NORTHWESTERN UNIVERSITY

Efficient Scaling Embodied AI: Systems, Models, and Empirical Laws

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Qinjie Lin

EVANSTON, ILLINOIS

December 2025

ABSTRACT

Efficient Scaling Embodied AI: Systems, Models, and Empirical Laws

Qinjie Lin

Embodied artificial intelligence is undergoing a paradigm shift from single-task systems to general-purpose agents, a transition demanding unprecedented scale. The economics of real-world learning present the obstacle to this goal, creating a critical need for efficiency in both the underlying computational infrastructure and the learning models. This thesis confronts these efficiency challenges through a dual strategy of systematic infrastructure engineering and efficient modeling strategies. We introduce EMS®, a sky computing framework to automate large-scale learning pipelines across heterogeneous resources, and RoboFlow, a containerization system that modularizes workflows to ensure reproducibility. On the modeling front, we develop efficient models, including a Markov game formulation for safe, collision-free navigation and the Switch Trajectory Transformer, which uses a Mixture-of-Experts (MoE) architecture for computationally efficient multi-task learning. Finally, we establish empirical scaling laws for LLM-driven embodied agents to guide resource-efficient research. Together, this work provides an economically viable blueprint to efficient scaling decision-making in embodied AI, paving the way for agents that can operate safely and efficiently in complex, human-centric environments.

Acknowledgements

I would like to express my deepest gratitude to the many people who have supported and guided me throughout my doctoral journey. This thesis would not have been possible without their wisdom, encouragement, and patience.

First and foremost, I am profoundly grateful to my advisor, Han Liu. Thank you for your unwavering guidance through numerous meetings, for teaching me how to write, for offering me this PhD opportunity, and for your steadfast support during my candidate exam, prospectus, and final defense. Your mentorship has been the foundation of this work.

I extend my sincere thanks to my committee members, Manling Li, Stephen Xia, and Xinyu Xing, for their timely feedback and for graciously agreeing to serve on my committee. Your insights have significantly strengthened this thesis.

I am also grateful to the professors I have had the privilege to collaborate with. 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. Thank you to Professor Satish Kumar from University of Southern California's Department of Computer Science; our collaboration at Meta extended my research into hierarchical planning, and you guided me in writing papers and provided innovative ideas. Thank you to Prof. Kai Zhang from the University of North Carolina at Chapel Hill for our collaboration in the time series domain.

My time at Northwestern University, particularly in the Magics Lab, has been enriched by my brilliant colleagues. I am especially grateful to Guo Ye for our close collaboration in Robotics, and to Weijian Li and Hong-Yu Chen in the Time Series domain. Thank you to Nabeel Rehemtulla

and Ved G. Shah for the collaboration on astronomy projects; I learned a great deal from you both. Thank you to Wan Zhang and Chen Lee for our close collaboration on the time series foundational model. I also want to thank all my labmates and friends, including Zhihan Zhou, Tim Tsz-Kit Lau, Jiayi Wang, Chenwei Xu, Weimin Wu, Jerry Yao-Chieh Hu, Mattson Thieme, Ning Wang, Qiankai Cao, Yukun Ma, Robin Luo, Dennis We, Lining Mao, and many others. Additionally, I am deeply thankful for my friends. Thank you to Qihua Chen for invaluable support in my downtime and uptime. Thank you to Julia Golden for the endless encouragement, countless dinners, and for always reminding me to celebrate small victories. Thank you to Terri Moi for your patience and support.

Finally, I owe my deepest thanks to my family for their endless love. To my parents, Qingwei Lin and Ruiyan Chen, thank you for your unconditional support. To my brothers, Peijie Lin, Zengjie Lin, and Qinhong Lin, and my grandparents, Meilan Dai and Zifa Lin, thank you for everything.

Statement of Funding Source: During my Ph.D. years (2020 Fall - 2025 Fall), I have been supported by the NSF AI Institute under award AST-2421845-001, gift funds from AbbVie Research and Dolby Research, and by Research and Teaching Assistantships from Northwestern University.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Embodied artificial intelligence is undergoing a paradigm shift from a 'one-model-per-task' approach towards the pursuit of general intelligence. For example, a robot that performs a single, repetitive task is a machine; a robot that understands the abstract concept of 'opening' and can interact with novel doors and containers exhibits general intelligence. This transition mirrors the recent revolution in natural language processing, where a single foundation model like ChatGPT OpenAI (2025) replaced specialized models for sentiment analysis, translation, summarization, etc. For robotics, scalability enables this same leap. It allows for the development of a unified world model that can be taught thousands of skills, rather than building a new system for each one. The field currently operates in a pre-foundation model era, where knowledge gained from one task remains siloed and does not accelerate learning on the next. Existing work Black et al.; Brohan et al. (2023); Li et al. (2024); Ichter et al. (2023); Ahn et al. (2022); Kim et al. (2024) shows promise in pursuing general intelligence, but the field is still on its way, and a successful general embodied AI has not yet been created.

Reaching this goal requires overcoming the fundamental economics of real-world learning. Unlike learning from static web datasets, each data point for an embodied agent incurs a direct time cost from real-time interaction and a financial cost from hardware operation and maintenance. For example, a million interactions in real-world can take weeks or months to collect, whereas a million data points from the web can be processed in minutes. This economic reality makes data efficiency a primary objective. An effective approach requires not only sample-efficient models but also an infrastructure designed as a 'data flywheel', where every experience from a fleet of robots is used to continuously refine a central world model and redeploy the improved intelligence. Realizing this

data flywheel, in turn, depends on both efficient infrastructure for massive-scale computation and modeling design for general intelligence.

This thesis builds the efficient infrastructure and models required to power this data flywheel. Our contribution is threefold. First, we engineer the computational backbone, introducing systems like Roboflow Lin et al. (2022b) and EMS® Lin et al. (2023) to manage the complex data and training loops at scale. Second, we develop efficient models designed for this infrastructure, demonstrating their effectiveness in settings like mobile robot navigation Ye et al. (2020) and multi-task learning Lin et al. (2022a). Finally, we establish general predictive scaling laws for LLM-driven embodied agents, enabling more efficient model development by providing a principled way to forecast performance and guide investment in more complex decision-making architectures.

The value of this integrated approach is twofold. First, it addresses the true system-level bottleneck in robotic learning. A data-efficient model is ineffective if the infrastructure cannot process its experiences and redeploy improvements in a timely manner. Conversely, a rapid data pipeline is wasted on a model that is too sample-inefficient to learn from the data or too computationally heavy for on-device inference. Our co-design methodology resolves this issue. Second, by presenting a solution that considers the end-to-end process—from physical interaction to data processing to model updates—this work provides a blueprint for economic viability. It moves beyond isolated algorithmic improvements to offer a template for engineering a complete, scalable system, establishing a practical path from research to real-world application.

This dissertation is organized by our core contributions. Chapter 2 presents our systems for scalable experimentation: EMS® and RoboFlow. Chapter 3 details our models for safe navigation and efficient multi-task learning. Chapter 4 establishes scaling laws for LLM-driven agents. Finally, Chapter 5 concludes with a summary of contributions and an outlook on future research.

CHAPTER 2

Systematic Infrastructure Engineering

This chapter details the design and implementation of the foundational infrastructure required to enable large-scale, reproducible research in embodied AI. We first introduce EMS®, a sky computing framework designed to automate the management of complex, long-running training pipelines across heterogeneous compute resources, from on-premise GPUs to public cloud instances. EMS®provides the backbone for the ”data flywheel” by efficiently orchestrating data collection, model training, and evaluation at scale. Complementing this, we present RoboFlow, a containerization system that modularizes research workflows. By encapsulating code, dependencies, and configurations, RoboFlow ensures that experiments are reproducible, and easily shareable, addressing a critical bottleneck in collaborative and long-term research projects. Together, these systems provide the robust, scalable, and reproducible infrastructure upon which the subsequent model-centric contributions are built.

2.1. EMS®: A Massive Computational Experiment Management System towards Data-driven Robotics

We propose EMS®, a cloud-enabled massive computational experiment management system supporting high-throughput computational robotics research. Compared to existing systems, EMS® features a sky-based pipeline orchestrator which allows us to exploit heterogeneous computing environments painlessly (e.g., on-premise clusters, public clouds, edge devices) to optimally deploy large-scale computational jobs (e.g., with more than millions of computational hours) in an integrated fashion. Cornerstoned on this sky-based pipeline orchestrator, this paper introduces three abstraction layers of the EMS® software architecture: (i) Configuration management layer focusing on automatically enumerating experimental configurations; (ii) Dependency management layer

focusing on managing the complex task dependencies within each experimental configuration; (iii) Computation management layer focusing on optimally executing the computational tasks using the given computing resource. Such an architectural design greatly increases the scalability and reproducibility of data-driven robotics research leading to much-improved productivity. To demonstrate this point, we compare EMS® with more traditional approaches on an offline reinforcement learning problem for training mobile robots. Our results show that EMS® outperforms more traditional approaches in two magnitudes of orders (in terms of experimental high throughput and cost) with only several lines of code change. We also exploit EMS® to develop mobile robot, robot arm, and bipedal applications, demonstrating its applicability to numerous robot applications [12].

2.1.1. Introduction

We are witnessing the emergence of massive computation as a fundamental strategy in modern robotics research. For example, OpenAI uses 6,144 CPU cores and 8 GPUs to train a robot hand manipulation policyAndrychowicz et al. (2020) for about 50 hours, and Deepmind takes 340 million training steps to train AlphaGoSilver et al. (2016), with 50 GPUs for about 500 hours. More examples include not only traditional hardware robots such as legged locomotionHwangbo et al. (2019); Bouman et al. (2020); Gangapurwala et al. (2021); Tsounis et al. (2020); Heess et al. (2017); Tan et al. (2018); Xie et al. (2021); Nachum et al. (2019); Peng et al. (2020), manipulationGu et al. (2017); Pinto and Gupta (2016); Mahler et al. (2016); Nagabandi et al. (2020); Stengel-Eskin et al. (2022); Strudel et al. (2020); Lee et al. (2017); Bateux et al. (2018); Florence et al. (2022); Mees and Burgard, and navigationKahn et al. (2018); Shacklett et al. (2021); Houston et al. (2020); Surmann et al. (2020); Ye et al. (2020); Fang et al. (2019); Savinov et al. (2018); Gupta et al. (2017); Lyu et al. (2022), but also software robots like conversational agentsFloridi and Chiriatti (2020); Devlin et al. (2018); Radford et al. (2018); Liu et al. (2019); Sanh et al. (2019) and game AIBerner et al. (2019); Mnih et al. (2013); Team (2019); Mathieu et al. (2021); Jiang et al. (2018). Though

[1]Project Website: sites.google.com/view/project-emsr
[2]Documentation: github.com/emsr-project/emsr_examples

these research fields are seemingly diversified, they address the same scientific question: "Which configuration is optimal for a given utility?" A principled approach that systematically answers this question is massive computational experimentsMonajemi et al. (2016), which complements two other important scientific methods for modern robotics research: mathematical analysis and physical experimentation. Compared to massive computational experiments, mathematical analysis can not be conducted in many realistic settings since it generally requires stylized simplification of formal models, while physical experimentation is not applicable on larger scales due to its expensiveness in time and budget. To implement the massive computation strategy, the most crucial step is to develop an experiment management system Monajemi et al. (2016).

