji Bekki, Gisella Clementini, Richard de Grijs, Jim Emerson, Martin AT Groenewegen, Valentin D Ivanov, et al. The vmc survey–xxv. the 3d structure of the small magellanic cloud from classical cepheids. *Monthly Notices of the Royal Astronomical Society*, 472(1):808–827, 2017.

Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 130, page 136. Citeseer, 2015.

Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Ioanid Roşu. A dynamic model of the limit order book. *The Review of Financial Studies*, 22(11): 4601–4641, 2009.

Slavek M Rucinski. The short-period end of the contact binary period distribution based on the all-sky automated survey. *Monthly Notices of the Royal Astronomical Society*, 382(1):393–396, 2007.

Francesco Rundo, Francesca Trenta, Agatino Luigi Di Stallo, and Sebastiano Battiato. Machine learning for quantitative finance applications: A survey. *Applied Sciences*, 9(24):5574, 2019.

P. Sánchez-Sáez, I. Reyes, C. Valenzuela, F. Förster, S. Eyheramendy, F. Elorrieta, F. E. Bauer, G. Cabrera-Vives, P. A. Estévez, M. Catelan, G. Pignata, P. Huijse, D. De Cicco, P. Arévalo, R. Carrasco-Davis, J. Abril, R. Kurtev, J. Borissova, J. Arredondo, E. Castillo-Navarrete,

D. Rodriguez, D. Ruz-Mieres, A. Moya, L. Sabatini-Gacitúa, C. Sepúlveda-Cobo, and E. Camacho-Iñiguez. Alert Classification for the ALeRCE Broker System: The Light Curve Classifier. *aj*, 161 (3):141, March 2021. doi: 10.3847/1538-3881/abd5c1.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016. URL https://proceedings.mlr.press/v48/santoro16.html.

Jeffrey D Scargle. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. *Astrophysical Journal, Part 1, vol. 263, Dec. 15, 1982, p. 835-853.*, 263:835–853, 1982.

Johannes Schimunek, Philipp Seidl, Lukas Friedrich, Daniel Kuhn, Friedrich Rippmann, Sepp Hochreiter, and Günter Klambauer. Context-enriched molecule representations improve few-shot drug discovery. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XrMWUuEevr.

Antonio Sclocchi and Matthieu Wyart. On the different regimes of stochastic gradient descent. *Proceedings of the National Academy of Sciences*, 121(9):e2316301121, 2024.

Philipp Seidl, Philipp Renz, Natalia Dyubankova, Paulo Neves, Jonas Verhoeven, Jorg K Wegner, Marwin Segler, Sepp Hochreiter, and Gunter Klambauer. Improving few-and zero-shot reaction template prediction using modern hopfield networks. *Journal of chemical information and modeling*, 62(9):2111–2120, 2022. URL https://pubs.acs.org/doi/full/10.1021/acs.jcim.1c01065.

Cuneyt Sevim, Asil Oztekin, Ozkan Bali, Serkan Gumus, and Erkam Guresen. Developing an early warning system to predict currency crises. *European Journal of Operational Research*, 237(3):1095–1104, 2014. URL https://www.sciencedirect.com/science/article/abs/pii/S0377221714001829.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 90:106181, 2020.

Dev Shah, Haruna Isah, and Farhana Zulkernine. Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies*, 7(2):26, 2019.

Ved G Shah, Alex Gagliano, Konstantin Malanchev, Gautham Narayan, LSST Dark Energy Science Collaboration, et al. Oracle: A real-time, hierarchical, deep-learning photometric classifier for the lsst. *arXiv preprint arXiv:2501.01496*, 2025.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

Oleksandr Shchur, Ali Caner Turkmen, Nick Erickson, Huibin Shen, Alexander Shirkov, Tony Hu, and Bernie Wang. Autogluon–timeseries: Automl for probabilistic time series forecasting. In *International Conference on Automated Machine Learning*, pages 9–1. PMLR, 2023.

Aditi Sheshadri, Marshall Borrus, Mark Yoder, and Thomas Robinson. Midlatitude error growth in atmospheric gcms: The role of eddy growth rate. *Geophysical Research Letters*, 48(23): e2021GL096126, 2021. URL https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2021GL096126.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Leslie F Sikos. Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, 32:200892, 2020.

Samuel L. Smith and Quoc V. Le. A Bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations (ICLR)*, 2018.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations (ICLR)*, 2018.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Bharath K Sriperumbudur and Gert RG Lanckriet. On the convergence of the concave-convex procedure. In *Nips*, volume 9, pages 1759–1767, 2009. URL https://proceedings.neurips.cc/paper_files/paper/2009/hash/8b5040a8a5baf3e0e67386c2e3a9b903-Abstract.html.

Hui Su, Zhi Tian, Xiaoyu Shen, and Xunliang Cai. Unraveling the mystery of scaling laws: Part I. *arXiv preprint arXiv:2403.06563*, 2024.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015. URL https://arxiv.org/abs/1503.08895.

Muhammad Syafrudin, Ganjar Alfian, Norma Latif Fitriyani, and Jongtae Rhee. Performance analysis of iot-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing. *Sensors*, 18(9):2946, 2018.

Mingtian Tan, Mike Merrill, Vinayak Gupta, Tim Althoff, and Tom Hartvigsen. Are language models actually useful for time series forecasting? *Advances in Neural Information Processing Systems*, 37:60162–60191, 2024.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious. *arXiv preprint arXiv:2501.00656*, 2025.

The Mosaic Research Team. Introducing DBRX: A new state-of-the-art open LLM. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024.

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Dat Thanh Tran, Alexandros Iosifidis, Juho Kanniainen, and Moncef Gabbouj. Temporal attention-augmented bilinear network for financial time-series data analysis. *IEEE transactions on neural networks and learning systems*, 30(5):1407–1418, 2018.

Dat Thanh Tran, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Data normalization for bilinear structures in high-frequency financial time-series. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 7287–7292. IEEE, 2021.

Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of statistical physics*, 52:479–487, 1988. URL https://link.springer.com/article/10.1007/bf01016429.

Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th conference on business informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017a.

Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning to detect price change indications in financial markets. In *2017 25th European signal processing conference (EUSIPCO)*, pages 2511–2515. IEEE, 2017b.

Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing*, 93:106401, 2020.

R Tylenda, M Hajduk, T Kamiński, A Udalski, I Soszyński, MK Szymański, M Kubiak, G Pietrzyński, R Poleski, K Ulaczyk, et al. V1309 scorpii: merger of a contact binary. *Astronomy & Astrophysics*, 528:A114, 2011.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

United States International Trade Commission. The 2021 commodity price surge: Causes and impacts on trade flows. https://www.usitc.gov/research_and_analysis/tradeshifts/2021/special_topic, 2021. Accessed: 08.09.2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

V. Ashley Villar, Griffin Hosseinzadeh, Edo Berger, Michelle Ntampaka, David O. Jones, Peter Challis, Ryan Chornock, Maria R. Drout, Ryan J. Foley, Robert P. Kirshner, Ragnhild Lunnan, Raffaella Margutti, Dan Milisavljevic, Nathan Sanders, Yen-Chen Pan, Armin Rest, Daniel M. Scolnic, Eugene Magnier, Nigel Metcalfe, Richard Wainscoat, and Christopher Waters. SuperRAENN: A Semisupervised Supernova Photometric Classification Pipeline Trained on Pan-STARRS1 Medium-Deep Survey Supernovae. *apj*, 905(2):94, December 2020. doi: 10.3847/1538-4357/abc6fd.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

James Wallbridge. Transformers for limit order books. *arXiv preprint arXiv:2003.00130*, 2020.

Bohan Wang, Jingwen Fu, Huishuai Zhang, Nanning Zheng, and Wei Chen. Closing the gap between the upper bound and lower bound of Adam's iteration complexity. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S Yu. Visual domain adaptation with manifold embedded distribution alignment. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 402–410, 2018.

Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and S Yu Philip. Generalizing to unseen domains: A survey on domain generalization. *IEEE transactions on knowledge and data engineering*, 35(8):8052–8072, 2022.

Ken Wang. MicroLlama-300M. https://github.com/keeeeenw/MicroLlama, 2024.

Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and Jun Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. *arXiv preprint arXiv:2405.14616*, 2024a.

Xinlei Wang, Caomingzhe Si, Jinjin Gu, Guolong Liu, Wenxuan Liu, Jing Qiu, and Junhua Zhao. Electricity-consumption data reveals the economic impact and industry recovery during the pandemic. *Scientific Reports*, 11(1):19960, 2021a. URL https://www.nature.com/articles/s41598-021-98259-3.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yunzhong Qiu, Jianmin Wang, and Mingsheng Long. Timexer: Empowering transformers for time series forecasting with exogenous variables. *arXiv preprint arXiv:2402.19072*, 2024b.

Zhe Wang, Tianzhen Hong, Han Li, and Mary Ann Piette. Predicting city-scale daily electricity consumption using data-driven models. *Advances in Applied Energy*, 2:100025, 2021b. URL https://www.sciencedirect.com/science/article/pii/S2666792421000184.

Bruce W Weber. Next-generation trading in futures markets: A comparison of open outcry and order matching systems. *Journal of Management Information Systems*, 16(2):29–45, 1999.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014. URL https://arxiv.org/abs/1410.3916.

Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

Darrell Whitley, Soraya Rana, and Robert B Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology*, 7(1):33–47, 1999.

Michael Widrich, Bernhard Schäfl, Milena Pavlović, Hubert Ramsauer, Lukas Gruber, Markus Holzleitner, Johannes Brandstetter, Geir Kjetil Sandve, Victor Greiff, and Sepp Hochreiter. Modern Hopfield networks and attention for immune repertoire classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Alan FT Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Distributed autonomous robotic systems 4*, pages 273–282. Springer, 2000.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024a.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. 2024b.

Jeffrey M Wooldridge, Mokhtarul Wadud, and Jenny Lye. *Introductory econometrics: Asia pacific edition with online study tools 12 months*. Cengage AU, 2016. URL https://books.google.com.tw/books/about/Introductory_Econometrics_Asia_Pacific_E.html?id=wXdLDwAAQBAJ&redir_esc=y.

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *International Conference on Learning Representations (ICLR)*, 2024.

Dennis Wu, Jerry Yao-Chieh Hu, Teng-Yun Hsiao, and Han Liu. Uniform memory retrieval with larger capacity for modern hopfield models. 2024. URL https://arxiv.org/abs/2404.03827.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/bcc0d400288793e8bdcd7c19a8ac0c2b-Abstract.html.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *International Conference on Learning Representations*, 2023a.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis, 2023b. URL https://arxiv.org/abs/2210.02186.

*Dennis Wu, *Jerry Yao-Chieh Hu, *Weijian Li, Bo-Yu Chen, and Han Liu. Stanhop: Sparse tandem hopfield model for memory-enhanced time series prediction. *arXiv preprint arXiv:2312.17346*, 2023c. * These authors contributed equally to this work.

Jingge Xiao, Yile Chen, Gao Cong, Wolfgang Nejdl, and Simon Gottschalk. Flextsf: A universal forecasting model for time series with variable regularities, 2024. URL https://arxiv.org/abs/2410.23160.

Aolin Xu and Maxim Raginsky. Information-theoretic analysis of generalization capability of learning algorithms. 30, 2017.

Chenwei Xu, Yu-Chao Huang, Jerry Yao-Chieh Hu, Weijian Li, Ammar Gilani, Hsi-Sheng Goan, and Han Liu. Bishop: Bi-directional cellular learning for tabular data with generalized sparse modern hopfield model. 2024. URL https://arxiv.org/abs/2404.03830.

Xiaojie Xu and Yun Zhang. House price forecasting with neural networks. *Intelligent Systems with Applications*, 12:200052, 2021.

Hao Xue and Flora D. Salim. Promptcast: A new prompt-based learning paradigm for time series forecasting, 2023. URL https://arxiv.org/abs/2210.08964.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, JunTao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Xinhao Yao, Hongjin Qian, Xiaolin Hu, Gengze Xu, and Yong Liu. Theoretical insights into fine-tuning attention mechanism: Generalization and optimization, 2024. URL https://arxiv.org/abs/2410.02247.

Yuanyuan Yao, Dimeng Li, Hailiang Jie, Hailiang Jie, Tianyi Li, Jie Chen, Jiaqi Wang, Feifei Li, and Yunjun Gao. Simplets: An efficient and universal model selection framework for time series forecasting. *Proc. VLDB Endow.*, 16(12):3741–3753, August 2023. ISSN 2150-8097. doi: 10.14778/3611540.3611561. URL https://doi.org/10.14778/3611540.3611561.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations (ICLR)*, 2020.

Bin Yu. Rates of Convergence for Empirical Processes of Stationary Mixing Sequences. *The Annals of Probability*, 22(1):94 – 116, 1994. doi: 10.1214/aop/1176988849.

Guoqi Yu, Jing Zou, Xiaowei Hu, Angelica I Aviles-Rivero, Jing Qin, and Shujun Wang. Revitalizing multivariate time series forecasting: Learnable decomposition with inter-series dependencies and intra-series variations modeling. *arXiv preprint arXiv:2402.12694*, 2024.

Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 8980–8987, 2022.

Alan L Yuille and Anand Rangarajan. The concave-convex procedure (cccp). *Advances in neural information processing systems*, 14, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/hash/a012869311d64a44b5a0d567cd20de04-Abstract.html.

Alan L Yuille and Anand Rangarajan. The concave-convex procedure. *Neural computation*, 15(4): 915–936, 2003. URL https://ieeexplore.ieee.org/abstract/document/6788812.

Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020. doi: 10.1109/ACCESS.2020.2983149.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023. URL https://ojs.aaai.org/index.php/AAAI/article/view/26317.

Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Joshua M. Susskind. Stabilizing transformer training by preventing attention entropy collapse. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

Gemma Zhang, Thomas Helfer, Alexander T. Gagliano, Siddharth Mishra-Sharma, and V. Ashley Villar. Maven: a multimodal foundation model for supernova science. *Machine Learning: Science and Technology*, 5(4):045069, December 2024. doi: 10.1088/2632-2153/ad990d.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How does critical batch size scale in pre-training? In *International Conference on Learning Representations (ICLR)*, 2025a.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024a.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022a.

*Wan Zhang, *Qinjie Lin, *Weijian Li, Chan Lee, Han Liu, and Kai Zhang. Hopformer: Homogeneity-pursuit transformer for time series forecasting. under review. * These authors contributed equally to this work, 2025b.

Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. *arXiv preprint arXiv:2207.01186*, 2022b.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=vSVLM2j9eie`.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=vSVLM2j9eie`.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need Adam: A Hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024b.

Zihao Zhang and Stefan Zohren. Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units. *arXiv preprint arXiv:2105.10430*, 2021.

Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.

Haojun Zhao and Ferdinand Mom. Picotron: Distributed training framework for education and research experimentation, 2025. URL `https://github.com/huggingface/picotron`.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. In *Proceedings of the VLDB Endowment*, 2023.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17325`.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pages 27268–27286. PMLR, 2022. URL `https://proceedings.mlr.press/v162/zhou22g.html`.

Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. Dnabert-2: Efficient foundation model and benchmark for multi-species genome. *arXiv preprint arXiv:2306.15006*, 2023.

Zhihan Zhou, Weimin Wu, Harrison Ho, Jiayi Wang, Lizhen Shi, Ramana V Davuluri, Zhong Wang, and Han Liu. Dnabert-s: Learning species-aware dna embedding with genome foundation models. *arXiv preprint arXiv:2402.08777*, 2, 2024.

Hubert Zimmermann. Osi reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.

Xiaoxiong Zuo, Yihan Tao, Yang Huang, Zhixuan Kang, Huaxi Chen, Chenzhou Cui, Jiashu Pan, Xiao Kong, Xiaoyu Tang, Henggeng Han, Haiyang Mu, Yunfei Xu, Dongwei Fan, Guirong Xue, Ali Luo, and Jifeng Liu. FALCO: a Foundation model of Astronomical Light Curves for time dOmain astronomy. *arXiv e-prints*, art. arXiv:2504.20290, April 2025. doi: 10.48550/arXiv.2504.20290.

APPENDIX A

Supplement Materials for Chapter 2

A.1. Adaptive Batch Size Schedules for Distributed Training of Language Models with Data and Model Parallelism

A.1.1. Additional Details of The Proposed Algorithm

Note that the use of PyTorch `FSDP` does not lead to significance difference in the implementation of the norm test compared to its `DDP` implementation. We assume that the gradients of different parameter shards are concatenated together in the following computation to simplify the representation.

A.1.1.1. The Overall Algorithm.

Algorithm 6 DDP-Norm or FSDP-Norm for AdamW

```
Input: $w_1 \in \mathbb{R}^d$, $m_0 = v_0 = \mathbf{0}_d \in \mathbb{R}^d$, $(\alpha, \lambda, \varepsilon, \beta_1, \beta_2) \in (0, \infty)^5$, $\mathcal{D}_n = \{z_i\}_{i \in [\![n]\!]} \subset \mathcal{Z}$, number
       of workers $J \in \mathbb{N}^*$, number of gradient accumulation steps $M \in \mathbb{N}^*$, number of training
       samples $N \in \mathbb{N}^*$, step counter $k = 1$, processed sample counter $i = 0$, initial (global) batch
       size $b_0$, initial microbatch size $b_{0,J}^M = b_0/(JM)$
  while $i < N$ do
      Sample the i.i.d. data batch (indices) $\mathcal{B}_k$ uniformly from $[\![n]\!]$ of size $b_k \coloneqq |\mathcal{B}_k|$
      Split $\mathcal{B}_k$ evenly to each worker $j \in \{ J \}$, each with $\mathcal{B}_{k,j}$ of size $b_{k,J}$
      for all $j = 1, \dots, J$ in parallel do
          Split $\mathcal{B}_{k,j}$ evenly to each gradient accumulation step $m \in \{ M \}$, each with $\mathcal{B}_{k,j}^m$ of size
          $b_{k,J}^M$
          Initialize $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) = \mathbf{0}_d$
          for $m = 1, \dots, M$ do
              Compute $\frac{1}{M} \nabla \mathrm{L}_{\mathcal{B}_{k,j}^m}(w_k)$
              Accumulate gradients $\nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) \leftarrow \nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) + \frac{1}{M} \nabla \mathrm{L}_{\mathcal{B}_{k,j}^m}(w_k)$
          end for
      end for
      Compute the batch gradient $g_k \coloneqq \nabla \mathrm{L}_{\mathcal{B}_k}(w_k)$ with *all-reduce*
      Compute the approximate gradient variance $\widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k))$ with *all-reduce*
          $$\widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k)) \coloneqq \frac{1}{J} \sum_{j \in [\![J]\!]} \Big( \nabla \mathrm{L}_{\mathcal{B}_{k,j}}(w_k) - g_k \Big)^2$$
      Compute the approximate norm test statistic
          $$\mathsf{T}_k \equiv \mathsf{T}(w_k; \mathcal{B}_k, \eta) \coloneqq \frac{\left\| \widehat{\mathrm{Var}}_{i \in \mathcal{B}_k}(\nabla \ell_i(w_k)) \right\|_1}{\eta^2 \|g_k\|^2}$$
      if $\mathsf{T}_k > b_k$ then
          Increase the next global batch size $b_{k+1} = \lceil \mathsf{T}_k \rceil$
          Round up the microbatch size $b_{k+1,J}^M = \lceil b_{k+1}/(JM) \rceil$
          Update the minibatch size $b_{k+1,J} = M b_{k+1,J}^M$
          Update the global batch size again $b_{k+1} = J b_{k+1,J}$
      else
          $b_{k+1} = b_k$
      end if
      $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$                                    ▷ AdamW
      $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
      $\widehat{m}_k = m_k \odot (1 - \beta_1^k)^{-1}$
      $\widehat{v}_k = v_k \odot (1 - \beta_2^k)^{-1}$
      $w_{k+1} = (1 - \alpha\lambda) w_k - \alpha \widehat{m}_k \odot (\widehat{v}_k^{\frac{1}{2}} + \varepsilon)^{-1}$
      $k \leftarrow k + 1$
      $i \leftarrow i + b_k$
  end while
```

A.1.2. Proofs of Main Text

We give a brief sketch of the omitted proof of the main text in this section. Notice that the convergence analysis of the norm test for Adam largely follows that in Wang et al. (2023), where more details and remarks of the analysis and rationales of its derivation can be found.

Remark A.1.1. Despite the similarity of the proof techniques, we emphasize that our setting requires less restrictive assumptions. While Wang et al. (2023) assume the stochastic oracle of the gradient (i.e., batch gradient in our case) has coordinate-wise affine variance, i.e., for any batch of samples $\mathcal{B} \subseteq \mathcal{D}_n$ and $(\sigma, \tau) \in (0, \infty)^2$, we have

$$(\forall i \in [\![d]\!])(\forall w \in \mathbb{R}^d) \quad \mathbb{E}\Big[(\partial_i \mathrm{L}_{\mathcal{B}}(w))^2\Big] \leqslant \sigma^2 + \tau^2(\partial_i \mathrm{L}(w))^2.$$

We do not impose this global condition which is often difficult to verify in practical scenarios, but instead we increase the (next) batch size such that the condition of the coordinate-wise (exact variance) norm test with constant $\eta \in (0, 1)$ is satisfied at the current iterate $w_k \in \mathbb{R}^d$ with the current batch $\mathcal{B}_k \subseteq \mathcal{D}_n$:

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\Big[(\partial_i \mathrm{L}_{\mathcal{B}_k}(w_k) - \partial_i \mathrm{L}(w_k))^2\Big] \leqslant \eta^2(\partial_i \mathrm{L}(w_k))^2,$$

which implies

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\Big[(\partial_i \mathrm{L}_{\mathcal{B}_k}(w_k))^2\Big] \leqslant (1 + \eta^2)(\partial_i \mathrm{L}(w_k))^2,$$

which is also known as the coordinate-wise *expected strong growth* (E-SG) condition (Lau et al., 2024c). Note that the coordinate-wise (E-SG) condition implies the coordinate-wise *relaxed growth* (RG) condition (Bottou et al., 2018), adopting the nomenclature in Khaled and Richtárik (2023):

$$(\forall i \in [\![d]\!]) \quad \mathbb{E}_k\Big[(\partial_i \mathrm{L}_{\mathcal{B}_k}(w_k))^2\Big] \leqslant \sigma^2 + \tau^2(\partial_i \mathrm{L}(w_k))^2,$$

where $\tau^2 = 1 + \eta^2$ and $\sigma \in (0, \infty)$. Recall that we only require such a condition to hold at the current iterate w_k with the current batch $\mathcal{B}_k$, through the enforcement of the coordinate-wise (exact variance) norm test. Even though the exact variance test is not implemented in practice but its approximate version instead, this is often a good heuristic to justify the convergence of the test.

Additional notation. To simplify notation, we denote the full gradient by $\mathrm{G}_k := \nabla \mathrm{L}(w_k)$, and its ith coordinate by $\mathrm{G}_{k,i} := \partial_i \mathrm{L}(w_k)$.

A.1.2.1. Technical Lemmas. We state without proof the following technical lemmas from Wang et al. (2023).

Lemma A.1.1. Let $0 < \beta_1^2 < \beta_2 < 1$ and consider a sequence of real numbers $(a_n)_{n\in\mathbb{N}^*} \subset \mathbb{R}$. Let $b_0 > 0$, $b_k = \beta_2 b_{k-1} + (1-\beta_2)a_k^2$, $c_0 = 0$ and $c_k = \beta_1 c_{k-1} + (1-\beta_1)a_k$. We have the following inequality

$$\sum_{k=1}^{K} \frac{|c_k|^2}{b_k} \leqslant \frac{(1-\beta_1)^2}{(1-\beta_2)(1-\beta_1/\sqrt{\beta_2})^2}\left(\log\left(\frac{b_K}{b_0}\right) - K\log\beta_2\right). \tag{A.1.1}$$

Lemma A.1.2. Consider the Adam iterates $(w_k)_{k\in\mathbb{N}^*}$ generated by (2.1.6). Then we have

$$(\forall k \in \mathbb{N}^*) \quad |w_{k+1,i} - w_{k,i}| \leqslant \alpha \frac{1-\beta_1}{\sqrt{1-\beta_2}\sqrt{1-\beta_1^2/\beta_2}} \leqslant \alpha \frac{1-\beta_1}{\sqrt{1-\beta_2}\sqrt{1-\beta_1/\beta_2}}.$$

Proof Sketch. This is due to the definition of the Adam iterate and Cauchy–Schwarz's inequality. □

A.1.2.2. Proof of Theorem 2.1.1. Since the proof of Theorem 2.1.1 is highly similar to that in Wang et al. (2023), we just provide a proof sketch. We state the formal theorem as follows.

Theorem A.1.1 (Formal version of Theorem 2.1.1)**.** Suppose that Assumption 2.1.1 holds. Let $(w_k)_{k\in\mathbb{N}^*}$ be the Adam iterates generated by (2.1.6), where the batch size $b_k := |\mathcal{B}_k|$ is chosen such that the coordinate-wise (exact variance) norm test with constant $\eta \in (0,1)$ is satisfied at each iteration $k \in \mathbb{N}^*$. Then, if $0 < \beta_1 \leqslant \sqrt{\beta_2} - 8(1+\eta^2)(1-\beta_2)/\beta_2^2$ and $\beta_2 \in (0,1)$, we have

$$\text{(A.1.2)}\quad \sum_{k=1}^{K}\mathbb{E}[\|\nabla \mathrm{L}(w_k)\|]$$
$$\leqslant \sqrt{c_2 + 2c_1\sum_{i=1}^{d}\left[\log\left(2(K+1)\sum_{i=1}^{d}\sqrt{v_{0,i}+\sigma^2} + 24d\frac{\tau^2 c_1}{\sqrt{\beta_2}}\log\left(d\frac{\tau^2 c_1}{\sqrt{\beta_2}}\right) + \frac{12\tau^2}{\sqrt{\beta_2}}c_2\right)\right]}$$
$$\times\sqrt{2(K+1)\sum_{i=1}^{d}\sqrt{v_{0,i}+\sigma^2} + 24d\frac{\tau^2 c_1}{\sqrt{\beta_2}}\log\left(d\frac{\tau^2 c_1}{\sqrt{\beta_2}}\right) + \frac{12\tau^2}{\sqrt{\beta_2}}c_2},$$

where $v_{0,i}$ is the i-coordinate of v_0, $\tau^2 = 1+\eta^2$, $\sigma \in (0,\infty)$,

$$c_1 := \frac{32L\alpha\left(1+\beta_1/\sqrt{\beta_2}\right)^3}{(1-\beta_2)\left(1-\beta_1/\sqrt{\beta_2}\right)^3} + \frac{16\beta_1^2\sigma(1-\beta_1)}{\beta_2\sqrt{1-\beta_2}\left(1-\beta_1/\sqrt{\beta_2}\right)^3} + \frac{64(1+\sigma^2)\sigma^2L^2\alpha^2 d}{\beta_2^2\left(1-\beta_1/\sqrt{\beta_2}\right)^4\sigma(1-\beta_2)^{3/2}},$$
$$c_2 := \frac{8(1-\beta_1/\sqrt{\beta_2})}{\alpha(1-\beta_1)} + \frac{32}{\beta_2\left(1-\beta_1/\sqrt{\beta_2}\right)^2}\sum_{i=1}^{d}\mathbb{E}\left[\frac{\mathrm{G}_{1,i}^2}{\sqrt{\widetilde{v}_{1,i}}}\right] + 2c_1\sum_{i=1}^{d}\left(\log\left(\frac{1}{\sqrt{\beta_2 v_{0,i}}}\right) - K\log\beta_2\right),$$
$$u_k := \frac{w_k - \beta_1 w_{k-1}/\sqrt{\beta_2}}{1-\beta_1/\sqrt{\beta_2}}.$$

The proof consists of deriving a descent lemma on the sequence $u_k := \frac{w_k-\beta_1 w_{k-1}/\sqrt{\beta_2}}{1-\beta_1/\sqrt{\beta_2}}$.

Lemma A.1.3. Suppose that all the assumptions in Theorem A.1.1 hold. We also define the function $\varphi_k := \mathbb{E}\left[-\alpha\left\langle \mathrm{G}_k, \mathrm{G}_k\odot\widetilde{v}_{k+1}^{-\frac{1}{2}}\right\rangle\right]$. Then we have

$$\mathbb{E}[\mathrm{L}(u_{k+1})]$$
$$\leqslant \mathbb{E}[\mathrm{L}(u_k)] - \frac{\alpha(1-\beta_1)}{4(1-\beta_1/\sqrt{\beta_2})}\mathbb{E}\left[-\alpha\left\langle \mathrm{G}_k, \mathrm{G}_k\odot\widetilde{v}_k^{-\frac{1}{2}}\right\rangle\right] + \frac{2\alpha\sigma\sqrt{1-\beta_2}}{(1-\beta_1^2/\beta_2)^2}\sum_{i=1}^{d}\left[\frac{g_{k,i}^2}{v_{k,i}}\right]$$
$$+ \frac{4\alpha\tau^2}{(1-\beta_1/\sqrt{\beta_2})^2\sqrt{\beta_2}}\sum_{i=1}^{d}\mathbb{E}\left[\frac{1}{\beta_2}\varphi_{k-1} - \varphi_k\right] + \sum_{i=1}^{d}\frac{2\alpha\sigma\sqrt{1-\beta_2}}{(1-\beta_1)(1-\beta_1/\sqrt{\beta_2})}\mathbb{E}\left[\frac{m_{k,i}^2}{v_{k,i}}\right]$$
$$+ \frac{64d(1+\tau^2)\tau^2L^2\alpha^3}{\beta_2^2(1-\beta_1/\sqrt{\beta_2})^3(1-\beta_1)\sigma\sqrt{1-\beta_2}}\cdot\mathbb{E}\left[\left\|m_{k-1}\odot v_{k-1}^{-\frac{1}{2}}\right\|^2\right]$$
$$+ \sum_{i=1}^{d}\frac{2\alpha\sigma\beta_1^2\sqrt{1-\beta_2}}{\beta_2(1-\beta_1)(1-\beta_1/\sqrt{\beta_2})}\mathbb{E}\left[\frac{m_{k-1,i}^2}{v_{k-1,i}}\right]$$
$$+ L\mathbb{E}\left[4\alpha^2\left(\frac{\beta_1/\sqrt{\beta_2}}{1-\beta_1/\sqrt{\beta_2}}\right)^2\left\|m_{k-1}\odot v_{k-1}^{-\frac{1}{2}}\right\|^2 + 3\alpha^2\left(\frac{1}{1-\beta_1/\sqrt{\beta_2}}\right)^2\left\|m_k\odot v_k^{-\frac{1}{2}}\right\|^2\right].$$

Proof Sketch. This bound is derived by bounding the "first-order term" and the "second-order term", similar to the derivation of a descent lemma for Lipschitz smooth functions but on the sequence $(u_k)_{k\in\mathbb{N}^*}$. $\square$

Lemma A.1.4. Suppose that all the assumptions in Theorem A.1.1 hold. Then we have

$$\sum_{k=1}^{K+1}\sum_{i=1}^{d}\mathbb{E}[\tilde{v}_{k,i}^{\frac{1}{2}}] \leqslant 2(K+1)\sum_{i=1}^{d}\sqrt{v_{0,i}+\sigma^2}+\frac{24d\tau^2c_1}{\sqrt{\beta_2}}\log\left(\frac{d\tau^2c_1}{\sqrt{\beta_2}}\right)+\frac{12\tau^2c_2}{\sqrt{\beta_2}}.$$

Proof Sketch. This bound is derived by a *divide-and-conquer* approach, considering the cases $|\mathrm{G}_{k,i}| \geqslant \sigma/\tau$ and $|\mathrm{G}_{k,i}| \leqslant \sigma/\tau$ respectively. $\square$

Proof Sketch of Theorem 2.1.1. The final bound is derived by first summing the inequality in Lemma A.1.3 with the assumed condition of (β_1, β_2). Further application of Lemma A.1.1, Cauchy–Schwarz's inequality and Lemma A.1.4 implies the desired result. $\square$

A.1.3. Details of Numerical Experiments

We provide a summary table for the architecture of the language models we pretrained. More details of these models can be found at Wang (2024); Zhang et al. (2024a); Geng and Liu (2023).

Model	MicroLlama 300M	TinyLlama 1.1B	OpenLlama 3B
n_{params}	304.6M	1.1B	3.4B
d_{model}	2048	2048	2048
n_{layers}	12	22	26
n_{heads}	12	32	32
d_{head}	64	64	100

Table A.1. Specifications of models

We also summarize the training hyperparameters of the three sets of experiments in the following tables.

Model	MicroLlama 300M
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	2048
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	256
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table A.2. Training hyperparameters for MicroLlama 300M

A.1.3.1. MicroLlama 300M.

Model	TinyLlama 1.1B
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	2048
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	128
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table A.3. Training hyperparameters for TinyLlama 1.1B

A.1.3.2. TinyLlama 1.1B.

Model	OpenLlama 3B
Training samples (sequences)	2000000
Learning rate schedule	Linear warmup + cosine decay
Learning rate warmup (samples)	20000 (1% of training samples)
Sequence length (tokens)	512
Optimizer	AdamW
Optimizer scaling rule	None
(β_1, β_2)	$(0.9, 0.95)$
ε	10^{-8}
Peak learning rate	0.0004
Minimum learning rate	0.00004
Base micro batch size	4
Maximum micro batch size	8
Base global batch size	128
Maximum global batch size	8192
Base gradient accumulation steps	16
Data-parallel size	4
Weight decay	0.1
Weight decay skip bias	No
Precision	`bfloat16`
Gradient clipping	1.0
Test interval	1

Table A.4. Training hyperparameters for OpenLlama 3B

APPENDIX B

Supplement Materials for Chapter 3

B.1. RDAS: A Low Latency and High Throughput Raw Data Engine for Machine Learning Systems

B.1.1. Additional Background

Robotics system. ROS (Robotics Operating System) (Quigley et al., 2009) is a system providing communication layers on top of machine operating systems to a computation cluster. Heterogeneous robots can use it to communicate with each other or a centralized cloud system. However, ROS only provides an underlying communication infrastructure and it does not deal with a higher level regarding data processing, which is where RDAS can step in.

Data format. Some other file formats are popular in data-logging scenarios. For instance, Ros Bag file format [1] is the format that Ros uses to store Ros messages in files. However, it is bound to Ros's ecosystem. Another general-purpose message recording data format is the MCAP format [2]. However, both Bag and MCAP's disadvantage compared the PCAP format our data engine uses is that they capture data from the application layer while PCAP captures data from the lower network layer according to the 7-layer OSI model (Zimmermann, 1980). Capturing from the network layer is much faster. Besides, network layer has the additional network information that helps network monitor and analysis. It is crucial for the scenarios that put the requirement for low latency, high throughput and high robustness to extreme.

[1] `https://wiki.ros.org/Bags/Format/2.0`
[2] `https://foxglove.dev/blog/introducing-the-mcap-file-format`

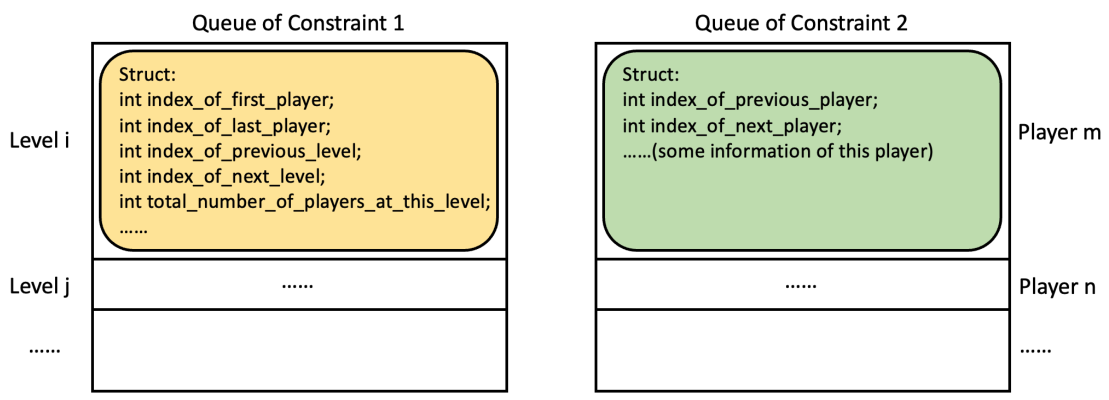

Queue of Constraint 1
Queue of Constraint 2
Level i
Struct:
int index_of_first_player;
int index_of_last_player;
int index_of_previous_level;
int index_of_next_level;
int total_number_of_players_at_this_level;
……
Struct:
int index_of_previous_player;
int index_of_next_player;
……(some information of this player)
Player m
Level j
……
……
Player n
……
……

Figure B.1. Detailed demonstration of the under-the-hood data structure of the minimal example of the message book implementation. Using the game matching example, the queue of Constraint 1 stores all the skill levels, and the queue of Constraint 2 stores all the players. Both queues are implemented using std::vector. Each level is a Struct that includes the index of the first player in this level and the index of the last player in the queue of Constraint 2 so that it provides O(1) time to access the linked list of players in that level. Besides, each level Struct also includes the index of both the previous level and the next level in the queue of Constraint 1 so that the retrieval code can easily traverse all levels according to their order.