Though some general-purpose experiment management systems have been developedMonajemi et al. (2016); cod; pyw, they are not directly applicable to modern robotics research due to the challenge that a sophisticated intelligent robot pipeline generally requires the usage of heterogeneous computational resources ranging from edge devices, on-premise deployment, public cloud, etc. In contrast, most existing systems are cluster or cloud-specific. To illustrate the multi-cloud nature of data-driven robotics applications, we consider an example of developing a reinforcement learning-based retail navigation robot. In this application, data collection should be run in an edge robot network, the decision model could be deployed on some public cloud (e.g., AWS, GCP, Azure), while the very intensive data storage has better to be on some on-premise cluster due to the budget concern. In this scenario, developers need to repeatedly transfer data between different components in the pipeline. More sky-based pipelines are illustrated in Figure 2.1.

To handle the above challenge, we propose EMS®(Experiment Management System for Robots), a cloud-enable massive computational experiment management system for robotics research (See Figure 2.2 for more details). Unlike most existing experiment management systemsJoshi (2020a,b); Bisong (2019b,a); Moritz et al. (2018); Meng et al. (2016); Lin et al. (2022b); Richie-Halford and Rokem (2018) which are specific to a particular cloud or cluster environment, EMS® supports a full-fledged orchestration of sky-based experiment pipelines, enabling different clouds or clusters

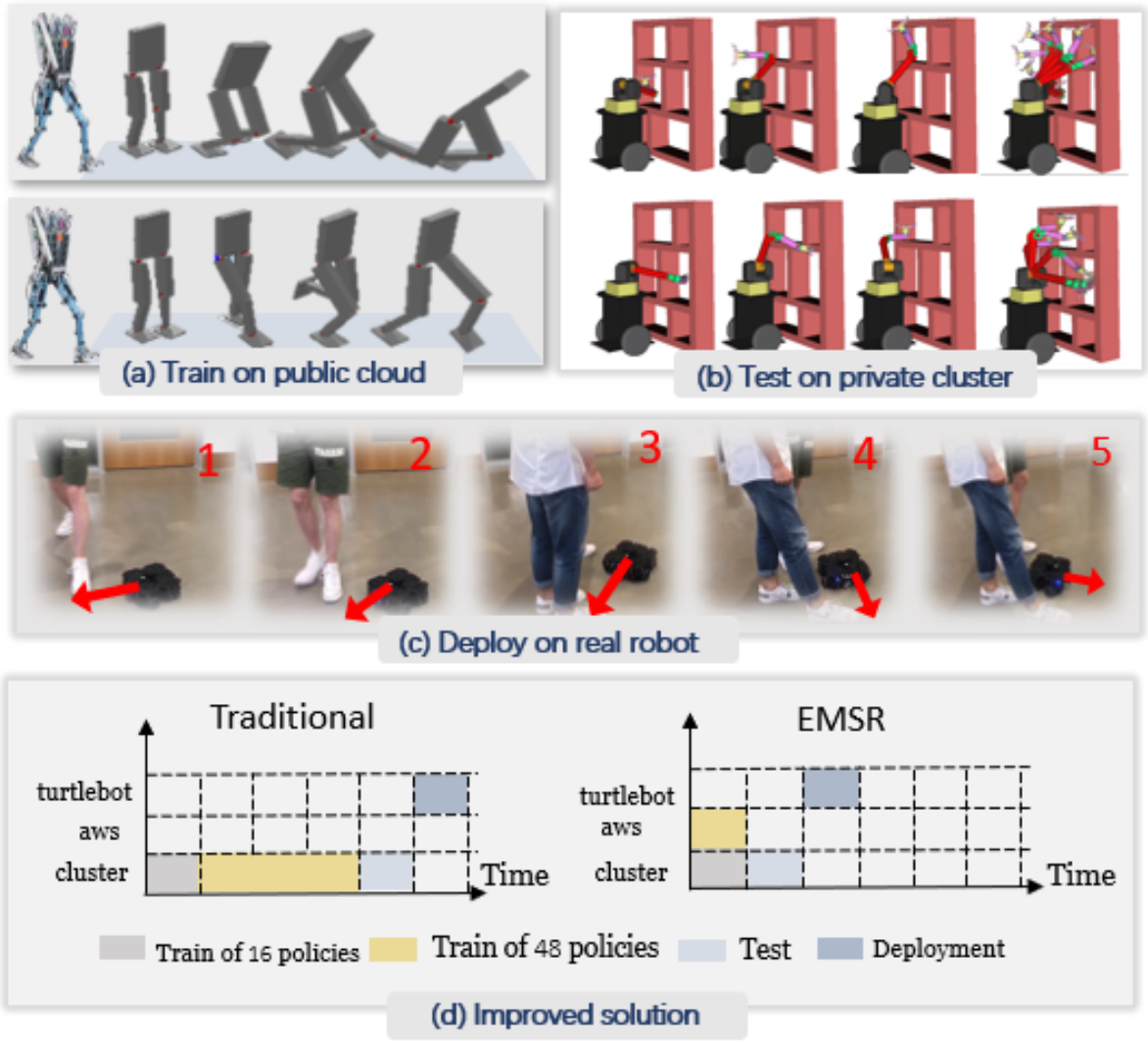


Figure 2.1. Examples of EMS® on robotics experiments. (a) shows bipedal training tasks on public cloud computation, (b) illustrates motion planning tasks of a robot arm, on a private cluster, (c) demonstrates collision avoidance performance of turtlebot, using onboard computation. These show sky-features of robot pipelines. (d) shows traditional pipeline and improved pipeline in EMS®. It consists of training 64 policies, testing the trained policies, and deploying the best policy on turtlebot. Specifically, the cluster can afford parallel training for a maximum of 16 policies within 20 minutes and the other 48 are scheduled on aws or the same cluster. The testing tasks are both scheduled on the cluster and deployments are on turtlebot's onboard computation in 20 minutes.

to run various stages of a robotics research pipeline. In particular, the system provides a highly sophisticated abstraction of the underlying computation platforms such that researchers only need to change one line of code to run the experiment across different computation environments, without

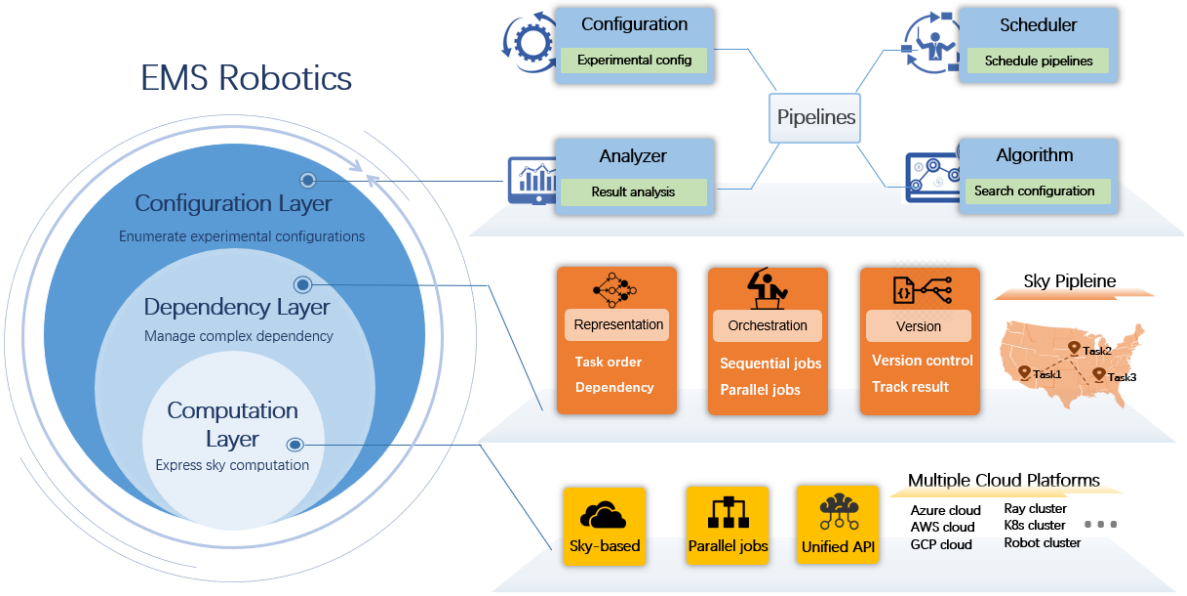


Figure 2.2. An overview of EMS®. EMS® enables sky-based pipeline orchestration. It consists of three layers: (i) Computation management layer abstracting clouds computing and selecting cloud on which various jobs are supposed to run on; (ii) Dependency management layer managing complex dependency in the pipeline; (iii) Configuration management layer automatically enumerating experimental configurations.

worrying about the data transfer and cloud service interface. In addition, EMS® automatically optimizes resource allocation according to given time and budget constraints.

The EMS® architecture consists of three abstraction layers, shown in Figure 2.2. (i) A sky-based computation layer providing a unified and easy-to-use API for submitting, monitoring, and canceling massive-scale computational jobs. This layer abstracts out the implementation details of all computing resources to free higher-level layers from the perceived complexity of using different cloud services. (ii) A dependency management layer using a DAG (directed acyclic graph) representation to orchestrate a task pipeline within an experiment. This layer features a sophisticated version controller which allows us to best reuse results from previous experiments. (iii) A configuration management layer providing a unified configuration programming interface to manage and optimize the schedule of massive amount of computational experiments. Such a layered design makes computational robotics research more efficient and scalable.

The rest of the paper is organized as follows. Section 2.1.2 introduces work related to cloud robotics and experiment management system. Section 2.1.3 illustrates EMS® architecture and discusses some implementation detail. Section 2.1.4 demonstrates experimental results on the improved productivity, reproducibility, and scalability of EMS®.

2.1.2. Related Work

This section summarizes related work on experiment management system and cloud robotics system.

Experiment management system. An experiment management system Monajemi et al. (2019) is a software stack that automates the process of experimental job management, output harvesting, data analysis, reporting, and publication of code and data. Some general-purpose experiment management systems include ClsterJobMonajemi et al. (2016), codalabcod, pywrenpyw, Kubernetes batchkub (b), and AWS batchaws (c). They facilitate researchers to conduct million-CPU-hour experiments in a painless and reproducible way. The most relevant works to this paper are the machine-learning pipeline management systems Xin et al. (2021). Such systems fall into two categories: (i) machine learning pipeline management service from a cloud provider, like AWS sagemakerJoshi (2020a), Microsoft Azure MLJoshi (2020b), and GCP MLopsBisong (2019b). (ii) automatic installation packages on general bare-metal servers, including Ray JobMoritz et al. (2018), KubeflowBisong (2019a), TensorFlow Extended (TFX)Baylor et al. (2017), MLlibMeng et al. (2016), MetaFLowArisdakessian et al. (2020), and ScikitLearnGarreta et al. (2017). Though significant progress has been made, these systems are not designed for robotics research which requires heterogeneous computational resources. To bridge this gap, EMS® resorts to a sky-based pipeline orchestration framework.

Cloud robotics system. A cloud robotics system uses cloud resources to enable greater memory, computational power, and interconnectivity for robotics applications. Early works focus on facilitating the seamless integration of robot and edge devices into both on-premises clusters and public cloud services Saha and Dasgupta (2018); Kehoe et al. (2015); Goldberg and Kehoe (2013); Shakya et al.