B.1.2. Message Book Implementation

In the C++ implementation, we use the built-in std::vector data structure to implement a linked list as each queue in the message book. Such an implementation combines the advantages of both contiguous arrays and linked lists. It provides both $O(1)$ time complexity for random access and for element add/delete operations. Figure B.1 demonstrates the implementation in the two-constraint scenario.

There are auxiliary low-level data structures to support major operations in the message book. There are two hash maps. One maps the id to the index of the corresponding level in the queue of Constraint 1 and the other maps the id to the index of the corresponding entity (player) in the queue of Constraint 2. Besides, there are two vectors called FreeVector. One stores the indices of entries in the queue of Constraint 1 that are free to be occupied and the other stores the indices of entries in the queue of Constraint 2 that are free to be occupied. Last but not least, there are two int variables storing the indices of the first level and the last level in the queue of Constraint 1. The major operations that the message book supports are as follows.

B.2. SWGA: A Distributed Hyperparameter Search Method for Time Series Prediction Models

B.2.1. Hyperparameter

The hyperparameters we tuned for each model are as follows.

B.2.2. Dataset Information

We use 10 different multivariate time series datasets in the paper. They are all commonly used time series datasets. These datasets are from different domains, of different resolutions, and have different numbers of variates. We chose such diverse multivariate time series datasets to demonstrate our method's general efficacy. The following are some brief introductions and a table including the details of the datasets we used.

Beijing PM2.5[3] includes hourly multivariate data from 2010 to 2014. SML2010[4] is a month of home monitoring multivariate data of the resolution of 15 minutes. Appliance Energy[5] has 4 months of energy use data of 10-minute resolution. Individual Household Electricity[6] is 4 years of electricity use. The Exchange dataset[7] includes daily exchange rates in eight different countries from 1990 to 2016. The ETT(Electricity Transformer Temperature) dataset[8] datasets are multivariate time series. There are two collection sources of them with labels 1 and 2. There are two collection resolutions that are 1 hour and 15 minutes. So, there are four specific datasets in this category: ETTh1, ETTh2, ETTm1, and ETTm2. The Traffic dataset[9] consists of hourly road occupancy rates

[3]https://archive.ics.uci.edu/dataset/381/beijing+pm2+5+data
[4]https://archive.ics.uci.edu/dataset/274/sml2010
[5]https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction
[6]https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption
[7]https://github.com/laiguokun/multivariate-time-series-data
[8]https://github.com/zhouhaoyi/ETDataset
[9]http://pems.dot.ca.gov

Table B.1. Hyperparameters For Models

Model	Hyperparameter	Space
LSTM	learning_rate	[1e-6, 1e-1]
	num_layers	[4, 64]
	hidden_size	[4, 128]
	max_epochs	[5, 100]
	batch_size	{16, 32, 64, 128, 256, 512}
	dropout	[0.1, 0.5]
Catboost	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	iterations	[5, 80]
	depth	[4, 12]
	random_strength	[1, 8]
	l2_leaf_reg	[1e-3, 1e3]
	bagging_temperature	[0, 10]
Lightgbm	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	n_estimators	{5, 10, 20, 40, 80}
	max_depth	{4, 6, 8, 10]}
	lambda_l2	{16, 32, 64, 128}
XGBoost	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	n_estimators	[5, 80]
	max_depth	[4, 12]
	reg_lambda	[1e-3, 1e3]
iTransformer	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	d_model	{32, 64, 96, 128, 160, 192, 224, 256 }
	encoder_layers	[1, 10]
DLinear	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	d_model	{32, 64, 96, 128, 160, 192, 224, 256 }
	encoder_layers	[1, 10]
PatchTST	learning_rate	{1e-2, 1e-3, 1e-4, 1e-5, 1e-6}
	d_model	{32, 64, 96, 128, 160, 192, 224, 256 }
	encoder_layers	[1, 10]

from California's Department of Transportation on San Francisco Bay area freeways. All datasets are split into training, validation, and test sets in an 8:1:1 ratio chronologically.

(AE: Appliance Energy, IHE: Individual Household Electricity)

Table B.2. Dataset Details

Dataset	Number of Samples	Number of Variates
Beijing PM2.5	43824	13
SML2010	4137	24
AE	19735	29
IHE	2075259	9
Exchange	7589	8
ETTh1	17421	7
ETTh2	17421	7
ETTm1	69681	7
ETTm2	69681	7
Traffic	17544	862

B.2.3. Computation Hardware and Software

All experiments are conducted on a cluster (except the distributed compute node experiment), where each node has 8 NVIDIA GEFORCE RTX 2080 Ti GPUs and 4 12-core Intel XEON Silver 4214 @ 2.20GHz. The total RAM is 790GB. The operating system is Ubuntu 18.04. The random seed we used was $\{1, 2, 5, 10, 24\}$. The major software and framework we used are PyTorch[10], scikit-learn[11], and Ray[12].

For the scalability experiments, the computing setup consists of computation nodes equipped with 16 Intel(R) Xeon(R) Gold 6230R CPUs and 1 A100 GPU each, with a combined RAM capacity of 1024G.

B.3. STanHop: Sparse Tandem Hopfield Model for Memory-Enhanced Time Series Prediction

B.3.1. Related Works and Limitations

Transformers for Time Series Prediction. As suggested in section 3.3.3 and (Hu et al., 2023; Ramsauer et al., 2020), besides the additional memory functionalities, the Hopfield layers act as

[10]https://pytorch.org/
[11]https://scikit-learn.org/stable/
[12]https://www.ray.io/

promising alternatives for the attention mechanism. Therefore, we discuss and compare STanHop-Net with existing transformer-based time series prediction methods here.

Transformers have gained prominence in time series prediction, inspired by their success in Natural Language Processing and Computer Vision. One challenge in time series prediction is managing transformers' quadratic complexity due to the typically long sequences. To address this, many researchers have not only optimized for prediction performance but also sought to reduce memory and computational complexity. LogTrans (Li et al., 2019) proposes a transformer-based neural network for time series prediction. They propose a convolution layer over the vanilla transformer to better capture local context information and a sparse attention mechanism to reduce memory complexity. Similarly, Informer (Zhou et al., 2021) proposes convolutional layers in between attention blocks to distill the dominating attention and a sparse attention mechanism where the keys only attend to a subset of queries. Reformer (Kitaev et al., 2020) replaces the dot-product self-attention in the vanilla transformer with a hashing-based attention mechanism to reduce the complexity. Besides directly feeding the raw time series inputs to the model, many works focus on transformer-based time series prediction by modeling the decomposed time series. Autoformer (Wu et al., 2021) introduces a series decomposition module to its transformer-based model to separately model the seasonal component and the trend-cyclical of the time series. FEDformer (Zhou et al., 2022) also models the decomposed time series and they introduce a block to extract signals by transforming the time series to the frequency domain.

Compared to STanHop, the above methods do not model multi-resolution information. Besides, Reformer's attention mechanism sacrifices the global receptive field compared to the vanilla self-attention mechanism and our method, which harms the prediction performance.

Some works intend to model the multi-resolution or multi-scale signals in the time series with a dedicated network design. Pyraformer (Liu et al., 2021a) designs a pyramidal attention module to extract the multi-scale signals from the raw time series. Crossformer (Zhang and Yan, 2022) proposes a multi-scale encoder-decoder architecture to hierarchically extract signals of different resolutions from the time series. Compared to these methods. STanHop adopts a more fine-grained multi-resolution modeling mechanism that is capable of learning different sparsity levels for signals in the data of different resolutions.

Furthermore, all of the above works on time series prediction lack the external memory retrieval module as ours. Thus, our STanHop method and its variations have a unique advantage in that we have a fast response to real-time unexpected events.

Hopfield Models and Deep Learning. Hopfield Models (Hopfield, 1984, 1982; Krotov and Hopfield, 2016) have garnered renewed interest in the machine learning community due to the connection between their memory retrieval dynamics and attention mechanisms in transformers via the Modern Hopfield Models (Wu et al., 2024; Hu et al., 2024b, 2023; Ramsauer et al., 2020). Furthermore, these modern Hopfield models enjoy superior empirical performance and possess several appealing theoretical properties, such as rapid convergence and guaranteed exponential memory capacity. By viewing modern Hopfield models as generalized attentions with enhanced memory functionalities, these advancements pave the way for innovative Hopfield-centric architectural designs in deep learning (Hu et al., 2024a; Hoover et al., 2023; Seidl et al., 2022; Fürst et al., 2022; Ramsauer et al., 2020). Consequently, their applicability spans diverse areas like physics (Krotov, 2023), biology (Schimunek et al., 2023; Kozachkov et al., 2023; Widrich et al., 2020), reinforcement learning (Paischer et al., 2022), tabular learning (Xu et al., 2024), and large language models (Hu et al., 2024a; Fürst et al., 2022).

This work pushes this line of research forward by presenting a Hopfield-based deep architecture (StanHop-Net) tailored for memory-enhanced learning in noisy multivariate time series. In particular, our model emphasizes in-context memorization during training and bolsters retrieval capabilities with an integrated external memory component.

Sparse Modern Hopfield Model. Our work extends the theoretical framework proposed in (Hu et al., 2023) for modern Hopfield models. Their primary insight is that using different entropic regularizers can lead to distributions with varying sparsity. Using the Gibbs entropic regularizer, they reproduce the results of the standard dense Hopfield model (Ramsauer et al., 2020) and further propose a sparse variant with the Gini entropic regularizer, providing improved theoretical guarantees. However, their sparse model primarily thrives with data of high intrinsic sparsity. To combat this, we enrich the link between Hopfield models and attention mechanisms by introducing *learnable sparsity* and showing that the sparse model from (Hu et al., 2023) is a specific case of our model when setting $\alpha = 2$. Unlike (Hu et al., 2023), our generalized sparse Hopfield model ensures adaptable sparsity across various data types without sacrificing theoretical integrity. By making this sparsity learnable, we introduce the `GSH` layers. These new Hopfield layers adeptly learn and store sparse representations in any deep learning pipeline, proving invaluable for inherently noisy time series data.

Memory Augmented Neural Networks. The integration of external memory mechanisms with neural networks has emerged as a pivotal technique for machine learning, particularly for tasks requiring complex data manipulation and retention over long sequences, such as open question answering, and few-shot learning.

Neural Turing Machines (NTMs) (Graves et al., 2014) combine the capabilities of neural networks with the external memory access of a Turing machine. NTMs use a differentiable controller (typically an RNN) to interact with an external memory matrix through read and write heads. This design allows NTMs to perform complex data manipulations, akin to a computer with a read-write memory. Building upon this, Graves et al. (2016) further improve the concept through Differentiable Neural Computers (DNCs), which enhance the memory access mechanism using a differentiable attention process. This includes a dynamic memory allocation and linkage system that tracks the relationships between different pieces of data in memory. This feature makes DNCs particularly adept at tasks that require complex data relationships and temporal linkages.

Concurrently, Memory Networks (Weston et al., 2014) showcase the significance of external memory in tasks requiring complex reasoning and inference. Unlike traditional neural networks that rely on their inherent weights to store information, Memory Networks incorporate a separate memory matrix that stores and retrieves information across different processing steps. This capability allows the network to maintain and manipulate a “memory” of past inputs and computations, which is particularly crucial for tasks requiring persistent memory, such as question-answering systems where the network needs to remember context or facts from previous parts of a conversation or text to accurately respond to queries. This concept is further developed into the End-to-End Memory Networks (Sukhbaatar et al., 2015), which extend the utility of Memory Networks (Weston et al., 2014) beyond the limitations of traditional recurrent neural network architectures, transitioning them into a fully end-to-end trainable framework, thereby making them more adaptable and easier to integrate into various learning paradigms.

A notable application of memory-augmented neural networks is in the domain of one-shot learning. The concept of meta-learning with memory-augmented neural networks, as explored by Santoro et al. (2016), has demonstrated the potential of these networks to rapidly adapt to new tasks by leveraging their external memory, highlighting their versatility and efficiency in learning. They showcase the potential of these networks to adapt rapidly to new tasks, a crucial capability in scenarios where data availability is limited. Complementing this, Kaiser et al. (2017) focus on enhancing the recall of rare events and is particularly notable for its exploration of memory-augmented neural networks designed to improve the retention and recall of infrequent but significant occurrences, highlighting the potential of external memory modules in handling rare data challenges. This is achieved through a unique soft attention mechanism, which dynamically assigns relevance weights to different memory entries, enabling the model to draw on a broad spectrum of stored experiences. This approach not only facilitates the effective recall of rare events but also adapts to new data, ensuring the memory’s relevance and utility over time.

In all above methods, (Kaiser et al., 2017) is closest to this work. However, our approach diverges in two key aspects: **(Enhanced Generalization):** Firstly, our external memory enhancements are external plugins with an option for fine-tuning. This design choice avoids over-specialization on rare events, thereby broadening our method’s applicability and enhancing its generalization capabilities across various tasks where the frequency and recency of data are less pivotal. (**Adaptive Response over Rare Event Memorization):** Secondly, our approach excels in real-time adaptability. By integrating relevant external memory sets tailored to specific inference tasks, our method can rapidly respond and improve performance, even without the necessity of prior learning. This flexibility contrasts with the primary focus on *memorizing* rare events in (Kaiser et al., 2017).

B.3.1.1. Limitations. The proposed generalized sparse modern Hopfield model shares the similar inefficiency due to the $\mathcal{O}(d^2)$ complexity[13]. In addition, the effectiveness of our memory enhancement methods is contingent on the relevance of the external memory set to the specific inference task. Achieving a high degree of relevance in the external memory set often necessitates considerable human effort and domain expertise, just like our selection process detailed in appendix B.3.5.2. This requirement could potentially limit the model's applicability in scenarios where such resources are scarce or unavailable.

[13]As a side note, Hu et al. (2024c) provides a characterization of the computational efficiency of all possible efficient variants of the modern Hopfield model from the fine-grained complexity theory.

B.3.2. Proofs of Main Text

B.3.2.1. Lemma 3.3.1.

Our proof relies on verifying that $\Psi^\star$ meets the criteria of Danskin's theorem.

Proof of Lemma 3.3.1. Firstly, we introduce the notion of convex conjugate.

Definition B.3.1. Let $F(\boldsymbol{p}, \boldsymbol{z}) \coloneqq \langle \boldsymbol{p}, \boldsymbol{z} \rangle - \Psi^\alpha(\boldsymbol{p})$. The convex conjugate of Ψ^α, $\Psi^\star$ takes the form:

$$\Psi^\star(\boldsymbol{z}) = \operatorname*{Max}_{\boldsymbol{p}\in\Delta^M} \langle \boldsymbol{p}, \boldsymbol{z} \rangle - \Psi^\alpha(\boldsymbol{p}) = \operatorname*{Max}_{\boldsymbol{p}\in\Delta^M} F(\boldsymbol{p}, \boldsymbol{z}). \tag{B.3.1}$$

By Danskin's theorem (Danskin, 2012; Bertsekas et al., 1999), the function $\Psi^\star$ is convex and its partial derivative with respect to $\boldsymbol{z}$ is equal to that of F, i.e. ${}^{\partial\Psi^\star}/_{\partial\boldsymbol{z}} = {}^{\partial F}/_{\partial\boldsymbol{z}}$, if the following three conditions are satisfied for $\Psi^\star$ and F:

(i) $F(\boldsymbol{p}, \boldsymbol{z}) : \mathcal{P} \times \mathbb{R}^M \to \mathbb{R}$ is a continuous function, where $\mathcal{P} \subset \mathbb{R}^M$ is a compact set.

(ii) F is convex in $\boldsymbol{z}$, i.e. for each given $\boldsymbol{p} \in \mathcal{P}$, the mapping $\boldsymbol{z} \to F(\boldsymbol{p}, \boldsymbol{z})$ is convex.

(iii) There exists an unique maximizing point $\widehat{\boldsymbol{p}}$ such that $F(\widehat{\boldsymbol{p}}, \boldsymbol{z}) = \operatorname{Max}_{\boldsymbol{p}\in\mathcal{P}} F(\boldsymbol{p}, \boldsymbol{z})$.

Since both $\langle \boldsymbol{p}, \boldsymbol{z} \rangle$ and Ψ^α are continuous functions and every component of $\boldsymbol{p}$ is ranging from 0 to 1, the function F is continuous and the domain $\mathcal{P}$ is a compact set. Therefore, condition (i) is satisfied.

Since we require $\boldsymbol{p} \in \Delta^M$ (i.e. $\mathcal{P} = \Delta^M$) to be probability distributions, for any fixed $\boldsymbol{p}$, $F(\boldsymbol{p}, \boldsymbol{z}) = \langle \boldsymbol{p}, \boldsymbol{z} \rangle - \Psi^\alpha(\boldsymbol{p})$ reduces to an affine function depending only on input $\boldsymbol{z}$. Due to the inner product form, this affine function is convex in $\boldsymbol{z}$, and hence condition (ii) holds for all given $\boldsymbol{p} \in \mathcal{P} = \Delta^M$.

Since, for any given $\boldsymbol{z}$, α-EntMax only produces one unique probability distribution $\boldsymbol{p}^\star$, condition (iii) is satisfied. Therefore, from Danskin's theorem, it holds

$$\boldsymbol{\nabla}_z \Psi^\star(\boldsymbol{z}) = \frac{\partial F}{\partial \boldsymbol{z}} = \frac{\partial}{\partial \boldsymbol{z}}(\langle \boldsymbol{p}, \boldsymbol{z} \rangle - \Psi^\alpha(\boldsymbol{p})) = \boldsymbol{p} = \alpha\text{-}\mathrm{EntMax}(\boldsymbol{z}). \tag{B.3.2}$$

□

B.3.2.2. Lemma 3.3.2.

Our proof is built on (Hu et al., 2023, Lemma 2.1). We first derive $\mathcal{T}$ by utilizing Lemma 3.3.1 and Remark B.3.1, along with the convex-concave procedure (Yuille and Rangarajan, 2003, 2001). Then, we show the monotonicity of minimizing $\mathcal{H}$ with $\mathcal{T}$ by constructing an iterative upper bound of $\mathcal{H}$ which is convex in $\boldsymbol{x}_{t+1}$ and thus, can be lowered iteratively by the convex-concave procedure.

Proof. From Lemma 3.3.1, the conjugate convex of Ψ, $\Psi^\star$, is always convex, and, therefore, $-\Psi^\star$ is a concave function. Then, the energy function $\mathcal{H}$ defined in (3.3.3) is the sum of the convex function $\mathcal{H}_1(\boldsymbol{x}) \coloneqq \frac{1}{2}\langle \boldsymbol{x}, \boldsymbol{x}\rangle$ and the concave function $\mathcal{H}_2(\boldsymbol{x}) \coloneqq -\Psi^\star(\Xi^{\mathbb{T}}\boldsymbol{x})$.

Furthermore, by definition, the energy function $\mathcal{H}$ is differentiable.

Every iteration step of convex-concave procedure applied on $\mathcal{H}$ gives

$$\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_1(\boldsymbol{x}_{t+1}) = -\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{x}_t), \tag{B.3.3}$$

which implies that

$$\boldsymbol{x}_{t+1} = \boldsymbol{\nabla}_{\boldsymbol{x}}\Psi(\Xi\boldsymbol{x}_t) = \Xi\,\alpha\text{-EntMax}(\Xi^{\mathbb{T}}\boldsymbol{x}_t). \tag{B.3.4}$$

On the basis of (Yuille and Rangarajan, 2003, 2001), we show the decreasing property of (3.3.3) over t via solving the minimization problem of energy function:

$$\underset{\boldsymbol{x}}{\text{Min}}[\mathcal{H}(\boldsymbol{x})] \quad = \quad \underset{\boldsymbol{x}}{\text{Min}}[\mathcal{H}_1(\boldsymbol{x}) + \mathcal{H}_2(\boldsymbol{x})], \tag{B.3.5}$$

which, in convex-concave procedure, is equivalent to solve the iterative programming

$$\boldsymbol{x}_{t+1} \quad \in \quad \underset{\boldsymbol{x}}{\operatorname{argmin}}[\mathcal{H}_1(\boldsymbol{x}) + \langle \boldsymbol{x}, \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{x}_t)\rangle], \tag{B.3.6}$$

for all t. The concept behind this programming is to linearize the concave function $\mathcal{H}_2$ around the solution for current iteration, $\boldsymbol{x}_t$, which makes $\mathcal{H}_1(\boldsymbol{x}_{t+1}) + \langle \boldsymbol{x}_{t+1}, \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{x}_t)\rangle$ convex in $\boldsymbol{x}_{t+1}$.

The convexity of $\mathcal{H}_1$ and concavity of $\mathcal{H}_2$ imply that the inequalities

$$\mathcal{H}_1(\boldsymbol{x}) \geqslant \mathcal{H}_1(\boldsymbol{y}) + \langle(\boldsymbol{x}-\boldsymbol{y}), \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_1(\boldsymbol{y})\rangle, \tag{B.3.7}$$

$$\mathcal{H}_2(\boldsymbol{x}) \leqslant \mathcal{H}_2(\boldsymbol{y}) + \langle(\boldsymbol{x}-\boldsymbol{y}), \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{y})\rangle, \tag{B.3.8}$$

hold for all $\boldsymbol{x}, \boldsymbol{y}$, which leads to

$$\mathcal{H}(\boldsymbol{x}) = \mathcal{H}_1(\boldsymbol{x}) + \mathcal{H}_2(\boldsymbol{x}) \tag{B.3.9}$$

$$\leqslant \mathcal{H}_1(\boldsymbol{x}) + \mathcal{H}_2(\boldsymbol{y}) + \langle(\boldsymbol{x}-\boldsymbol{y}), \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{y})\rangle \coloneqq \mathcal{H}_U(\boldsymbol{x}, \boldsymbol{y}), \tag{B.3.10}$$

where the upper bound of $\mathcal{H}$ is defined as $\mathcal{H}_U$. Then, the iteration (B.3.6)

$$\boldsymbol{x}_{t+1} \in \underset{\boldsymbol{x}}{\operatorname{argmin}}[\mathcal{H}_U(\boldsymbol{x}, \boldsymbol{x}_t)] = \underset{\boldsymbol{x}}{\operatorname{argmin}}[\mathcal{H}_1(\boldsymbol{x}) + \langle \boldsymbol{x}, \boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{H}_2(\boldsymbol{x}_t)\rangle], \tag{B.3.11}$$

can make $\mathcal{H}_U$ decrease iteratively and thus decreases the value of energy function $\mathcal{H}$ monotonically, i.e.

$$\mathcal{H}(\boldsymbol{x}_{t+1}) \leqslant \mathcal{H}_U(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t) \leqslant \mathcal{H}_U(\boldsymbol{x}_t, \boldsymbol{x}_t) = \mathcal{H}(\boldsymbol{x}_t), \tag{B.3.12}$$

for all t. eq. (B.3.10) shows that the retrieval dynamics defined in (3.3.2) can lead the energy function $\mathcal{H}$ to decrease with respect to the increasing t. □

B.3.2.3. Lemma 3.3.3. To prove the convergence property of retrieval dynamics $\mathcal{T}$, first we introduce an auxiliary lemma from (Sriperumbudur and Lanckriet, 2009).

Lemma B.3.1 ((Sriperumbudur and Lanckriet, 2009), Lemma 5)**.** Following Lemma 3.3.3, $\boldsymbol{x}$ is called the fixed point of iteration $\mathcal{T}$ with respect to $\mathcal{H}$ if $\boldsymbol{x} = \mathcal{T}(\boldsymbol{x})$ and is considered as a generalized fixed point of $\mathcal{T}$ if $\boldsymbol{x} \in \mathcal{T}(\boldsymbol{x})$. If $\boldsymbol{x}^\star$ is a generalized fixed point of $\mathcal{T}$, then, $\boldsymbol{x}^\star$ is a stationary point of the energy minimization problem (B.3.5).

Proof. Since the energy function $\mathcal{H}$ monotonically decreases with respect to increasing t in Lemma 3.3.2, we can follow (Hu et al., 2023, Lemma 2.2) to guarantee the convergence property of $\mathcal{T}$ by checking the necessary conditions of Zangwill's global convergence. After satisfying these conditions, Zangwill global convergence theory ensures that all the limit points of $\boldsymbol{x}_{t}{}_{t=0}^{\infty}$ are generalized fixed points of the mapping $\mathcal{T}$ and it holds $\lim_{t\to\infty} \mathcal{H}(\boldsymbol{x}_t) = \mathcal{H}(\boldsymbol{x}^\star)$, where $\boldsymbol{x}^\star$ are some generalized fixed points of $\mathcal{T}$. Furthermore, auxiliary Lemma B.3.1 implies that $\boldsymbol{x}^\star$ are also the stationary points of energy function $\mathcal{H}$. Therefore, we guarantee that $\mathcal{T}$ can iteratively lead the query $\boldsymbol{x}$ to converge to the local optimum of $\mathcal{H}$. $\square$

B.3.2.4. Theorem 3.3.1.

Proof. Compared with $\mathcal{T}_{\text{Dense}}$, $\mathcal{T}$ only sample partial memory patterns near $\boldsymbol{\xi}_\mu$, which gives the tighter bound:

$$\|\mathcal{T}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| \quad\leqslant\quad \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\|. \tag{B.3.13}$$

For $2\geqslant\alpha\geqslant 1$: Then, we derive the upper bound on $\|\mathcal{T}_{\text{dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\|$ based on (Hu et al., 2023, Theorem 2.2):

$$\|\mathcal{T}_{\text{dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| = \left\|\sum_{\nu=1}^{M}[\text{Softmax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x})]_\nu\boldsymbol{\xi}_\nu-\boldsymbol{\xi}_\mu\right\| \tag{B.3.14}$$

$$= \left\|\sum_{\nu=1,\nu\neq\mu}^{M}[\text{Softmax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x})]_\nu\boldsymbol{\xi}_\nu-(1-\text{Softmax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x}))\boldsymbol{\xi}_\mu\right\| \tag{B.3.15}$$

$$\leqslant 2\widetilde{\epsilon}m, \tag{B.3.16}$$

where $\widetilde{\epsilon}\coloneqq(M-1)\exp\left\{-\beta\widetilde{\Delta}_\mu(\boldsymbol{x})\right\}=(M-1)\exp\left\{-\beta(\langle\boldsymbol{\xi}_\mu,\boldsymbol{x}\rangle-\text{Max}_{\nu\in[M]}\langle\boldsymbol{x},\boldsymbol{\xi}_\nu\rangle)\right\}$. Consequently, (3.3.6) results from above and (Ramsauer et al., 2020, Theorem 4,5).

For $\alpha\geqslant 2$. Following the setting of α-EntMax in (Peters et al., 2019), the equation

$$\text{2-EntMax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x})=\text{Sparsemax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x}) \tag{B.3.17}$$

holds. According to the closed form solution of Sparsemax in (Martins and Astudillo, 2016), it holds

$$[\text{Sparsemax}(\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x})]_\mu\leqslant[\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x}]_\mu-[\beta\boldsymbol{\Xi}^{\mathbb{T}}\boldsymbol{x}]_{(\kappa)}+\frac{1}{\kappa}, \tag{B.3.18}$$

for all $\mu \in [M]$. Then, the sparsemax retrieval error is

$$\|\mathcal{T}_{\text{Sparsemax}}(\boldsymbol{x}) - \boldsymbol{\xi}^{\mu}\| = \left\|\Xi \operatorname{Sparsemax}\left(\beta \Xi^{\mathbb{T}} \boldsymbol{x}\right) - \boldsymbol{\xi}^{\mu}\right\| = \left\|\sum_{\nu=1}^{\kappa} \boldsymbol{\xi}_{(\nu)} \left[\operatorname{Sparsemax}\left(\beta \Xi^{\mathbb{T}} \boldsymbol{x}\right)\right]_{(\nu)} - \boldsymbol{\xi}^{\mu}\right\|$$

$$\text{(By (B.3.18))} \qquad \leqslant m + m\beta \left\|\sum_{\nu=1}^{\kappa} \left(\left[\Xi^{\mathbb{T}} \boldsymbol{x}\right]_{(\nu)} - \left[\Xi^{\mathbb{T}} \boldsymbol{x}\right]_{(\kappa)} + \frac{1}{\beta\kappa}\right) \frac{\boldsymbol{\xi}_{(\nu)}}{m}\right\|$$

$$\text{(B.3.19)} \qquad \leqslant m + d^{1/2} m\beta \left[\kappa \left(\operatorname*{Max}_{\nu\in[M]} \langle \boldsymbol{\xi}_{\nu}, \boldsymbol{x}\rangle - \left[\Xi^{\mathbb{T}} \boldsymbol{x}\right]_{(\kappa)}\right) + \frac{1}{\beta}\right].$$

By the first inequality of Theorem 3.3.1, for $\alpha \geqslant 2$, we have

$$\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_{\mu}\| \leqslant \|\mathcal{T}_{\text{Sparsemax}}(\boldsymbol{x}) - \boldsymbol{\xi}_{\mu}\| \leqslant m + d^{1/2} m\beta \left[\kappa \left(\operatorname*{Max}_{\nu\in[M]} \langle \boldsymbol{\xi}_{\nu}, \boldsymbol{x}\rangle - \left[\Xi^{\mathbb{T}} \boldsymbol{x}\right]_{(\kappa)}\right) + \frac{1}{\beta}\right],$$

which completes the proof of (3.3.7). ☐

B.3.2.5. Lemma 3.3.4.

Our proof, built on (Hu et al., 2023, Lemma 2.1), proceeds in 3 steps:

- **(Step 1.)** We establish a more refined well-separation condition, ensuring that patterns $\{\boldsymbol{\xi}_{\mu}\}_{\mu\in[M]}$ are well-stored in $\mathcal{H}$ and can be retrieved by $\mathcal{T}$ with an error ϵ at most R.
- **(Step 2.)** This condition is then related to the cosine similarity of memory patterns, from which we deduce an inequality governing the probability of successful pattern storage and retrieval.
- **(Step 3.)** We pinpoint the conditions for exponential memory capacity and confirm their satisfaction.

Since the generalized sparse Hopfield shares the same well-separation condition (shown in below Lemma B.3.2), it has the same exponential memory capacity as the sparse Hopfield model (Hu et al., 2023, Lemma 3.1). For completeness, we restate the proof of (Hu et al., 2023, Lemma 3.1) below.

Step 1. To analyze the memory capacity of the proposed model, we first present the following two auxiliary lemmas.

Lemma B.3.2. [Corollary 3.1.1 of (Hu et al., 2023)] Let $\delta := \|\mathcal{T}_{\text{Dense}} - \boldsymbol{\xi}_\mu\| - \|\mathcal{T} - \boldsymbol{\xi}_\mu\|$. Then, the well-separation condition can be formulated as:

$$\Delta_\mu \geqslant \frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right) + 2mR. \tag{B.3.20}$$

Furthermore, if $\delta = 0$, this bound reduces to well-separation condition of Softmax-based Hopfield model.

Proof of Lemma B.3.2. Let $\mathcal{T}_{\text{Dense}}$ be the retrieval dynamics given by the dense modern Hopfield model (Ramsauer et al., 2020), and $\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$ and $\|\mathcal{T}_{\text{Dense}}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|$ be the retrieval error of generalized sparse and dense modern Hopfield model, respectively. By Theorem 3.3.1, we have

$$\|\mathcal{T}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\|. \tag{B.3.21}$$

By (Ramsauer et al., 2020, Lemma A.4), we have

$$\|\mathcal{T}_{\text{Dense}}(\boldsymbol{x}) - \boldsymbol{\xi}_\mu\| \leqslant 2\widetilde{\epsilon}m, \tag{B.3.22}$$

where $\widetilde{\epsilon} := (M-1)\exp\left\{-\beta\widetilde{\Delta}_\mu\right\} = (M-1)\exp\left\{-\beta\left(\langle\boldsymbol{\xi}_\mu, \boldsymbol{x}\rangle - \text{Max}_{\nu\in[M]}\langle\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\nu\rangle\right)\right\}$. Then, by the Cauchy-Schwartz inequality

$$|\langle\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu\rangle - \langle\boldsymbol{x}, \boldsymbol{\xi}_\mu\rangle| \leqslant \|\boldsymbol{\xi}_\mu - \boldsymbol{x}\| \cdot \|\boldsymbol{\xi}_\mu\| \leqslant \|\boldsymbol{\xi}_\mu - \boldsymbol{x}\|m, \quad \forall\mu \in [M], \tag{B.3.23}$$

we observe that $\widetilde{\Delta}_\mu$ can be expressed in terms of Δ_μ:

$$\widetilde{\Delta}_\mu \leqslant \Delta_\mu - 2\|\boldsymbol{\xi}_\mu - \boldsymbol{x}\|m = \Delta_\mu - 2mR, \tag{B.3.24}$$

where R is radius of the sphere S_μ. Thus, inserting the upper bound given by (B.3.22) into (3.3.6), we obtain

$$\|\mathcal{T}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| \leqslant \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| \leqslant 2\tilde{\epsilon}m \tag{B.3.25}$$

$$\leqslant 2(M-1)\exp\{-\beta(\Delta_\mu-2mR)\}m. \tag{B.3.26}$$

Then, for any given $\delta := \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| - \|\mathcal{T}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| \leqslant 0$, the retrieval error $\|\mathcal{T}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\|$ has an upper bound:

$$\|\mathcal{T}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\| \leqslant 2(M-1)\exp\{-\beta(\Delta_\mu-2mR+\delta)\}m-\delta \leqslant \|\mathcal{T}_{\text{Dense}}(\boldsymbol{x})-\boldsymbol{\xi}_\mu\|. \tag{B.3.27}$$

Therefore, for $\mathcal{T}$ to be a mapping $\mathcal{T}: S_\mu \to S_\mu$, we need the well-separation condition

$$\Delta_\mu \geqslant \frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right)+2mR. \tag{B.3.28}$$

□

Lemma B.3.3 ((Hu et al., 2023; Ramsauer et al., 2020))**.** If the identity

$$ac + c\ln c - b = 0, \tag{B.3.29}$$

holds for all real numbers $a, b \in \mathbb{R}$, then c takes a solution:

$$c = \frac{b}{W_0(\exp(a+\ln b))}. \tag{B.3.30}$$

Proof of Lemma B.3.3. We restate the proof of (Hu et al., 2023, Lemam 3.1) here for completeness.

With the given equation $ac + c\ln c - b = 0$, we solve for c by following steps:

$$\begin{aligned}
ac + c\ln c - b &= 0,\\
a + \ln c &= \frac{b}{c},\\
\frac{b}{c} + \ln\left(\frac{b}{c}\right) &= a + \ln b,\\
\frac{b}{c}\exp\left(\frac{b}{c}\right) &= \exp(a + \ln b),\\
\frac{b}{c} &= W_0(\exp(a + \ln b)),\\
c &= \frac{b}{W_0(\exp(a + \ln b))}.
\end{aligned}$$

$\square$

Then, we present the main proof of Lemma 3.3.4.