(2020); Liu and Xu (2019). Specifically, DAvinCiArumugam et al. (2010) implements a software framework on the Hadoop cluster to scale robotic development and RoboFlowLin et al. (2022b) is a cloud-based workflow management system orchestrating the pipelines on Kubernetes cluster. FetchCorefet and Formantfor robotics build web-based robot management systems, automating monitoring and controlling robotics. To leverage the power of public clouds, RoboEarthWaibel et al. (2011) and RapyutaHunziker et al. (2013) propose a system architecture, enabling robots to delegate their intense computational tasks to the Amazon cloud service. FogRosLiang et al. (2021); Ichnowski et al. (2022) automates robotics deployment on public cloud and SmartCloudStauffer (2022) facilitates robot interactions with public cloud services. HondaNishimiya and Imai (2021) proposes a serverless architecture for service robot management systems on AWS and Robomakeraws (b) provides a cloud-based simulation service for scaling robotic applications. Most of these systems focus on robot deployment. In contrast, EMS® focuses on massive experiment management.

2.1.3. System Architecture and Implementation

The EMS® architecture consists of thee abstraction layers: (i) a computation management layer expressing sky-based computation, (ii) a dependency management layer providing sky-based pipeline orchestration, and (iii) a configuration management layer enumerating experimental configurations.

EMS® uses sky computing for pipelines orchestration In particular, it exploits an intercloud broker to optimally mediate multi-cloud computing by abstracting away the deployment details of the underlying clouds. This is different from the two related types of multi-cloud solutions. One is partitioned multi-cloud enabling different corporate teams to run their workloads on different clouds or on-premise clusters. The other is portable multi-cloud enabling the same application to run on multiple clouds. Examples include many third-party cloud applications (e.g. Confluentcon, DatabricksIlijason (2020), SnowflakeDageville et al. (2016), Trifactatri), uniform low interface across multiple clouds (e.g. KubernetesSayfan (2017), Google Anthosant, Azure ARCazu, AWS Outpostsaws (a)), and previous sky computing providing uniform infrastructure-as-a-service

for applicationsKeahey et al. (2009); Monteiro et al. (2014). In contrast to these works, sky computing provides a set of uniform high-level APIs for different cloud services so that users can split the experiment pipelines across different clouds. In addition, the intercloud broker enables selecting different clouds for job execution while optimizing customized metrics (such as price or performance). To conduct sky-based pipeline orchestration, EMS® implements an intercloud broker in the computation abstraction layer as described in Section 2.1.3.1.

2.1.3.1. Computation Management Layer. The computation layer features an intercloud broker that hides all the implementation details of the multi-cloud computation. As shown in Figure 2.3, the input of the intercloud broker is a set of computational jobs. It parses the job list and submits the jobs to cloud services (e.g. AWS batch) or cluster services (e.g. Kubernetes, Ray Cluster) for execution. After job submission, it monitors the job output and provides execution feedback to the parent process. We describe each job as a Python `dataclass` object whose attributes include job name, job resource demand like CPU number, GPU number, memory size, and job executed commands, target cloud name.

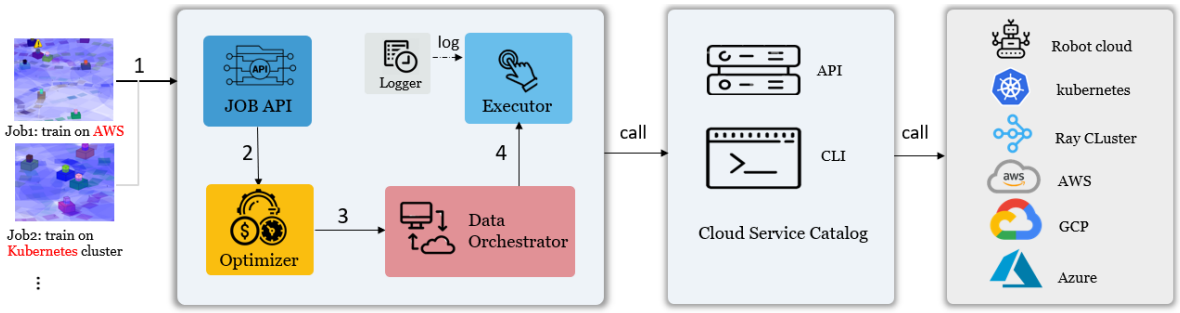


Figure 2.3. The computation layer architecture. The intercloud broker consists of the middle two panels. It creates an interface for jobs and clouds. It takes job descriptions as input and executes jobs on target clouds.

The intercloud broker consists of the following components (1) **Job APIs**: Job APIs provide a set of unified interfaces for developers to submit jobs to clouds, monitor and log running jobs on clouds, and cancel submitted jobs on clouds. Available APIs are shown in Table 2.1. (2) **Optimizer**: Given developers' submitted job descriptions, the optimizer generates optimal execution plans including moving data to a different region and selecting the suitable service providers. For example,

if developers specify a job on AWS batch, the optimizer determines AWS batch instance types, number of instances, and regions of the instances. (3) **Cloud Service Catalog**: This catalog is a list of open service interfaces, provided by cloud or cluster vendors. The service in this catalog can be a third-party library like boto3Garnaat (2018), or a command line like Azure CLI. (4) **Executor**: The executor creates required resources and executes commands on the resources allocated by the optimizer. (5) **Data Orchestratio**n: Data orchestration manages the data related to jobs, like code folder, raw data, and generated model data. If computation jobs are launched on the same region as the data, then it mounts the data-related storage to the instance, executing jobs. If they are in different regions, it copies the related data to the job region before launching jobs. (6) **Logger**: Loggers keep pulling jobs output from clouds. It provides real-time job execution status for the parent process.

Table 2.1. Description of job APIs.

Job APIs	**Description**
`submit_job()`	Submit jobs to clouds. The resource requirement, execution command lines, conda environment, and related data storage are specified in the `job.json`.
`status_job()`	List job status on the clouds. Status can be submitted, running, canceled, failed, or succeeded.
`cancel_job()`	Cancel jobs on the clouds.
`watch_job()`	Pull job output from clouds and print the content.

Built upon the computation management layer, the dependency management layer orchestrates task pipelines within one experiment. The architecture is illustrated in Figure 2.4. Specifically, this layer takes a pipeline description as input, generates a version number and job description for each task in the pipeline, and then submits the job description in both sequential and parallel order.

To perform these functions, the dependency management layer consists of four major components: (1) **DAG Representation**: We use a directed acyclic graph (DAG) to represent the execution dependency between tasks. Each task in the pipelines describes the computation requirement, source file directory paths, dependency tasks, and input arguments. (2) **Pipeline Orchestrator**: Given a DAG representation, the pipeline orchestrator generates job descriptions for each task and

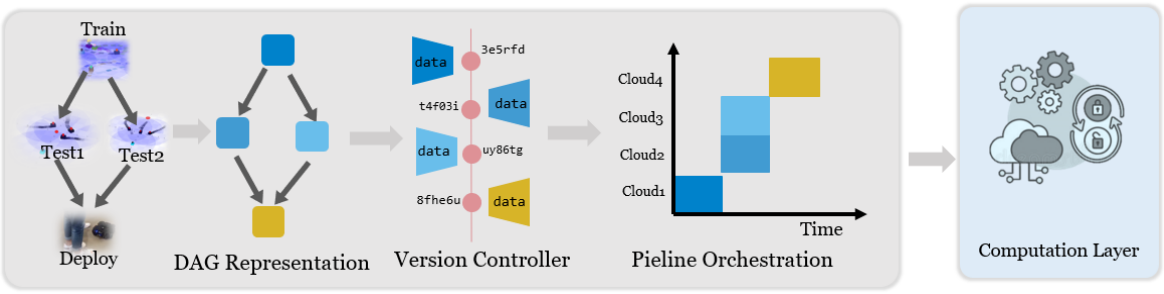


Figure 2.4. The dependency management layer, managing a turtlebot pipeline. Rectangles represent tasks and trapezoids represent version seed generated from the version controller.

determines the execution order of the task. It also detects the last modification time of the related data files in each task. If the detected time is later than the last task execution time, it submits the job for execution, otherwise, it keeps the previous results. (3) **Version Controller**: Version Controller tracks every pipeline experiment metadata like code, data, timing information, and pipeline results. Given a DAG representation, the version controller generates a version number for each task in the pipeline. This version number changes if pipeline name or task name or task dependency changes. The version number is used as a directory name, which is created for saving all relevant information about the task.

2.1.3.2. Configuration Management Layer. Built upon the dependency management layer, the configuration management layer features a flexible programming interface for users to specify the whole configuration space for massive computational experiments. It specifies the set of computational jobs (along with their resource sharing constraints) serving as the input to the computation management layer. It is also responsible for harvesting the results from all the executed experiments for downstream analytics. Specifically, there are heterogeneous categories of configurations like hyperparameters for models, resource directory paths, and settings for the dependency management layer and computation management layer, etc. To manage massive computational experiments, we need to frequently modify and automatically generate the configurations. We implement a configuration manager to handle this task.

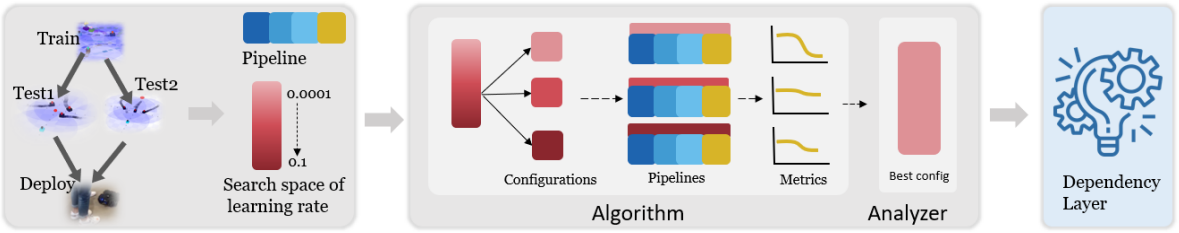


Figure 2.5. The configuration management layer, managing a turtlebot pipeline. Given pipeline representation and search space of configurations, the algorithm module suggests configurations to pipelines, and the analyzer module finds the configuration with the best metric.

As shown in Figure 2.5, the configuration manage layer consists of three components. (i) The configuration component provides a flexible programming interface for users to describe the search space. Like hyperparameters of models (batch size or learning rate). Given configuration space and pipeline, algorithm, and analyzer component together automates the process of running pipelines. (ii) The algorithm component generates and schedules configurations for pipelines according to the given search space. (iii) The analyzer component aims at finding the best configurations for the pipelines.

2.1.3.3. System Implementation

The implementation of the computation layer, the dependency management layer, and the configuration management layer comprises 92% in Python, 5% in Rust, and 3% in Shell scripts. Specifically, the computation layer abstracts three clouds, including AWS batch service, Ray Cluster service, and Kubernetes Cluster Service. We set up Ray Cluster and Kubernetes Cluster on two on-premise clusters and connect them with real robots. To access cloud service and cluster service, we use AWS Boto3 library, Kubernetes command lines, and Ray Job/Core APIs. For transferring data between different clouds, we use network filesystems like AWS EFS and SSHFS to mount data from different clouds. Then, for the dependency management layer, we exploit Python Hashlib to manage tasks in pipelines. To perform scalable configuration searching, we exploit Ray library to run, monitor, and analyze pipelines.

2.1.4. Experimental Results

We exploit EMS® to develop an offline reinforcement learning(RL) pipeline for training mobile robot navigation policy and demonstrate its productivity, scalability, and reproducibility. Also, we develop three realistic robotics applications to demonstrate that EMS® is applicable to numerous intelligent robots. In particular, we aim to answer the following questions: (1) How well does EMS® improve productivity and scalability upon an evaluation baseline? (2) What advantages do the computation layer, dependency management layer, and configuration management layer provide for EMS®? (3) Is it difficult to connect EMS® 's applicability to robotics?