Proof of Lemma 3.3.4. Since the generalized Hopfield model shares the same well-separation condition as the sparse Hopfield model (Hu et al., 2023), the proof of the exponential memory capacity automatically follows that of (Hu et al., 2023). We restate the proof of (Hu et al., 2023, Corollary 3.1.1) here for completeness.

(Step 2.) & (Step 3.) Here we define $\Delta_{\min}$ and $\theta_{\mu\nu}$ as $\Delta_{\min} := \mathrm{Min}_{\mu\in[M]}\Delta_\mu$ and the angle between two patterns $\boldsymbol{\xi}^\mu$ and $\boldsymbol{\xi}^\nu$, respectively. Intuitively, $\theta_{\mu\nu} \in [0,\pi]$ represent the pairwise correlation of two patterns the two patterns and hence

$$\Delta_{\min} = \underset{1\leqslant\mu\leqslant\nu\leqslant M}{\mathrm{Min}}\left[m^2(1-\cos\theta_{\mu\nu})\right] = m^2[1-\cos\theta_{\min}], \tag{B.3.31}$$

where $\theta_{\min} := \mathrm{Min}_{1\leqslant\mu\leqslant\nu\leqslant M}\,\theta_{\mu\nu} \in [0,\pi]$.

From the well-separation condition (B.3.2), we have

$$\Delta_\mu \geqslant \Delta_{\min} \geqslant \frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right) + 2mR. \tag{B.3.32}$$

Hence, we have

$$m^2[1-\cos\theta_{\min}] \geqslant \frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right)+2mR. \tag{B.3.33}$$

Therefore, we are able to write down the probability of successful storage and retrieval, i.e. minimal separation $\Delta_{\min}$ satisfies Lemma B.3.2:

$$P\left(m^2[1-\cos\theta_{\min}] \geqslant \frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right)+2mR\right)=1-p. \tag{B.3.34}$$

By (Olver et al., 2010, (4.22.2)), it holds

$$\cos\theta_{\min} \leqslant 1-\frac{\theta_{\min}^2}{5} \quad \text{for} \quad 0 \leqslant \cos\theta_{\min} \leqslant 1, \tag{B.3.35}$$

and hence

$$P\left(M^{\frac{2}{d-1}}\theta_{\min} \geqslant \frac{\sqrt{5}M^{\frac{2}{d-1}}}{m}\left[\frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right)+2mR\right]^{\frac{1}{2}}\right)=1-p. \tag{B.3.36}$$

Here we introduce $M^{2/d-1}$ on both sides in above for later convenience.

Let $\omega_d := \frac{2\pi^{d+1/2}}{\Gamma\left(\frac{d+1}{2}\right)}$, be the surface area of a d-dimensional unit sphere, where $\Gamma(\cdot)$ represents the gamma function. By (Brauchart et al., 2018, Lemma 3.5), it holds

$$1-p \geqslant 1-\frac{1}{2}\gamma_{d-1}5^{\frac{d-1}{2}}M^2m^{-(d-1)}\left[\frac{1}{\beta}\ln\left(\frac{2(M-1)m}{R+\delta}\right)+2mR\right]^{\frac{d-1}{2}}, \tag{B.3.37}$$

where γ_d is characterized as the ratio between the surface areas of the unit spheres in $(d-1)$ and d dimensions, respectively: $\gamma_d := \frac{1}{d}\frac{\omega_{d-1}}{\omega_d}$.

Since $M=\sqrt{p}C^{\frac{d-1}{4}}$ is always true for $d, M \in \mathbb{N}_+$, $p \in [0,1]$ and some real values $C \in \mathbb{R}$, we have

$$5^{\frac{d-1}{2}}C^{\frac{d-1}{2}}m^{-(d-1)}\left\{\frac{1}{\beta}\ln\left[\frac{2\left(\sqrt{p}C^{\frac{d-1}{4}}-1\right)m}{R+\delta}\right]+\frac{1}{\beta}\right\}^{\frac{d-1}{2}} \leqslant 1. \tag{B.3.38}$$

Then, we rearrange above as

$$\frac{5C}{m^2\beta}\left\{\ln\left[\frac{2\left(\sqrt{p}C^{\frac{d-1}{4}}-1\right)m}{R+\delta}\right]+1\right\}-1\leqslant 0, \tag{B.3.39}$$

and identify

$$a:=\frac{4}{d-1}\left\{\ln\left[\frac{2m(\sqrt{p}-1)}{R+\delta}\right]+1\right\},\quad b:=\frac{4m^2\beta}{5(d-1)}. \tag{B.3.40}$$

By Lemma B.3.3, we have

$$C=\frac{b}{W_0(\exp\{a+\ln b\})}, \tag{B.3.41}$$

where $W_0(\cdot)$ is the upper branch of the Lambert W function. Since the domain of the Lambert W function is $x>(-1/e,\infty)$ and the fact $\exp\{a+\ln b\}>0$, the solution for (B.3.41) exists. When the inequality (B.3.38) holds, we arrive the lower bound on the exponential storage capacity M:

$$M\geqslant\sqrt{p}C^{\frac{d-1}{4}}. \tag{B.3.42}$$

In addition, by the asymptotic expansion of the Lambert W function (Hu et al., 2023, Lemma 3.1), it also holds $M\geqslant M_{\text{Dense}}$, where M_{Dense} is the memory capacity of the dense modern Hopfield model (Ramsauer et al., 2020). □

B.3.3. Methodology Details

B.3.3.1. The Multi-Step GSH Updates. `GSH` inherits the capability of multi-step update for better retrieval accuracy, which is summarized in below algorithm 7 for a given number of update steps κ. In practice, we find that a single update suffices, consistent with our theoretical finding (3.3.6) of Theorem 3.3.1.

B.3.3.2. `GSHPooling` and `GSHLayer`. Here we provide the operational definitions of the `GSHPooling` and the `GSHLayer`.

Algorithm 7 Multi-Step Generalized Sparse Hopfield Update for `GSH`

```
Input: $\kappa \in \mathbb{R} \geqslant 1, \boldsymbol{Q} \in \mathbb{R}^{\text{len}_Q \times D_Q}, \boldsymbol{K} \in \mathbb{R}^{\text{len}_K \times D_K}, \boldsymbol{W}_V \in \boldsymbol{R}^{D_K \times d}, \boldsymbol{W}_K \in \boldsymbol{R}^{D_K \times D_Q}$
  for $i \to 1$ to $\kappa$ do
      $\boldsymbol{Q}^{\text{new}} = \alpha\text{-EntMax}\Big(\beta \boldsymbol{Q}(\boldsymbol{K}\boldsymbol{W}_K)^{\mathbb{T}}\Big)\boldsymbol{K}$ Hopfield Update
      $\boldsymbol{Q} \leftarrow \boldsymbol{Q}^{\text{new}}$
  end for
  $\boldsymbol{Z} = \boldsymbol{Q}\boldsymbol{W}_V$
  return $\boldsymbol{Z}$
```

Definition B.3.2 (Generalized Sparse Hopfield Pooling (`GSHPooling`))**.** Given inputs $\boldsymbol{Y} \in \mathbb{R}^{\text{len}_Y \times D_Y}$, and len_Q query patterns $\boldsymbol{Q} \in \mathbb{R}^{\text{len}_Q \times D_K}$ the 1-step Sparse Adaptive Hopfield Pooling update is

$$\texttt{GSHPooling}(\boldsymbol{Y}) = \alpha\text{-EntMax}\Big(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{D_k}\Big)\boldsymbol{V}, \tag{B.3.43}$$

Here we have $\boldsymbol{K}, \boldsymbol{V}$ equal to $\boldsymbol{V} = \boldsymbol{Y}\mathbf{W}_K\mathbf{W}_V, \boldsymbol{K} = \boldsymbol{Y}\boldsymbol{W}_K$, and $\mathbf{W}_V \in \mathbb{R}^{D_K \times D_K}, \mathbf{W}_K \in \mathbb{R}^{D_K \times D_K}$. Where d is the dimension of K. And the query pattern $\boldsymbol{Q}$ is a learnable variable, and is independent from the input, the size of len_Q controls how many query patterns we want to store.

Definition B.3.3 (Generalized Sparse Hopfield Layer (`GSHLayer`))**.** Given inputs $\boldsymbol{Y} \in \mathbb{R}^{\text{len}_Y \times D_Y}$, and len_Q query patterns $\boldsymbol{Q} \in \mathbb{R}^{\text{len}_Q \times D_K}$ the 1-step Sparse Adaptive Hopfield Layer update is

$$\texttt{GSHLayer}(\boldsymbol{R}, \boldsymbol{Y}) = \alpha\text{-EntMax}\Big(\boldsymbol{R}\boldsymbol{Y}^T/\sqrt{D_k}\Big)\boldsymbol{Y}, \tag{B.3.44}$$

Here $\boldsymbol{R}$ is the input and $\boldsymbol{Y}$ can be either learnable weights or given as an input.

B.3.3.3. Example: Memory Retrieval for Image Completion. The standard memory retrieval mechanism of Hopfield Models contains two inputs, the query $\boldsymbol{x}$ and the associative memory set Ξ. The goal is to retrieve an associated memory $\boldsymbol{\xi}$ most similar to the query $\boldsymbol{x}$ from the stored memory set Ξ. For example, in (Ramsauer et al., 2020), the query $\boldsymbol{x}$ is a corrupted/noisy image from CIFAR10, and the associative memory set Ξ is the CIFAR10 image dataset. All images are flattened into vector-valued patterns. This task can be achieved by taking the query as $\boldsymbol{R} = \boldsymbol{x}$ and the associative memory set as $\boldsymbol{Y} = \Xi$ for `GSHLayer` with fixed parameters. After steps of updates, we expect the output of the `GSHLayer` to be the recovered version of $\boldsymbol{x}$.

B.3.3.4. Pseudo Label Retrieval. Here, we present the use of the memory retrieval mechanism from modern Hopfield models to generate pseudo-labels for queries $\boldsymbol{R}$, thereby enhancing predictions. Given a set of memory patterns $\boldsymbol{Y}$ and their corresponding labels $\boldsymbol{Y}_{\text{label}}$, we concatenate them together to form the *label-included* memory set $\widetilde{\boldsymbol{Y}}$. Take CIFAR10 for example, we can concatenate the flatten images along with their one-hot encoded labels together as the memory set. For the query, we use the input with padded zeros concatenated at the end of it. The goal here is to "retrieve" the padding part in the query, which is expected to be the retrieved "pseudo-label". Note that this pseudo-label will be a weighted sum over all other labels in the associative memory set. An illustration of this mechanism can be found in fig. 3.11. For the retrieved pseudo-label, we can either use it as the final prediction, or use it as pseudo-label to provide extra information for the model.

B.3.3.5. Algorithm for STanHop-Net. Here we summarize the STanHop-Net as below algorithm.

Algorithm 8 STanHop-Net

```
Input: $L \geqslant 1, \boldsymbol{Z} \in \mathbb{R}^{T \times C \times D^0_{hidden}}$
    for $\ell \to 1$ to $L$ do
        $\boldsymbol{Z}^\ell_{\text{enc}} = \texttt{STanHop}\Big(\text{Coarse-Graining}\Big(\boldsymbol{Z}^{\ell-1}_{\text{enc}}, \Delta\Big)\Big)$ encoder forward
    end for
    $\boldsymbol{Z}^0_{\text{dec}} = \boldsymbol{E}_{\text{dec}}$ learnable positional embedding
    for $\ell \to 1$ to $L$ do decoder forward
        $\widetilde{\boldsymbol{Z}}^\ell_{\text{dec}} = \texttt{STanHop}\Big(\boldsymbol{Z}^{\ell-1}_{\text{dec}}\Big)$
        $\widehat{\boldsymbol{Z}}^\ell_{\text{dec}} = \texttt{GSH}\Big(\boldsymbol{Z}^\ell_{\text{dec}}, \boldsymbol{Z}^\ell_{\text{enc}}\Big)$
        $\check{\boldsymbol{Z}}^\ell_{\text{dec}} = \text{LayerNorm}\Big(\widehat{\text{Z}}^\ell_{\text{dec}} + \widetilde{\text{Z}}^\ell_{\text{dec}}\Big)$
        $\boldsymbol{Z}^\ell_{\text{dec}} = \text{LayerNorm}\Big(\check{\text{Z}}^\ell_{\text{dec}} + \text{MLP}\Big(\check{\text{Z}}^\ell_{\text{dec}}\Big)\Big)$
    end for
    return $\boldsymbol{Z}^L_{\text{dec}} \in \mathbb{R}^{\frac{P}{T} \times C \times D_{\text{hidden}}}$
```

B.3.4. Additional Numerical Experiments

Here we provide additional experimental investigations to back up the effectiveness of our method.

B.3.4.1. Numerical Verification's of Theoretical Results.

Faster Fixed Point Convergence and Better Generalization. In fig. B.2, to support our theoretical results in section 3.3.4, we numerically analyze the convergence behavior of the `GSH`, compared with the dense modern Hopfield layer `Hopfield`.

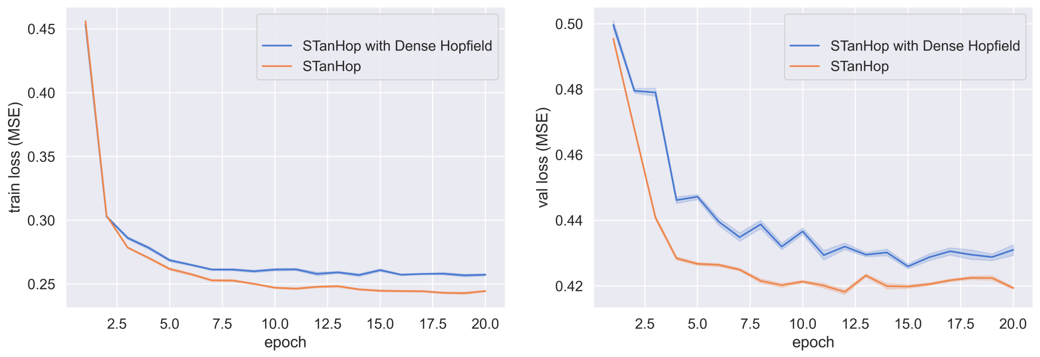


Figure B.2. The training and validation loss curves of STanHop (D), i.e. STanHop-Net with dense modern Hopfield `Hopfield` layer, and STanHop-Net with `GSH` layer. The results show that the generalized sparse Hopfield model enjoys faster convergence than the dense model and also obtain better generalization.

In fig. B.2, we plot the loss curves for STanHop-Net using both generalized sparse and dense modern models on the ETTh1 dataset for the multivariate time series prediction tasks.

The results reveal that the generalized sparse Hopfield model (`GSH`) converges faster than the dense model (`Hopfield`) and also achieves better generalization. This empirically supports our theoretical findings presented in Theorem 3.3.1, which suggest that the generalized sparse Hopfield model provides faster retrieval convergence with enhanced accuracy.

Memory Capacity and Noise Robustness. Following (Hu et al., 2023), we also conduct experiments verifying our memory capacity and noise robustness theoretical results (Lemma 3.3.4 and Theorem 3.3.1), and report the results in fig. B.3. The plots present average values and standard deviations derived from 10 trials.

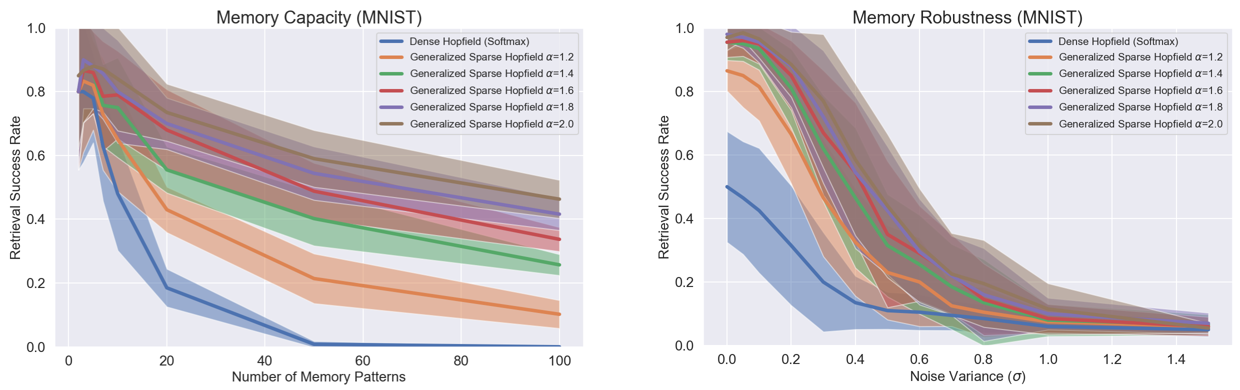


Figure B.3. **Left:** Memory Capacity measured by successful half-masked retrieval rates. **Right:** Memory Robustness measured by retrieving patterns with various noise levels. A query pattern is considered accurately retrieved if its cosine similarity error falls below a specified threshold. We set error threshold of 20% and β=0.01 for better visualization. We plot the average and variance from 10 trials. These findings demonstrate the generalized sparse Hopfield model's ability of capturing data sparsity, improved memory capacity and its noise robustness.

Regarding memory capacity (displayed on the left side of fig. B.3), we evaluate the generalized sparse Hopfield model's ability to retrieve half-masked patterns from the MNIST dataset, in comparison to the Dense modern Hopfield model (Ramsauer et al., 2020).

Regarding robustness against noisy queries (displayed on the right side of fig. B.3), we introduce Gaussian noises of varying variances (σ) to the images.

These findings demonstrate the generalized sparse Hopfield model's ability of capturing data sparsity, improved memory capacity and its noise robustness.

B.3.4.2. Computational Cost Analysis of Memory Modules. Here we analyze the computational cost between the **Plug-and-Play** memory plugin module and the baseline. We evaluate 2 matrices: (i) the number of floating point operations (flops) (ii) number of parameters of the model. Note that for **Plug-and-Play** module, the parameter amount will not be affected by the size of external memory set. The result can be found in fig. B.4 and fig. B.5.

B.3.4.3. Ablation Studies.

Hopfield Model Ablation. Beside our proposed generalized sparse modern Hopfield model, we also test STanHop-Net with 2 other existing different modern Hopfield models: the dense modern

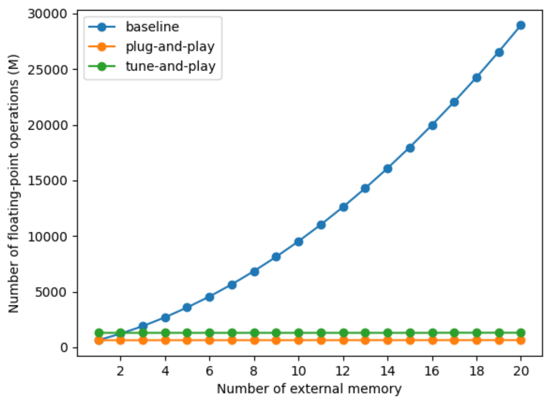


Figure B.4. The number of floating-point operations (flops) (in millions) comparison between **Plug-and-Play**, **Tune-and-Play** and the baseline. The result shows that the **Plug-and-Play**, **Tune-and-Play** successfully reduce the required computational cost to process an increased amount of data.

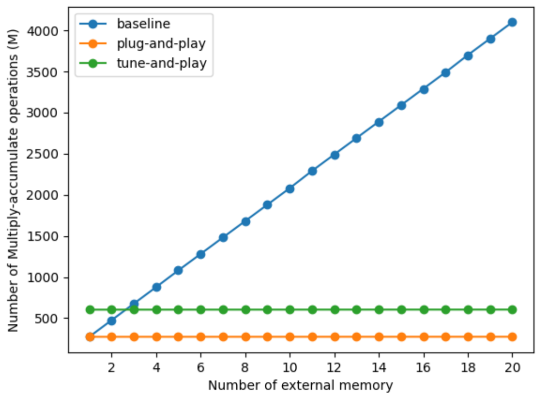


Figure B.5. The number of Multiply–accumulate operations (MACs) (in millions) comparison between **Plug-and-Play**, **Tune-and-Play** and the baseline. The result shows that both of our memory plugin modules face little MACs increasement while the baseline model MACs increase almost linearly w.r.t. the input size.

Hopfield model (Ramsauer et al., 2020) and the sparse modern Hopfield model (Hu et al., 2023). We report their results in table 3.6.

We terms them as **STanHop-Net (D)** and **STanHop-Net (S)** where (D) and (S) are for “Dense” and “Sparse” respectively.

Component Ablation. In order to evaluate the effectiveness of different components in our model, we perform an ablation study by removing one component at a time. In below, we denote Patch Embedding as (PE), StanHop as (SH), Hopfield Pooling as (HP), Multi-Resolution as (MR). We also denote their removals with “w/o” (i.e., without.)

For w/o PE, we set the patch size P equals 1. For w/o MR, we set the coarse level Δ as 1. For w/o SH and w/o HP, we replace those blocks/layers with an MLP layer with GELU activation and layer normalization.

The results are showed in table B.3. From the ablation study results, we observe that removing the STanHop block gives the biggest negative impact on the performance. Showing that the STanHop block contributes the most to the model performance. Note that patch embedding also provides a notable improvement on the performance. Overall, every component provides a different level of performance boost.

Table B.3. **Component Ablation.** We conduct component ablation by separately removing Patch Embedding (PE), STanHop (SH), Hopfield Pooling (HP), and Multi-Resolution (MR). We report the mean MSE and MAE over 10 runs, with variances omitted as they are all $\leqslant 0.15\%$. The results indicate that while every single component in STanHop-Net provides performance boost, the impact of STanHop block on model performance is the most significant among all other components.

Models		STanHop		w/o PE		w/o SH		w/o HP		w/o MR	
	Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.294	0.360	0.318	0.368	0.305	0.365	0.306	0.363	0.307	0.363
	48	0.340	0.387	0.357	0.389	0.352	0.393	0.346	0.387	0.348	0.385
	168	0.420	0.452	0.454	0.476	0.480	0.500	0.434	0.455	0.447	0.464
	336	0.450	0.472	0.501	0.524	0.530	0.535	0.462	0.473	0.482	0.486
	720	0.512	0.520	0.540	0.538	0.610	0.581	0.524	0.526	0.537	0.531
WTH	24	0.292	0.341	0.318	0.365	0.335	0.375	0.340	0.374	0.325	0.373
	48	0.363	0.402	0.386	0.421	0.414	0.439	0.385	0.420	0.391	0.427
	168	0.499	0.515	0.504	0.521	0.507	0.519	0.503	0.525	0.520	0.509
	336	0.499	0.515	0.514	0.529	0.532	0.541	0.513	0.528	0.533	0.542
	720	0.548	0.556	0.570	0.565	0.569	0.565	0.539	0.548	0.555	0.557

B.3.4.4. The Impact of Varying α. We examine the impact of increasing the value of α on memory capacity and noise robustness. It is known that as α approaches infinity, the α-entmax operation transitions to a hardmax operation (Peters et al., 2019; Correia et al., 2019). Furthermore, it is also known that memory pattern retrieval using hardmax is expected to exhibit perfect retrieval ability, potentially offering a larger memory capacity than the softmax modern Hopfield model in pure retrieval tasks (Millidge et al., 2022). Our empirical investigations confirm that higher values of α frequently lead to higher memory capacity. We report results only up to $\alpha = 5$, as we observed that values of α greater than 5 consistently lead to numerical errors, especially under float32 precision. It is crucial to note, that while the hardmax operation (realized when $\alpha \to \infty$) may maximize memory capacity, its lack of differentiability renders it unsuitable for gradient descent-based optimization.

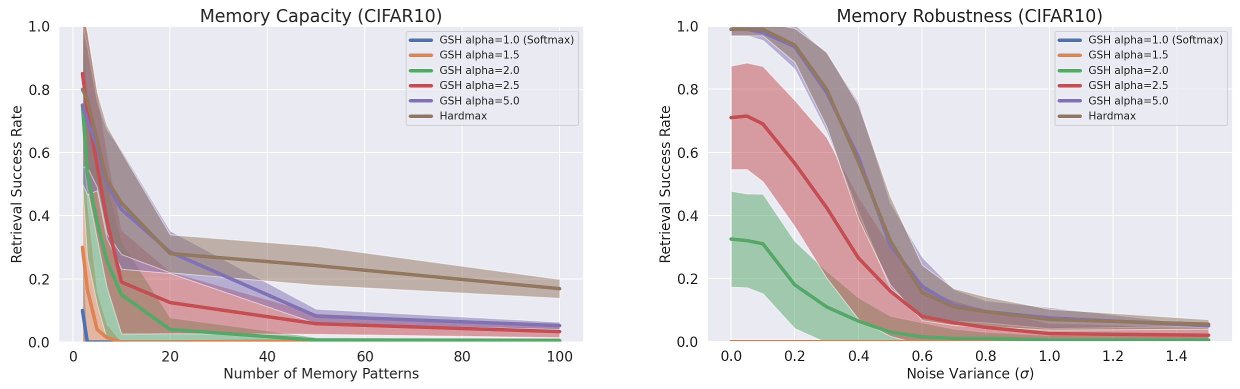
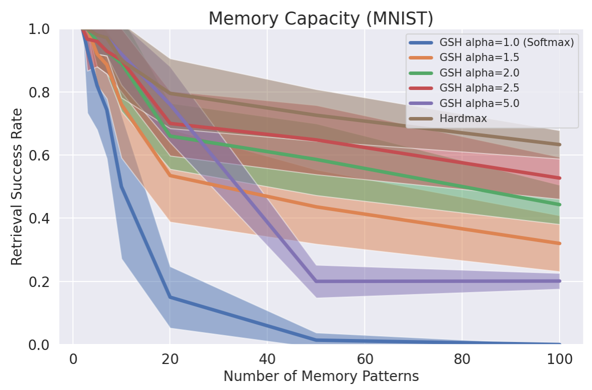


Figure B.6. **Left:** Memory Capacity measured by successful half-masked retrieval rates w.r.t. different values of α on CIFAR10. **Right:** Memory Robustness measured by retrieving patterns with various noise levels on CIFAR10. A query pattern is considered accurately retrieved if its cosine similarity error falls below a specified threshold. We set error threshold of 20% and $\beta = 0.01$ for better visualization. We plot the average and variance from 10 trials. We can see that using hardmax (argmax) normally gives the best retrieval result as it retrieves only the most similar pattern w.r.t. dot product distance. And setting $\alpha = 5$ approximately gives the similar result while having $\alpha = 5$ keeps the overall mechanism differentiable.



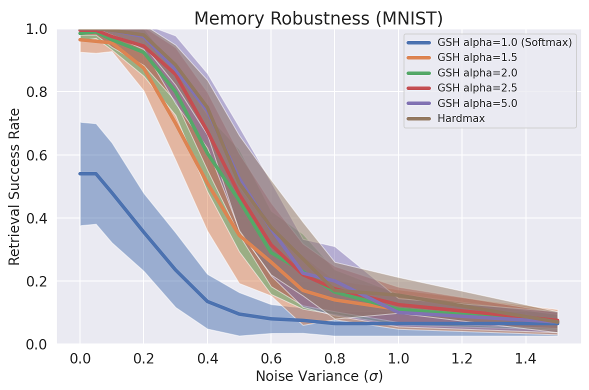


Figure B.7. **Left:** Memory Capacity measured by successful half-masked retrieval rates w.r.t. different values of α on MNIST. **Right:** Memory Robustness measured by retrieving patterns with various noise levels on MNIST. A query pattern is considered accurately retrieved if its cosine similarity error falls below a specified threshold. We set error threshold of 20% and $\beta = 0.1$ for better visualization. We plot the average and variance from 10 trials. We can see that using hardmax (argmax) normally gives the best retrieval result as it retrieves only the most similar pattern w.r.t. dot product distance. And setting $\alpha = 5$ approximately gives the similar result while having $\alpha = 5$ keeps the overall mechanism differentiable.

B.3.4.5. Memory Usage of STanHop and learnable α. To compare the memory and GPU usage, we benchmark STanHop with STanHop-(D) (using *dense* modern Hopfield layers). In below fig. B.8 and fig. B.9, we report the footprints of from the weight and bias (wandb) system (Biewald et al., 2020). The figures clearly demonstrate that the computational cost associated with learning an additional α for adaptive sparsity is negligible.

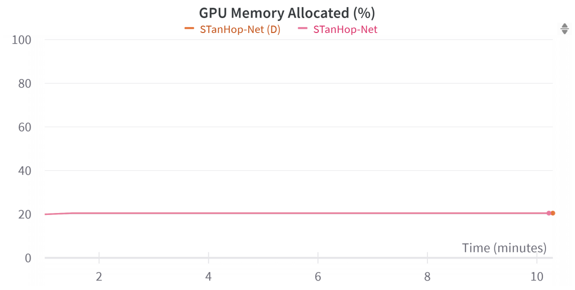


Figure B.8. The GPU memory allocation between STanHop-Net with and without learnable alpha (STanHop-Net (D)). We can see that with learnable alpha does not significantly increase reuired GPU memory.

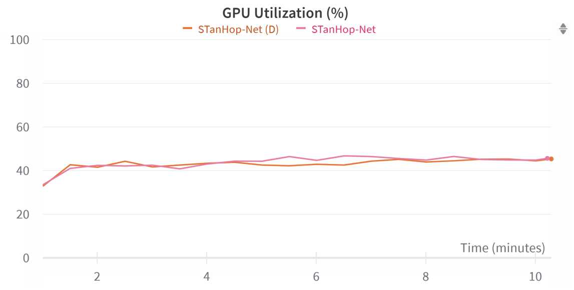


Figure B.9. The percentage of gpu utilization between STanHop-Net with and without learnable alpha (STanHop-Net (D)). We can see that with learnable alpha does not significantly increase the GPU utilization.

B.3.4.6. Time Complexity Analysis of STanHop-Net. Here we use the same notation as introduced in the main paper. Let T be the number of input length on the time dimension, D_{hidden} be the hidden dimension, P be patch size, D_{out} be the prediction horizon, len_Q be the size of query pattern in `GSHPooling`.

- Patch embedding: $\mathcal{O}(D_{\text{hidden}} \times T) = \mathcal{O}(T)$.
- Temporal and Cross series GSH: $\mathcal{O}(D_{\text{hidden}}^2 \times T^2 \times P^{-2}) = \mathcal{O}(T^2 \times P^{-2}) = \mathcal{O}(T^2)$
- Coarse graining: $\mathcal{O}(D_{\text{hidden}}^2 \times T) = \mathcal{O}(T)$
- GSHPooling : $\mathcal{O}(D_{\text{hidden}} \times \text{len}_Q \times T^2 \times P^{-2}) = \mathcal{O}(\text{len}_Q \times T^2)$
- PlugMemory: $\mathcal{O}(T^2)$
- TuneMemory: $\mathcal{O}((T + D_{\text{out}})^2)$

Additionally, the number of parameters of STanHop-Net 0.78 million. As a reference, with batch size of 32, input length of 168, STanHop-Net requires 2 minutes per epoch for the ETTh1 dataset. Meanwhile, STanHop-Net (D) also requires 2 minutes per epoch under same setting.

B.3.4.7. Hyperparameter Sensitivity Analysis. We conduct experiments exploring the parameter sensitivity of STanHop-Net on the ILI dataset. Our results (in table B.5 and table B.6) show that STanHop-Net is not sensitive to hyperparameter changes.

The way that we conduct the hyperparameter sensitivity analysis is that to show the model's sensitivity to a hyperparameter h, we change the value of h in each run and keep the rest of the hyperparameters' values as default and we record the MAE and MSE on the test set. We train and evaluate the model 3 times for 3 different values for each hyperparameter. We analyze the model's sensitivity to 7 hyperparameters respectively. We conduct all the experiments on the ILI dataset.

Table B.4. Default Values of Hyperparameters in Sensitivity Analysis

Parameter	Default Value
seg_len (patch size)	6
window_size (coarse level)	2
e_layers (number of encoder layers)	3
d_model	32
d_ff (feedforward dimension)	64
n_heads (number of heads)	2

Table B.5. MAEs for each value of the hyperparameter with the rest of the hyperparameters as default. For each hyperparameter, each row's MAE score corresponds to the hyperparameter value inside the parentheses in the same order. For example, when lr is 1e-3, the MAE score is 1.313.

lr	seg_len	window_size	e_layers	d_model	d_ff	n_heads
(1e-3, 1e-4, 1e-5)	(6, 12, 24)	(2,4,8)	(3,4,5)	(32, 64, 128)	(64, 128, 256)	(2,4,8)
1.313	1.313	1.313	1.313	1.313	1.313	1.313
1.588	1.311	1.285	1.306	1.235	1.334	1.319
1.673	1.288	1.368	1.279	1.372	1.302	1.312

Table B.6. MSEs for each value of the hyperparameter with the rest of the hyperparameters as default. For each hyperparameter, each row's MSE score corresponds to the hyperparameter value inside the parentheses in the same order. For example, when lr is 1e-3, the MSE score is 3.948.

lr	seg_len	window_size	e_layers	d_model	d_ff	n_heads
(1e-3, 1e-4, 1e-5)	(6, 12, 24)	(2,4,8)	(3,4,5)	(32, 64, 128)	(64, 128, 256)	(2,4,8)
3.948	3.948	3.948	3.948	3.948	3.948	3.948
5.045	3.968	3.877	3.915	3.566	3.983	3.967
5.580	3.865	4.267	3.834	4.078	3.866	3.998

B.3.4.8. Additional Multiple Instance Learning Experiments. We also evaluate the efficacy of the proposed `GSH` layer on Multiple Instance Learning (MIL) tasks. In essence, MIL is a type of supervised learning whose training data are divided into bags and labeling individual data points is difficult or impractical, but bag-level labels are available (Ilse et al., 2018). We follow (Ramsauer et al., 2020; Hu et al., 2023) to conduct the multiple instance learning experiments on MNIST.

Model. We first flatten each image and use a fully connected layer to project each image to the embedding space. Then, we perform `GSHPooling` and use linear projection for prediction.

Hyperparameters. For hyperparameters, we use hidden dimension of 256, number of head as 4, dropout as 0.3, training epoch as 100, optimizer as AdamW, initial learning rate as $1e-4$, and we also use the cosine annealing learning rate decay.

Baselines. We benchmark the Generalized Sparse modern Hopfield model (GSH) with the Sparse (Hu et al., 2023) and Dense (Ramsauer et al., 2020) modern Hopfield models.

Dataset. For the training dataset, we randomly sample 1000 positive and 1000 negative bags for each bag size. For test data, we randomly sample 250 positive and 250 negative bags for each bag size. We set the positive signal to the images of digit 9, and the rest as negative signals. We vary the bag size and report the accuracy and loss curves of 10 runs on both training and test data.

Results. Our results (shown in the figures below) demonstrate that `GSH` converges faster than the baselines in most settings, as it can adapt to varying levels of data sparsity. This is consistent with our theoretical findings in Theorem 3.3.1, which state that `GSH` achieves higher accuracy and faster fixed-point convergence compared to the dense model.