2.1.4.1. Experiment Setting. We train a decision transformerChen et al. (2021) based robot agent in the Four Rooms environment of Gym MinigridChevalier-Boisvert et al. (2018), where the agent navigates in a maze composed of four rooms interconnected by four gaps in the walls. We implement the problem using a four-task pipeline. Firstly, a data acquisition task that applies PPOSchulman et al. (2017) to interact with the Four Rooms environment to collect data. Secondly, a data preprocessing task which transforms collected data (represented by state, action, reward) into a trajectory representation. Thirdly, a model training task that trains a decision transformer model using one hyperparameter set (experiment seed, head number, and embedding size) of the transformer model. Fourthly, an evaluation task that evaluates the trained model in the Four Rooms environments over 100 goals and reports accumulated rewards of the model in each goal. More details are referred to in the original paperChen et al. (2021). We implement EMS® on four cloud computing environments: the AWS Batch computing, a Kubernetes cluster, a Ray cluster, and a robot cluster. To make consistent benchmarks, we deploy all these environments on the AWS EC2 p3.8xlarge (Ubuntu 20.04) instances[3]. To simulate a multi-cloud setting, we deploy the four computing clusters in four different AWS VPCs and connect them through AWS site-to-site VPN.

[3]We also use the same instance for AWS Batch job submission

2.1.4.2. Overall Performance. We run a robotics daily routine pipeline on three systems (Naive, Single-cloud EMS, EMSR) and compare their productivity, scalability, and reproducibility. Specifically, Naive runs decision transformer's original code from Chen et al. (2021) on one AWS EC2 p3.8xlarge instance. Single cloud-EMS runs the robotic pipeline in the subsection2.1.4.1 using EMS® but only enabling AWS batch computing. EMSR is the full-fledged implementation of EMS®. We use these systems to run 16 pipelines at the beginning of every hour from 1:00 pm to 10:00 pm. Each pipeline conducts model training with different hyperparameters (experiment seed, head number, and embedding size) using 1 GPU, and reports the averaged time duration, money cost, and evaluation reward of pipelines every hour. In addition, we use three systems to repeatedly run the pipeline at 2:00 pm multiple times (8, 16, 24, 32) within one hour and report the averaged time duration and money cost, evaluation reward of pipelines.

We collect 3 evaluation metrics: (i) We measure **productivity** by total_output and total_input where total_input is the averaged money cost and duration time of running pipelines, and total_output is averaged evaluation reward of running pipelines. Given the same total_output, less total_input means more productivity. (ii) We measure **scalability** by money cost and duration time of different pipeline frequency, which is the total number of pipeline runs within one hour. (iii) We measure **reproducibility** by the evaluation reward of running pipelines. Moreover, the money cost on simulated on-premise devices is considered free and not counted towards instance hours[4].

We reports producitivity in Figure 2.6, and reports scalibility and reproducibility in Figure 2.7. We highlight three key findings: (1) EMS® improves the productivity of naive approach and single cloud-based system due to its sky-based architecture. Single-cloud EMS's cost is lower than Naive since its version controller in the dependency manage layer enables reducing duration by recycling the data preprocessing (2) EMS® has the highest scalability due to its sky-based computation layer optimizes the cost by scheduling pipeline on an on-premise device (3) EMS® reproduces the pipeline results with different number of pipeline runs.

[4]In reality, there is a tiny cost on maintaining the on-premise cluster, but this detail does not change the main conclusion drawn from these experiments.

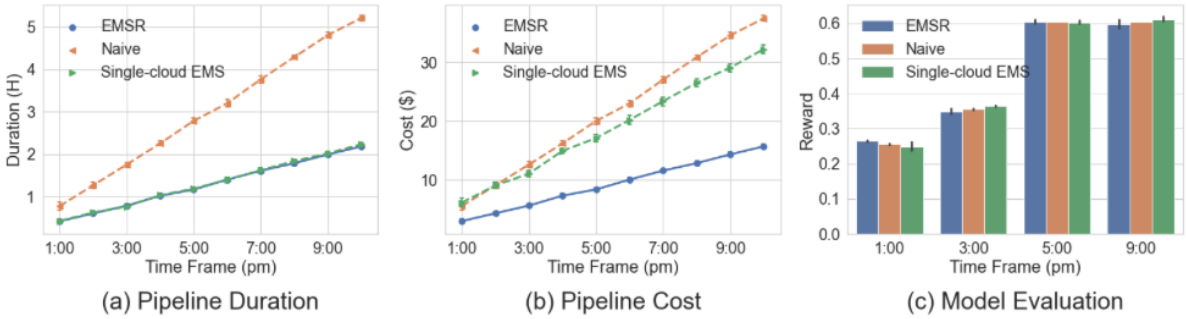


Figure 2.6. Productivity of EMS® in a robotics daily routine pipeline. We run the pipeline every 1 hour from 1:00 pm to 10:00 pm and report the pipeline duration in (a), the cost in (b), and evaluation rewards in (c). The blue line and bar represent EMS®, implemented on an on-premise cluster and AWS batch service. The green line and bar represent single-cloud EMS, implemented by EMS® only enabling AWS batch service. The orange line and bar represent the naive pipeline, implemented on one EC2 instance. As expected, EMS® requires less time and cost than the other methods.

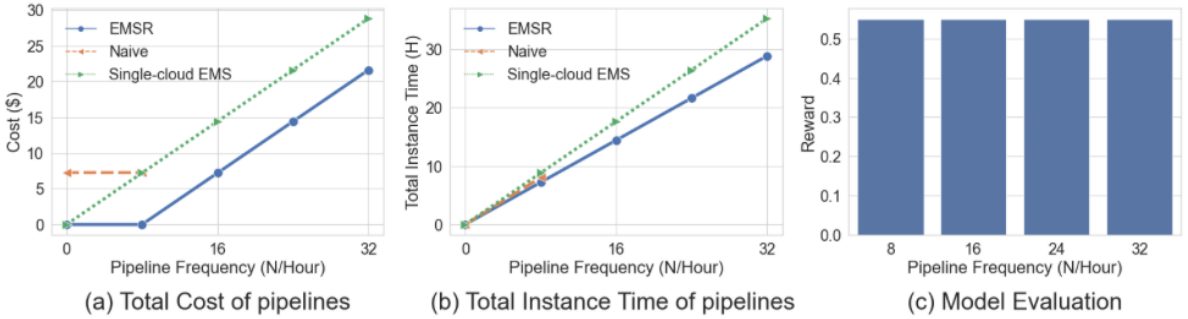


Figure 2.7. Scalibility and reproducibility of EMS® in a robotics daily routine pipeline. We run different number of pipelines every hour and report averaged duration in (a), averaged cost in (b), averaged evaluation rewards in (c). The blue line and bar represents EMS®, implemented on an on-premise cluster and AWS batch service. The green line and bar represents single-cloud EMS, implemented on EMS® only enabling AWS batch service. As expected, EMS® requires less time and cost for running the same number of pipelines than the others. Also, EMS® reproduces the model performance by running the pipeline multiple times.

2.1.4.3. Performance Gain from Each of the Abstraction Layers. Computation layer. A key benefit of the sky-based computation layer is to allow EMS® to scale jobs while maintaining low time duration and money cost. In Figure 2.8a and Figure 2.8b, we evaluate this ability on parallel workloads of empty jobs, where each job requests 1 CPU hour. Compared with 3 existing job management systems (AWS-Batch, Kubernetes-Batch, Ray-Job), we observe that the total time duration of EMS® near-perfectly stays the same when the total number of jobs increases. This is

due to the sky-based computation leveraging public computing for parallel-jobs scheduling. Since Kubernetes-batch and Ray-job are deployed on an EC2 instance with only 96 CPU cores to schedule the job, the time increases linearly with respect to the number of jobs. In Figure 2.8b, we observe that the instance hours of the computation layer are lower than the others. This is due to the sophisticated data orchestrator reducing the time of resource transfer.

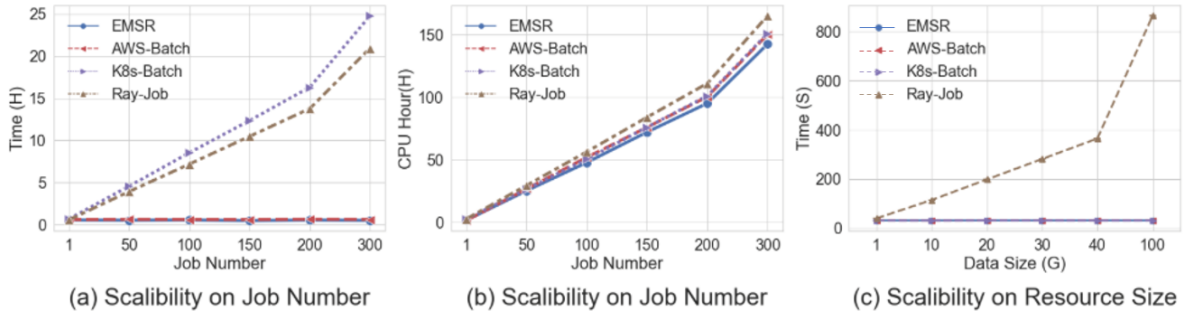


Figure 2.8. Scalability of computation layer in EMS®. We report total time duration in (a) and total instance hours of jobs in (b) with increasing job numbers. We also report single job time duration with increasing data size of jobs in (c). The blue line represents the computation layer, the red line represents the AWS Batch computing, the purple line represents the Kubernetes Batch job and the brown line represents Ray Job. The computing environment of Kubernetes Batch job and Ray Job is set up with mounted data volumes.

Another benefit of the computation layer is the ability to scale the dataset size of a job while maintaining a low time duration. In Figure 2.8c, we see that EMS®'s job duration almost stays the same even dataset size of the job increases. Compared to Ray-job, it outperforms in two magnitudes of order. The data orchestrator utilizes a networked file system to mount the source file to the jobs and this reduces the total data transfer time.

Dependency management layer. To evaluate the performance of the dependency layer, we track the data preprocessing time of the pipeline experiment from Section 2.1.4.2 and compare it with Naive. In Figure 2.10a, we observe that the time of EMS® stays almost the same with the dataset size increases while Naive increases linearly. This is because the version controller of EMS® tracks preprocessed results from the previous pipeline runs and the new pipeline only spends time on the newly arrived data from the data collection task.

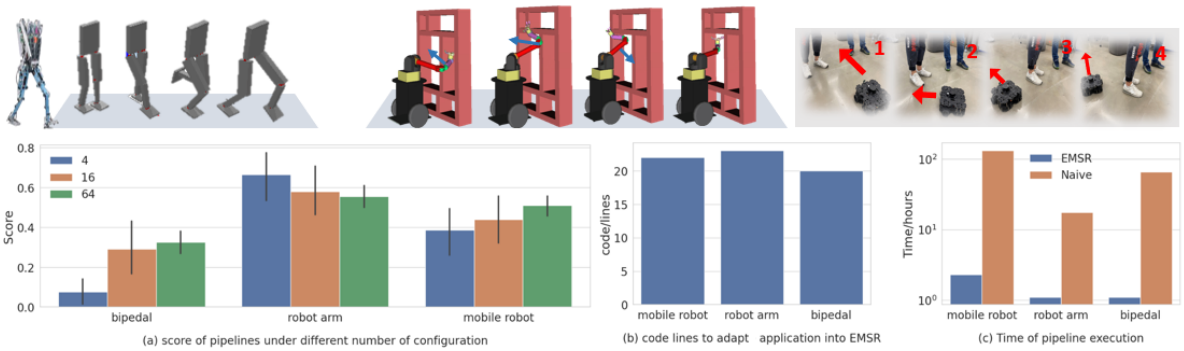


Figure 2.9. Statics of three case studies - bipedal walking, motion planning of a 7-dof robot arm, and collision avoidance of turtlebot. In (a), we report the average score of pipelines under different configurations (4,16,64). The score of bipedal and turtlebot is normalized reward and the score of the robot arm is normalized path cost. We also report the number of code lines for adapting naive implementation into EMS® and the time duration in hours of 64 pipeline execution.