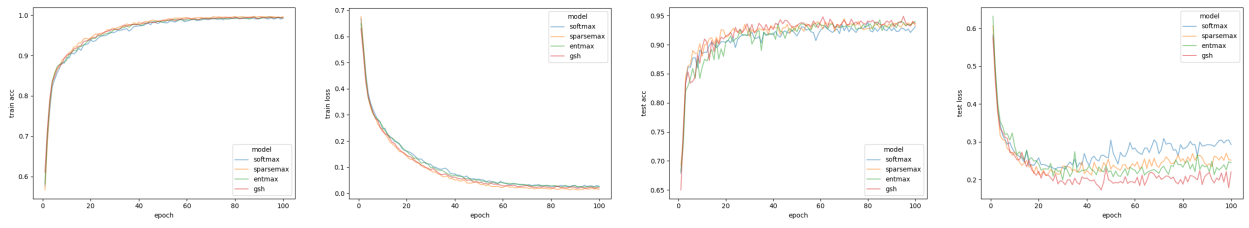


Figure B.10. The MIL experiment with bag size 5. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

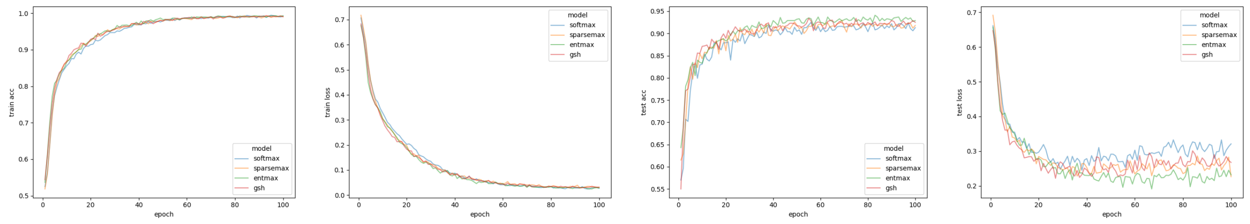


Figure B.11. The MIL experiment with bag size 10. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

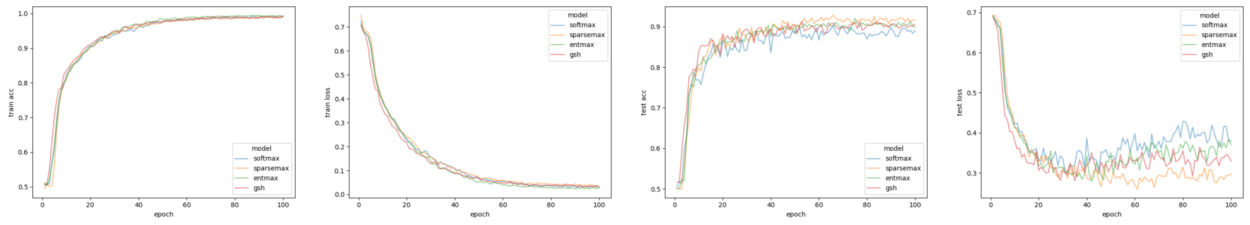


Figure B.12. The MIL experiment with bag size 20. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

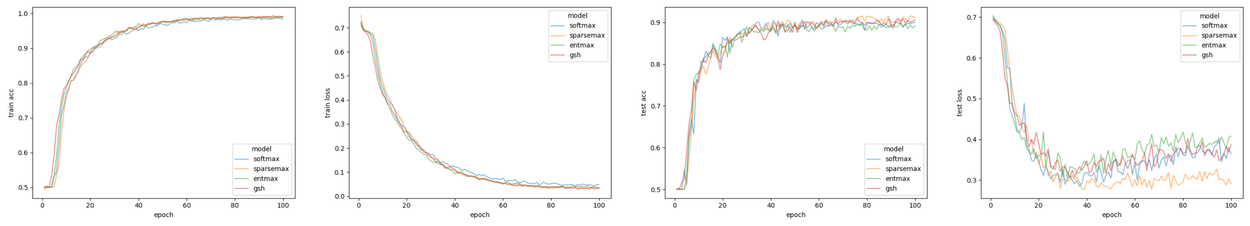


Figure B.13. The MIL experiment with bag size 30. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

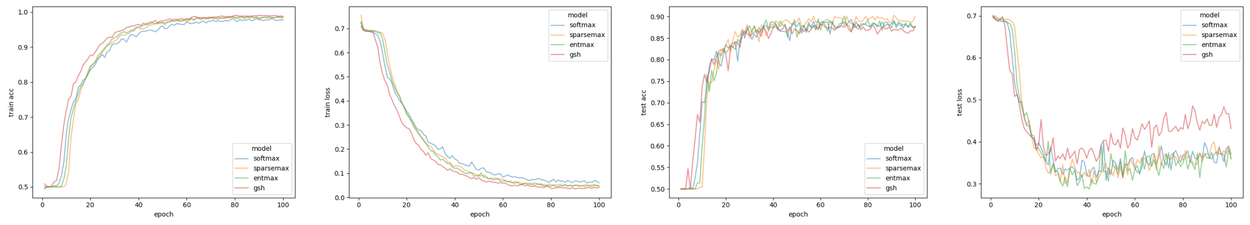


Figure B.14. The MIL experiment with bag size 50. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

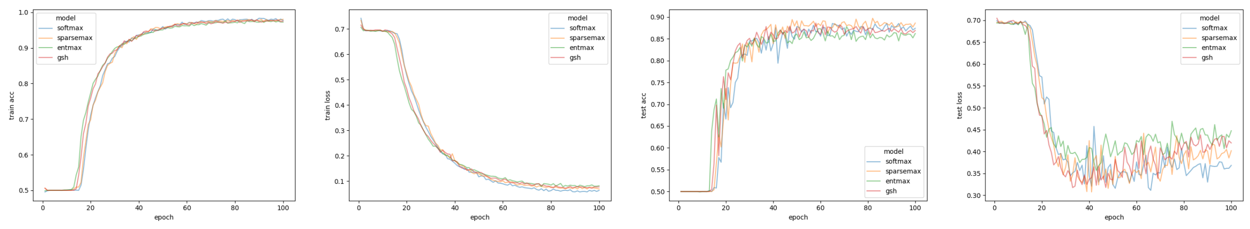


Figure B.15. The MIL experiment with bag size 100. From left to right: (1) Training data accuracy curve (2) Training data loss curve (3) Test data accuracy curve (4) Test data accuracy curve

B.3.4.9. STanHop-Net Outperforms DLinear in Settings Dominated by Multivariate Correlations: A Case Study. The performance of STanHop-Net in the main text, as presented in table 3.6, does not show a clear superiority over DLinear (Zeng et al., 2023). We attribute this to the nature of the common benchmark datasets in table 3.6, which are not dominated by multivariate correlations.

To verify our conjecture, we employ a strongly correlated multivariate time series dataset as a test bed, representing a practical scenario where multivariate correlations are the predominant source of predictive information in input features. In such a scenario, following the same setting in section 3.3.5.1, our experiments show that STanHop-Net consistently outperforms DLinear. We also compare with another top-tier channel-independent method, PatchTST (Nie et al., 2022), and results show that we outperform it.

Specifically, we follow the experimental settings outlined in section 3.3.5.1, but with a specific focus on cases involving *small* lookback windows. This emphasis aims to reduce autoregressive correlation in data with smaller lookback windows, thereby increasing the dominance of multivariate correlations. We refer just a scenario as ***C**ross-**S**ectional **R**egression*[14] (CSR) in time series modelling (Reneau et al., 2023; Fama and French, 2020; Andrews, 2005) and (Wooldridge et al., 2016, Part 1).

Dataset. We evaluate our model on the synthetic dataset[15] generated in the ICML2023 Feature Programming paper (Reneau et al., 2023). Feature programming is an automated, programmable method for feature engineering. It produces a large number of predictive features from any given input time series. These generated features, termed *extended features*, are by construction highly correlated. The synthetic dataset, containing 44 extended features derived from the taxi dataset (see (Reneau et al., 2023, Section D.1) for the generation details), is thereby a strongly correlated multivariate time series dataset.

[14]From a technical standpoint, this setting doesn't fall under CSR, yet we refer to it as CSR to emphasize its utilization of one time slice for predicting the next. Cross-sectional regression is valuable for understanding the simultaneous impact of variables across different entities (i.e., multivariate correlations), providing insights that are complementary to those obtained from autoregressive time-series analysis.

[15]https://www.dropbox.com/scl/fo/cg49nid4ogmd6f6sdbwqz/h?rlkey=jydf3ay1fzeotl5ws2s0cjnlc&dl=0

Baseline. We mainly compare our performance with DLinear (Zeng et al., 2023) as it showed comparable performance in table 3.6. We also include an additional comparison with PatchTST (Nie et al., 2022) to further showcase our method's advantage in capturing multivariate correlations.

Setting. For both STaHop-Net and DLinear, we use the same hyperparameter setup as we used in the ETTh1 dataset. For PatchTST, we used the same hyperparameter setup as its original paper. We also conduct two ablation studies with varying lookback windows. We report the mean MAE, MSE and R2 score over 5 runs.

Results. Our results (table B.7) demonstrate that STanHop-Net consistently outperforms DLinear and PatchTST when multivariate correlations dominate the predictive information in input features. Importantly, our ablation studies show that increasing the lookback window size, which reduces the dominance of multivariate correlations, results in DLinear's performance becoming comparable to, rather than being consistently outperformed by, STanHop-Net. This explains why DLinear exhibits comparable performance to STanHop-Net in table 3.6, when the datasets are not dominated by multivariate correlations.

Channel-Independent and Channel-Dependent Methods. Recent studies have compared Channel-Independent (CI) (Zeng et al., 2023; Nie et al., 2022) and Channel-Dependent (CD) (Yu et al., 2024; Liu et al., 2023; Chen et al., 2023b; Luo and Wang, 2024) methods for time series prediction. CI methods, like **DLinear** and **PatchTST**, treat multivariate time series as multiple independent uni-variate tasks, often using simpler architectures like linear layers (Zeng et al., 2023). Conversely, CD methods leverage operations like convolution and channel-wise attention to capture inter-channel relationships and enhance performance.

Surprisingly, CI methods perform well with minimal model complexity and parameters, and sometimes even outperform CD methods, as noted in (Zeng et al., 2023; Nie et al., 2022). However, (Han et al., 2023) points out that while CD benefits from increased capacity, CI's simple architecture and limited samples in current benchmarks can significantly boost performance. Existing datasets are often too small for CD models to showcase their full potential, leading to potential underestimation of CD model effectiveness on popular benchmarks like ETTh1, ILI etc.

Table B.7. Comparison between DLinear, PatchTST and StanHop-Net on the synthetic dataset generated in (Reneau et al., 2023). This dataset is by construction a strongly correlated multivariate time series dataset. We report the mean MAE, MSE and R2 score over 5 runs. The $A \rightarrow B$ denotes the input horizon A and prediction horizon B. **CSR (Cross-Sectional Regression):** Focuses on using single time step information from multivariate time series for predictions. **Ablation1:** With a prediction horizon of 1, the lookback window is increasing, thereby the dominance of multivariate correlations is decreasing. **Ablation2:** With a prediction horizon of 2, the lookback window is increasing, thereby the dominance of multivariate correlations is decreasing. Our results aligns with our expectations: STanHop-Net uniformly beats DLinear (Zeng et al., 2023) and PatchTST (Nie et al., 2022) in the Cross-Sectional Regression (CSR) setting. Importantly, our ablation studies show that increasing the lookback window size, which reduces the dominance of multivariate correlations, results in DLinear's performance becoming comparable to, rather than being consistently outperformed by, STanHop-Net. This explains why DLinear exhibits comparable performance to STanHop-Net in table 3.6, when the datasets are not dominated by multivariate correlations.

lookback → horizon		DLinear			PatchTST			STanHop-Net		
		MSE	MAE	R^2	MSE	MAE	R^2	MSE	MAE	R^2
CSR	$1 \rightarrow 1$	0.896	0.615	0.256	0.332	**0.341**	0.728	**0.329**	0.375	**0.633**
	$1 \rightarrow 2$	1.193	0.794	0.001	0.460	**0.417**	0.625	**0.417**	0.428	**0.552**
	$1 \rightarrow 4$	1.211	0.806	-0.002	0.734	0.546	0.401	**0.592**	**0.522**	**0.383**
	$1 \rightarrow 8$	1.333	0.868	-0.100	1.227	0.744	-0.001	**0.812**	**0.636**	**0.182**
	$1 \rightarrow 16$	1.305	0.846	-0.069	1.755	0.947	-0.409	**1.028**	**0.734**	**-0.058**
Ablation1	$2 \rightarrow 1$	0.514	0.504	0.573	0.332	**0.355**	0.729	**0.328**	0.366	**0.710**
	$4 \rightarrow 1$	0.373	0.417	0.690	0.332	**0.357**	0.729	**0.328**	0.364	**0.712**
	$8 \rightarrow 1$	0.328	0.380	**0.727**	**0.322**	0.361	0.737	**0.327**	0.367	**0.715**
	$16 \rightarrow 1$	**0.319**	0.372	**0.736**	0.324	0.371	0.735	**0.323**	**0.361**	**0.717**
Ablation2	$2 \rightarrow 2$	0.771	0.632	0.359	0.458	0.426	**0.626**	**0.424**	**0.425**	0.630
	$4 \rightarrow 2$	0.423	0.439	**0.645**	0.460	0.429	**0.624**	**0.410**	**0.415**	0.643
	$8 \rightarrow 2$	0.647	0.441	0.646	0.439	0.429	**0.642**	**0.402**	**0.412**	0.655
	$16 \rightarrow 2$	**0.419**	0.435	**0.652**	0.447	0.446	0.635	**0.435**	**0.433**	**0.626**

B.3.5. Experiment Details

Here we present the details of experiments in the main text.

B.3.5.1. Experiment Details of Multivariate Time Series Predictions without Memory Enhancements.

Datasets. These datasets, commonly benchmarked in literature (Zhang and Yan, 2022; Wu et al., 2021; Zhou et al., 2021).

- **ETT (Electricity Transformer Temperature) (Zhou et al., 2021):** ETT records 2 years of data from two counties in China. We use two sub-datasets: ETTh1 (hourly) and ETTm1 (every 15 minutes). Each entry includes the "oil temperature" target and six power load features.
- **ECL (Electricity Consuming Load):** ECL records electricity consumption (Kwh) for 321 customers. Our version, sourced from (Zhou et al., 2021), covers hourly consumption over 2 years, targeting "MT 320".
- **WTH (Weather):** WTH records climatological data from approximately 1,600 U.S. sites between 2010 and 2013, measured hourly. Entries include the "wet bulb" target and 11 climate features.
- **ILI (Influenza-Like Illness):** ILI records weekly data on influenza-like illness (ILI) patients from the U.S. Centers for Disease Control and Prevention between 2002 and 2021. It depicts the ILI patient ratio against total patient count.
- **Traffic:** Traffic records hourly road occupancy rates from the California Department of Transportation, sourced from sensors on San Francisco Bay area freeways.

Training. We use Adam optimizer to minimize the MSE Loss. The coefficients of Adam optimizer, betas, are set to (0.9, 0.999). We continue training till there are $\texttt{Patience} = 3$ consecutive epochs where validation loss doesn't decrease or we reach 20 epochs. Finally, we evaluate our model on test set with the best checkpoint on validation set.

Table B.8. Dataset Sources

Dataset	URL
ETTh1 & ETTm1	`https://github.com/zhouhaoyi/ETDataset`
ECL	`https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014`
WTH	`https://www.ncei.noaa.gov/data/local-climatological-data/`
ILI	`https://archive.ics.uci.edu/ml/datasets/seismic-bumps`
Traffic	`https://www.kaggle.com/shrutimechlearn/churn-modelling`

Hyperparameters. For hyperparameter search, for each dataset, we conduct hyperparameter optimization using the "Sweep" feature of Weights and Biases (Biewald et al., 2020), with 200 iterations of random search for each setting to identify the optimal model configuration. The search space for all hyperparameters are reported in table B.9.

Table B.9. STanHop-Net hyperparameter space.

Parameter	Distribution
Patch size P	[6, 12, 24]
FeedForward dimension	[64, 128, 256]
Number of encoder layer	[1, 2, 3]
Number of pooling vectors	[10]
Number of heads	[4, 8]
Number of stacked STanHop blocks	[1]
Dropout	[0.1, 0.2, 0.3]
Learning rate	[5e-4, 1e-4, 1e-5, 1e-3]
Input length on ILI	[24, 36, 48, 60]
Input length on ETTm1	[24, 48, 96, 192, 288, 672]
Input length on other dataset	[24, 48, 96, 168, 336, 720]
Course level	[2, 4]
Weight decay	[0.0, 0.0005, 0.001]

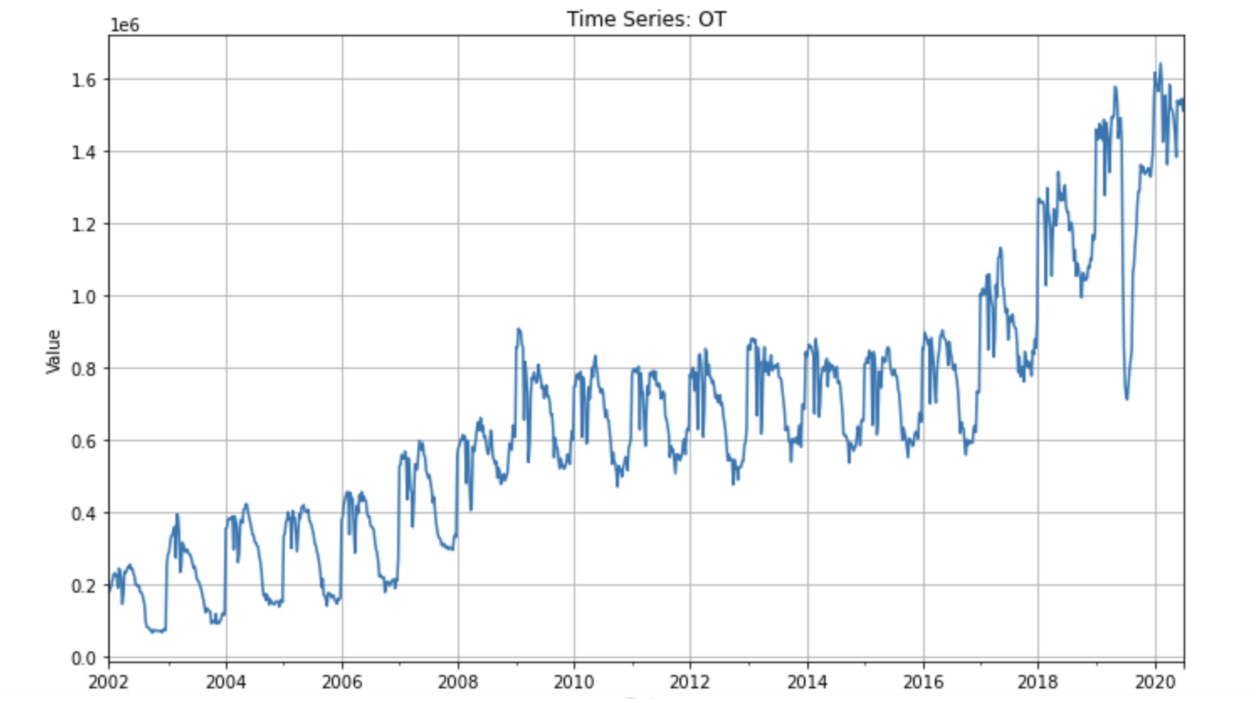


Figure B.16. The visualization of ILI dataset "OT" variate.

B.3.5.2. External Memory Plugin Experiment Details. The hyperparameter of the external memory plugin experiment can be found in table B.9. For ILI_OT, we set the input length as 24, feed forward dimension as 32 and hidden dimension as 64. For prediction horizon 60, we set the input length as 48, feed-forward dimension as 128 and hidden dimension as 256. For ETTh1, we use the same hyperparameter set found via random search in table 3.6.

For the "bad" external memory set intervals, we pick 40 and 200 for ILI_OT and ETTh1, which represents 40 timesteps (weeks) earlier and 200 timesteps (hours) earlier. For ILI dataset, we set the memory set size as 15 and for ETTh1, we set it as 20.

For **Case 3** (ETTh1), we select construct the external memory pattern with interval 168 timesteps earlier (equivalent to 1 week). For **Case 4** (ETTm1), we select construct the external memory pattern with interval 672 timesteps earlier (equivalent to 1 week).

B.3.6. Additional Theoretical Background

Remark B.3.1. Peters et al. (2019) provide a closed-form expression for α-EntMax as

$$\alpha\text{-}\mathrm{EntMax}(\boldsymbol{z}) = [(\alpha-1)\boldsymbol{z}-\tau(\boldsymbol{z})]^{\frac{1}{\alpha-1}}, \tag{B.3.45}$$

where we denote $[t]_+ \coloneqq \max\{t,0\}$, and τ is the threshold function $\mathbb{R}^M \to \mathbb{R}$ such that $\sum_{\mu=1}^{M}[(\alpha-1)\boldsymbol{z}-\tau(\boldsymbol{z})]^{\frac{1}{\alpha-1}} = 1$ satisfies the normalization condition of probability distribution.

B.4.

B.4.1. Limitations

One of the limitations of our work is that the futures dataset is proprietary and not open sourced due to the license.

B.4.2. Reproducibility

The code to reproduce the results of this paper is accessible at this link: `https://www.dropbox.com/scl/fo/9pw1wagwglelh2ftohy8h/AJXCZ_Bd4hh7qAM-SOlwEWk?rlkey=7xrqjetu317dcm61b3z486j4n`
`st=nbcgbit3&dl=0`

B.4.3. Background on Limit Order Book (LOB)

Global exchanges use matching engines to pair orders from bid and ask sides of market participants. The Limit Order Book is the essential data structure organizing these orders, reflecting market supply and demand. Three common order types exist: **1)** *Market orders* are requests to buy or sell a specified number of shares at the best available price, usually executed immediately. **2)** *Limit orders* are requests to buy or sell a specified number of shares at a specified price, often queued for matching due to price constraints. **3)** *Cancel orders* are requests to withdraw previously submitted limit orders.

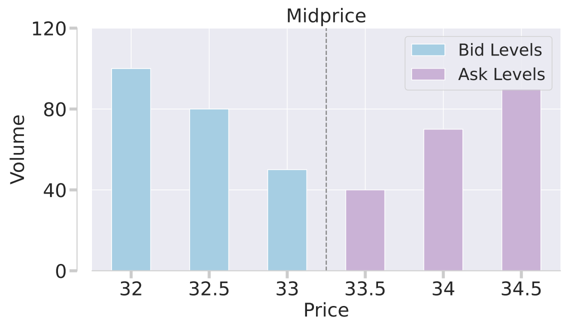


Figure B.17. **Visualizing Limit Order Book Data:** three *bid* and three *ask* levels of varying price and volume are shown, as well as the *mid-price*.

Table B.10. **CHF-2023 Features.** CHF-2023 consists of three feature sets: a set of 20 *Basic LOB* features, a set of 26 *Time-insensitive* features, and a set of 20 *Time-sensitive* features.

Feature Set	Definitions	Details
Basic LOB	$u_1 = \{P_i^{\text{bid}}, V_i^{\text{bid}}, P_i^{\text{ask}}, V_i^{\text{ask}}\}_{i=1}^{n}$	5-level LOB Data
Time-insensitive	$u_2 = \{(P_i^{\text{bid}} - P_i^{\text{ask}}), (P_i^{\text{bid}} + P_i^{\text{ask}})/2\}_{i=1}^{n}$	Spread & Mid-Price
	$u_3 = \{P_n^{\text{ask}} - P_1^{\text{ask}}, P_1^{\text{bid}} - P_n^{\text{bid}}, \lvert P_{i+1}^{\text{ask}} - P_i^{\text{ask}}\rvert, \lvert P_{i+1}^{\text{bid}} - P_i^{\text{bid}}\rvert\}_{i=1}^{n-1}$	Price Differences
	$u_4 = \left\{\frac{1}{n}\sum_{i=1}^{n} P_i^{\text{ask}}, \frac{1}{n}\sum_{i=1}^{n} P_i^{\text{bid}}, \frac{1}{n}\sum_{i=1}^{n} V_i^{\text{ask}}, \frac{1}{n}\sum_{i=1}^{n} V_i^{\text{bid}}\right\}$	Price & Volume Means
	$u_5 = \left\{\sum_{i=1}^{n}(P_i^{\text{ask}} - P_i^{\text{bid}}), \sum_{i=1}^{n}(V_i^{\text{ask}} - V_i^{\text{bid}})\right\}$	Accumulated Differences
Time-sensitive	$u_6 = \left\{\mathrm{d}P_i^{\text{ask}}/\mathrm{d}t, \mathrm{d}P_i^{\text{bid}}/\mathrm{d}t, \mathrm{d}V_i^{\text{ask}}/\mathrm{d}t, \mathrm{d}V_i^{\text{bid}}/\mathrm{d}t\right\}_{i=1}^{n}$	Price & Volume Derivation

B.4.4. Detailed Information of Datasets and Models in Benchmark

B.4.4.1. Models for Mid-price Trend Prediction.

- **MLP, LSTM, CNN1, CNN2, CNNLSTM.** Tsantekidis et al. (2017b) used Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM) to predict future mid-price movements. The same authors (Tsantekidis et al., 2017a) also proposed a Convolutional Neural Network (CNN) model (CNN1), which employs a standard CNN architecture with convolutional layers followed by fully connected layers. In 2020, the same research team (Tsantekidis et al., 2020) proposed another CNN model (CNN2) and a CNNLSTM model based on the described LSTM and CNN2. The key improvement of CNN2 is the use of causal convolutions with "full" padding, ensuring all convolutional layers produce outputs with the same number of time steps. This matches labels to their correct time steps and prevents future data from influencing past predictions. The CNNLSTM model merges CNN2 and LSTM, using CNN2 for feature extraction and passing the features to the LSTM module for classification. In our experiments, we adjusted the CNN kernel size to match the different number of features in each dataset.

- **DAIN.** Passalis et al. (2019) introduce Deep Adaptive Input Normalization (DAIN) for time series forecasting. The main innovation of the method is learning to adaptively normalize input data during model training via two feed-forward layers, an adaptive shifting layer and scaling layer, instead of using fixed schemes like z-score normalization. Besides, they use a gating layer to suppress irrelevant features. DAIN explores three possible neural network architectures: MLP, CNN, and RNN. We choose the MLP architecture as it demonstrates the highest empirical performance.
- **CTABL, BINCTABL.** Tran et al. (2018) propose the Temporal-Attention-Augmented Bilinear Layer (TABL) model. The Bilinear Layer (BL) performs two linear transformations on the input data along the feature and temporal dimensions to capture how stock prices interact at a given point in time and how prices at an index progress over time. However, within a BL, it is unclear how different time instance representations interact. TABL addresses this by incorporating a temporal attention mechanism into the BL model to learn the importance of each time instance relative to others. The input time series is first passed through the feature dimension transformation, then through the temporal attention mechanism, and finally through the temporal dimension transformation. In our experiments, we use the C(TABL) variant of TABL for its superior empirical performance over A(TABL) and B(TABL). Tran et al. (2021) improve upon TABL with BINCTABL, which incorporates a Bilinear Normalization (BiN) strategy that normalizes the data along the temporal and feature dimensions with learnable parameters to scale the normalization. The authors compare BiN as a simpler and more intuitive approach compared to DAIN when using TABL networks.

- **DEEPLOB.** Zhang et al. (2019) propose Deep Convolutional Neural Networks for Limit Order Books (DeepLOB). The authors employ a mid-price-based smooth data labeling method to reduce noise and eliminate minor oscillations. DEEPLOB combines convolutional layers and an Inception Module to extract features from the noisy financial data, then uses a Long Short-Term Memory (LSTM) layer to capture longer time dependencies among the extracted features. In our experiments, we adjust the kernel size of the last convolution layer for different datasets.
- **DEEPLOBATT.** Zhang and Zohren (2021) introduce DeepLOB-Attention, an encoder-decoder model built upon their previous work, DEEPLOB. DEEPLOBATT utilizes DEEPLOB as the encoder and an Attention model (Luong et al., 2015) as the decoder. The authors investigated both Attention and Seq2Seq as the decoder in their paper. Since DEEPLOBATT outperforms DEEPLOB-Seq2Seq in nearly all experiments, we choose the DEEPLOBATT model in our experiments and adjust the kernel size of the convolutional block for different datasets according to the feature dimensions.
- **TLONBoF.** Passalis et al. (2020) propose Temporal Logistic Neural Bag-of-Features (TLo-NBoF) (2020), a refined Bag-of-Features (BoF) formulation to better capture the dynamics of time series data compared to existing BoF methods. They achieved this by introducing a novel adaptive scaling mechanism and replacing the Gaussian density estimation of regular BoF with a logistic kernel. The input time series is first passed through a 1-D convolution layer to extract relationships between time instances. Then, the TLONBoF formulation uses a learnable logistic kernel and codebook to aggregate the feature vectors into short-term, mid-term, and long-term histograms that capture the overall behaviors of the input time series. This adaptive method removes the need for sophisticated initialization methods and facilitates smoother hyperparameter tuning.
- **DLA.** Guo and Chen (2022) propose a Dual-Stage Temporal Attention-Based Deep Learning Architecture (DLA). It uses a dual-stage temporal attention mechanism to

repeatedly highlight the most important time instances in input time series data. In the first stage, temporal attention is performed on the input data to learn the importance of each time instance relative to the others. The output of the attention is passed through a stacked Gated Recurrent Unit (GRU) network to further enhance the representational state of the input. In the second stage, temporal attention weights are adaptively assigned to the hidden states of the GRU network. The model's performance is benchmarked against other models in the literature, such as CTABL (Tran et al., 2018), DEEPLOB (Zhang et al., 2019), and TLONBoF (Passalis et al., 2020).

- **TRANSLOB.** Wallbridge (2020) proposes TransLOB, a new model architecture for LOB data that uses Transformer blocks (Vaswani et al., 2023). The TRANSLOB architecture consists of a causal convolution module and a causal transformer block to stay consistent with the nature of time-series data. The convolutional module comprises 5 1-D convolution layers with increasing dilation to capture relationships between short-term and long-term time instances. The transformer block includes 2 masked self-attention encoders to determine important time instance representations. The performance of TRANSLOB is benchmarked against other state-of-the-art models such as CTABL, DEEPLOB, and CNN-LSTM.

B.4.4.2. Models for Mid-price Forecasting. Besides the LOB Models including MLP, CNN1, LSTM, BINCTABL, DAIN, and TRANSLOB mentioned above, we also include 4 state-of-the-art time series forecasting models.

- **PatchTST.** Nie et al. proposes PatchTST (Nie et al., 2022), a multivariate time series forecasting model based on vanilla Transformer architecture. It has two major novelties. First, it uses the same model to predict each variate of a multivariate time series model independently, making it technically a univariate time series model. Second, to better capture time series' local pattern, it models the time series in patch-wise (each patch includes multiple consecutive time steps) rather than timestep-wise.

- **iTransformer.** To better capture variate-centric information rather than only temporal dimension, Liu et al. (2023) proposes iTransformer. It uses the vanilla transformer components in a novel way where it first embeds each series of a multivariate time series into a token and then applies attention across all the tokens representing different series.
- **DLinear.** Besides transformer-based models, linear models are also showing top performance in time series forecasting. Zeng et al. (2023) questions Transformer architecture's capability to properly capture temporal correlations in time series data. They propose a simple baseline architecture, DLinear. It is a one-layer linear model with a decomposition component. It first decomposes the raw time series input into a trend component and a seasonal component. Then, two separate one-layer linear layers are applied on these two components. Lastly, two components sum up in to the final prediction. DLinear outperformans most of the Transformer-based time series prediction model and achieve top-tier time series forecasting performance.
- **TimeMixer.** Along the line of better modeling temporal correlations from the time series data, Wang et al. (2024a) proposes TimeMixer. It is a linear-layer-based model. Its major novelty is to first preprocess the raw input time series into multiple time series of different resolutions by downsampling. Then, they mix pairs of time series of different resolutions by adding one of the time series to a transformation of the other. The transformation consists of two linear layers and an intermediate GELU activation function. Lastly, they use a linear layer to regress from the mixed time series to produce the final prediction.

B.4.5. Hyperparameters

table B.11, table B.12, and table B.13 include the hyperparameters we use in MPTP and MPRF. For Mid-Price Trend Prediction (MPTP), we adhere to the hyperparameters specified in the original papers for the models. We apply a patience of 8 for MPTP on the FI dataset and a patience of 3 for the CHF dataset. For Mid-Price Return Forecasting (MPRF), we perform a grid search for the batch size, including {32, 64, 128} and the learning rate, including {0.01, 0.001, 0.0001, 0.00001} on horizon 5. We employ a patience of 15 for the FI dataset and a patience of 2 for the CHF dataset.

Table B.11. Hyperparameters for Mid-Price Trend Prediction on the FI-2010 Dataset. Dashes mean the corresponding module does not exist in the model architecture.

	FI						
Model	**Learning Rate**	**Optimizer**	**Batch Size**	**Epochs**	**Dropout**	**MLP Hidden**	**RNN Hidden**
LSTM	0.001	Adam	32	100	0	64	40
MLP	0.001	Adam	64	100	0.1	128	-
CNN1	0.0001	Adam	64	100	-	32	-
CTABL	0.01	Adam	256	200	0.1	-	-
DAIN	0.0001	RMSprop	32	100	0.5	512	-
DEEPLOB	0.01	Adam	32	100	-	-	64
CNNLSTM	0.001	RMSprop	32	100	0.1	32	32
CNN2	0.001	RMSprop	32	100	-	32	-
TRANSLOB	0.0001	Adam	32	150	0.1	64	-
TLONBoF	0.0001	Adam	128	100	0.5	512	-
BINCTABL	0.001	Adam	128	200	0.1	-	-
DEEPLOBATT	0.001	Adam	32	100	-	-	64
DLA	0.01	Adam	256	100	0.5	-	100

Table B.12. Hyperparameters for Mid-Price Trend Prediction on CHF-2023 Dataset

	CHF						
Model	**Learning Rate**	**Optimizer**	**Batch Size**	**Epochs**	**Dropout**	**MLP Hidden**	**RNN Hidden**
LSTM	0.001	Adam	1024	100	0	64	40
MLP	0.001	Adam	2048	100	0.1	128	-
CNN1	0.0001	Adam	1024	100	-	32	-
CTABL	0.01	Adam	2048	200	0.1	-	-
DAIN	0.0001	RMSprop	2048	100	0.5	512	-
DEEPLOB	0.01	Adam	1024	100	-	-	64
CNNLSTM	0.001	RMSprop	2048	100	0.1	32	32
CNN2	0.001	RMSprop	1024	100	32	-	-
TRANSLOB	0.001	Adam	2048	150	0.1	64	-
TLONBoF	0.0001	Adam	1024	100	0.5	512	-
BINCTABL	0.001	Adam	1024	200	0.1	-	-
DEEPLOBATT	0.001	Adam	1024	100	-	-	64
DLA	0.001	Adam	1024	100	0.5	-	100

Table B.13. Hyperparameters for Mid-Price Return Forecasting on the FI-2010 Dataset

				FI				
Model	**Learning Rate**	**Optimizer**	**Batch Size**	**Epochs**	**Dropout**	**MLP Hidden**	**RNN Hidden**	**Transformer Hidden**
MLP	0.0001	Adam	32	50	0.1	128	-	-
LSTM	0.01	Adam	64	50	0.2	64	32	-
CNN1	0.0001	Adam	32	50	-	-	32	-
BINCTABL	0.0001	Adam	64	50	0.1	128	32	-
DAIN	0.0001	Adam	128	50	0.5	-	-	-
TRANSLOB	0.0001	Adam	128	50	0.1	64	-	60
PATCHTST	0.0001	Adam	128	50	0.3	-	-	128
DLINEAR	0.0001	Adam	128	50	-	-	-	-
ITRANSFORMER	0.0001	Adam	128	50	0.1	-	-	512
TIMEMIXER	0.0001	Adam	128	50	0.1	16	-	-

B.4.6. Additional MPRF Results on Multivariate Synthetic Datasets

To further demonstrate CVML's ability to capture cross-variate correlation, we generate a synthetic dataset and test CVML's performance on it. Compared to real datasets with latent cross-variate correlations such as FI-2010 and CHF-2023, the synthetic dataset has well-defined cross-variate correlations specified in the data generation code. The synthetic dataset's target is a synthetic electricity price, the other variates are electricity load, electricity production and temperature. We generate the data in a way that the temperature affects the electricity load (e.g. cold and hot temperature increase the electricity load), the electricity load and the production affect the electricity price (e.g. higher load increases the price and higher production lower the price).

B.4.6.1. Synthetic Time Series Data Generation. The synthetic dataset consists of multiple time series components with complex relationships and non-linear interactions. The data generation process follows a hierarchical structure where intermediate variables influence the final target variable (electricity price). There are totally six varieties: electricity price, load, production, temperature, supply_margin, price_volatility. These varieties have cross-variate correlations. For

Table B.14. **Time Series Model Performance with and without CVML** on the Synthetic Dataset using basic LOB features.