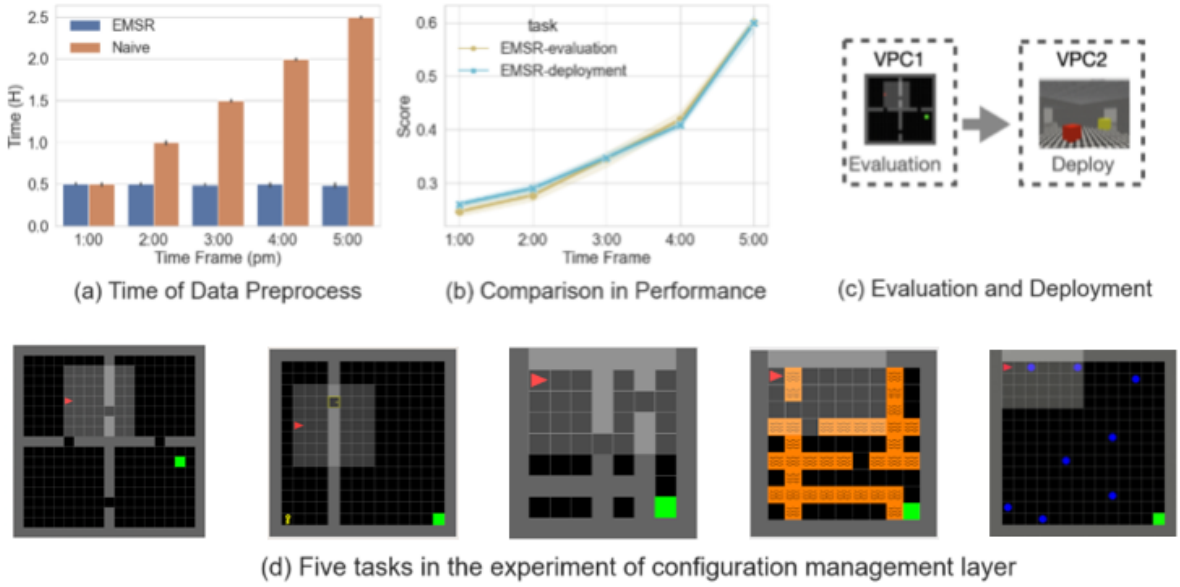


Figure 2.10. Performance of the pipeline management layer in EMS®. In (a), we report the data preprocessing duration of pipelines running on different time frames. In (b), we also report the evaluation score and deployment score of the trained models in pipelines. (c) illustrates the experiment setting of (b), where the evaluation cloud and deployment cloud is different. From left to right in (d), they are Four Rooms, Door Keys, Simple Crossing and Lava Crossing, Dynamics Obstacle Avoidance.

Also, the dependency management layer brings EMS® the ability to schedule sky-based pipeline as demonstrated in Figure 2.10b, allowing us to evaluate models on one cloud (VPC1) and deploy the models on another cloud (VPC2).

Configuration management layer. The configuration management layer is important for hyperparameter search in many data-driven workloads.We run the same pipeline in Section 2.1.4.2 for five different tasks, as shown in Figure 2.10d. Specifically, for each task, we search configuration among 1, 8, 16, 32, and 64 sets of hyperparameters, which are head number, layer number, embedding size, activation functions, batch size, and experiment seeds for the decision transformer model. For each set, we report the best model evaluation reward among the hyperparameters. We report the average evaluation reward and variance of the performance on 100 goals in Table 2.2 and observe that a larger amount of hyperparameters delivers better-performed models.

Table 2.2. Best evaluation score of decision transformer models in different hyperparameter number.

Number	Four Rooms	Door Keys	Simple Crossing	Lava Crossing	Dynamic Obstacles
1	0.460 ± 0.03	0.892 ± 0.08	0.789 ± 0.28	0.798 ± 0.01	0.654 ± 0.11
8	0.481 ± 0.05	0.934 ± 0.05	0.820 ± 0.12	0.855 ± 0.02	0.685 ± 0.09
16	0.520 ± 0.03	0.954 ± 0.07	0.824 ± 0.11	0.909 ± 0.03	0.720 ± 0.12
32	0.523 ± 0.02	0.953 ± 0.08	0.850 ± 0.10	0.920 ± 0.03	0.719 ± 0.08
64	0.524 ± 0.03	0.960 ± 0.02	0.888 ± 0.12	0.920 ± 0.02	0.722 ± 0.10

2.1.4.4. Case Studies. We exploited EMS® to develop three data-driven robotics applications. **(i)** Developing collision avoidance policy of turtlebot robot. This pipeline consists of training collision avoidance policy in Stage simulation Vaughan (2008) with PPO, testing in stage, and deployment on a real robot Amsters and Slaets (2019). **(ii)** Developing motion planning policy of a 7-dof robot arm in OpenRave simulation. This pipeline consists of collecting datasets in OpenRave Diankov (2010) with BIT* Gammell et al. (2020), training NEXT Chen et al. (2019) motion plan policy, and testing in simulation. **(iii)** Developing a bipedal walking policy. We model Flame robot Solomon et al. (2013); Hobbelen et al. (2008) in pybullet Erwin and Yunfei (2016), train walking policy using PPO, and testing policy in pybullet. The details of these implementation are referred to in paper Ye et al. (2020); Long et al. (2018); Gammell et al. (2020); Liang et al. (2017). Specifically, the naive implementation of these applications can only leverage computation on a AWS EC2 instance of p3.8xlarge and EMS® integration can leverage AWS batch computation to run pipelines. Each running pipeline of them has one configuration and requires resource of one p3.8xlarge instance.

In the experiment, we run 4,16,64 pipelines with different configurations(seed, batch size. and learning rate) and report average score, total execution time and code line number in Figure 2.9. As shown in subfigure 2.9a, the average score is higher with more configuration of pipelines, which justifies the importance of massive computation in EMS®. Also, subfigure 2.9b shows that adapting existing robotic applications EMS® only requires a few of code lines, which implies that EMS® is applicable to numerous intelligent robotics applications. Then we compare the time cost of EMS® with naive implementation and demonstrate the less time cost of EMS® than the naive with the same pipeline workloads.

2.1.5. Conclusion

We propose EMS®, a cloud-enabled massive computational experiment management system supporting massive data-driven robotic routine experiments. By exploiting it to develop an offline reinforcement learning pipeline for mobile robot navigation, we demonstrate the scalability, reproducibility, and productivity of EMS®. Then, we show that EMS® is applicable to numerous data-driven robotics applications, through three realistic robot examples. Future works will consider developing more sophisticated methods to optimize the cost of computation scheduling and integrate EMS®with a robotic deployment system in a continuous integration and continuous delivery (CI/CD) fashion.

2.2. RoboFlow: a Data-centric Workflow Management System for Developing AI-enhanced Robots

We propose RoboFlow, a cloud-based workflow management system orchestrating the pipelines of developing AI-enhanced robots. Unlike most traditional robotic development processes that are essentially process-centric, RoboFlow is data-centric. This striking property makes it especially suitable for developing AI-enhanced robots in which data play a central role. More specifically, RoboFlow models the whole robotic development process into 4 building modules (1. data

processing, 2. algorithmic development, 3. back testing and 4. application adaptation) interacting with a centralized data engine. All these building modules are containerized and orchestrated under a unified interfacing framework. Such an architectural design greatly increases the maintainability and re-usability of all the building modules and enables us to develop them in a fully parallel fashion. To demonstrate the efficacy of the developed system, we exploit it to develop two prototype systems named "Egomobility" and "Egoplan". Egomobility provides general-purpose navigation functionalities for a wide variety of mobile robots and Egoplan solves path planning problems in high dimensional continuous state and action spaces for robot arms. Our result shows that RoboFlow can significantly streamline the whole development lifecycle and the same workflow is applicable to numerous intelligent robotic applications.

2.2.1. Introduction

We propose RoboFlow, a cloud-based workflow management system for developing data-centric and AI-enhanced robots. This work is done in the context that significant progresses have been made in robotics development and a paradigm shift from process-centric development to data-centric development is being witnessed, especially for learning robots. Specifically, traditional robot development workflow Arumugam et al. (2010); Bouziane et al. (2017); Kato et al. (2018); Lei et al. (2016); Dennis et al. (2016); Huang et al. (2020) is essentially process-centric, which emphasizes more on designing, developing and integrating different "processing modules" interacting and inter-operating with each other in a complex fashion. Such a process-centric robotic development model, though natural for humans, is not suitable for developing modern AI-enhanced robotic systems (aka., learning robots) that are essentially data-centric Finn et al. (2017); Gu et al. (2017); Kretzschmar et al. (2016); Zeng et al. (2018); Ebert et al. (2018); Mahler et al. (2017). Some key reasons are that the development processes of AI-enhanced robots generally involve managing and interacting with massive amounts of data, and even after the systems have been deployed, continuous modification and improvement are still needed when more data get acquired. Such

an extra "data-centric" dimension of learning robots causes a dramatic increase in both coding complexity and maintaining complexity of the traditional process-centric robotic development workflow, thus a new data-centric robotic development workflow is crucially is needed. To bridge this gap, we propose RoboFlow.

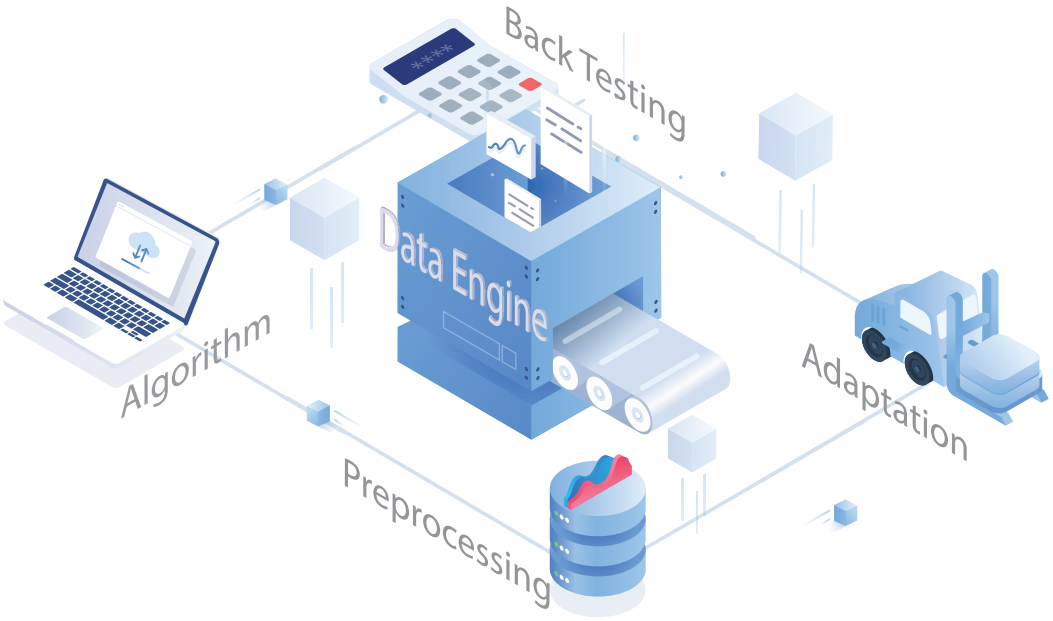


Figure 2.11. An overview of RoboFlow. The robot development pipeline consists of 4 modules interacting with a centralized data engine. The data engine manages the large-scale dataset, and publishes data from robot. The data preprocess module encodes raw input to data that algorithm development module can easily parse. The algorithms development module develops customized control policy. The back testing module tests the policy in various environments. The application adaption module deploys the learned policy in real world.