	MSE (↓)			Corr (↑)			R^2 (↑)		
Model	K=1	K=5	K=10	K=1	K=5	K=10	K=1	K=5	K=10
PatchTST-CVML	**0.807**	**0.7538**	**0.7589**	**0.4298**	**0.485**	**0.48**	**0.1818**	**0.2346**	**0.2296**
PatchTST	1.0508	1.0441	1.0382	0.3907	0.3915	0.3991	-0.0654	-0.0602	-0.054
DLinear-CVML	**0.7825**	**0.7733**	**0.8743**	**0.4585**	**0.4639**	**0.3364**	**0.2067**	**0.2147**	**0.1124**
DLinear	0.8861	0.8847	0.8868	0.3188	0.3209	0.3159	0.1016	0.1017	0.0997
iTransformer-CVML	**0.8544**	**0.8183**	**0.8179**	**0.4011**	**0.4235**	**0.4317**	**0.1337**	**0.1691**	**0.1697**
iTransformer	1.1667	1.161	1.1706	0.339	0.3627	0.3628	-0.1829	-0.1789	-0.1884
TimeMixer-CVML	**0.7469**	0.7704	**0.7539**	**0.493**	0.4693	**0.4865**	**0.2427**	0.2177	**0.2347**
TimeMixer	0.7622	**0.7596**	0.7781	0.4774	**0.4782**	0.4586	0.2272	**0.2287**	0.2101

example, an increasing load of electricity leads to an increasing electricity price. An increasing production of electricity leads to a decreasing electricity price. When the temperature is too high or too cold, the load will increase. We introduce the detailed generation process of each variate as follows.

Temperature Generation: Let $t \in \{0, 1, ..., n-1\}$ represent the time index for n hourly observations. The temperature time series T_t is generated as:

$$T_t = 20 + 10\sin(\text{seasonal\_cycle}) + 5\sin(\text{daily\_cycle}) + \epsilon_T + \delta_H - \delta_C \tag{B.4.1}$$

where:

- $\text{seasonal\_cycle} = \frac{2\pi t}{24 \times 365.25}$
- $\text{daily\_cycle} = \frac{2\pi t}{24}$
- $\epsilon_T \sim \mathcal{N}(0, 2^2)$ represents random fluctuations
- $\delta_H \sim \text{Bernoulli}(0.05) \times 8$ represents heat waves
- $\delta_C \sim \text{Bernoulli}(0.05) \times 8$ represents cold snaps

Load Generation: The electricity load L_t is modeled as (including base load, daily pattern, AC usage, heating, seasonal pattern, weekday effect, and a random noise):

$$\begin{aligned} L_t = 1000 + 200\sin(\text{daily\_cycle}) + 150\mathbb{1}_{T_t>25} + 100\mathbb{1}_{T_t<10} \\ + 100\sin(\text{seasonal\_cycle}) + 50\mathbb{1}_{\text{weekday}} + \epsilon_L \end{aligned} \tag{B.4.2}$$

where $\epsilon_L \sim \mathcal{N}(0, 30^2)$ and $\mathbb{1}$ represents the indicator function.

Production Generation: The production capacity P_t is generated through multiple components:

$$P_t = (1.1L_t + 100\sin(\text{daily\_cycle}) + \epsilon_P) \times M_t \times O_t \tag{B.4.3}$$

where:

- $\epsilon_P \sim \mathcal{N}(0, 30^2)$
- $M_t \sim$ Categorical$([1.0, 0.7], [0.9, 0.1])$ represents maintenance periods
- $O_t \sim$ Categorical$([1.0, 0.3], [0.98, 0.02])$ represents outages

Electricity Price Generation (Target Variable): The final price Y_t is generated through a complex interaction of components:

$$\begin{aligned} Y_t = ((50 + 0.08L_t - 0.04P_t + 0.7(T_t - 20)^2 \\ + 15\sin(\text{daily\_cycle}) + 10\sin(\text{seasonal\_cycle})) \\ \times R_t \times S_t \times D_t + \epsilon_Y) \end{aligned} \tag{B.4.4}$$

where:

- $R_t \sim$ Categorical$([1.0, 1.5, 0.7], [0.7, 0.15, 0.15])$ represents regime changes
- S_t represents price spikes triggered by conditions:

$$S_t = \begin{cases} \sim \text{Categorical}([1.0, 2.5], [0.7, 0.3]) & \text{if } C_t = 1 \\ 1.0 & \text{otherwise} \end{cases}$$

where $C_t = 1$ if any of the following conditions are met:

- $L_t/P_t > 0.9$ (high demand relative to production)
- $T_t > 30$ (very hot weather)
- $T_t < 0$ (very cold weather)
- $P_t/P_{\text{base},t} < 0.5$ (significant production issues)

- $D_t \sim \text{Categorical}([1.0, 0.4], [0.97, 0.03])$ represents sudden price drops
- $\epsilon_Y \sim \mathcal{N}(0, (0.3 \times 100)^2)$ represents price noise

Derived Features. Additional features are computed from the primary variables:

$$\text{supply\_margin}_t = \frac{P_t - L_t}{L_t}$$

$$\text{price\_volatility}_t = \sigma(\{Y_{t-23}, ..., Y_t\})$$

where σ represents the rolling standard deviation over a 24-hour window.

As shown in table B.15, this synthetic time series dataset, although it was defined in a specific domain (electricity) for better interpretability, includes generic patterns that could be found across different domains of multivariate time series. Specifically, it includes the following patterns: multiple seasonality (e.g. weekly/monthly sales cycles, weekday/weekend differences in web traffic), non-linear relationships, temporary anomalies and recoveries (e.g. electricity outage), periods of high volatility followed by calmer periods (e.g. viral content spread on social media), multi-factor interactions (e.g. inventory-price-demand relationships in supply chain) and stochastic components that mirror real-world randomness. CVML's good performance on this synthetic dataset shows meaningful values for the broader time series forecasting field.

Table B.15. **Time Series Model Performance with and without CVML** on the synthetic electricity price dataset. The % column indicates the percentage improvement from adding CVML.

	MSE (↓)						Corr (↑)						R^2 (↑)					
Model	K=1	K=2	K=3	K=5	K=10	%	K=1	K=2	K=3	K=5	K=10	%	K=1	K=2	K=3	K=5	K=10	%
PatchTST-CVML	**0.653**	**1.071**	**1.370**	**1.893**	**2.646**	3.1	**0.070**	**0.113**	**0.165**	**0.191**	**0.241**	86.2	**0.005**	**0.007**	**0.028**	**0.033**	**0.054**	958.3
PatchTST	0.654	1.077	1.406	1.949	2.795		0.081	0.082	0.079	0.092	0.085		0.003	0.002	0.002	0.004	0.001	
DLinear-CVML	**0.650**	**1.042**	**1.352**	**1.796**	**2.548**	5.9	**0.104**	**0.192**	**0.205**	**0.291**	**0.313**	174.9	**0.010**	**0.035**	**0.040**	**0.082**	**0.089**	814.3
DLinear	0.652	1.073	1.402	1.945	2.782		0.080	0.081	0.074	0.083	0.084		0.006	0.006	0.005	0.006	0.005	
iTransformer-CVML	**0.654**	1.084	**1.402**	**2.002**	**2.649**	14.6	**0.054**	**0.070**	**0.088**	**0.121**	**0.249**	140.5	**0.002**	**-0.005**	**0.005**	**-0.024**	**0.053**	104.8
iTransformer	0.683	**1.183**	1.582	2.279	3.401		0.045	0.045	0.033	0.063	0.056		-0.041	-0.096	-0.123	-0.165	-0.216	
TimeMixer-CVML	**0.642**	**1.033**	**1.329**	**1.807**	**2.494**	4.6	**0.160**	**0.221**	**0.257**	**0.298**	**0.353**	61.1	**0.022**	**0.043**	**0.056**	**0.076**	**0.109**	194.2
TimeMixer	0.657	1.075	1.394	1.888	2.643		0.083	0.110	0.135	0.201	0.271		-0.001	0.004	0.011	0.035	0.055	

B.4.7. Additional MPTP results on Bitcoin Dataset

We conduct further MPTP benchmark experiments on a public Bitcoin LOB datasets [16]. Our conclusion from benchmarking on the FI and CHF datasets is still valid on the crypto dataset, which is that there is limited generalizability of current LOB model architectures and the underlying characteristics of LOB data from different assets are different. As shown in table B.16, the ranking of the LOB model performance is also different from the one on the FI-2010 dataset and CHF-2023 dataset.

B.4.8. Full MPTP Results on Different Feature Sets

table B.17 and table B.18 include the full results for LOB models on the MPTP task on the basic feature set and the basic+time_insensitive feature set. They support fig. 3.13.

B.4.9. Computational Resources

For all the experiments, we use a computation cluster with 1 node of 8 Nvidia A100 GPUs and 3 nodes of 8 Nvidia 2080Ti GPUs. The RAM is 1TB per node and there are 96 CPU cores per node.

[16]https://www.kaggle.com/datasets/siavashraz/bitcoin%2Dperpetualbtcusdtp%2Dlimit%2Dorder%2Dbook%2Ddata/data

Table B.16. **Mid-price Trend Prediction F1 Scores (Mean&Standard Deviation) on Basic LOB data + time-insensitive features + time-sensitive features**. We provide the F1 scores on mid-price trend predictions across horizons {1,2,3,5,10} on for the Bitcoin LOB dataset. The model performance ranking is not consistent with that on the FI-2010 and CHF-2023 datasets , further confirming that models' prediction power for one asset is not automatically transferable to another asset.

Model	K=1	K=2	K=3	K=5	K=10	avg
MLP	92.4 (0.2)	93.2 (0.1)	93.8 (0.1)	94.3 (0.1)	95.3 (0.0)	93.8
LSTM	86.2 (2.8)	94.7 (0.2)	95.2 (0.5)	96.3 (0.1)	97.2 (0.1)	94.0
CNN1	94.5 (0.7)	96.3 (0.2)	96.9 (0.2)	96.9 (0.1)	97.7 (0.1)	96.5
CTABL	53.9 (0.2)	64.3 (0.0)	72.1 (0.2)	80.9 (0.2)	92.3 (0.2)	72.7
DeepLOB	**97.9** (0.1)	**98.4** (0.0)	98.5 (0.1)	98.1 (0.0)	98.3 (0.0)	**98.3**
DAIN	34.7 (0.3)	53.3 (0.4)	66.9 (0.6)	79.8 (0.1)	87.1 (0.1)	64.4
CNN-LSTM	97.2 (0.1)	97.7 (0.2)	97.9 (0.1)	97.7 (0.1)	98.1 (0.0)	97.7
CNN2	97.1 (0.1)	98.1 (0.0)	98.2 (0.1)	97.9 (0.1)	98.1 (0.0)	97.9
TransLOB	95.9 (0.8)	98.2 (0.2)	98.3 (0.2)	98.0 (0.2)	98.4 (0.1)	97.8
TLONBOF	56.5 (1.0)	70.0 (0.9)	78.1 (0.7)	88.2 (0.3)	94.8 (0.1)	77.5
BinCTABL	50.5 (0.4)	60.3 (0.5)	65.1 (1.7)	73.1 (0.2)	90.4 (0.4)	67.9
DeepLOBAtt	97.6 (0.2)	98.1 (0.5)	**98.7** (0.2)	**98.4** (0.1)	**98.5** (0.1)	**98.3**
DLA	57.1 (2.0)	69.2 (0.3)	74.4 (0.4)	81.0 (0.2)	89.0 (0.2)	74.2
avg	77.8	84.0	87.2	90.8	95.0	87.0

Table B.17. **Mid-price Trend Prediction F1 Scores (Mean&Standard Deviation) on Basic LOB data**. 13 models relevant in the literature are benchmarked to compare their F1 scores on across horizons {1,2,3,5,10} on the basic LOB feature set of the FI-2010 and CHF-2023 datasets. For each horizon, the best model is bolded, and the next best model is underlined.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	35.014 (4.545)	42.242 (2.038)	46.759 (0.918)	46.061 (0.904)	47.210 (2.887)	38.788 (1.487)	44.328 (0.760)	45.958 (0.518)	45.650 (0.210)	37.952 (1.522)
LSTM	64.809 (1.377)	57.882 (0.613)	65.205 (0.198)	66.898 (0.747)	58.850 (0.926)	47.280 (0.459)	49.762 (0.210)	50.710 (0.265)	48.411 (0.202)	43.972 (0.522)
CNN1	27.608 (0.000)	30.815 (0.096)	54.783 (4.891)	62.882 (0.828)	63.955 (0.705)	42.357 (1.899)	47.936 (0.912)	49.203 (0.587)	47.676 (0.390)	43.661 (1.056)
CTABL	67.353 (0.585)	60.531 (0.213)	66.186 (0.198)	70.736 (0.367)	71.244 (1.092)	43.902 (2.157)	45.986 (0.675)	46.429 (1.193)	46.432 (0.417)	43.867 (0.879)
DEEPLOB	70.018 (1.160)	62.357 (0.577)	70.403 (1.010)	75.924 (0.089)	77.551 (0.285)	45.926 (2.140)	48.324 (1.646)	49.166 (1.854)	47.465 (1.230)	41.233 (1.290)
DAIN	79.767 (0.050)	70.202 (0.110)	79.851 (0.036)	87.041 (0.029)	91.816 (0.075)	45.130 (0.759)	49.014 (0.274)	49.290 (0.211)	46.928 (0.211)	40.445 (0.529)
CNNLSTM	27.620 (0.000)	29.656 (1.210)	34.060 (2.43)	44.248 (10.898)	54.872 (6.768)	45.067 (3.705)	47.342 (1.175)	48.556 (2.505)	47.740 (1.173)	**46.209 (0.198)**
CNN2	27.620 (0.000)	27.914 (1.978)	33.315 (1.909)	50.086 (11.537)	61.930 (5.501)	42.357 (1.899)	47.936 (0.912)	49.203 (0.587)	47.676 (0.390)	43.661 (1.056)
TRANSLOB	51.020 (12.632)	40.976 (8.987)	50.876 (10.709)	60.748 (1.541)	59.715 (0.859)	**49.189 (1.454)**	**50.379 (0.618)**	50.739 (0.323)	48.690 (0.426)	45.931 (0.273)
TLONBoF	37.549 (1.759)	40.181 (3.560)	41.551 (2.348)	48.991 (1.371)	60.702 (6.665)	43.961 (1.370)	45.935 (0.839)	46.556 (0.382)	45.935 (0.970)	43.544 (1.084)
BINCTABL	**80.985 (0.055)**	**71.168 (0.371)**	**80.734 (0.044)**	**87.553 (0.037)**	**92.074 (0.042)**	44.800 (0.608)	46.041 (0.647)	46.063 (0.479)	45.759 (0.182)	44.124 (0.263)
DEEPLOBATT	69.435 (0.011)	62.936 (0.015)	59.100 (0.203)	73.083 (0.007)	77.028 (0.020)	47.955 (1.213)	50.176 (0.993)	**50.744 (0.332)**	**49.104 (0.231)**	43.938 (1.4981)
DLA	76.410 (0.028)	65.966 (0.012)	77.858 (0.006)	85.713 (0.005)	51.617 (0.010)	44.136 (5.060)	48.060 (3.583)	47.797 (3.055)	46.992 (0.337)	38.347 (1.026)
Mean	55.016	58.290	65.311	69.695	72.525	46.660	48.685	49.246	47.398	43.357

B.4.10. Full CVML Results on MPRF

table B.19 includes the full mean and standard deviation results for the time series models using CVML. table B.20 contains ablation results on CVML using its two ablated variants, CVML-abla1

Table B.18. **Mid-price Trend Prediction F1 Scores (Mean&Standard Deviation) on Basic LOB data + time-insensitive features**. The F1 scores across horizons {1,2,3,5,10} for the FI-2010 and CHF-2023 datasets on the basic LOB + time-insensitive feature set of the FI-2010 and CHF-2023 datasets. For each horizon, the best model is bolded, and the next best model is underlined.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	45.849 (1.995)	44.475 (1.324)	47.855 (0.570)	44.487 (1.209)	49.629 (0.564)	40.343 (1.641)	44.818 (2.101)	45.975 (0.924)	45.336 (0.712)	37.242 (0.679)
LSTM	74.261 (0.051)	65.072 (0.223)	72.295 (0.579)	76.537 (1.496)	60.141 (1.992)	48.779 (0.556)	50.464 (0.495)	51.178 (0.154)	48.805 (0.108)	45.511 (1.654)
CNN1	60.554 (13.194)	61.928 (0.308)	70.664 (0.174)	77.906 (0.520)	80.107 (1.099)	47.367 (1.303)	49.236 (0.847)	50.400 (0.461)	48.316 (0.376)	44.365 (1.459)
CTABL	77.336 (0.107)	68.265 (0.483)	76.910 (0.217)	82.573 (0.388)	84.356 (0.203)	44.012 (2.707)	46.187 (1.791)	45.916 (0.567)	45.667 (0.490)	43.930 (0.634)
DEEPLOB	79.047 (0.075)	69.773 (0.216)	78.797 (0.087)	85.249 (0.090	88.471 (0.177)	46.316 (2.756)	47.581 (1.547)	49.692 (1.162)	47.435 (1.381)	41.116 (2.251)
DAIN	79.935 (0.028)	**80.190 (0.102)**	79.905 (0.072)	87.151 (0.022)	92.222 (0.024)	45.773 (0.391)	48.475 (0.214)	48.607 (0.281)	46.767 (0.231)	41.913 (1.216)
CNNLSTM	38.279 (12.537)	36.006 (1.015)	36.181 (0.456)	36.845 (0.342)	76.312 (1.819)	44.696 (0.908)	48.517 (0.877)	49.729 (0.439)	47.392 (0.448)	45.782 (0.610)
CNN2	28.558 (0.666)	33.529 (1.818)	36.469 (0.541)	39.401 (2.606)	66.267 (7.428)	45.365 (1.145)	48.162 (0.672)	49.027 (0.372)	46.843 (1.082)	44.855 (0.766)
TRANSLOB	69.436 (5.014)	59.600 (3.084)	70.478 (0.758)	75.294 (2.101)	71.545 (6.861)	**51.213(0.655)**	**52.294 (0.202)**	**51.752 (0.293)**	48.984 (0.269)	**46.418 (0.540)**
TLONBoF	43.105 (8.616)	40.792 (1.035)	51.367 (1.609)	59.992 (3.051)	65.607 (3.224)	49.888 (0.896)	49.506 (0.680)	49.426 (0.288)	47.120 (0.375)	44.761 (0.195)
BINCTABL	**81.024 (0.034)**	71.547 (0.265)	**80.896 (0.016)**	**87.849 (0.055)**	**92.541 (0.081)**	46.309 (1.214)	46.844 (0.981)	47.218 (1.409)	46.388 (0.567)	44.313 (0.295)
DEEPLOBATT	75.652 (0.013)	58.878 (0.111)	68.990 (0.103)	67.337 (0.119)	56.923 (0.034)	48.388 (0.578)	50.737 (0.260)	51.192 (0.567)	**49.229 (0.214)**	43.939 (0.886)
DLA	77.143 (0.004)	67.721 (0.007)	78.241 (0.002)	85.417 (0.002)	58.703 (0.005)	48.128 (0.516)	50.080 (0.309)	50.092 (0.398)	47.888 (0.734)	39.499 (0.818)
Mean	63.860	58.290	65.311	69.695	72.525	46.660	48.685	49.254	47.398	43.357

and CVML-abla2. CVML-abla1 focuses exclusively on cross-variate correlations ang ignores temporal relationships by performing convolution with a kernel size of 1. CVML-abla2 uses depthwise convolution. We set the number of groups in the convolution equal to the number of input channels, which allows the model to focus exclusively on temporal correlations and ignore cross-variate correlations. The results demonstrate how these ablated versions compare to the original CVML design, highlighting the architecture's dual-correlation approach and its impact on model performance.

Table B.19. **Full MPRF Results for table 3.12 with Mean and Std on Basic LOB data with CVML**

Model	Metric	K=1	K=2	K=3	K=5	K=10
PatchTST	MSE	0.653 (0.004)	1.071 (0.016)	1.370 (0.024)	1.893 (0.043)	2.646 (0.018)
	Corr	0.070 (0.036)	0.113 (0.037)	0.165 (0.061)	0.191 (0.045)	0.241 (0.011)
	R^2	0.005 (0.005)	0.007 (0.015)	0.028 (0.017)	0.033 (0.022)	0.054 (0.007)
DLinear	MSE	0.650 (0.003)	1.042 (0.004)	1.352 (0.011)	1.796 (0.008)	2.548 (0.028)
	Corr	0.104 (0.021)	0.192 (0.011)	0.205 (0.021)	0.291 (0.007)	0.313 (0.024)
	R^2	0.010 (0.004)	0.035 (0.004)	0.040 (0.007)	0.082 (0.004)	0.089 (0.010)
iTransformer	MSE	0.654 (0.002)	1.084 (0.014)	1.402 (0.007)	2.002 (0.046)	2.649 (0.014)
	Corr	0.054 (0.039)	0.070 (0.007)	0.088 (0.014)	0.121 (0.027)	0.249 (0.018)
	R^2	0.002 (0.024)	-0.005 (0.013)	0.005 (0.005)	-0.024 (0.024)	0.053 (0.005)
TimeMixer	MSE	0.642 (0.004)	1.033 (0.006)	1.329 (0.007)	1.807 (0.029)	2.494 (0.056)
	Corr	0.160 (0.009)	0.221 (0.009)	0.257 (0.007)	0.298 (0.009)	0.353 (0.008)
	R^2	0.022 (0.006)	0.043 (0.005)	0.056 (0.005)	0.076 (0.015)	0.109 (0.020)

Table B.20. **MPRF Results on Ablated CVMLs**. The comparison to CVML's results in table 3.12 demonstrates CVML's ability to capture cross-variate correlations and temporal correlations.

		CVML-abla1 (Cross-variate)					CVML-abla2 (Temporal)				
Model	Metric	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
PatchTST	MSE	0.6565	1.1168	1.3936	1.9292	2.7402	0.6531	1.0763	1.4063	1.9501	2.7983
	Corr	0.0102	0.0572	0.1080	0.1204	0.1443	0.0826	0.0813	0.0784	0.0909	0.0835
	R^2	-0.0005	-0.0349	0.0107	0.0138	0.0206	0.0046	0.0026	0.0016	0.0031	-0.0002
DLinear	MSE	0.6501	1.0681	1.3680	1.842	0.25945	0.6526	1.0727	1.4016	1.9446	2.7819
	Corr	0.0789	0.1031	0.1750	0.243	0.2710	0.0773	0.0810	0.0744	0.0814	0.0860
	R^2	0.009	0.0102	0.0288	0.0283	0.0726	0.0055	0.0060	0.0048	0.0059	0.0057
iTransformer	MSE	0.7341	1.272	1.488	1.9398	3.1331	0.6967	1.1605	1.5865	2.2183	3.4300
	Corr	0.0372	0.061	0.069	0.1178	0.1000	0.0368	0.0460	0.0280	0.0646	0.0517
	R^2	-0.1188	-0.179	-0.056	0.0084	-0.1199	-0.0617	-0.0754	-0.1263	-0.1340	-0.2260
TimeMixer	MSE	0.637	1.038	1.329	1.806	2.508	0.6558	1.0791	1.4006	1.9031	2.6729
	Corr	0.180	0.2142	0.257	0.296	0.342	0.0837	0.1047	0.1186	0.2114	0.2617
	R^2	0.030	0.039	0.057	0.077	0.104	0.0006	0.0000	0.0271	0.0271	0.0446

B.4.11. More Details of Experiments on Using Different Values for Alpha

For trend prediction, understanding the impact of the threshold parameter α on the classification of financial data into Up, Down, or Stationary categories is essential. We start by setting $\alpha = 0.00002$ for the CHF dataset, matching the parameters used in the FI-2010 study for direct comparison. We then experiment with different values of α, specifically $\alpha = 0.00001$ and $\alpha = 0.00004$, to see how changes in this threshold affect the trend prediction model. These tests reveal clear patterns, which we display in Figures B.18, B.19, and B.20.

As α increases, fewer price movements qualify as Up or Down, and more get the Stationary label. This happens because a higher α demands a larger price change to classify movements as

Up or Down. In contrast, a lower α lets even small price changes trigger a change from Stationary. Adjusting α thus allows us to fine-tune the model's sensitivity to price movements, tailoring it to specific financial datasets or desired sensitivity levels.

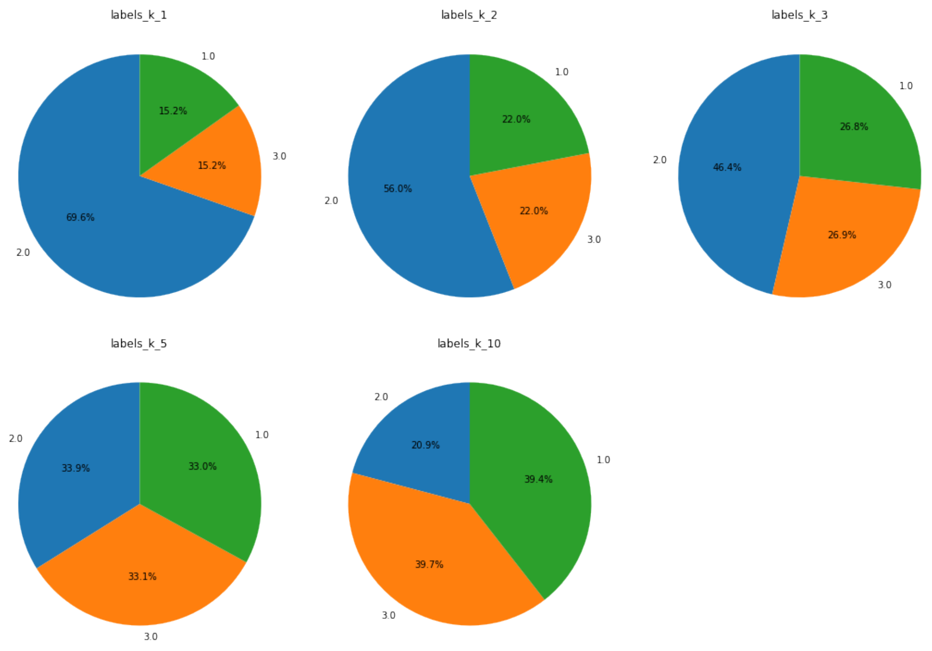


Figure B.18. Label Percentage with $\alpha = 0.00001$. 1.0: Up, 2.0: Stationary, 3.0: Down

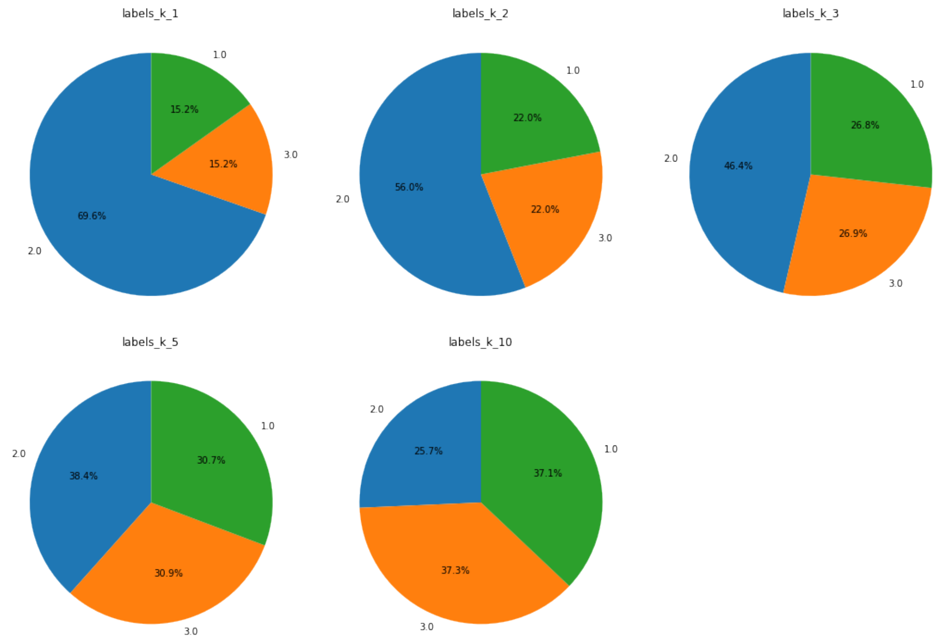


Figure B.19. Label Percentage with $\alpha = 0.00002$. 1.0: Up, 2.0: Stationary, 3.0: Down

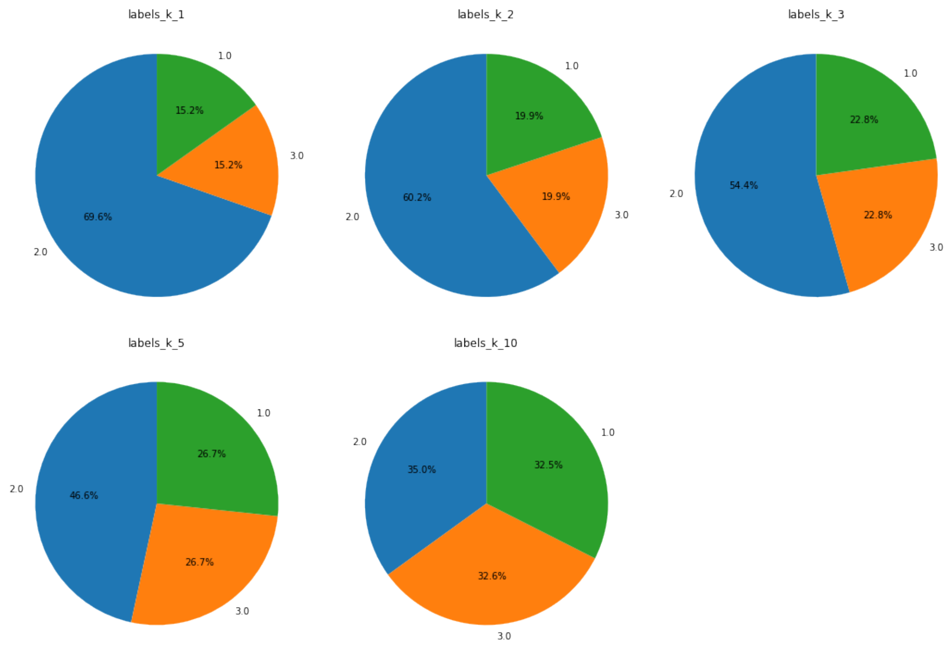


Figure B.20. Label Percentage with $\alpha = 0.00004$. 1.0: Up, 2.0: Stationary, 3.0: Down

Table B.21. **Mid-price Trend Prediction F1 Scores on Basic LOB data features for alpha=0.00004**.

	CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10
MLP	42.179	42.623	44.683	43.670	39.676
LSTM	50.533	49.607	45.793	47.074	44.035
CNN1	44.180	47.410	44.294	44.485	43.375
CTABL	40.567	40.847	40.301	41.336	39.884
DEEPLOB	49.497	47.882	46.313	46.244	43.379
DAIN	46.920	44.002	44.659	46.027	41.730
CNNLSTM	48.993	47.387	46.308	46.046	43.793
CNN2	47.072	47.272	45.851	45.195	44.296
TRANSLOB	47.431	44.890	45.193	44.260	43.675
TLONBoF	41.419	42.129	40.742	41.930	40.319
BiN-CTABL	44.335	42.938	42.707	42.020	40.013
DEEPLOBATT	49.245	48.819	46.836	47.283	44.575
DLA	47.439	47.194	42.998	44.598	44.183
Mean	46.139	45.615	44.437	44.628	42.533

B.4.12. Experiments on Using Different Number of Levels of the LOB Data

This section investigates how varying the number of levels in the Limit Order Book (LOB) data impacts model performance. Limit Order Books record buying and selling interest at various price levels and are essential for understanding market dynamics. We conduct experiments using Basic LOB data with the number of levels set to 2, 5, and 10. These experiments cover both trend prediction and return forecasting tasks, providing insights into the optimal level of detail needed for effective modeling.

Table B.22. **Mid-price Trend Prediction F1 Scores on Basic LOB data features for alpha=0.00001**.

	CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10
MLP	37.160	43.472	46.296	45.630	35.976
LSTM	47.929	48.284	50.695	50.484	43.474
CNN1	37.980	47.830	49.090	48.806	41.873
CTABL	45.024	45.697	47.326	44.944	44.788
DEEPLOB	48.541	47.775	50.258	47.477	38.535
DAIN	46.037	49.413	49.427	46.764	40.413
CNNLSTM	46.630	48.118	50.219	48.151	46.529
CNN2	45.833	49.907	50.226	50.272	44.971
TRANSLOB	50.066	50.895	50.824	49.692	46.628
TLONBoF	44.520	47.214	46.323	46.934	44.268
BiN-CTABL	44.151	44.391	45.379	45.584	44.094
DEEPLOBATT	46.917	49.485	50.723	49.926	44.128
DLA	47.439	49.996	50.136	47.377	36.444
Mean	45.248	47.883	48.994	47.849	42.471

Table B.23. **Mid-price Trend Prediction F1 Scores on 2 Levels of LOB Data**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	50.129	47.427	51.967	54.914	52.135	39.284	43.894	47.373	45.938	39.194
LSTM	63.049	56.332	62.413	64.816	62.992	45.546	49.411	<u>50.455</u>	**48.496**	44.446
CNN1	27.607	54.638	59.043	62.135	63.582	41.827	46.37	49.585	46.486	44.066
CTABL	62.837	57.618	62.278	64.526	66.870	42.843	45.12	46.767	46.201	43.261
DEEPLOB	62.860	56.910	63.237	66.150	69.092	42.716	47.772	48.706	47.88	41.84
DAIN	<u>80.240</u>	<u>70.604</u>	80.171	87.300	<u>91.944</u>	42.421	48.569	49.276	46.43	40.659
CNNLSTM	27.620	38.466	37.521	55.728	63.487	43.736	46.467	49.513	47.546	45.443
CNN2	32.536	51.300	59.027	64.464	63.577	40.458	47.429	47.416	48.031	<u>46.229</u>
TRANSLOB	59.367	53.775	58.834	59.119	59.647	**49.064**	**50.553**	**50.659**	<u>48.375</u>	**46.35**
TLONBoF	41.306	38.593	61.975	64.761	62.018	44.182	45.799	46.48	46.306	43.447
BiN-CTABL	**81.001**	**71.754**	**80.762**	<u>87.453</u>	**92.070**	44.27	44.04	45.977	46.177	43.517
DEEPLOBATT	63.553	58.110	64.063	67.241	66.445	<u>46.533</u>	48.812	49.591	48.163	41.297
DLA	79.689	70.186	<u>80.426</u>	**87.467**	52.417	45.275	<u>49.681</u>	49.407	47.184	38.708
Mean	56.292	55.824	63.209	68.160	66.636	43.704	47.225	48.570	47.155	42.958

Table B.24. **Mid-price Trend Prediction F1 Scores on 5 Levels of LOB Data**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	52.794	40.386	48.657	52.382	49.696	39.506	45.029	45.268	45.681	40.506
LSTM	67.331	58.830	65.323	67.757	63.964	47.494	**49.760**	**51.036**	<u>48.519</u>	44.913
CNN1	27.608	41.759	60.526	62.797	66.061	44.615	49.182	49.903	48.000	45.299
CTABL	66.114	58.894	66.118	71.151	70.679	39.754	45.039	46.313	45.650	43.467
DEEPLOB	69.551	61.271	69.204	73.892	75.685	**47.815**	45.593	45.579	45.070	38.657
DAIN	79.949	<u>70.409</u>	80.017	<u>87.202</u>	<u>91.902</u>	44.611	48.540	48.874	46.530	39.390
CNNLSTM	27.620	33.814	46.195	58.340	58.298	37.710	45.054	43.430	45.737	46.422
CNN2	27.620	30.295	38.296	62.642	67.814	<u>47.251</u>	48.248	50.136	47.565	43.717
TRANSLOB	61.506	53.964	59.260	60.364	61.561	46.320	<u>49.216</u>	<u>50.823</u>	47.853	<u>45.845</u>
TLONBoF	38.524	38.180	36.624	66.409	64.650	44.738	47.358	46.801	46.336	44.310
BiN-CTABL	**81.049**	**71.030**	**80.737**	**87.563**	**92.041**	45.134	46.116	45.965	45.781	44.234
DEEPLOBATT	67.191	63.967	67.897	74.033	75.205	46.818	48.918	50.525	**48.729**	**46.580**
DLA	<u>80.214</u>	69.412	<u>80.224</u>	87.092	54.495	34.711	41.011	41.816	46.870	36.906
Mean	57.467	53.247	61.468	70.125	68.619	43.498	46.851	47.386	46.794	43.014

Table B.25. **Mid-price Trend Prediction F1 Scores on 10 Levels of LOB Data**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	48.934	44.339	46.866	50.816	53.545	N/A	N/A	N/A	N/A	N/A
LSTM	66.601	58.858	65.400	67.607	59.442	N/A	N/A	N/A	N/A	N/A
CNN1	60.262	56.685	62.396	65.374	66.932	N/A	N/A	N/A	N/A	N/A
CTABL	69.677	63.095	69.761	74.298	74.625	N/A	N/A	N/A	N/A	N/A
DEEPLOB	71.415	62.300	70.862	75.947	77.880	N/A	N/A	N/A	N/A	N/A
DAIN	79.866	70.224	79.785	<u>87.034</u>	<u>91.758</u>	N/A	N/A	N/A	N/A	N/A
CNNLSTM	64.611	54.301	62.427	65.801	66.689	N/A	N/A	N/A	N/A	N/A
CNN2	61.623	50.089	59.730	62.955	61.092	N/A	N/A	N/A	N/A	N/A
TRANSLOB	61.120	54.331	60.015	62.266	60.777	N/A	N/A	N/A	N/A	N/A
TLONBoF	57.320	51.144	56.688	65.032	65.900	N/A	N/A	N/A	N/A	N/A
BiN-CTABL	**81.042**	**71.372**	**80.761**	**87.587**	**92.078**	N/A	N/A	N/A	N/A	N/A
DEEPLOBATT	69.302	64.018	66.629	72.033	71.523	N/A	N/A	N/A	N/A	N/A
DLA	<u>79.932</u>	<u>70.259</u>	<u>79.883</u>	86.778	51.224	N/A	N/A	N/A	N/A	N/A
Mean	64.818	57.239	64.175	68.775	66.252	N/A	N/A	N/A	N/A	N/A

B.4.12.1. Mid-price Trend Prediction.

B.4.12.2. Mid-price Return Forecasting.

B.4.13. Training Epochs of Models

In all the experiments, we use the early stopping strategy. The following table shows the actual number of training epochs that each model was trained for before triggering the early stopping condition.

Table B.26. **Mid-price Return Forecasting on 2 Levels of LOB data**.

		FI-2010					CHF-2023				
Model	Metric	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
	MSE	0.655	$\underline{1.073}$	1.396	1.932	2.751	0.217	0.218	0.219	0.141	0.371
MLP	Corr	0.058	0.084	0.095	0.113	0.136	0.000	0.000	0.000	0.000	0.000
	R2	0.002	0.005	0.008	0.012	0.017	0.000	0.000	0.000	0.000	0.000
	MSE	**0.653**	**1.071**	**1.393**	**1.927**	**2.743**	**0.216**	**0.218**	**0.219**	**0.141**	**0.371**
LSTM	Corr	**0.069**	**0.091**	**0.104**	**0.121**	**0.142**	**0.010**	**0.011**	**0.011**	**0.013**	**0.008**
	R2	**0.005**	**0.008**	**0.0105**	**0.014**	**0.019**	0.000	0.000	0.000	0.000	0.000
	MSE	$\underline{0.654}$	**1.071**	$\underline{1.394}$	1.930	2.747	**0.216**	0.218	0.219	0.141	0.371
CNN1	Corr	$\underline{0.065}$	0.089	$\underline{0.101}$	$\underline{0.115}$	$\underline{0.138}$	0.000	-0.001	0.001	0.000	0.000
	R2	0.003	0.006	0.009	0.013	0.018	0.000	0.000	0.000	0.000	0.000
	MSE	0.655	1.074	1.400	1.940	2.768	0.217	0.218	0.219	0.141	0.371
CNNLSTM	Corr	0.051	0.069	0.078	0.094	0.114	0.000	0.000	0.000	0.000	0.000
	R2	0.002	0.004	0.005	0.008	0.011	0.000	0.000	0.000	0.000	0.000
	MSE	0.657	1.080	1.409	1.961	2.794	**0.216**	0.218	0.219	0.141	0.371
CNN2	Corr	0.024	0.060	0.067	0.079	0.098	0.000	0.000	0.000	0.000	0.000
	R2	-0.002	-0.001	-0.001	-0.003	0.002	0.000	0.000	0.000	0.000	0.000
	MSE	0.659	1.086	1.421	1.979	2.841	0.219	0.217	0.220	0.142	0.371
PatchTST	Corr	0.069	0.078	0.078	0.084	0.086	0.000	0.000	0.000	0.000	0.000
	R2	-0.004	-0.006	-0.009	-0.012	-0.015	-0.052	-0.055	-0.035	-0.002	0.007
	MSE	0.695	1.136	1.478	2.038	2.892	0.226	**0.216**	0.227	0.142	0.371
DLinear	Corr	0.026	0.024	0.024	0.034	0.042	0.000	0.000	0.000	0.000	0.000
	R2	-0.058	-0.052	-0.049	-0.042	-0.034	-0.043	0.000	-0.036	-0.013	-0.002
	MSE	0.699	1.164	1.525	2.116	3.054	0.217	0.218	0.219	0.142	0.371
iTransformer	Corr	0.039	0.038	0.041	0.059	0.071	0.000	0.000	0.000	0.000	0.000
	R2	-0.065	-0.078	-0.083	-0.082	-0.092	-0.085	-0.089	-0.066	-0.074	-0.075
	MSE	0.671	1.099	1.435	1.983	2.827	0.218	0.219	0.220	0.141	0.371
TimeMixer	Corr	0.033	0.045	0.049	0.064	0.077	0.001	0.001	0.001	0.001	0.001
	R2	-0.021	-0.018	-0.019	-0.014	-0.011	-0.006	-0.005	-0.005	-0.003	-0.002

B.5. Hopformer: Homogeneity-Pursuit Transformer for Time Series Forecasting

B.5.1. Appendix / supplemental material

B.5.1.1. Proof of Theorem 3.5.1.

Lemma B.5.1 ((Leung and Barron, 2006)). Let M be a finite set of candidate models with $|M|$ denoting its cardinality. For each model $m \in M$, define $\widehat{r}_m = \mathbb{E}\|\widehat{g}_m - \eta\|^2$ as the empirical risk estimate of model m.

Table B.27. **Mid-price Return Forecasting on 5 Levels of LOB data**.

		FI-2010					CHF-2023				
Model	Metric	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	MSE	0.658	1.075	1.398	1.933	2.749	**0.216**	**0.217**	**0.218**	0.140	**0.370**
	Corr	0.062	0.073	0.087	0.114	0.143	0.045	**0.076**	0.057	0.081	0.066
	R2	-0.003	0.003	0.006	0.011	0.017	**0.001**	**0.006**	**0.003**	**0.006**	**0.004**
LSTM	MSE	0.654	1.072	1.395	1.930	2.745	2.747	0.218	0.219	0.141	0.371
	Corr	0.064	0.086	0.099	0.118	0.140	0.011	0.005	0.012	0.014	0.008
	R2	0.003	0.006	0.009	0.012	0.018	0.000	-0.000	0.000	0.000	0.000
CNN1	MSE	**0.653**	**1.071**	**1.393**	**1.928**	**2.742**	0.217	0.220	0.220	**0.136**	0.364
	Corr	**0.067**	**0.089**	**0.105**	**0.121**	**0.145**	**0.068**	0.056	**0.076**	**0.191**	**0.136**
	R2	0.004	0.007	0.010	0.0142	0.0204	- 0.002	-0.008	-0.002	0.036	0.018
CNNLSTM	MSE	0.656	1.078	1.406	1.949	2.766	2.788	0.218	0.219	0.141	0.371
	Corr	0.042	0.061	0.071	0.093	0.121	-0.000	0.000	0.000	0.002	0.001
	R2	0.000	0.001	0.001	0.003	0.004	-0.000	0.000	-0.000	0.000	-0.000
CNN2	MSE	0.658	1.079	1.406	1.950	2.777	0.217	0.218	0.219	0.141	0.371
	Corr	0.041	0.069	0.083	0.096	0.119	0.011	0.011	0.011	0.014	0.008
	R2	-0.003	3.797	0.001	0.002	0.007	0.000	0.000	0.000	0.000	0.000
PatchTST	MSE	0.659	1.086	1.421	1.979	2.841	0.306	0.310	0.293	0.185	0.490
	Corr	0.070	0.077	0.078	0.084	0.086	0.000	0.000	0.000	-0.001	0.000
	R2	-0.004	-0.006	-0.009	-0.012	-0.016	-0.415	-0.423	-0.335	-0.311	-0.320
DLinear	MSE	0.702	1.1412	1.488	2.058	2.926	0.288	0.289	0.289	0.183	0.466
	Corr	0.021	0.027	0.028	0.031	0.036	0.000	0.000	0.000	0.000	0.000
	R2	-0.07	-0.057	-0.057	-0.052	-0.046	-0.330	-0.322	-0.317	-0.296	-0.255
iTransformer	MSE	0.696	1.157	1.426	2.124	3.026	0.224	0.223	0.221	0.144	0.386
	Corr	0.040	0.043	0.044	0.058	0.072	0.000	0.000	0.000	-0.001	-0.001
	R2	-0.060	-0.072	-0.069	-0.086	-0.082	-0.0362	-0.020	-0.010	-0.020	-0.041
TimeMixer	MSE	0.677	1.099	1.432	1.983	2.823	0.284	0.294	0.265	0.179	0.490
	Corr	0.030	0.050	0.056	0.065	0.081	0.000	0.000	0.000	0.000	0.000
	R2	-0.032	-0.018	-0.016	-0.014	-0.009	-0.311	-0.346	-0.208	-0.260	-0.320

Define the aggregation weights w_m are given by:

(B.5.1)
$$w_m = \frac{\pi_m \exp(-\widehat{r}_m/4)}{\sum_{m' \in M} \pi_{m'} \exp(-\widehat{r}_{m'}/4)},$$

where $\pi_m = \exp(-C_m)$ satisfying $\sum_{m \in M} \pi_m \leqslant 1$ with C_m denote complexity penalty term associated with model m.

Table B.28. **Mid-price Return Forecasting on 10 Levels of LOB data**.

		FI-2010					CHF-2023				
Model	Metric	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
	MSE	0.645	1.055	1.378	1.904	2.723	N/A	N/A	N/A	N/A	N/A
MLP	Corr	0.170	0.181	0.177	0.187	0.177	N/A	N/A	N/A	N/A	N/A
	R2	0.018	0.023	0.021	0.026	0.027	N/A	N/A	N/A	N/A	N/A
	MSE	0.639	1.042	1.349	**1.885**	2.745	N/A	N/A	N/A	N/A	N/A
LSTM	Corr	0.205	0.218	0.219	**0.199**	0.181	N/A	N/A	N/A	N/A	N/A
	R2	0.042	0.048	**0.048**	**0.039**	0.032	N/A	N/A	N/A	N/A	N/A
	MSE	0.631	1.034	1.350	1.891	**2.716**	N/A	N/A	N/A	N/A	N/A
CNN1	Corr	0.226	0.219	0.214	0.198	**0.190**	N/A	N/A	N/A	N/A	N/A
	R2	0.051	0.048	0.046	**0.039**	**0.036**	N/A	N/A	N/A	N/A	N/A
	MSE	0.659	1.044	1.351	1.910	2.766	N/A	N/A	N/A	N/A	N/A
CNNLSTM	Corr	0.159	0.204	0.217	0.196	0.171	N/A	N/A	N/A	N/A	N/A
	R2	0.051	0.048	0.046	**0.039**	**0.036**	N/A	N/A	N/A	N/A	N/A
	MSE	**0.620**	**1.026**	**1.345**	1.943	2.797	N/A	N/A	N/A	N/A	N/A
CNN2	Corr	**0.239**	**0.222**	**0.229**	0.181	0.112	N/A	N/A	N/A	N/A	N/A
	R2	**0.056**	**0.049**	0.045	0.007	0.001	N/A	N/A	N/A	N/A	N/A
	MSE	0.662	1.088	1.419	1.968	2.831	N/A	N/A	N/A	N/A	N/A
PatchTST	Corr	-0.061	-0.060	0.087	-0.053	0.077	N/A	N/A	N/A	N/A	N/A
	R2	-0.054	-0.058	-0.015	-0.067	-0.037	N/A	N/A	N/A	N/A	N/A
	MSE	0.655	1.077	1.404	1.952	2.802	N/A	N/A	N/A	N/A	N/A
DLinear	Corr	0.054	0.059	0.056	0.052	0.042	N/A	N/A	N/A	N/A	N/A
	R2	0.003	0.002	0.003	0.002	0.001	N/A	N/A	N/A	N/A	N/A
	MSE	0.658	1.087	1.426	1.985	2.857	N/A	N/A	N/A	N/A	N/A
iTransformer	Corr	0.074	0.076	0.063	0.086	0.081	N/A	N/A	N/A	N/A	N/A
	R2	-0.002	-0.007	-0.013	-0.023	-0.021	N/A	N/A	N/A	N/A	N/A
	MSE	0.660	1.081	1.410	1.937	2.735	N/A	N/A	N/A	N/A	N/A
TimeMixer	Corr	0.060	0.077	0.086	0.119	0.161	N/A	N/A	N/A	N/A	N/A
	R2	-0.005	-0.002	-0.001	0.010	0.023	N/A	N/A	N/A	N/A	N/A

Then, the unbiased risk estimate for $\widehat{\mu}$ satisfies:

$$\widehat{r} = \sum_{m \in M} w_m \widehat{r}_m < \min_{m \in M} \{\widehat{r}_m + 4C_m\} \tag{B.5.2}$$

Table B.29. **Average Training Epochs for Mid-price Trend Prediction for Basic LOB features**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	1.3	5.0	4.7	3.3	3.3	4.0	5.8	3.2	3.8	3.2
LSTM	8.0	6.0	8.3	12.7	10	5.4	4.6	4.2	3.4	3.6
CNN1	1.0	1.0	8.7	13.3	11	5.4	6	5	4.4	3.0
CTABL	5.7	3.3	5.7	4.3	4.7	3.6	2.0	2.2	3.4	2.2
DEEPLOB	11.3	8.8	8.0	7.5	4.8	6.2	6.2	6.2	4.8	3.0
DAIN	8.3	7.0	8.0	5.7	6.0	4.8	5.0	4.4	3.8	1.2
CNNLSTM	1.0	1.0	4.0	8.3	10.0	4.0	2.8	4.4	5.6	3.4
CNN2	1.0	1.0	3.0	6.7	6.7	3.4	3.0	5.0	4.2	3.4
TRANSLOB	8.2	8.8	5.6	13	8.8	3.8	1.8	2.4	3.2	2.0
TLONBoF	3.3	4.7	2.7	3	11.7	3.6	3.4	4.2	4.0	3.2
BiN-CTABL	3.6	5.6	4.4	3.8	2.8	3.8	3.4	3.2	2.0	2.2
DEEPLOBATT	6	7.5	4.5	4.3	5.8	3.0	3.0	2.6	2.8	2.4
DLA	17	12.8	26	28.2	13.2	5.6	5.6	4.4	3.8	3.2

Table B.30. **Average Training Epochs for Mid-price Trend Prediction for Basic LOB + time insensitive features**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	6.7	6.0	6.7	2.7	4.7	3.4	3.6	3.4	3.6	4.2
LSTM	9.3	9.7	11.3	11.7	6.7	4.0	2.4	4.0	3.6	2.8
CNN1	17.7	17.7	17	20	19.3	5.4	4.4	5.2	5.2	3.6
CTABL	4.0	5.7	4.3	2.7	6.7	1.8	2.4	2.2	2.6	3.6
DEEPLOB	11.5	9.3	8.0	8.0	6.0	4.6	3.8	3.8	4.4	3.2
DAIN	6.7	5.0	5.7	5.0	5.3	4.0	4.0	1.6	1.8	1.0
CNNLSTM	12.3	11.0	7.3	4.7	6.0	2.6	3.4	4.2	5.2	3.8
CNN2	2.3	9.7	6.7	5.0	4.0	4.4	4.4	5.0	4.2	4.0
TRANSLOB	22.2	19.2	21.2	17.6	12.3	2.2	3.0	2.4	2.2	3.2
TLONBoF	3.3	4.0	5.0	6.7	11.3	6.4	4.4	3.4	3.6	4.2
BiN-CTABL	3.2	2.2	4	4.2	2.8	3.0	2.0	2.2	2.6	2.2
DEEPLOBATT	4.0	4.3	3.3	3.0	3.3	3.2	3.8	3.0	2.8	1.8
DLA	13.2	11.4	9.8	6.8	6.8	4.2	4.8	4.6	3.6	3.2

Table B.31. **Average Training Epochs for Mid-price Trend Prediction for Basic LOB + time insensitive + time sensitive features**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	3.7	4.3	3.7	3.0	3.3	2.4	4.4	4.0	3.4	3.2
LSTM	11.7	13.0	12.0	11.0	3.0	4.0	3.8	3.8	3.0	3.8
CNN1	18.0	22.3	16.7	19.0	20.7	5.2	3.6	5.0	5.2	3.6
CTABL	5.0	6.0	5.3	7.0	8.0	2.4	1.8	1.4	4.0	3.4
DEEPLOB	10.0	12.0	10.3	7.0	5.8	2.8	3.6	4.0	3.8	2.4
DAIN	5.3	6.0	5.3	6.3	4.7	5.0	3.4	1.0	1.2	1.0
CNNLSTM	8.0	6.3	7.0	6.0	5.7	2.6	2.2	3.4	4.8	2.6
CNN2	6.7	6.7	6.3	6.7	3.3	3.8	2.4	3.2	2.6	3.4
TRANSLOB	16.6	19.6	18.6	16.6	9	3.0	2.0	1.4	2.2	2.0
TLONBoF	6.7	17.0	16.3	17.0	10.7	5.2	4.0	4.0	2.2	2.6
BiN-CTABL	2.2	9.2	7.8	4.8	3.8	4.0	2.4	3.0	1.6	3.0
DEEPLOBATT	4.3	4.3	4.3	2.3	3.0	3.0	4.0	3.0	3.2	1.8
DLA	2.8	5.4	7.4	9.2	2.2	4.4	3.8	3.2	2.2	2.8

Table B.32. **Average Training Epoch for Mid-price Return Forecasting**.

	FI-2010					CHF-2023				
Model	K=1	K=2	K=3	K=5	K=10	K=1	K=2	K=3	K=5	K=10
MLP	3.1	2.6	5	9	7.6	3	3	3	2.6	3
LSTM	3	5	7	2	1.6	4	5	5	4.6	4.3
CNN1	3	5.6	5	5.3	3.3	3.3	5	2	3	2
CNNLSTM	4	3.3	4.3	3.3	3.3	3	3	3.3	3.3	3
CNN2	2.3	2.3	3	1.3	2.6	3.3	5	5	4	3
PatchTST	8	7	7	6	7.3	5	4	4	5	5
DLinear	14	14	13	15	14	9	8	8	9	10
iTransformer	2.6	2.3	2	2	1.3	2.3	2.6	2	3.3	2
TimeMixer	9	8.3	8.6	9.3	8	4.6	6	5	5	4

Proof. Let $\widehat{m} \in \arg\min_{m\in M}\{\widehat{r}_m + 4C_m\}$. Then, we can write:

$$\widehat{r}_m = 4\left[\log\frac{\pi_m}{w_m} - \log\sum_{m'}\pi_{m'}\exp(-\widehat{r}_{m'}/4)\right] = \widehat{r}_{\widehat{m}} + 4\left[C_{\widehat{m}} - \log\frac{w_m}{\pi_m} + \log w_{\widehat{m}}\right].$$

Therefore:

$$\widehat{r} = \sum_{m\in\mathcal{M}} w_m\widehat{r}_m = \widehat{r}_{\widehat{m}} + 4[C_{\widehat{m}} - D(w\|\pi) + \log w_{\widehat{m}}] < \min_{m\in\mathcal{M}}\{\widehat{r}_m + 4C_m\}.$$

□

The proof of Theorem 3.5.1 follows from letting

$$\widehat{r}_p = \frac{1}{\sigma^2}\sum_{k=1}^{n}\left(X_{t,k}^{(i)} - g_{\widehat{\omega}_{\mathbf{p}}}(\mathbf{Z}_{t,k}^{(i)})\right)^2 + 2|\mathbf{p}|$$

B.5.1.2. Proof of Theorem 3.5.2.

Lemma B.5.2 ((Xu and Raginsky, 2017))**.** Consider a pair of random variables X and Y with joint distribution $P_{X,Y}$, Let $\overline{X}$ be an independent copy of X, $\overline{Y}$ be an independent copy of Y, such that $P_{\overline{X},\overline{Y}} = P_X \otimes P_Y$. Let f be σ-subgaussian under $P_{\overline{X},\overline{Y}} = P_X \otimes P_Y$, then

$$\left|\mathbb{E}[f(X,Y)] - \mathbb{E}[f(\overline{X},\overline{Y})]\right| \leqslant \sqrt{2\sigma^2 I(X;Y)},$$

Let $X = \boldsymbol{R}_T$, $Y = W$ and $f(s,w) = \frac{1}{T-d}\sum_{i=d+1}^{T}\ell(w,x_i)$, then we can express the empirical risk as $L_{\boldsymbol{R}_T}(w) = f(\boldsymbol{R}_T, w)$, and the population risk $L_{\mathcal{P}}(w) = \mathbb{E}_{\mathcal{P}}[f(\boldsymbol{R}_T, w)]$.

Thus, the generalization error reformulates as

$$\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})] = \mathbb{E}[f(\overline{\boldsymbol{R}_T}, \overline{\mathbf{W}})] - \mathbb{E}[f(\boldsymbol{R}_T, \mathbf{W})].$$

Definition B.5.1 (Stationarity)**.** A random sequence S_∞ is stationary when all its finite-dimensional distributions are time-invariant: for all t and all non-negative integers i and j, the random vectors $S_{t:t+i}$ and $S_{t+j:t+i+j}$ have the same distribution.

Definition B.5.2 (β-Mixing)**.** Consider a stationary random sequence S_∞ defined on a probability space $(\Omega, \Sigma, P_\infty)$. Denote $S_{i:j} := (S_i, S_{i+1}, \ldots, S_j), S_\infty := S_{-\infty:\infty}$ an infinite dimensional sequence. Denote $\mathbb{P}_{i:j}$ and $\mathbb{P}_\infty$ as the associated joint distributions, and $\sigma_{i:j} = \sigma(S_{i:j})$ and $\sigma_\infty = \sigma(S_\infty)$ as the σ-fields. Let P_0 be the restriction of P_∞ to $\sigma_{-\infty:0}$, P_a be the restriction of P_∞ to $\sigma_{a:\infty}$, and $P_{0\otimes a}$ be the restriction of P_∞ to $\sigma(S_{\infty:0}, S_{a:\infty})$. The coefficient of absolute regularity, or β-mixing coefficient, β_a, is given by

$$\beta_a := \|P_0 \times P_a - P_{0\otimes a}\|_{TV},$$

where $\|\cdot\|_{TV}$ is the total variation norm. A stochastic process is absolutely regular, or β-mixing, if $\beta_a \to 0$ as $a \to \infty$.

Lemma B.5.3. Assuming the loss function $\ell(w, R)$ is σ-subgaussian, if the target time series is stationary and β-mixing, then there exists a constant $a > 0$ such that $2am \leqslant T$ ensuring that the empirical loss $f(\boldsymbol{R}_T, w)$ is $\frac{\sigma}{\sqrt{m}}$-subgaussian.

Proof. We divide the sequence $S_{d+1:T}$ into $2m$ blocks of length a, such that $2ma + d = T$. Identify the blocks as follows:

$$U_j = \{R_i : 2(j-1)a + d + 1 \leqslant i \leqslant (2j-1)a + d\},$$

$$V_j = \{R_i : (2j-1)a + d + 1 \leqslant i \leqslant 2ja + d\}.$$

Let $\mathbf{U}$ be the sequence of odd blocks, and let $\mathbf{V}$ be the sequence of even blocks. With the mixing conditions (need β-mixing assumption), we choose a large enough so that the odd blocks are almost independent, but at the same time small enough so that the odd blocks behave similarly to the original mixing sequence. Let $\mathbf{U}'$ be a sequence of identically distributed independent blocks such that each block has the same distribution as a block from the original sequence:

$$\mathcal{L}(U'_j) = \mathcal{L}(U_j) = \mathcal{L}(U_1).$$

By the linearity of subgaussianity, the block-wise empirical average is given by:

$$\widehat{f}_{U_j}(w) = \frac{1}{a} \sum_{i \in U_j} \ell(w, R_i)$$

remains σ-subgaussian. By (Yu, 1994), for the odd blocks sequence $f_{\mathbf{U}}(w) = \frac{1}{m} \sum_{j=1}^{m} \widehat{f}_{U_j}(w)$, we have

$$\log \mathbb{E}\Big[e^{\lambda(f_{\mathbf{U}}(w) - \mathbb{E}(f_{\mathbf{U}}(w)))}\Big] \lesssim \log \mathbb{E}\Big[e^{\lambda(f_{\mathbf{U}'}(w) - \mathbb{E}(f_{\mathbf{U}'}(w)))}\Big] \leqslant \frac{\lambda^2 \sigma^2}{2m},$$

which implies that $f_{\mathbf{U}}(w)$ is $\frac{\sigma}{\sqrt{m}}$-subgaussian.

Applying the same construction to the sequence of even blocks, the total empirical loss $f(\boldsymbol{R}_T, w)$ can be viewed as the sum of two $\frac{\sigma}{\sqrt{m}}$-subgaussian variables. By the linearity property of subgaussianity, it follows that $f(\boldsymbol{R}_T, w)$ is also $\frac{\sigma}{\sqrt{m}}$-sub-Gaussian.

□

Applying Lemma B.5.2 and Lemma B.5.3, we obtain the generalization bound as in (3.5.11)

$$\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})] \leqslant \sqrt{\frac{2\sigma^2}{m} I(\boldsymbol{R}_T; \mathbf{W})}.$$

We follow the blocking technique from the proof above to divide $S_{d+1:N}$ into $2m$ blocks of length a, ensuring that sufficiently spaced blocks are approximately independent under the assumption of mixing.

By the data processing inequality, we have

$$I(\mathbf{W} + \Delta\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}) \leqslant I(\Delta\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}, \mathbf{W}).$$

Using the chain rule and the assumption that $\mathbf{W}$ is independent of $\boldsymbol{R}_T$, the mutual information can be decomposed as

$$I(\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}) + I(\Delta\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}, \mathbf{W}) = I(\Delta\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}, \mathbf{W}).$$

Combining these results with 3.5.11, and apply the standard inequality that mutual information is always upper bounded by entropy ($I\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}; \boldsymbol{R}_T\Big) \leqslant H\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}\Big)$), we obtain the generalization bound

$$\begin{aligned}
\mathbb{E}[L_{\mathcal{P}}(\mathbf{W}) - L_{\boldsymbol{R}_T}(\mathbf{W})] &\leqslant \sqrt{\frac{2\sigma^2}{m} I(\Delta\mathbf{W}; \boldsymbol{R}_T \mid P_{\mathbf{W}|\boldsymbol{R}_T}, \mathbf{W})} \\
&= \sqrt{\frac{2\sigma^2}{m} I\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}; \boldsymbol{R}_T \mid P_{\mathbf{W}_{\{q,k,v\}}|\boldsymbol{R}_T}, \mathbf{W}\Big)} \\
&\leqslant \sqrt{\frac{2\sigma^2}{m} H\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}\Big)} \\
&\leqslant \sqrt{\frac{6\sigma^2}{m} qr \sum_{i\in\mathcal{I}} (d_{in} + d_{out})},
\end{aligned}$$

where $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i \in \mathbb{R}^{d_{in}\times d_{out}}$. The last inequality holds because entropy is upper bounded by the uniform distribution over its possible support set.

Remark. For a more general setting that each attention head $i \in \mathcal{I}$ with:

- Query matrix: $\mathbf{W}_q^i \in \mathbb{R}^{d_{\text{in}}\times d_k}$,
- Key matrix: $\mathbf{W}_k^i \in \mathbb{R}^{d_{\text{in}}\times d_k}$,
- Value matrix: $\mathbf{W}_v^i \in \mathbb{R}^{d_{\text{in}}\times d_v}$.

Thus, the total entropy bound is:

$$H\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}\Big) \leqslant qr \sum_{i\in\mathcal{I}} [2(d_{\text{in}} + d_k) + (d_{\text{in}} + d_v)].$$

Applying the mutual information bound, we conclude:

$$I\Big(\big\{\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i\big\}_{i\in\mathcal{I}}; \boldsymbol{R}_T\Big) \leqslant qr \sum_{i\in\mathcal{I}} [2(d_{\text{in}} + d_k) + (d_{\text{in}} + d_v)].$$

Remark. Consider a pre-trained model with weights $\mathbf{W}_0$ (independent of $\boldsymbol{R}_T$) and LoRA's learned low-rank parameters $\Delta\mathbf{W} = \mathbf{BA}$. Since $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}$ and $\mathbf{W}_0$ are fixed, $I(\boldsymbol{R}_T; \mathbf{W}) = I(\boldsymbol{R}_T; \Delta\mathbf{W})$. If $\Delta\mathbf{W}$ contains d parameters in total (with a suitably discretized or bounded range), an upper bound on its entropy is $H(\Delta\mathbf{W}) \approx d \log |\text{Range}|$. For low-rank $\mathbf{A} \in \mathbb{R}^{m\times r}, \mathbf{B} \in \mathbb{R}^{r\times n}$, $d = r(m+n)$ which is much smaller than the full model's parameter count mn. Thus one can derive $I(\boldsymbol{R}_T; \mathbf{W}) \leqslant H(\Delta\mathbf{W}) = \mathcal{O}(r(m+n))$ (in bits). As r, m, n are modest for LoRA, the information bound is dramatically smaller. This back-of-the-envelope derivation aligns with the idea that LoRA's limited parameter count yields a provably smaller $I(\boldsymbol{R}_T; \mathbf{W})$, reinforcing why it generalizes well under the stability condition.

B.5.2. Additional Experiments on SPA

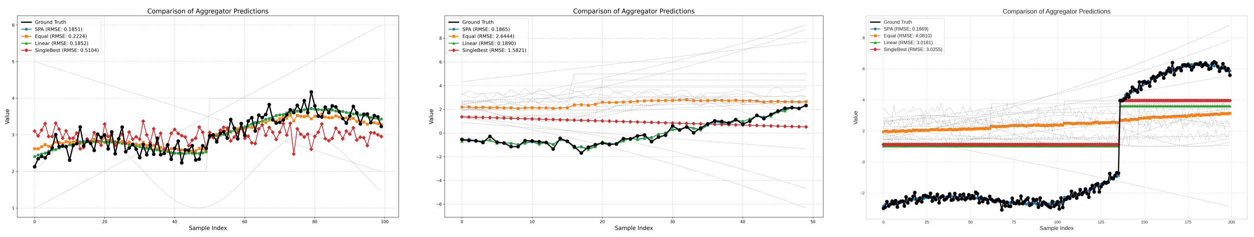


Figure B.21. Simulated grocery-sales example (grey curves are individual regressors encoding trend, seasonality, and promotion effects). Left: Linear mixture of three regressors over 100 time steps with added noise—SPA matches ordinary least-squares (LR) aggregation. Middle: Non-linear combination of 20 regressors over 50 time steps—SPA outperforms LR by capturing interaction effects. Right: Same non-linear setting over 200 time steps—SPA continues to beat LR, demonstrating robustness as series length grows.

B.5.3. Computation Resources

We run experiment on an Ubuntu server with 4 × 1080Ti GPUs and 96 CPUs.

We further provide supplementary analyses and visualizations to deepen our understanding of Hopformer's capabilities and limitations. Section B.5.4 describes the generation process for our synthetic datasets. Section B.5.5 evaluates Hopformer's robustness on real-world datasets and discusses observed limitations. In Section B.5.6, we visualize Hopformer's inference process, demonstrating the inference. Practical implementation [17] and reproducibility are detailed in Section B.5.7. Section B.5.8 integrates Hopformer with Chronos, Moirai (Woo et al., 2024a; Liu et al.,

[17]Data and code are available at https://www.dropbox.com/scl/fo/q4t08x79w1jxq2tnkz15d/ACXuC5n06yS17cH696oaP0g?rlkey=e9ogypv287u8232l06dzn0u28&dl=0.

2024b), and Lag-Llama (Rasul et al., 2024). Lastly, Section B.5.9 examines fine-tuning efficiency (Hu et al., 2022; Gupta et al., 2024a) on these models and Section B.5.10 visualizes the forecasting results of different models.

Through these evaluations, we conclusively illustrate Hopformer's robustness to varying context and prediction lengths and affirm its broad applicability, highlighting consistent performance gains when integrated with foundation models such as Chronos, Moirai, and Lag-Llama.

B.5.4. Synthetic Data

Synthetic Sale data - Sale1. We generate synthetic store sales data across 200 stores, each parameterized by independent and identically distributed (i.i.d.) parameters drawn from normal distributions. Each store i has unique parameters including amplitude A_i, frequency f_i, baseline sales B_i, and covariate sensitivities such as promotion effect P_i, temperature effect T_i, price effect C_i, promotion probability p_i, temperature noise σ_{T_i}, and sales noise σ_{S_i}. The mean and standard deviation of each of the parameters are in Table B.33. The sales data for each store at time t are generated as follows:

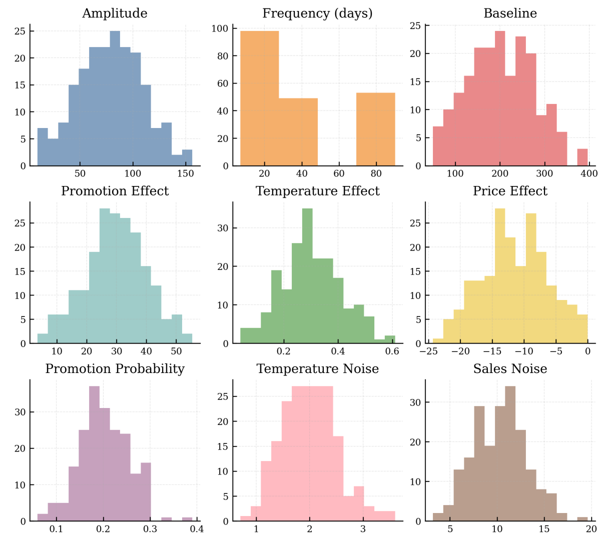
$$\text{Sales}_{i,t} = B_i + A_i \cdot \sin\left(\frac{2\pi t}{f_i}\right) + P_i \cdot \text{promotion}_{i,t} + T_i \cdot \text{temperature}_{i,t} + C_i \cdot \text{price}_{i,t} + \epsilon_{i,t}, \quad \epsilon_{i,t} \sim \mathcal{N}(0, \sigma_{S_i}^2)$$

Here, the promotion occurrence is sampled using the store-specific promotion probability p_i. The temperature follows a seasonal pattern with added Gaussian noise parameterized by σ_{T_i}. The term $\varepsilon_{i,t}$ represents Gaussian noise with standard deviation σ_{S_i}, capturing real-world randomness in sales. The covariates (promotion, temperature, and price) are known in advance and used as inputs for forecasting.