A high-level overview of RoboFlow is illustrated in Figure 2.11. In the most abstract fashion, the RoboFlow system divides the whole pipeline of developing AI-enhanced robots into 4 building modules (1. data processing, 2. algorithmic development, 3. back testing and 4. application adaptation) interacting with a centralized data engine. Specifically, the data engine can be viewed as an "oracle" that abstracts out all the data management details and is interacting (e.g., being queried or manipulated) with all the 4 building modules in an asynchronous fashion. Centered around the data engine, the 4 building modules follow an iterative spiral model. Unlike the classical spiral modelBoehm (1988) for software development, these building blocks mainly interact with data

engine, thus can be developed in a fully parallel fashion and each module could have different "versions". Such a data-centric design dramatically decreases the developing and maintaining complexities, making it especially suitable for developing AI-enhanced robots. The framework in Figure 2.11 is quite generic and described in a fully abstract way. To understand this framework better, we put it in a concrete context of developing learning robots. From a learning perspective, the data processing module transforms and encodes data into a state that other modules can easily parse. The algorithm module represents the planning policy of the robots, which is in charge of the robots' actions. The back testing module evaluates the policy of the learning robot based on large-scale collected or simulated data. The application adaptation module puts the developed policy into real-world environments once the back testing meets a designed criterion. More details about these modules will be provided in Section 3.

This paper has 3 major contributions: (i) we proposed a novel data-centric robot development model which improves upon the traditional routines in terms of development flexibility and maintainability. (ii) We implement a first prototype system using containerization techniques (specifically, Dockerdoc and Kubernetes kub (c)). (iii) Using this system, we develop two platforms: Egomobility and Egoplan. Egomobility exploits deep reinforcement learning to provide navigation ability for turtlebot and Egoplan uses imitation learning to solve complex motion planning problems for sawyer robots. These studies demonstrate the efficacy of RoboFlow in various intelligent robotics applications. The rest of this paper is organized as below. Section 2 introduces related work. Section 3 describes implementation of RoboFlow. Section 4 presents two platforms to showcase the usefulness of RoboFlow.

2.2.2. Related Work

The two most relevant lines of work related to RoboFlow are the workflow management system and data management engine for data-centric robot systems. For the workflow management system, earlier robot development pipelines try to simplify the workflow of robotics development and reduce

re-programming efforts. For example, Fetch roboticsRobotics launches a Workflow Builder which allows customers to design, implement, and redesign their own workflows. But their available tools are not designed for large-scale dataset processing. In the machine learning field, frameworks like Kubeflowkub (a) and MLFlowMLF have been developed to manage the workflow of model development but they are not designed for robot development. For the data management engine, researchers in bioinformatics community developed system tools to manage data workflows in an end-to-end fashionComeau et al. (2017); Backman and Girke (2016). Nevertheless, in robotics, few works emphasize on the data-centric aspect of the development processes except some proposals related to cyber-physical systems which view robots as data-gathering nodes Gil et al. (2007); Römer et al. (2002); Yu et al. (2004); Abdelzaher et al. (2004); Remy and Blake (2011); Mohamed et al. (2008); Gil et al. (2007). In recent work, Farzad et al. Khodadadi et al. (2015) proposed a cloud framework aiming to facilitate the development of IoT applications. but it is not straightforward on how to apply it to the more heavy-weighted data-centric robotic systems.

2.2.3. System Architecture and Software Implementation

The RoboFlow architecture builds upon the containerization and container orchestration techniques. More specifically, a container platform (e.g., the Docker) packages applications so that they can access a specific set of resources on a physical or virtual host. The main benefit, especially for developers, is that containers isolate different applications and are elastic, i.e., come and go as demanded by need. This is particularly useful for developing massive robotic systems in which developers may contribute code in different programming languages and application frameworks. In such scenarios, we could exploit a container platform to establish many containers to isolate and manage all the developed applications. To manage containers at scale, we can utilize a container orchestrate system (e.g., Kubernetes, Docker Swarm) to automate the deployment, management, scaling, networking, and availability of all the containers. In the rest of this section, we describe the system architecture and software implementation of the RoboFlow system.

System Architecture. Figure 2.12 illustrates the system architecture of RoboFlow. It has 4 essential modules(1.Data Preprocess 2.Algorithm Development 3.Back Testing 4.Application Adaptation) interacting with a centralized data engine. Each module is employed into a containerized environment by bundling it together with all related configuration files, libraries and dependencies. These module containers run isolated processes on the system, thus enabling RoboFlow to be developed in a fully parallel fashion. In addition, any change in these module containers is recorded, making version control easy to implement. These containers exploit DDS (Data Distribution Service) to manage real-time communication between them and the data engine. DDS implements a publish-subscribe pattern for sending and receiving data and each process running in RoboFlow are considered as DDS nodes to communicate data with other process. Also, a networked filesystem named Glusterfs is also ultilized for sharing large-scale files (e.g., large neural network models or training datasets) between modules. Such integration of DDS and Glusterfs make RoboFlow suitable for developing data-driven methods on robotics. RoboFlow also provides a web-based frontend to ease developers to monitor, analyze, and manage the robotic development process.

Software Implementation. To implement RoboFlow, we wrap each module as a `docker` image. For this, we specify the software environment (e.g., the operating system distribution and pre-installed packages) of an image in a `Dockerfile`. The obtained module images are installed with `juypter lab` and ROS2. Some modules may need additional packages. For example, the algorithm development module is equipped with the deep learning libraries `tensorflow` and `pytorch`, while the data engine image utilizes several robot simulators like `Gazebo`, `Stage` and `OpenRave`.The successfully built images are stored in a cloud-based storage space named `cargo`, which can be viewed as on-premise `dockerhub` of the RoboFlow system. During the run time, each instance of a built image is deployed as a container and RoboFlow utilizes `Kubernetes` to deploy and manage these containers. We exploit the `React` javascript library to implement a graphical user interface (GUI). Through this GUI users can choose desired versions of the docker images and allocate computational resources (e.g., CPUs or GPUs) to the container being created. Once such

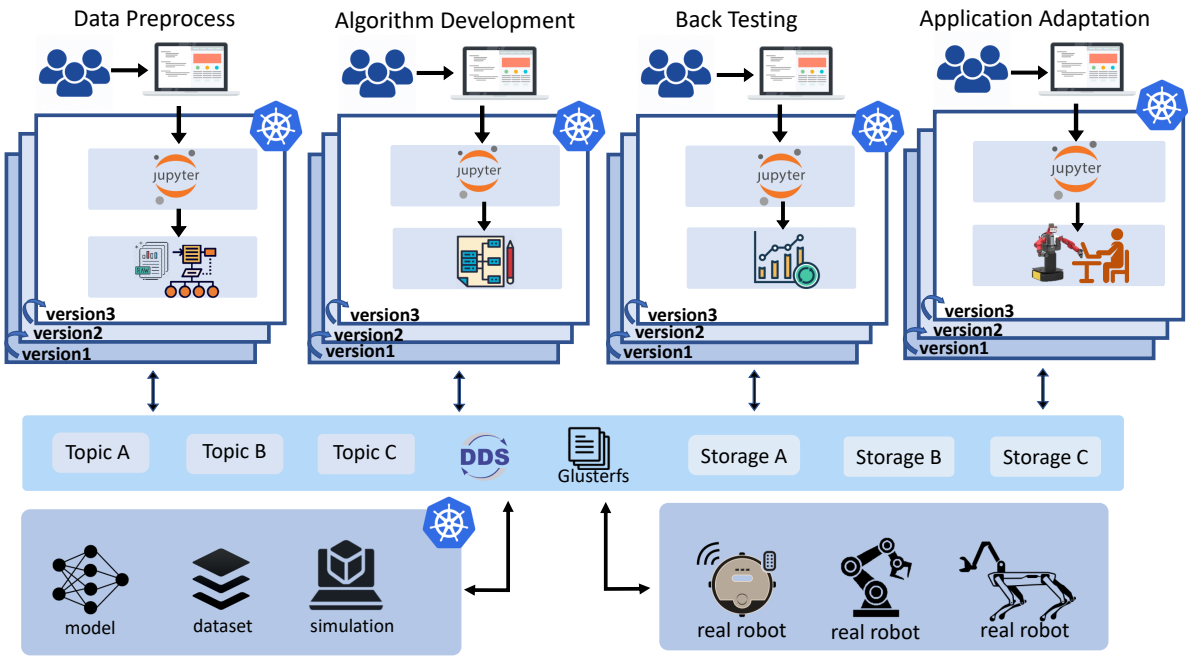


Figure 2.12. The architecture of RoboFlow. RoboFlow provides a graphical user interface for developers to access all containerized modules. These modules interact with the data engine through shared storage and DDS topics. The data engine consists of large-scale datasets and model data stored in the shared folder, simulation containers and robots connected to the RoboFlow system through DDS topics.

configuration information is submitted, it is turned into a `YAML` file which will be serialized and deployed by `Kubernetes`.

2.2.4. Two Case Studies and Performance Evalaution

In this section, we exploit RoboFlow to develop two prototype systems named "Egomobility" and "Egoplan". Egomobility provides a general-purpose navigation platform for managing a wide variety of mobile robots and Egoplan is a motion planning platform for robot arms. To demonstrate the efficacy of RoboFlow, we also conduct some performance evaluations of the two case studies.

Case Study 1: The Egomobility platform for Mobile Robots. In this study, we exploit RoboFlow to develop a mobile robot navigation platform named Egomobility, which is a data-centric AI-enhanced robot system using data from a `Stage` simulator Vaughan (2008) as training data. Within this environment, a mobile robot takes 3 raw laser frames and its velocity as input. Our goal

is to train a reinforcement learning policy that outputs a velocity guiding the robot avoiding dynamic obstacles. The deployment page of Egomobility is provided in [5].

Case Study 2: The Egoplan Platform for Arm Robots. In this study, we exploit RoboFlow to develop a robot arm motion planning Platform named EgoPlan. More specifically, we exploit the NEXT (Neural Exploration-Exploitation Trees) algorithm proposed inChen et al. (2020) to learn a motion plan policy for solving path planning problems in 7-dimensional state space and 7-dimensional action space. In each planning task, we simulate a robot arm and a shelf in the `openrave` Diankov and Kuffner (2008) simulator in our data engine. Each level of the shelf is horizontally divided into multiple bins. The task for the robot arm is to plan a collision-free path from an initial pose to grab an object placed in a shelf. The main strategy is to exploit BIT* Gammell et al. (2014) to solve the planning problem in a brutal force way but collect the data to learn a smarter motion planning policy.