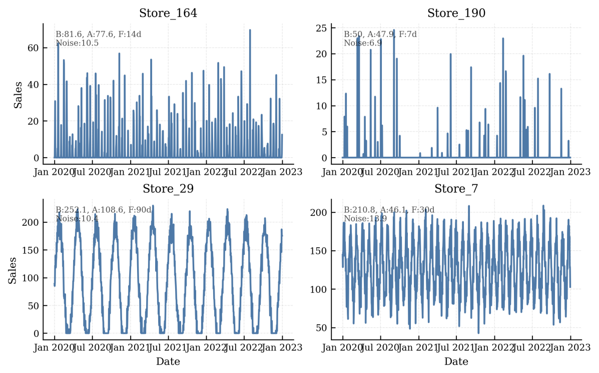
Table B.33. Mean and standard deviation of store-specific parameters

	A	f	B	P	T	C	p	σ_T, σ_S
Mean	80	{7,14,30,90}	200	30	0.3	-12	0.20	2.0, 10.0
Std. Dev.	30	–	80	10	0.1	5	0.05	0.5, 3.0

Figure B.22 illustrates key aspects of the synthetic SALE1 dataset. The left panel shows the distribution of store-specific parameters such as promotion effects, baseline sales, and trend coefficients, highlighting the diversity across stores. The right panel presents sample sales trajectories from randomly selected stores, demonstrating realistic seasonality and promotion-driven variability embedded in the generated data.



(a) Distribution of store-specific parameters used for synthetic data (SALE1) generation (e.g., promotion effect, baseline sales, trend).



(b) Sample sales trajectories from randomly selected stores in SALE1. Seasonal and promotional effects are clearly visible.

Figure B.22. Illustration of synthetic dataset (SALE1) generation. Left: parameter distribution across stores. Right: sample time series showing temporal dynamics.

Synthetic Sale data - Sale2. We simulate realistic sales patterns across 200 stores by parameterizing each store with a high-dimensional vector of behavioral and structural attributes sampled independently. Each is parameterized by independent and identically distributed (i.i.d) parameters drawn from normal distributions, where the mean and standard deviation are shown in Table B.34. For each store i, we define the daily sales $y_{i,t}$ as a combination of trends, seasonality, covariate interactions, and stochastic noise:

$$y_{i,t} = \underbrace{B_i + \alpha_i t + A_i \sin\left(\frac{2\pi t}{f_i}\right) + S_i \sin\left(\frac{2\pi t}{365}\right)}_{\text{trend + intra-cycle + annual seasonality}} + E_{i,t}^{\text{cov}} + \varepsilon_{i,t}$$

Here, B_i is the baseline sales, α_i is the trend coefficient, A_i is the amplitude of intra-cycle seasonality with frequency f_i, and S_i controls the strength of annual seasonality. The term $\varepsilon_{i,t} \sim \mathcal{N}(0, \sigma^2_{S_i})$ is Gaussian noise. The covariate effect $E^{\text{cov}}_{i,t}$ incorporates multiple nonlinear and interactive effects:

$$E^{\text{cov}}_{i,t} = \beta_{\text{promo}} \cdot \text{promo}_{i,t} + \beta_{\text{holiday}} \cdot \text{holiday}_t + \beta_{\text{weekend}} \cdot \text{weekend}_t + \beta_{\text{competitor}} \cdot \text{comp\_promo}_{i,t}$$
$$+ \beta_{\text{inventory}} \cdot \text{inventory}_{i,t} + m_{i,t} + \beta_{\text{temp}} \cdot (1 - |T_{i,t} - T^*_i|) + \beta_{\text{price}} \cdot \text{price}^{\eta_i}_{i,t}$$

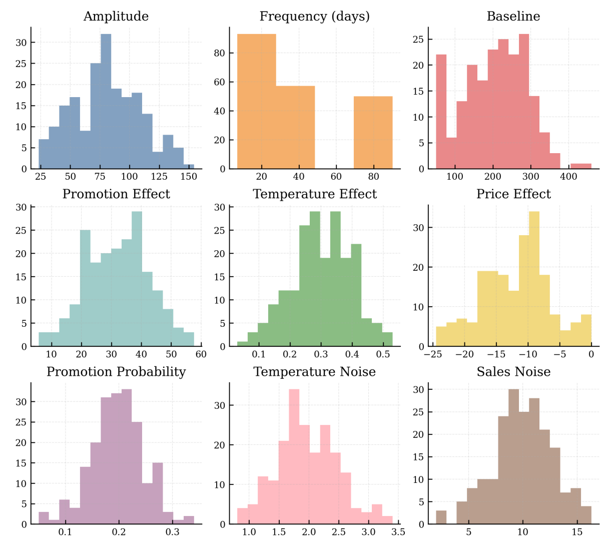
where: $\text{promo}_{i,t}$, holiday_t, and weekend_t are binary indicators for promotion, holiday, and weekend; $\text{comp\_promo}_{i,t}$ indicates competitor promotions; $\text{inventory}_{i,t}$ and $m_{i,t}$ represent inventory effects and exponentially decayed marketing memory, respectively; $T_{i,t}$ is the daily temperature and T^*_i is the store-specific optimal temperature; $\text{price}^{\eta_i}_{i,t}$ models nonlinear price elasticity with elasticity coefficient η_i. These represents known covariates in the dataset.

Table B.34. Parameter distributions used for complex synthetic data generation.

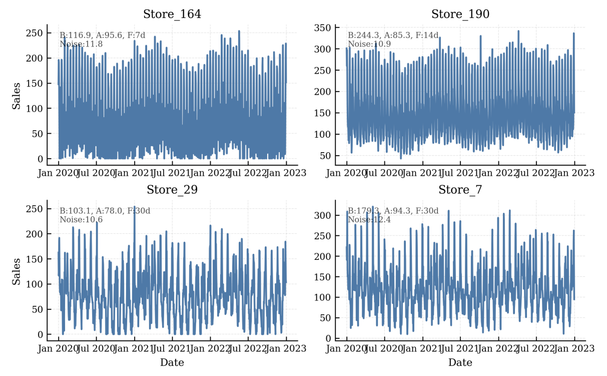
Symbol	$\mathcal{A}$	$\mathcal{B}$	f	β_{promo}	β_{weekend}	β_{holiday}	β_{temp}	T^*	β_{price}	η
Mean	80	200	{7,14,30,90}	30	15	40	0.3	22	-12	1.5
Std	30	80	—	10	5	15	0.1	3	5	0.3
Symbol	β_{comp}	β_{mkt}	λ	β_{inv}	S	α	p	p_{comp}	σ_T	σ_S
Mean	-15	0.4	0.85	0.2	30	0.01	0.2	0.15	2.0	10.0
Std	5	0.1	0.05	0.05	10	0.005	0.05	0.05	0.5	3.0

Table B.23 illustrates key aspects of the synthetic Sale2 dataset. The left panel shows the distribution of store-specific parameters such as promotion effects, baseline sales, and trend coefficients, highlighting the diversity across stores. The right panel presents sample sales trajectories from randomly selected stores, demonstrating realistic seasonality and promotion-driven variability embedded in the generated data.

Synthetic Electricity Load Generation. We simulate regional electricity load patterns using a high-fidelity parametric model that incorporates weather, calendar, infrastructure, and market



(a) Distribution of store-specific parameters used for synthetic data (SALE2) generation (e.g., promotion effect, baseline sales, trend).



(b) Sample sales trajectories from randomly selected stores in SALE2. Seasonal and promotional effects are clearly visible.

Figure B.23. Illustration of synthetic dataset (SALE2) generation. Left: parameter distribution across stores. Right: sample time series showing temporal dynamics.

dynamics. For each region r and time t, the electricity load $L_{r,t}$ is modeled as:

$$L_{r,t} = \underbrace{B_r + \alpha_r t}_{\text{trend}} + \underbrace{D_r(t) + W_r(t) + Y_r(t)}_{\text{daily, weekly, yearly patterns}} + E_{r,t}^{\text{cov}} + \varepsilon_{r,t}$$

Here, B_r is the base load, α_r is the regional trend (load growth), and $D_r(t)$, $W_r(t)$, $Y_r(t)$ are the daily, weekly, and yearly seasonal patterns. The covariate component $E_{r,t}^{\text{cov}}$ accounts for complex external influences:

$$\begin{aligned} E_{r,t}^{\text{cov}} &= \beta_{\text{temp}} \cdot f_T(T_{r,t}) + \beta_{\text{humid}} \cdot H_{r,t} + \beta_{\text{wind}} \cdot W_{r,t} + \beta_{\text{solar}} \cdot S_{r,t} \\ &\quad + \beta_{\text{weekend}} \cdot \text{Weekend}_t + \beta_{\text{holiday}} \cdot \text{Holiday}_t + \beta_{\text{DST}} \cdot \text{DST}_t + \beta_{\text{outage}} \cdot \text{PlannedOutage}_t \end{aligned}$$

The function $f_T(\cdot)$ captures the nonlinear sensitivity to temperature using region-specific parameters (e.g., piecewise or quadratic effects). Additional noise $\varepsilon_{r,t} \sim \mathcal{N}(0, \sigma_r^2)$ accounts for random fluctuations. Additionally, the nonlinear effect of temperature on load is modeled using asymmetric thresholds for heating and cooling:

$$f_T(T) = \gamma_{\text{cool}} \cdot \max(0, T - T^C)^2 + \gamma_{\text{heat}} \cdot \max(0, T^H - T)^2,$$

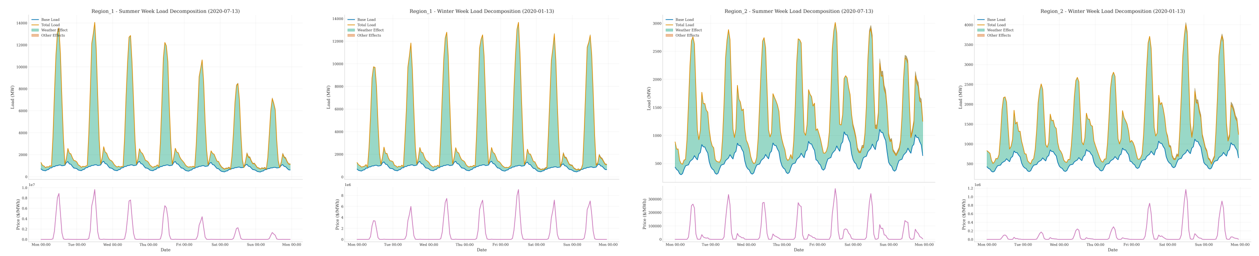
where: T is the observed temperature; T^C is the cooling threshold (e.g., 22°C); T^H is the heating threshold (e.g., 15°C); γ_{cool}, γ_{heat} are the sensitivity coefficients for cooling and heating loads. This formulation captures the fact that electricity demand rises nonlinearly when temperatures deviate from the comfort band.

For market simulation, we also define a dynamic electricity price:

$$P_{r,t} = \beta_{\text{base}} + \beta_{\text{peak}} \cdot \mathbb{I}[L_{r,t} > \text{Cap}_r]^{\gamma_r} + \nu_{r,t}$$

Here, $\mathbb{I}[L_{r,t} > \text{Cap}_r]$ is an indicator for capacity exceedance, with exponent γ_r modeling price spikes, and $\nu_{r,t} \sim \mathcal{N}(0, \sigma_P^2)$ denotes price noise. The framework supports renewable volatility and substitution through additional interaction terms.

Table B.35 summarizes the parameter ranges used to generate realistic electricity load and price patterns across regions. These ranges govern variability in demand behavior, weather sensitivity, calendar effects, and market responses, enabling the simulation of heterogeneous grid conditions reflective of residential, commercial, industrial, and mixed-use regions.



(a) Region 1 — Summer (b) Region 1 — Winter (c) Region 2 — Summer (d) Region 2 — Winter

Figure B.24. Load decomposition over one representative week for two regions across summer and winter. Each panel visualizes base demand, weather influence, calendar signals, and total load.

Table B.35. Parameter ranges used to generate region-specific electricity load and price patterns.

Category	**Symbol**	**Min**	**Max**
Base Load & Seasonality	B_r (base load, MW)	500	5000
	A_{year} (yearly amplitude)	0.15	0.35
	α_r (load trend)	-0.02	0.04
Temperature Sensitivity	T^* (optimal temp)	18.0	23.0
	T^C (cooling threshold)	22.0	27.0
	T^H (heating threshold)	12.0	18.0
	γ_{cool} (cooling slope)	0.02	0.06
	γ_{heat} (heating slope)	0.01	0.04
Weather Sensitivity	β_{humid}	0.001	0.005
	β_{wind}	-0.005	0.002
	β_{solar}	-0.02	-0.005
Calendar Effects	β_{weekend}	-0.25	-0.05
	β_{holiday}	-0.30	-0.10
	β_{DST}	-0.05	0.05
Infrastructure Effects	β_{outage}	-0.20	-0.05
Renewables	β_{renew} (substitution)	0.20	0.80
	$V_{r,t}$ (volatility)	0.01	0.05
Price Model	β_{base} (price base)	20	60
	β_{peak} (peak multiplier)	1.5	4.0
	σ_P (price volatility)	0.05	0.20
Capacity Constraints	Cap_r	0.80	0.95
	γ_r (price exponent)	1.5	3.0
Noise Components	σ_L (load noise)	0.01	0.05
	σ_P (price noise)	0.05	0.15

Figure B.24 visualizes the weekly decomposition of electricity load across two regions under contrasting seasonal conditions. The top row shows Region1's load components during summer and winter weeks, while the bottom row depicts the same for Region2. These plots highlight the distinct roles of temperature, calendar effects, and renewable volatility in shaping regional demand, and emphasize how seasonal dynamics interact differently across geographic and structural profiles.

B.5.5. Robustness on Real-world datasets

We evaluate the robustness of Hopformer on the real-world EPF dataset (Wang et al., 2024b) by varying the context length and prediction length. The results show that integrating Hopformer with Chronos consistently improves the performance of vanilla Chronos across all settings—zero-shot, full fine-tuning, and LoRA fine-tuning. Note that the performance of Hopformer variants differs slightly from the main experiment section, as we conducted additional hyperparameter tuning for the GBDT-based covariate regressors (e.g., XGBoost, LightGBM) to ensure a fairer comparison.

Table B.36. Forecasting performance of Hopformer, Chronos-bolt-small (Chronos), and their hybrid variants on the EPF dataset across varying **context length**. Each cell reports MASE. Bold values highlight the two best-performing models per row. The prediction length is fixed at 24 and number of rolling window is 20. The number of gradient steps for full fine tuning and LoRA fine tuning is 100.

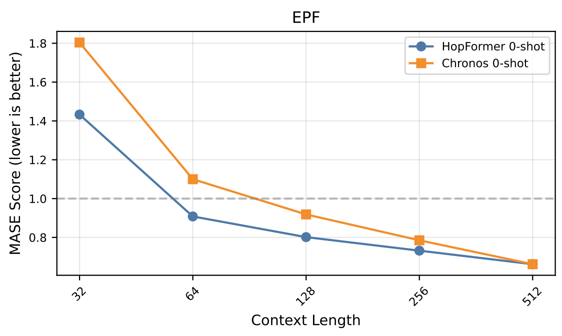
Models		**Hopformer**			**Cross-Sectional**				**Chronos**		
Variants		0-shot	Full	LoRA	SPA	Lasso	Best	Equal	0-shot	Full	LoRA
EPF	32	1.433	**1.423**	**1.429**	2.033	2.017	2.203	2.133	1.804	1.804	1.804
	64	0.908	**0.901**	**0.904**	1.343	1.327	1.513	1.444	1.100	1.091	1.096
	128	0.802	**0.794**	**0.795**	1.175	1.161	1.323	1.287	0.918	0.914	0.915
	256	0.732	**0.725**	**0.727**	1.135	1.116	1.218	1.174	0.785	0.787	0.788
	512	0.662	**0.655**	**0.658**	1.141	1.118	1.033	0.986	0.662	0.672	0.671

Table B.36 summarizes the Mean Absolute Scaled Error (MASE) of Hopformer (built on top of Chronos) and vanilla Chronos when the available context ranges from 32 to 512 time steps. Table B.37 summarizes the MASE of the models across varying prediction length. Four findings stand out. (i)**Zero-shot performance**: Hopformer consistently beats Chronos at every context length, with the largest gain ($\sim 37.1\%$) at the shortest window of 32 steps, as visualized in Figure B.25. This suggests that the covariate-driven expert pool provides valuable signal when historical information is scarce. (ii) **Fine-tuned performance**: After full-parameter or LoRA fine-tuning, Hopformer still yields lower error than Chronos. Removing covariate effects in the first stage appears to simplify

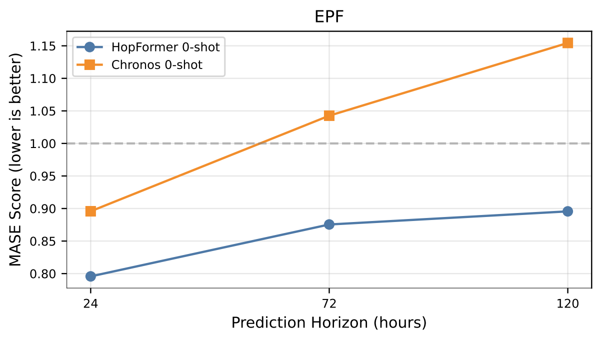
Table B.37. Forecasting performance of Hopformer, Chronos-bolt-small (Chronos), and their hybrid variants on the EPF dataset across varying **prediction length**. Each cell reports MASE. Bold values highlight the two best values per row. The context length is fixed at 256. The number of rolling windows is 10 for prediction length of 24 and 72, and is 4 for 120. The number of gradient steps for full fine tuning and LoRA fine tuning is 100.

Models		**Hopformer**			**Cross-Sectional**				**Chronos**		
Variants		0-shot	Full	LoRA	SPA	Lasso	Best	Equal	0-shot	Full	LoRA
EPF	24	0.796	**0.786**	**0.789**	1.209	1.188	1.299	1.194	0.896	0.904	0.904
	72	0.875	**0.864**	**0.866**	1.625	1.601	1.317	1.293	1.043	1.022	1.036
	120	**0.896**	0.919	**0.916**	1.856	1.822	1.354	1.363	1.154	1.165	1.151

the residual dynamics, making the subsequent transformer easier to adapt. (iii) **Cross-sectional aggregation**: In this two-covariate setting, SPA and Lasso deliver comparable accuracy, indicating that with a very small covariate set the sparsity prior in SPA offers little advantage over a standard ℓ_1-penalised regression. (iv) **Long-horizon prediction**: Hopformer also surpasses Chronos at every horizon, as visualized in Figure B.25. The largest gain ($\sim 25.8\%$) locates at the longest horizon of 120 steps, again demonstrating the value of the covariate-driven expert pool as the prediction window expands.



(a) Effect of context length on forecasting.



(b) Effect of prediction horizon on forecasting.

Figure B.25. Model robustness across varying context lengths and forecast horizons. Only the informative portion of each figure is shown for clarity.

B.5.5.1. Limitation and Discussion. Hopformer is designed to leverage future covariates; when these are absent its benefit naturally diminishes. Although Hopformer achieves strong results on most benchmarks, two practical constraints limit the breadth of our evaluation.

Illness and M5: On the Illness benchmark (Wang et al., 2024b) the series come with seven past covariates but no future ones, so the cross-sectional stage receives little forward-looking signal and Hopformer largely reduces to its residual transformer, yielding only marginal gains over Chronos. The situation is different for the M5 (Makridakis et al., 2022) competition data: rich item-level covariates are available, yet the full dataset contains around 30K hierarchically linked series and several million training points. Training the required gradient-boosted covariate regressors at that scale demands tens of gigabytes of system RAM and many hours of hyper-parameter search, which exceeded the fixed computational budget for this submission. Handling such industry-size datasets will require memory-efficient regressors (e.g., online trees) or sharded training pipelines—an important direction for future work.

Datasets without covariates: For univariate or multivariate series (Aksu et al., 2024; Godahewa et al., 2021; Makridakis and Hibon, 2000; Makridakis et al., 2018) that lack future covariates, the cross-sectional stage collapses to a zero-prediction expert, making Hopformer equivalent to its residual transformer sub-model (e.g., Chronos). Because this setting provides no opportunity to test the proposed aggregation mechanism, we omit those results from the main paper and treat pure-series forecasting as an orthogonal problem. Extending Hopformer with automated feature extraction or self-supervised pretraining may restore an advantage in covariate-free domains, and we plan to investigate this in future work.

B.5.6. Visualization of Hopformer Inference

Figure B.26 decomposes Hopformer's inference pipeline on the Sale1 benchmark. A clear pattern emerges: once the covariate signal (row 3) including promotional, temperature, and price effect is removed from input time series (row 2), the remaining series (row 3) become markedly smoother

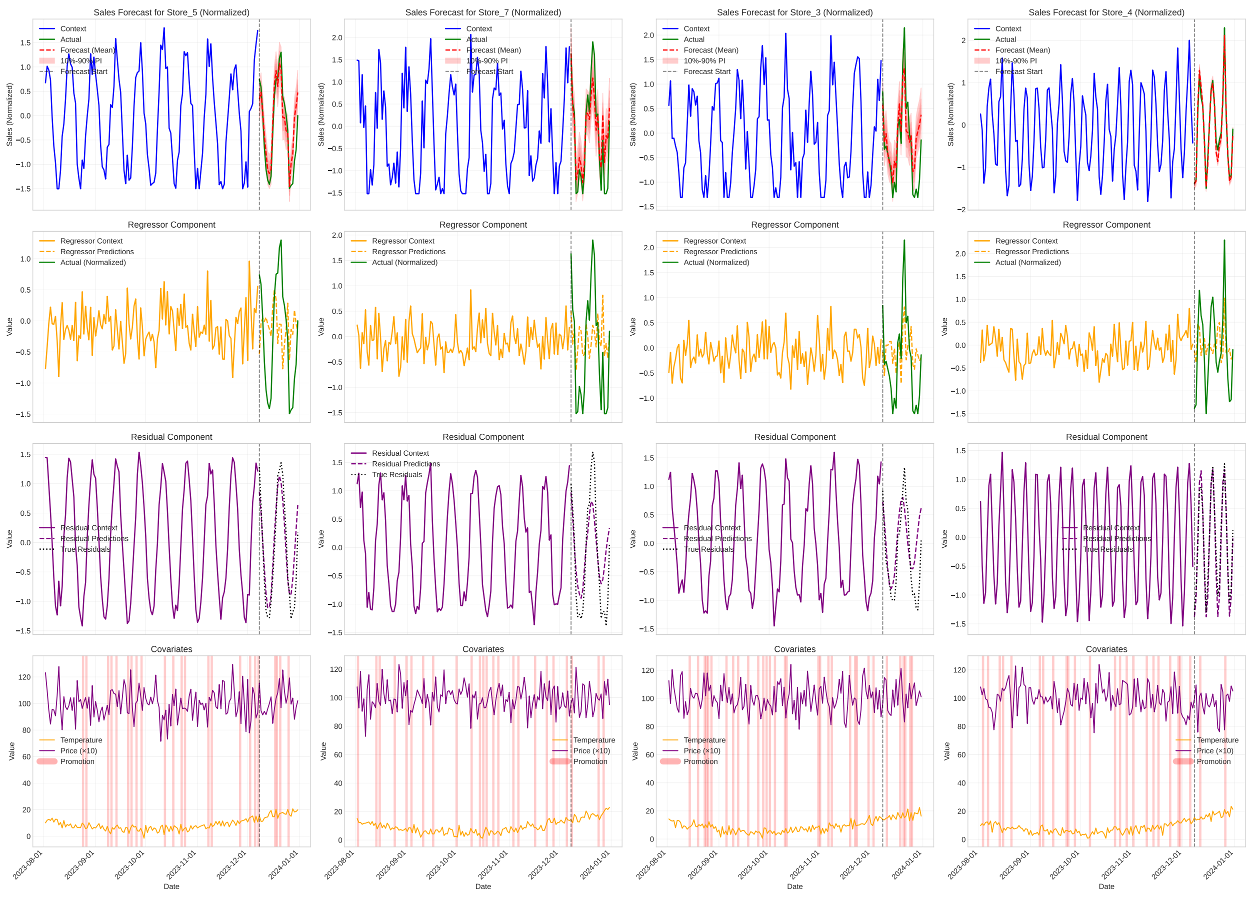


Figure B.26. Decomposition of Hopformer's prediction pipeline on the SALE1 dataset. Each column corresponds to a store. **Row 1**: ground truth and final forecast; **Row 2**: cross-sectional stage (covariate aggregation); **Row 3**: residual transformer output; **Row 4**: covariate trajectories. Removing covariate effects (Row 1 $\rightarrow$ 3) yields a smoother, quasi-periodic residual series, illustrating the division of labour between the two stages.

and more periodic, with promotional spikes eliminated. This confirms that the expert pool has successfully captured most exogenous variation, leaving the residual module to be a simpler and precisely structured pattern.

This qualitative evidence complements our quantitative results: Hopformer isolates high-variance exogenous effects in the first stage and hands a simpler forecasting task to the residual transformer, leading to the accuracy gains reported in Method Section.

B.5.7. Code usage

Here, we desribe the simple usage of Hopformer library [18] The `Predictor` wrapper exposes Hopformer through the same API as `autogluon.timeseries.Predictor`, so existing AutoGluon (Shchur et al., 2023) scripts require only a one–line replacement. Listing 1 trains a Hopformer model on the SALE1 dataset and produces 24-step forecasts. Any AutoGluon regressor (e.g. XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), CatBoost (Prokhorenkova et al., 2018)) can be dropped into the expert pool by editing the `regressor_types` list, and fine-tuning can be toggled with a single Boolean flag.

Listing 1 Minimal hopformer usage. The interface mirrors `autogluon.timeseries.Predictor`, enabling effortless integration into existing pipelines.

```python
from autogluon.timeseries import TimeSeriesDataFrame
from hopformer import Predictor          # Hopformer wrapper

# ---------- load & split data ----------
df = TimeSeriesDataFrame.from_path("store_sales_data.csv")
train, test = df.train_test_split(prediction_length=24)

# ---------- fit Hopformer ----------
model = Predictor(
    prediction_length       = 24,
    target                  = "target",
    known_covariates_real   = ["promotion", "temperature", "price"],
    regressor_types         = ["LR", "XGB", "GBM", "CAT"],
    aggregation_strategy    = ("SPA", {}),
    bolt_model_path         = "bolt_small",
)
model.fit(train, time_limit=120, fine_tune=True, use_lora=True)

# ---------- forecast ----------
context   = test.slice_by_timestep(-512-24, -24)    # context window
konwn_cov = test.slice_by_timestep(-24, None)       # future covariates
forecast  = model.predict(data=context, known_covariates=konwn_cov)
forecast.to_csv("forecasts.csv")
```

Table B.38. Forecasting performance (MASE and MAPE; lower is better) of four state-of-the-art foundation models—Chronos, Moirai, Moirai-MoE, and Lag-Llama—before and after integration with Hopformer's cross-sectional stage (denoted as "Hop"). "Uni" and 'Multi" denotes univariate and multivariate time series prediction. Experiments use 0-shot setting, a context length of 256, a forecast horizon of 24, and metric is averaged over 20 rolling windows. Moirais and Lag-Llama use 100 samples per forecast. Boldface highlights the best best-performing methods for each set of model variants. Highlighting Hop only for clearer view.

Models		**Chronos**		**Moirai**			**Moirai-moe**			**Lag-Llama**	
Variants		Uni	Hop	Uni	Mutli	Hop	Uni	Mutli	Hop	Uni	Hop
EPF	MASE	0.785	**0.732**	0.952	0.915	**0.821**	0.886	0.845	**0.768**	1.168	**0.773**
	MAPE	1.414	**1.355**	1.245	1.427	1.550	1.363	1.312	1.580	2.387	**1.626**
Elec.	MASE	0.706	**0.668**	1.185	1.199	**1.125**	0.994	1.029	**0.939**	1.421	**1.395**
	MAPE	0.074	**0.070**	0.128	0.125	**0.123**	0.109	0.116	**0.103**	0.172	**0.167**
Sale1	MASE	0.577	**0.576**	1.274	1.187	1.337	2.721	1.923	**1.878**	1.217	**1.183**
	MAPE	0.562	0.562	1.217	1.225	**1.073**	0.738	0.798	**0.714**	1.013	**0.885**
Sale2	MASE	0.561	**0.326**	0.702	0.707	**0.482**	0.652	0.705	**0.455**	0.702	**0.482**
	MAPE	0.555	**0.399**	0.875	0.956	**0.636**	0.703	0.781	**0.580**	0.881	**0.598**

B.5.8. Ablation Study: Hopformer with Moirai and Lag-Llama

Table B.38 investigates the effectiveness and generality of Hopformer's two-stage forecasting framework, particularly assessing the gain from its cross-sectional stage across time series foundation models—Chronos-bolt-small(Chronos), Moirai-small, Moirai-MoE-small, and Lag-Llama.

Three observations are particularly notable. **(i)** The inclusion of Hopformer's cross-sectional aggregation consistently improves forecasting performance across all four foundational models and datasets, highlighting its generalizability. For instance, on the EPF dataset, integrating the cross-sectional stage (Hop) reduces the Chronos MASE from 0.785 to 0.732 (5.3% improvement), Moirai from 0.952 (multi) to 0.821 (13.1%), Moirai-MoE from 0.886 (multi) to 0.768 (11.8%), and Lag-Llama from 1.168 to 0.773 (39.5%). **(ii)** These improvements underscore the robustness and broad compatibility of the Hopformer framework: regardless of differences in architectural complexity (e.g., MoE vs. standard transformer models), adding a dedicated cross-sectional aggregation stage effectively isolates and leverages covariate information. Such results demonstrate that the Hopformer

[18]Data and code are available at `https://tinyurl.com/32rn9dvy`.

paradigm—extracting and modeling residuals after explicit covariate adjustment—is a universally beneficial design choice, significantly boosting the predictive power of contemporary foundation models. **(iii)** Notably, Hopformer-enhanced models often outperform the native multivariate versions of Moirai and Moirai-MoE. For instance, on the Sale2 dataset, Hopformer applied to Chronos achieves a MASE of 0.326, compared to 0.707 and 0.705 from multivariate Moirai and Moirai-MoE respectively. This suggests that Hopformer offers a more effective and interpretable way to incorporate exogenous information than direct multivariate modeling. We visualize the forecasting results in SectionB.5.10.

B.5.9. Ablation Study: Fine Tuning Time Series Foundational Models

Through this ablation study, we provide insight into how finetuning could improve the forecasting performance of two state-of-the-art foundational models. Table B.39 compares the effectiveness of zero-shot forecasting with that of full-shot forecasting and of forecasting with a model that was finetuned using LoRA. All experiments use a context length of 512, a forecast horizon of 24, 100 Monte Carlo samples per forecast, and 20 rolling windows. Note that we performed zero shot forecasting on a computer different from the one used for zero-shot forecasting in the ablation study in Section B.5.8. For LoRA finetuning, we update only the query, key, value, and output projection matrices with rank=8, scaling factor $\alpha = 16$, and a 5 % dropout rate, in 100 gradient steps; full-shot models likewise train for 100 steps on all parameters.

Although the results varied across models and datasets, in most cases, the LoRA-finetuned model matches or even slightly outperforms the full-shot variant. For example, on the EPF dataset, full-shot Chronos attains an MASE of 0.674 and an MAPE of 1.252, while LoRA finetuning nearly matches in performance with MASE = 0.720 and MAPE = 1.265. An instance in which we see LoRA surpassing full-shot finetuning is Moirai on the EPF dataset: LoRA has MASE 0.709 and MAPE 0.108, which outperforms full-shot finetuning (0.849/0.127). These results demonstrate that a lightweight, 100-step LoRA update can recover—and in several cases exceed—the accuracy of full-shot training, though its benefit varies by dataset.

Table B.39. Forecasting performance (MASE and MAPE; lower is better) of Chronos and Moirai on the four datasets. Boldface highlights the best-performing metsosohod for each variant.

Models		**Chronos**			**Moirai**		
Variants		0-shot	Full	LoRA	0-shot	Full	LoRA
EPF	MASE	**0.662**	0.674	0.720	0.809	0.849	**0.709**
	MAPE	**1.180**	1.252	1.265	0.119	0.127	**0.108**
Sale1	MASE	1.095	**0.927**	0.971	0.621	0.527	**0.500**
	MAPE	0.951	**0.679**	0.707	0.823	0.677	**0.580**
Sale2	MASE	0.542	0.307	**0.301**	0.566	**0.545**	0.557
	MAPE	0.518	0.315	**0.309**	2.440	**2.222**	4.452
Elec.	MASE	0.817	0.770	**0.756**	0.984	**0.757**	0.870
	MAPE	0.083	0.079	**0.078**	**0.402**	0.407	0.442

B.5.10. Forecasting Visualizations from Table B.38.

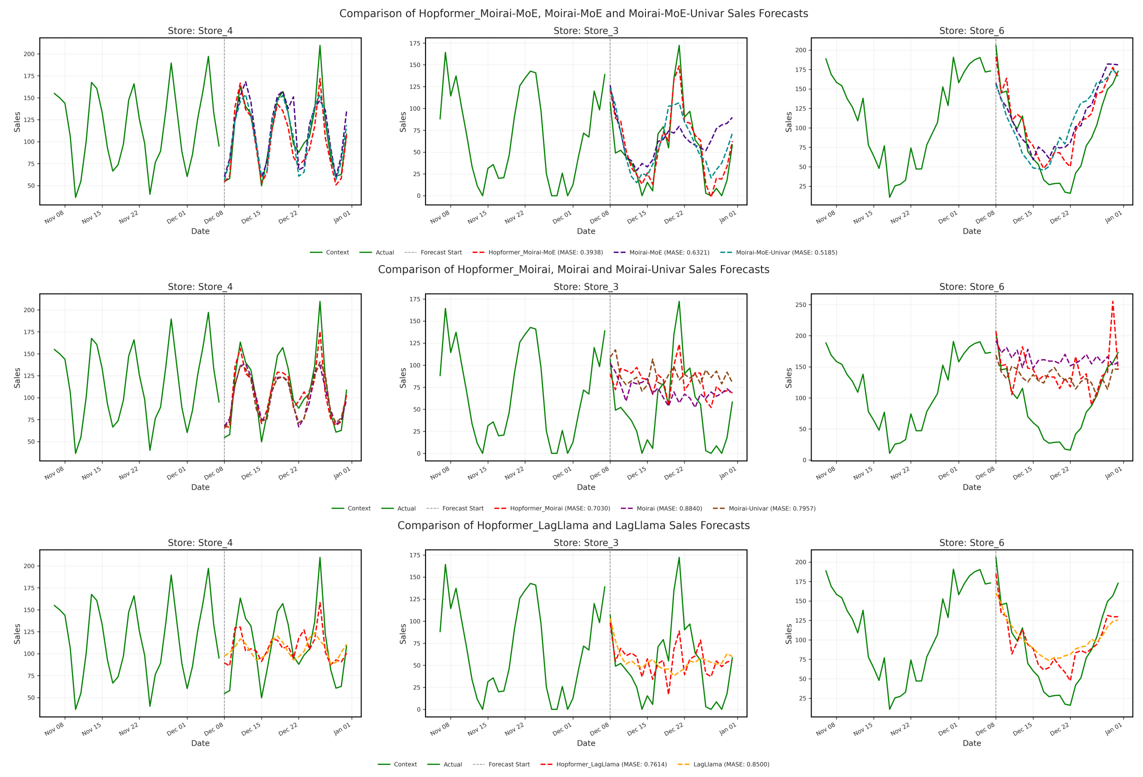


Figure B.27. Forecasting results on the `Sales1` dataset. top: Moirai-MoE; middle: Moirai; bottom: Lag-Llama.