Performance Evaluation. Through the above two case studies, the benefit of RoboFlow for managing a data-centric robotic system development pipeline is quite obvious: RoboFlow enables developers to develop all the component modules in a fully-parallel fashion. For example, in developing Egomobility, algorithm developers can join in the RoboFlow to develop the navigation algorithm independently without interfering with each other. The developers for the application adaptation module can simultaneously test many learned policies without the need of complex communication processes. For a more quantitative comparative analysis, we let the same team of developers to build the Egomobility and Egoplan platforms with and without using RoboFlow and document. During the development lifecyle, we carefully record different progress metrics. We observe at least 10 times improvement (3 months vs 1 week) in terms of development speed. One thing that needs to be careful is that the same team-first developed these two platforms without using RoboFlow and some significant experience has been acquired. However, even adjusted by this confounding factor, all the members in the team agree that at least 5 times more productivity should

[5]Project site: https://sites.google.com/u.northwestern.edu/roboflow

be achieved by using RoboFlow. In addition, due to the usage of sophisticated container-orchestration techniques, the resulting systems developed by RoboFlow is much more reliable and maintainable.

CHAPTER 3

Efficient Modeling Strategy

This chapter confronts the challenge of model efficiency. We first address the non-negotiable requirement of safety in human-centric environments. In joint work with Guo Ye, we formulate the problem of multi-agent navigation as a Markov game and utilize an adversarial training strategy to improve the model's performance. This approach develops a control policy that guarantees collision-free operation, providing the safety assurances necessary to scale data collection beyond controlled lab settings. Second, we tackle the problem of multi-task learning through the Switch Trajectory Transformer. This model employs a Mixture-of-Experts (MoE) architecture to increase model capacity without a proportional rise in computational cost. This design achieves high performance on diverse tasks while maintaining the inference efficiency required for deployment on resource-constrained robotic hardware.

3.1. Collision-free Navigation of Human-centered Robots via Markov Games

We exploit Markov games as a framework for collision-free navigation of human-centered robots. Unlike the classical methods which formulate robot navigation as a single-agent Markov decision process with a static environment, our framework of Markov games adopts a multi-agent formulation with one primary agent representing the robot and the remaining auxiliary agents form a dynamic or even competing environment. Such a framework allows us to develop a path-following type adversarial training strategy to learn a robust decentralized collision avoidance policy. Through thorough experiments on both simulated and real-world mobile robots, we show that the learnt policy outperforms the state-of-the-art algorithms in both sample complexity and runtime robustness.

3.1.1. Introduction

This paper studies the collision-free navigation problem of human-centered robots(Riener et al., 2006; Lam et al., 2010). Such robots need to interact, assist and cooperate with humans. One motivating example is grocery robots as shown in Figure 3.1, from which we see such robots must operate in a dynamic environment due to humans activities. Collision-free navigation(Borenstein and Koren, 1989; Van den Berg et al., 2008; Fox et al., 1997; Kretzschmar et al., 2016; Chiang et al., 2019; Bharadhwaj et al., 2018; Jun et al., 2019) is a fundamental required capability of human-centered robots.

The dynamic environment of human-centered robots brings difficulty for using traditional trajectory-based methods(Francis et al., 2019; Sartoretti et al., 2019) or reinforcement learning based methods which assume a static map (Chiang et al., 2019; Vasquez et al., 2014). More specifically, these methods formulate the robot navigation problem as a single-agent Markov decision process, then conduct either forward path planning (random tree search algorithms)(Faust et al., 2018; Chiang et al., 2015) or backward policy search (e.g., policy gradient algorithms)(Tai et al., 2017, 2018; Long et al., 2018, 2017) for collision-free navigation. They all assume the robot operating in a static environment, which is unrealistic for human-centered robots.

To handle this challenge, we exploit Markov games (Littman, 2001; Wang and Sandholm, 2003; Hu and Wellman, 2003; Casgrain et al., 2019) as a modeling framework for collision-free navigation. Unlike traditional trajectory-based methods or reinforcement learning based methods, our framework of Markov games adopts a multi-agent formulation with one primary agent representing the robot and the remaining auxiliary agents form a dynamic or even competing environment. This framework allows us to develop an adversarial training strategy to learn a robust decentralized collision avoidance policy.

Under the multi-agent formulation(Chen et al., 2017; Bu et al., 2008; Hernandez-Leal et al., 2017; Weinberg and Rosenschein, 2004; Raileanu et al., 2018), the primary agent representing robot is called a protagonist. Our method spawn several auxiliary agents who attack the protagonist in

Figure 3.1. A grocery robot navigating in a retail scenario. Such robots must operate in a dynamic environment due to humans activities. Image source: https://vosizneias.com/.

an adversarial way. These adversary agents get reward by resulting in protagonist not reaching the goal. A major contribution of this paper is the development of a novel path-following policy learning strategy for solving the Markov games. More specifically, all agents are modeled by a Markov process but we allow the auxiliary agents to have a global view of the system state while the primary agent (the robot) only have a local view. The Markov processes of the auxiliary agents are centrally configurable and are parameterized by a set of aggressiveness parameters (e.g., moving velocity, perception accuracy, etc.) that affect their capabilities being adversarial. By varying the aggressiveness parameters from tight to more relaxed (i.e., the auxiliary agents become more and more adversarial), the obtained parametric family of Markov decision processes form a regularization path that can be used to robustly train the primary agent's navigation policy using a path-following algorithm (More details are provided in Section 3.1.4). The same idea has also been exploited in developing the interior point method (Potra and Wright, 2000) in nonlinear optimization and parametric simplex method in linear optimization (Pang et al., 2017).

We conduct thorough numerical simulations to demonstrate the efficacy of the proposed method. We show that the learnt policy is simultaneously efficient in sample complexity and adaptive to

complex human-centered environment. We also conduct a sensitivity analysis to demonstrate the robustness of the proposed algorithm. In addition, we deploy the learnt policy on a real mobile robot equipped with only one low cost 2D LIDAR and show that the trained agent is directly deployable to complex environment.

3.1.2. Related work

This section summarizes some related work. Relevant literature includes Multi-agent collision-free navigation, Markov games formulation of mobile robots and adversarial learning.

3.1.2.1. Multi-Agent Collision-free Navigation. There are two collision-free navigation paradigms in multi-agent environments: the centralized approach vs decentralized approach. The centralized approach (Snape et al., 2011; Bareiss and van den Berg, 2015) assumes each agent has perfect knowledge of the other agents, which is unrealistic for human-centered robots since the primary agent (the robot) obvious can not know the perfect state of every auxiliary agent (humans). In contrast, the decentralized approach assumes the primary agent only has partial observation of the rest. Thus is more relevant to our setting. Related works on decentralized collision-free navigation methods include (Everett et al., 2018) and (Long et al., 2018). However, (Everett et al., 2018) only considers auxiliary agents with stochastic or prefixed policies while (Long et al., 2018) assumes the primary and auxiliary agents share the same policy. In contrast, our method models the auxiliary agents using an adversarial setting which improves both sample complexity and policy robustness of the primary agent.

3.1.2.2. Markov Games Formulation of Multi-agent Systems. Markov games are powerful at modeling multi-agent systems (Weinberg and Rosenschein, 2004; Hernandez-Leal et al., 2017). However, solving their equilibrium is nontrivial. To achieve tractable solution, one approach is to constrain the problem into a two-agent zero-sum game to learn a stationary policy for both protagonist and adversary (Pinto et al., 2017). Another approach is to exploit centralized actor-critic type methods (Lowe et al., 2017). A third approach is to constrain all the agents can only communicate

with their neighbors in a network setting (Zhang et al., 2018). None of these methods is readily applicable to our setting where the protagonist (the robot) is decentralized while the remaining agents are centralized.

3.1.3. Adversarial Learning

Motivated by the success of generative adversarial networks (Goodfellow et al., 2014), recent works show that training an agent in an adversarial setting could lead to improved performance. For example, in game environments, Trapit et al.(Bansal et al., 2017) and Adam et al.(Gleave et al., 2019) applied 2 competeting agents (one protagonist and one opponent) in the training process and obtained superior performance compared to the vanilla reinforcement learning methods. For driving games, (Pan et al., 2019) showed that the protagonist works better in the presence of a risk-seeking opponent. All these works exploit two-player zero-sum games. In contrast, we employ multi-agent general-sum Markov games which are more challenging to solve and motivate the development of a new path-following adversarial training strategy (More details are provided in Section 3.1.4).

3.1.4. Methods

In this section, we formulate the collision-free navigation problem as a Markov game and describe a path-following type strategy for solving it. Though our algorithmic framework is fully generic and applicable to any human-centered robot, we describe the algorithm using a simple 2D `Stage` simulation environment (Vaughan, 2008) with the hope of making the main idea more concrete to accessible to the audience. All the agents in this simulator exploits a 2D LIDAR with laser scan range 3.5m as sensing device. The maximum linear speed of the mobile robot in this simulator is 1m/s.

3.1.4.1. Collision-free Navigation Formulated as Markov Games. A human-centered robot needs to navigate in an environment with the interference of one or more moving humans. This scenario can be modeled as a multi-agent system with the robot as the primary agent (protagonist)

and the humans as auxiliary agents (adversaries). Each agent is modeled by a Markov decision process. We allow the adversaries to access the global state information of all other agents while the protagonist to access only local information of its nearby agents. Each agent aims at maximizing its accumulative reward under its own reward mechanism. These co-evolving and competitive agents thus form a Markov game(Littman, 1994):

$$\mathcal{M} = (S, O_1 \dots O_N, A_1 \dots A_N, T, R_1 \dots R_N).$$

Here N is the total number of agents with Agent 1 as the protagonist. S denotes the states of all agents. $O_1 \dots O_N, A_1 \dots A_N$ are the sets observations and actions for each agent. A_i is action of Agent i sampled from a stochastic policy π_i and the next state is generated by the state transition function $T : S \times A_1 \times \dots \times A_N \to S$. At each step, every agent gets a reward according to the state and corresponding action $r_i : S \times A_i \times \dots \times A_N \to \mathbb{R}$ along with an observation of the system state $o_i : S \to O$. While the adversarial agents $2, \dots, N$ have access to the global information S, the protagonist only have access to partial information o_i (More details are provided in Section 3.1.5). The objective for the i-th agent is to learn a policy that maximizes the cumulative discounted rewards

$$\mathcal{R}(i) := \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_i^{(t)})\Big], \tag{3.1.1}$$

where $\gamma \in (0, 1)$ is the discount factor and $r_i^{(t)}$ is the the reward received at the t-th step.

3.1.4.2. Path-following Policy Learning Strategy. The cumulative discounted reward of the protagonist $R(1)$ and those of the adversaries $R(2), \mathcal{R}(3) \dots, \mathcal{R}(N)$ are coupled since these adversary agents get reward by resulting in protagonist not reaching the goal. This forms a general-sum Markov game and it is nontrivial to solve its equilibrium. To proceed, we propose a path-following policy learning strategy. The main idea is to parameterize all the auxiliary agents by a set of aggressiveness parameters that affect their capabilities being adversarial. By varying the

aggressiveness parameters from tight to more relaxed, the auxiliary agents become more and more adversarial. In another word, the environment of the protagonist shifts from being stochastic to more adversarial. The obtained parametric family of Markov decision processes form a regularization path that can be used to robustly train the primary agent's navigation policy using a path-following strategy.