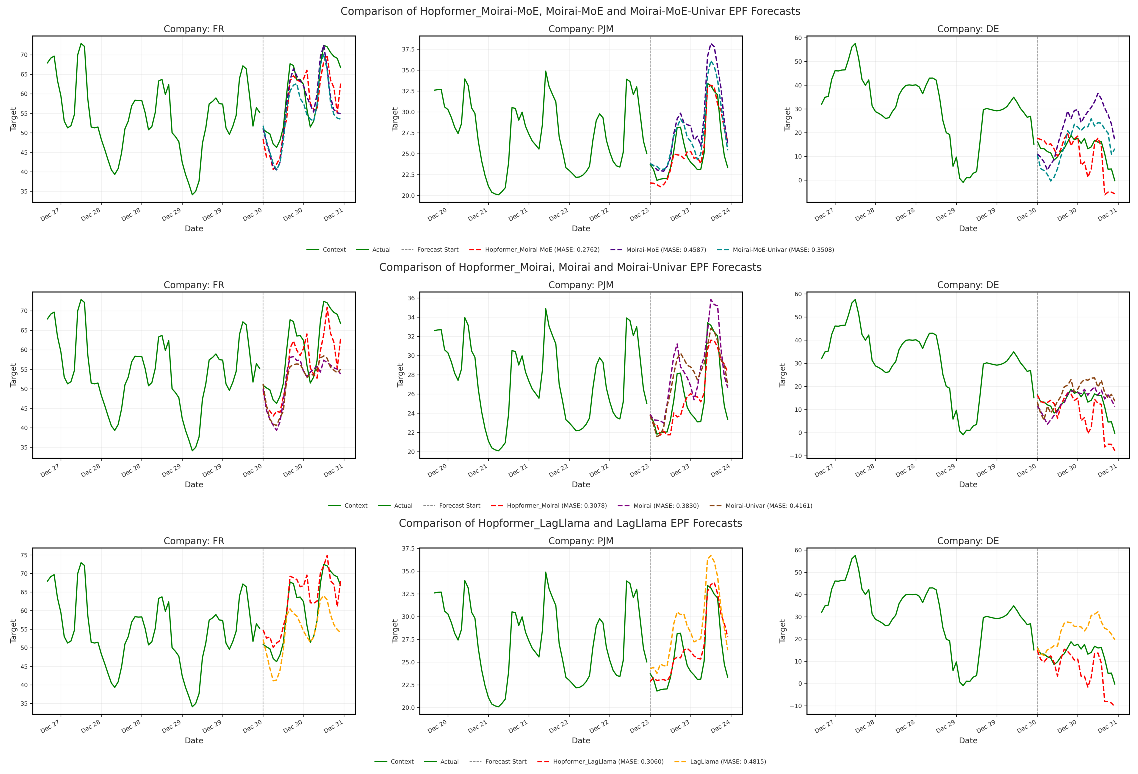


Figure B.28. Forecasting results on the `EPF` dataset. top: Moirai-MoE; middle: Moirai; bottom: Lag-Llama.

APPENDIX C

Supplement Materials for Chapter 4

C.1. StarEmbed: A Benchmark for Evaluating Pre-trained Light-Curve Embeddings in Variable Star Classification

C.1.1. Additional Experiments of Random Forest Classifier

In this section, we attach additional experiment results of using a random forest classifier instead of a MLP classifier to train on the embeddings.

Table C.1. Classification results of the **random forest classifier** with metrics including accuracy, precision, recall and F1 score. Each metric is averaged across different classes for each model. Each experiment repeats three times with random seed=42,100,200 respectively and the numbers show the mean (std).

Methods	Accuracy	Precision	Recall	F1
Astromer	0.6755 (0.0001)	0.0966 (0.0000)	0.1428 (0.0000)	0.1152 (0.0000)
Baseline RNN	0.6900 (0.0383)	0.5000 (0.0002)	<u>0.6600 (0.0089)</u>	0.5500 (0.0057)
Moirai-small	0.8220 (0.0012)	<u>0.7230 (0.0007)</u>	0.5082 (0.0048)	0.5507 (0.0030)
Chronos-tiny	<u>0.8620 (0.0008)</u>	0.7204 (0.0042)	0.5974 (0.0025)	<u>0.6363 (0.0022)</u>
Chronos-Bolt-tiny	0.8254 (0.0005)	0.7051 (0.0015)	0.5454 (0.0006)	0.5799 (0.0005)
Random Embeddings	0.6758 (0.0000)	0.0965 (0.0000)	0.1429 (0.0000)	0.1152 (0.0000)
Handcrafted Features	**0.9193 (0.0003)**	**0.8651 (0.0033)**	**0.7720 (0.0010)**	**0.8034 (0.0014)**

Table C.2. Classification results of the **MLP classifier** with metrics including accuracy, precision, recall and F1 score (**Add Astromer 2 over table 4.4**). Each metric is averaged across different classes for each model. Each experiment repeats three times with random seed=42,100,200 respectively and the numbers show the mean (std).

Methods	Accuracy	Precision	Recall	F1
Astromer 2	0.5345 (0.1056)	0.3720 (0.0364)	0.5384 (0.0406)	0.3875 (0.0514)

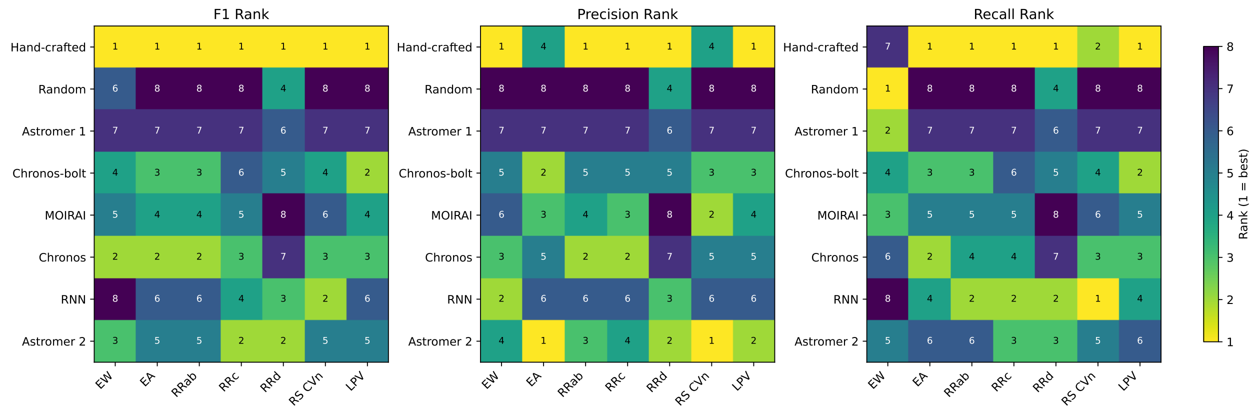
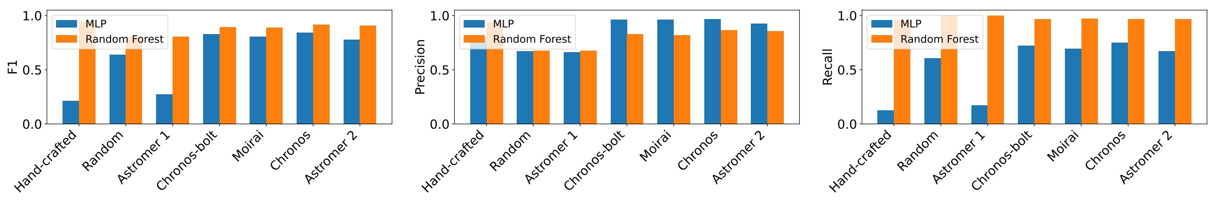
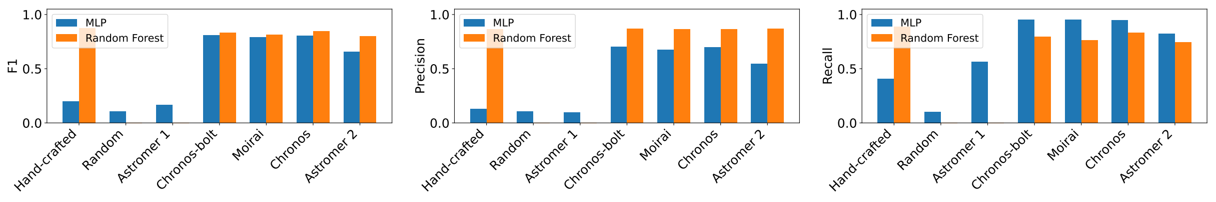


Figure C.1. Ranking of classification performance using the **random forest classifier** on each class on each metric. The results show that the random forest classifier with hand-crafted features achieves top ranks in most of the classes and time series pretrained models and RNN are ranked the second mostly.



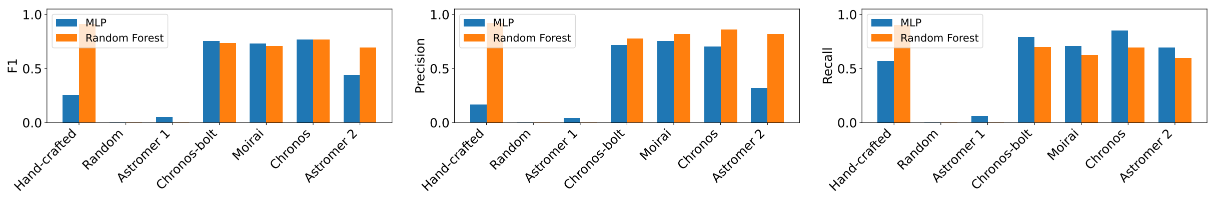
(a) F1 (b) Precision (c) Recall

Figure C.2. MLP vs. random forest performance for class **EW**.



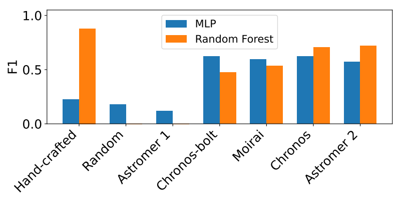
(a) F1 (b) Precision (c) Recall

Figure C.3. MLP vs. random forest performance for class **EA**.

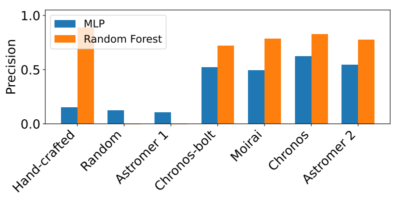


(a) F1 (b) Precision (c) Recall

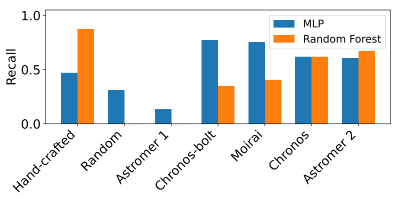
Figure C.4. MLP vs. random forest performance for class **RRab**.



(a) F1

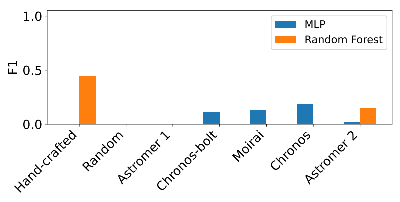


(b) Precision

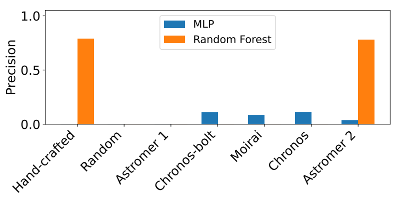


(c) Recall

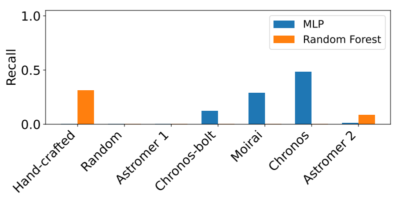
Figure C.5. MLP vs. random forest performance for class **RRc**.



(a) F1

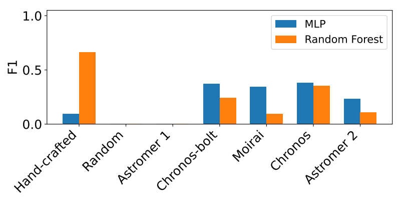


(b) Precision

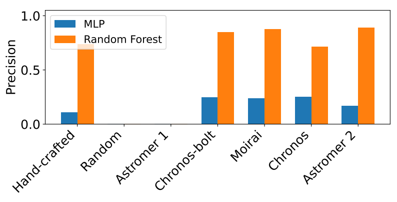


(c) Recall

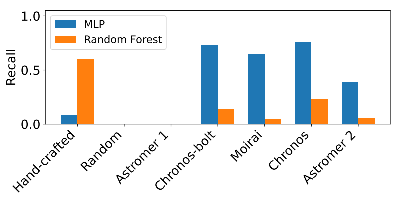
Figure C.6. MLP vs. random forest performance for class **RRd**.



(a) F1

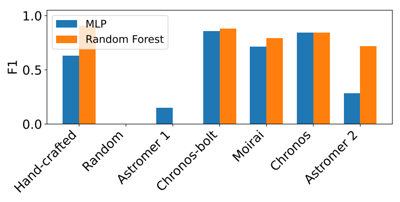


(b) Precision

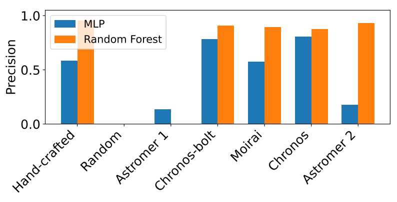


(c) Recall

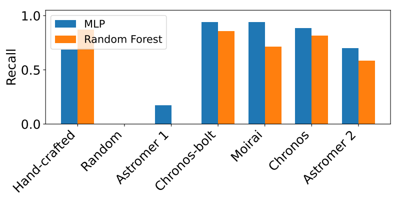
Figure C.7. MLP vs. random forest performance for class **RS CVn**.



(a) F1



(b) Precision



(c) Recall

Figure C.8. MLP vs. random forest performance for class **LPV**.

Figure C.2 to Figure C.8 demonstrate the comparison between the classification results of the MLP classifier and the random forest classifier. Each classifier exhibit advantages over the other on some of the metrics and classes.

Table C.3. Best hyper-parameters for baseline RNN

Method	Hyper-parameters	Training time (s)
Baseline RNN	`batch_size=256, learning_rate=0.0001, dropout=0.0`	90000

Table C.4. Best hyper-parameters for each model with the MLP classifier

Method	Hyper-parameters	Training epochs
Astromer 1	`batch_size=32, learning_rate=0.0001, dropout=0.0`	17
Astromer 2	`batch_size=32, learning_rate=0.0001, dropout=0.0`	23
Moirai-small	`batch_size=64, learning_rate=0.001, dropout=0.0`	11
Chronos-tiny	`batch_size=32, learning_rate=0.0001, dropout=0.0`	26
Chronos-Bolt-tiny	`batch_size=128, learning_rate=0.0001, dropout=0.1`	17
Random Embeddings	`batch_size=128, learning_rate=0.0001, dropout=0.0`	5
Hand-crafted Feat.	`batch_size=32, learning_rate=0.0001, dropout=0.1`	30

Table C.5. Best hyper-parameters for each model with the random-forest classifier

Method	Hyper-parameters	Training time (s)
Astromer 1	`max_depth=10, min_samples_split=2, n_estimators=100`	42.8
Astromer 2	`max_depth=30, min_samples_split=10, n_estimators=500`	450
Moirai-small	`max_depth=None, min_samples_split=2, n_estimators=200`	300
Chronos-tiny	`max_depth=None, min_samples_split=5, n_estimators=200`	300
Chronos-Bolt-tiny	`max_depth=None, min_samples_split=5, n_estimators=500`	630
Random Embeddings	`max_depth=10, min_samples_split=2, n_estimators=100`	42.6
Hand-crafted Feat.	`max_depth=20, min_samples_split=2, n_estimators=100`	32.9

C.1.2. Full List of Hyperparameters

C.1.3. Visualizations of Embeddings

We include the UMAP visualizations for the embeddings from each embedding models to provide more intuitions regarding the embedding space. As shown by Figure C.9, all time series pretrained models' embeddings are showing clear distinction and distribution of different clusters corresponding to different ground truth classes. In comparison, as a baseline, the random embeddings show no clear clusters at all. Astromer 1's embeddings and hand crafted features do not show clear clusters either. Astromer 2's embeddings show clearer cluster distribution but for some classes, the clusters are not distinctive with others either. These UMAP visualizations further demonstrate the promising potentials of using time series pretrained models as light curve embedding models.

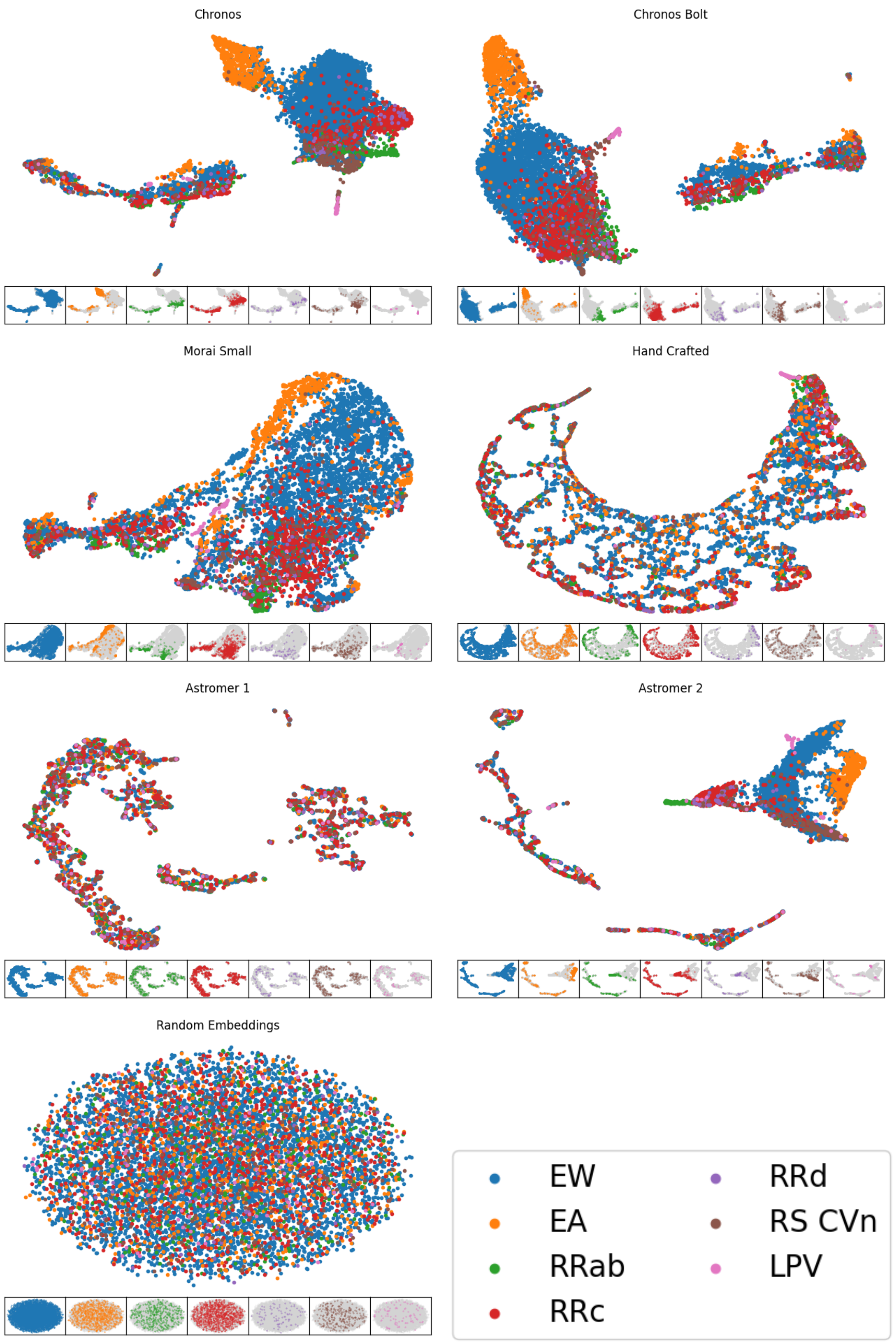


Figure C.9. UMAP projections for each embedding model included in our analysis using the test set. Inset plots at the bottom of each figure show clustering of different classes.

C.1.4. Full List of Hand-crafted Features

We select handcrafted features from the libraries of established software packages: `FATS` (Nun et al., 2015) and `light_curve` (Malanchev et al., 2021). Each feature described here is computed for each passband individually and the embeddings are formed by concatenating the feature lists of the g and r embeddings. Tables C.6 and C.7 show the full list of features and descriptions from `FATS` and `light_curve`, respectively.

Table C.6. `FATS` features and plain-language descriptions

Feature	**Intuitive one-sentence comment**
PeriodLS	Identifies the dominant repetition cycle in a light curve with the Lomb–Scargle periodogram.
Period_fit	Reports how unlikely the detected period is to arise from random noise (lower value = more significant).
Psi_CS	Quantifies how tightly points cluster when the light curve is folded on the detected period.
Psi_eta	Applies the η variability index to the folded curve, indicating scatter within one cycle.
Autocor_length	Gives the time lag after which successive measurements cease to resemble each other.
PairSlopeTrend	Compares rises versus falls in the last 30 points to flag any recent monotonic trend.
Freq{N}_harmonics_amplitude_{M}	Amplitude of the M-th harmonic of the N-th strongest frequency in a multi-sine fit.
Freq{N}_harmonics_rel_phase_{M}	Phase offset of that harmonic relative to its fundamental component (in radians, wrapped to $[-\pi, \pi]$).
CAR_sigma	Measures the short-timescale "jitter" amplitude captured by a continuous-time AR(1) model.
CAR_tau	Characteristic damping timescale over which variations decay in the same CAR(1) model.
CAR_mean	Long-term average magnitude implied by the CAR(1) fit.

See here for a detailed description:
`http://isadoranun.github.io/tsfeat/FeaturesDocumentation.html`

C.1.5. Astrophysical Description of Classes

Our dataset contains seven total classes of periodic variable stars: EW, EA, RRab, RRc, RRd, RS CVn, and LPV. Here, we provide a high-level astrophysical description for each of these classes, including each class' observational characteristics and utility. In some cases, multiple classes are closely related so we describe them together.

C.1.5.1. Eclipsing Binaries (EW, EA). Eclipsing binary stars are pairs of stars orbiting each other and aligned with the observer in such a way that either star periodically blocks the light from the other. When neither star is eclipsed, the system is at maximum brightness, but when one star is eclipsed by the other, the total brightness of the system is suppressed, giving the binary star system periodic light curve behavior. This type of variability is described as extrinsic because it is not due to astrophysical properties of the stars themselves. Sub-categorization of eclipsing binaries is based on the configuration of the stars in the pair.

EW-type eclipsing binaries (also called W Ursae Majoris-type after the original EW system) are contact binaries. In this case, the two stars, typically dwarf stars, share a common outer envelope which entirely encapsulates them. This common envelope allows for the exchange of mass and energy between the pair, equilibrating their temperature. Their light curves exhibit constant and smooth variations where the dips from either star being eclipsed are of similar or identical depth. EW-type variables typically have short periods ($0.2 \lesssim P$ [d] $\lesssim 0.5$), with a notably unsolved period cut-off at ~ 0.2 days (Rucinski, 2007; Drake et al., 2014a). These systems are astrophysically valuable, in part, because they are expected to emit gravitational waves due to their tight orbits, and they also have the possibility of merging and triggering transient events (Tylenda et al., 2011).

EA-type eclipsing binaries (also called Algol-type binaries after the original EA system) are detached binaries. In this case, the two stars are not in contact and thus can have different temperatures and more varied orbits, manifesting as different light curve properties. The ellipticity of the orbit and the brightnesses of the stars affect the spacing and depths of the brightness dips. Multiple additional factors can affect their light curves, e.g., the presence of an accretion disk. EA systems tend to have longer periods than EW systems due to their wider orbits: $0.3 \lesssim P$ [d] $\lesssim 100$. EA systems are key for studying binary stellar evolution and populations, especially the exchange of mass between stars in a binary.

C.1.5.2. Active Binaries (RS CVn). RS Canum Venaticorum (RS CVn) stars are also stars in binary systems and are characterized by one of the stars exhibiting large magnetic spots on its surface. This manifests as observable variability as the RS CVn stars also have rapid rotational velocities. The resulting light curve affect also depends on the difference between the star's rotational period and the systems orbital period, and most RS CVn systems are tidally locked, meaning the two periods are closely matched. Some RS CVn are also eclipsing binaries, so their light curves can also show variability due to eclipses. Their periods tend to be $3 \lesssim P$ [d] $\lesssim 14$. RS CVn systems serve as extreme testbeds for studying stellar magnetic phenomena and evolution.

C.1.5.3. RR Lyrae (RRab, RRc, RRd). RR Lyrae stars are low-mass stars exhibiting pulsations, cyclically expanding and contracting radially due to internal changes in opacity. Because RR Lyrae occur with only a small range of intrinsic brightnesses, their distances can be easily measured from their observed brightness. Among other utilities, this allows RR Lyrae to be used for measuring distances within the Milky Way and to nearby Galaxies. Their periods are also related to their chemical composition, so they can provide crucial information about Galactic structure and formation. Some RR Lyrae, mostly RRab, also exhibit long-term modulations of their light curve amplitudes and phases through the Blazhko effect, although the physical mechanism behind this effect remains uncertain.

RR Lyrae can occur in different pulsation modes which define the various RR Lyrae subclasses. RRab stars pulsate in the fundamental mode; RRc in the first-overtone; and RRd are double-mode pulsators. RRab stars have light curves with a rapid brightening episode followed by a gradual fading, producing a sawtooth-like pattern. RRab typically have periods $0.4 \lesssim P$ [d] $\lesssim 1.0$. RRc stars exhibit sinusoid-like variability with typical periods of $0.2 \lesssim P$ [d] $\lesssim 0.5$. They also tend to exhibit a constant, slow drift in their periods. RRd stars pulsate in both the fundamental mode and the first-overtone and thus show a combination of two periodic signals in their light curves, typically with the first-overtone dominating.

C.1.5.4. Long Period Variables (LPVs). LPVs are giant stars exhibiting pulsations with periods of $3 \lesssim P$ [d] $\lesssim 1000$. They include multiple subtypes each with different period-luminosity relations but we, as in (Drake et al., 2014a), consider these a single class. They pulsate with a similar mechanism to the RR Lyrae but have dramatically larger radii. Their outer layers are not very tightly bound to the rest of the star, which can lead to the star expunging mass and polluting the surrounding environment with gas. Thus, studying LPVs provides insights into the cycles of gas into and out of stars. Their period-luminosity relations also allow LPVs to act as distance measures.

C.1.6. Potentials of Fine-Tuned Time Series Pretrained Models

We perform a fine-tuning study on a small subset of ZTF light-curve data to further explore time series pretrained models' potential capacity on fitting on the light curve data. Taking Chronos as an example, we try fine-tuning Chronos on a small set of ZTF light curves to show that Chronos has the sufficient architecture and capacity to perfectly fit the light curve dynamics.

We select 20 g-band light curves—10 labeled class 1 (EW) and 10 labeled class 5 (RRc) to try fine-tuning the model to do light curve forecasting. We preprocess the irregular time series by removing observations flagged as poor quality, resampling the irregular time series to a uniform 0.0025 MJD grid, picking the single measurement with the smallest magnitude error per bin, and padding data with NaNs when no observation exists. Then, we pre-train Chronos-small model with context length = 128, forecast horizon = 32, token size = 4096, upper limit = 1.2, lower limit = 0.8, and min past = 64.

Figure C.10 illustrates two example forecasts. Top panel shows a success case. After about 50k gradient steps Chronos tracks magnitude variations to within less than or equal to 0.0001 mag MAE, confirming the architecture can memorize a light curve. The bottom panel shows a failure case where there are < 128 valid points, so we posit the model does not have enough context to predict the future. Other failure modes show the opposite extreme: 200 points with high irregularity also lead to poor forecasting. Uniform context sampling during training rarely lands on the same dense

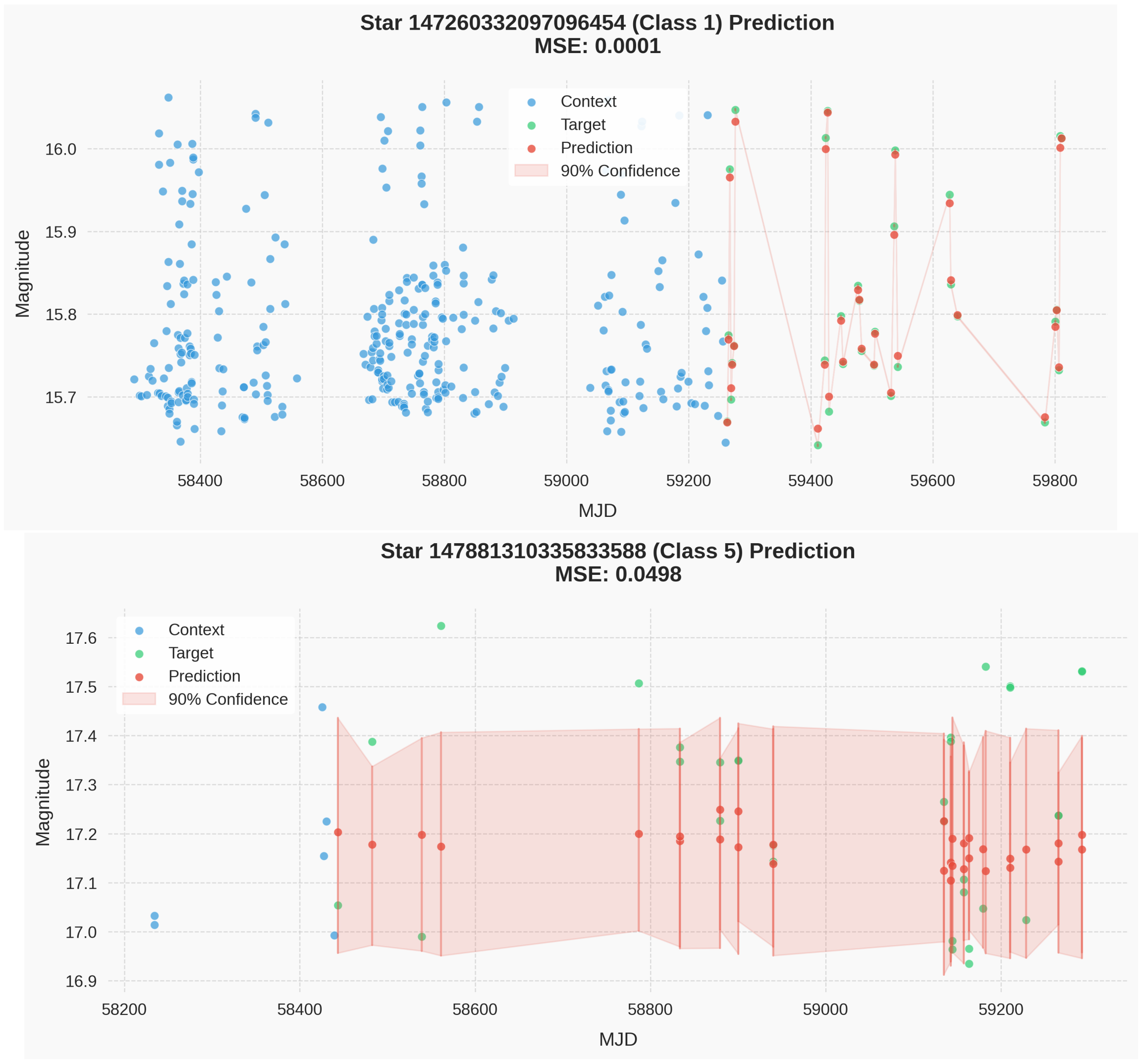


Figure C.10. Chronos forecasts on two example ZTF light curves used in our fine-tuning study.

segments encountered at inference, producing large MSE mtric (>0.005). These errors suggest that although Chronos is capable of perfectly fitting the light curve dynamics, it has some requirements to have a success. Chronos is sensitive to (i) very sparse sequences and (ii) long sequences where the fixed context window captures inconsistent local statistics.

We also conduct further experiments to explore fine-tunign or pre-training potentials of Chronos on the highly irregular ZTF data. Preliminary experiments of directly fine-tuning Chronos on the unprocessed raw time series shows extreme difficulties in decreasing the training loss. The training loss would saturate after first several training steps. Thus, we start with pre-processing the data into a more regular series by segmenting the raw series into a series of 24-hour buckets. Then, we calculate the weighted average of all the magnitude observations in each bucket using the inverse of the measurement error as the weights. After this processing, most consecutive values in the series have a gap of 24 hours and the general error of the series has been lowered. Then, we pick a very small subset from the total dataset and conduct fine-tuning and pre-training on it.

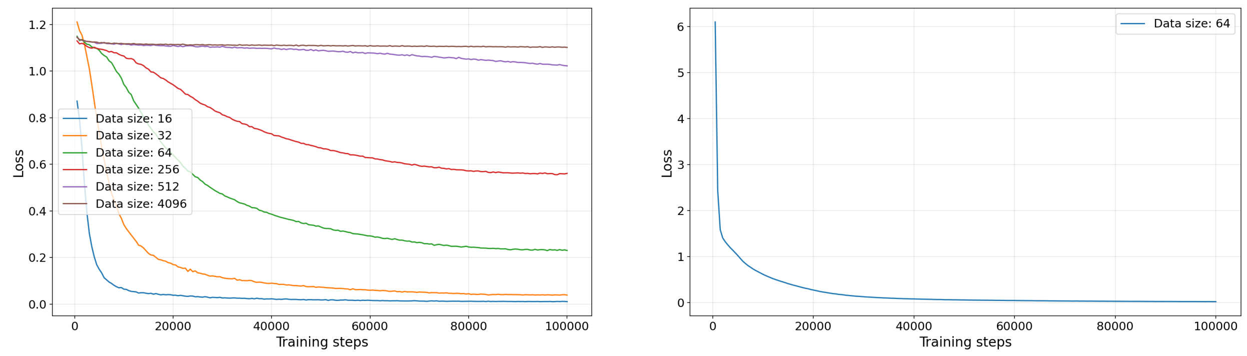


Figure C.11. Left: Training loss dynamics of fine-turning on different data sizes. Right: Training loss dynamics of pre-training on data size 64.

From Figure C.11, we can see that even with a very small data size (less than 5000 light curves), the training loss takes an extremely many steps to reach the saturated stage. As a reference, the Chronos original pre-training process took 200K steps. This demonstrates a limitation on the efficiency of fine-tuning or pre-training Chronos on the ZTF data. Further future work needs to be conducted to significantly improve the efficiency of training on much larger light curve datasets of more than millions of light curves.

Table C.7. `light_curve` features and plain-language descriptions

Feature	**Intuitive one-sentence comment**
Amplitude	Half the peak-to-peak range—how far the light curve swings between brightest and faintest points.
AndersonDarlingNormal	Scores how strongly the magnitude distribution departs from an ideal bell curve.
BeyondNStd	Proportion of data points that sit more than N standard deviations away from the mean, flagging outliers.
Cusum	Total vertical span of the running cumulative sum, revealing slow drifts or trends.
Eta	Von Neumann ratio: compares successive-point differences to overall scatter to catch rapid variability.
EtaE	Eta re-weighted by time gaps so uneven sampling doesn't skew the variability estimate.
InterPercentileRange(p)	Distance between the p and $(1 - p)$ quantiles—a robust width such as the IQR (when p=0.25).
Kurtosis	Indicates whether the distribution is more peaked or heavy-tailed than a normal curve.
LinearFit	Slope, error, and fit quality for a straight line that accounts for measurement uncertainties.
LinearTrend	Slope and error of a simple least-squares line that ignores the error bars.
MagnitudePercentageRatio	Ratio of inner to outer percentile widths, contrasting core spread with overall spread.
MaximumSlope	Steepest single-step change in magnitude per unit time between consecutive points.
Mean	Ordinary average magnitude.
Median	Mid-point magnitude that splits the data into equal halves.
MedianAbsoluteDeviation	Typical absolute distance from the median—a robust scatter measure.
MedianBufferRangePercentage	Fraction of points that fall inside a narrow buffer zone around the median.
OtsuSplit	Statistics describing the two groups produced by Otsu's automatic thresholding of magnitudes.
PercentAmplitude	Largest absolute deviation of any point from the median magnitude.
ReducedChi2	Reduced χ^2 showing how well the data match their (weighted) mean given the quoted errors.
Skew	Tells whether the distribution leans toward brighter or fainter extremes (positive or negative tail).
StandardDeviation	Classical root-mean-square scatter of the magnitudes.
StetsonK	Error-weighted "peakedness" measure that is robust to outliers in light-curve shape.
WeightedMean	Average magnitude that gives greater weight to points with smaller measurement errors.

See here for a detailed description:
`https://github.com/light-curve/light-curve-python`