More specifically, we consider the following set of aggressiveness parametrized of the auxiliary agents: (i) *Linear velocity*: The mobile robot contains linear and angular speed v_ℓ, v_ω, the latter does not have notable influence to adversaries' ability. Hence, we only consider the linear velocity in the range $v_\ell \in [0, 1.5]$. (ii) *Adversarial size*: We model the shape of each auxiliary agent by a square with width $s \in [0.1, 0.4]$. The larger size brings difficulty for the protagonist to conduct collision avoidance. (iii) *Perception accuracy*: Each adversarial agent can access the positions of other agents with an additive Gaussian noise with spheirical covariance matrix $N(0, 1/\rho^2 I)$. Here we set $\rho^2 \in [0.2, 1000]$. We parameterize the above agressiveness parameters using a one-dimensional parameter $\tau \in [0, 1]$ to be monotone increasing functions $v_\ell(\tau), s(\tau)$ and $\sigma(\tau)$ such that $v_\ell(0) = 0, v_\ell(1) = 1.5$; $s(0) = 0.1, s(1) = 0.4$; $\rho^2(0) = 0.2, \rho^2(1) = 1000$. Once the functions $v_\ell(\cdot), s(\cdot), \rho^2(\cdot)$ are fixed, the adversarial capabilities of the adversarial agents are indexed by the parameter τ. Our path-follwing strategy is to train the policy of the protagonist along the regularizaiton path by varying τ from 0 to 1. For simplicity, we set the functions $v_\ell(\cdot), s(\cdot), \rho^2(\cdot)$ to be linear in this paper. More sophisticated parameterization paradigms are also possible.

To train the protagonist's poicy along the path, we conduct reinforcement learning of the protagonist in an environment with adversaries of minimum difficulty $\tau = 0$. We then increase the aggressiveness level of the adversary agents to a higher level according to some pre-defined updating rule and use the learnt policy from the previous step for initialization. The process keeps going until reaching the hardest setting corresponding to $\tau = 1$ as Algorithm **??** shows.

The above path-follwoing algorithm requires an updating rule of the parameter τ. We consider 3 updating methods: (i) *discrete*, (ii) *sigmoid* and (iii) *linear*. Let E be the number of maximum

Algorithm 1 Path-following Policy Learning

Input: Transition model T of the Markov game

Initialize $\tau \leftarrow 0$; policies $\pi_1, \ldots, \pi_N$; max episodes E
for episode $= 1, \ldots, E$ **do**
  for m **steps do**
    Each agent selects action $a_i \sim \pi_i(o_i)$
    Step forward $s', r \leftarrow T(s, a_1, \ldots, a_N)$
    Each agent collects trajectory (s, s', a, r)
    Update π_i using collected data
  Update τ using the updating rule
return π_1^*

episodes and i be the episode index. The discrete rule generates a path with only 3 values of $\tau = 0, 0.5, 1$, the sigmoid rule generates a path by varying τ along a sigmoid-shaped curve $1/(1 + e^{-(i-E/2)})$. The linear rule generates a path by varying τ along a linear line from 0 to 1.

3.1.4.3. Protagonist Setting. To train the protagonist, we use the same PPO algorithm (Schulman et al., 2017) as described in (Long et al., 2018) with some minor adaptions (e.g., the input dimension of the LIDAR has different dimensions). For the purpose of completeness, we briefly describe the setups in this section to ease the audience.

Observation Space and Action Space: The observations of the protagonist contain laser readings $o_\ell \in \mathbb{R}^{3\times 360}$ (Representing data from the 3 most recent frames of the LIDAR scan, each of which is a 360-dimensional vector), current velocity $o_v \in \mathbb{R}^2$ including both linear and angular velocities and the target location's relative polar coordinates $o_g \in \mathbb{R}^2$ with respect to the agent's local coordinate system. Note that each value of the laser reading o_ℓ lies in between 0 to 3.5 which is the range of laser beam.

Reward Setting: To navigate the agent, the reward function is the sum of 4 components:

$$r^{(t)}(s^{(t)}, a^{(t)}) = r_d^{(t)} + r_g^{(t)} + r_c^{(t)} + r_w^{(t)}.$$

Here $r_d^{(t)} = w_d \cdot (\|\mathbf{p}_1^{(t-1)} - \mathbf{p}_g\|_2 - \|\mathbf{p}_1^{(t)} - \mathbf{p}_g\|_2)$ guides the agent to move towards the goal. $\mathbf{p}_1 = [\mathbf{p}_1^x, \mathbf{p}_1^y] \in \mathbb{R}^2$ and $\mathbf{p}_g^{(t)} = [\mathbf{p}_g^x, \mathbf{p}_g^y] \in \mathbb{R}^2$ are the current and goal positions of the protagonist.

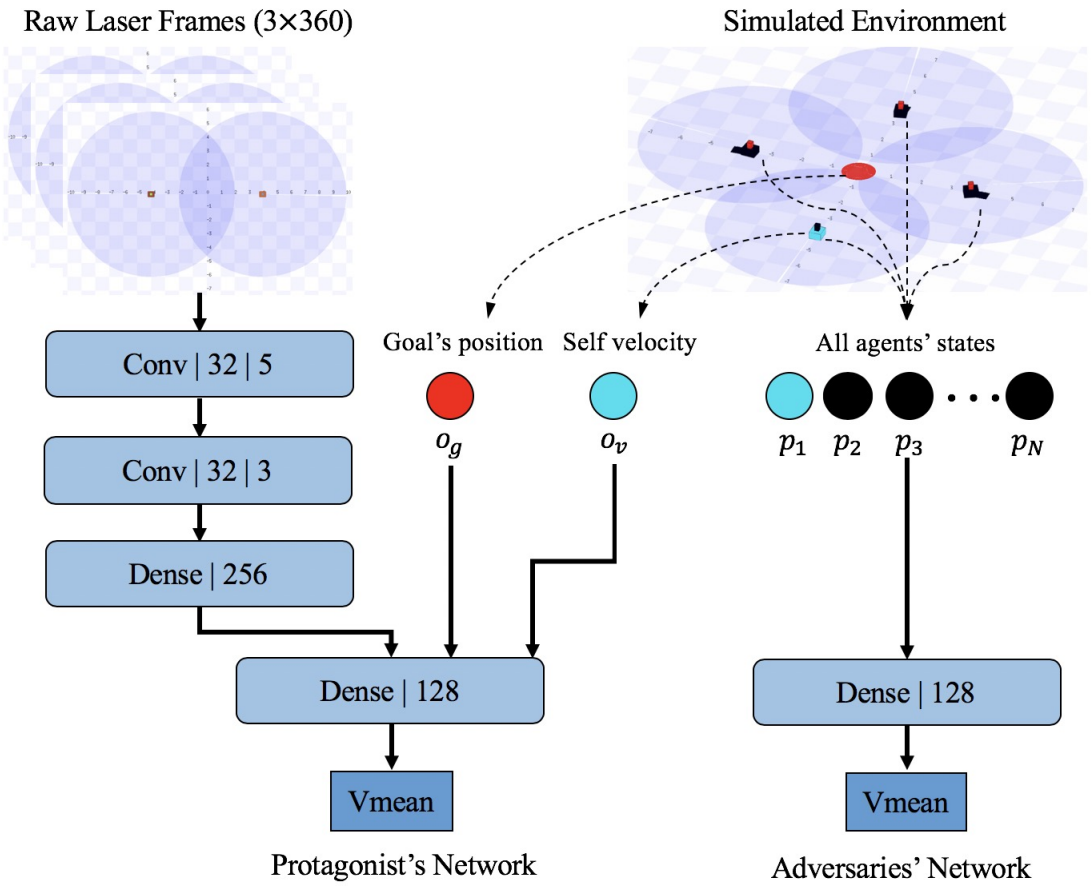
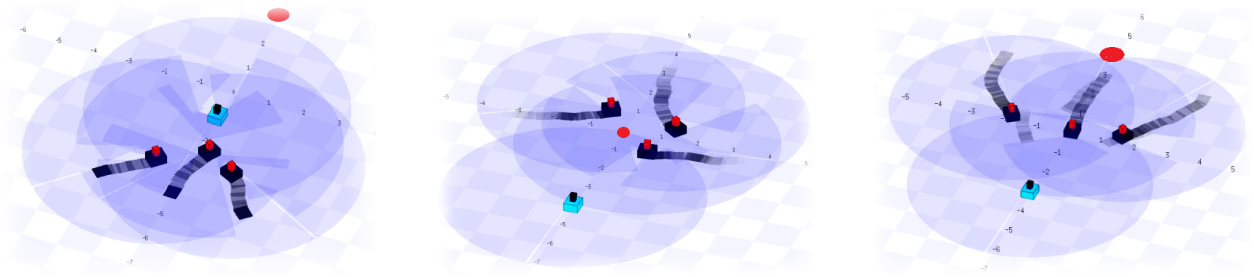


Figure 3.2. Network Architecture of Protagonist and Adversary. Conv represents convolution layer followed by its dimension and kernel size. Dense represents fully-connected layer with its dimension.

We set the weight $w_d = 2.5$. Reward for reaching the goal $r_g^{(t)} = 15$ if $\|\mathbf{p}_1^{(t)} - \mathbf{p}_g\|_2 < 0.5$. We set $r_c^{(t)} = -15$ if a collision happens and similarly $r_w^{(t)} = w_\omega \cdot |\omega^{(t)}|$ if $\omega^{(t)} > 1.05$ with $w_\omega = -0.1$ is used to force the protagonist moving more smoothly.

Network Architecture: The network architecture is shown in Figure 3.2. The LIDAR data is fed to two convolution layers and one 256-dimensional fully-connected (fc) layer. The processed laser information is then concatenated with the other two observations o_g and o_v. The concatenated vector is fed into a 128-dimensional fully-connected layer to generate the mean of the velocity v_{mean}. We then construct a Gaussian policy with v_{mean} as the mean parameter and a variance parameter v_{std} trained in the same way as in (Long et al., 2018). In addition to the policy network, the PPO algorithm also involves a value network which is nearly the same except the last fully-connected



(a) Chasing (b) Blocking (c) Crossing

Figure 3.3. The 3 Adversarial Paradigms: Chasing, Blocking and Crossing. The blue cube represents the protagonist, the black ones represents adversaries. The red circle denoting protagonist's goal location. The trajectories of adversaries present different attacking strategies.

layer outputs a value for judging the policy's performance. More details can be found in (Long et al., 2018).

Adversarial Setting: Unlike most multi-agent navigation methods which are either fully centralized with all agents share the same policy (Van Den Berg et al., 2011) or fully decentralized with all agents only have limited partial observations (Long et al., 2018; Chen et al., 2017), our setting have all adversaries centralized but with different policies and the protagonist decentralized. This allows us to fully harness the benefit of adversarial learning to obtain a more robust policy for the protagonist. Since all adversaries are centralized, each one has an observation containing the position information of all the agents: $o_a^{(t)} = [\mathbf{p}_1^{(t)} \dots \mathbf{p}_N^{(t)}, \mathbf{p}_g^{(t)}]$, where $\mathbf{p}_i^{(t)} = [\mathbf{p}_i^x, \mathbf{p}_i^y] \in \mathbb{R}^2$ represents the current position of the i-th agent. $\mathbf{p}_g^{(t)}$ is the goal position of the protagonist. The observation $o^{(t)}$ is fed into a neural network with the same architecture as in Figure 3.2 but each adversary agent independently trains the network parameters. Similarly, a Gaussian policy is constructed for each adversary with the output action $a^{(t)} = [v_l, v_w] \in \mathbb{R}^2$ containing linear and angular velocities. To further promote the diversity of the adversarial agents, we consider 3 adversarial paradigms shown in Figure 3.3. These paradigms have different attacking strategies characterized by different reward mechanisms. To train the protagonist, we simultaneously launch multiple instances of the adversarial agents under different paradigms. The policy networks of the adversaries from the same paradigm

share the same parameters. The protagonist is simultaneously trained in these instances using the path-following strategy. We describe the 3 adversarial paradigms below.

Chasing: In this paradigm, 3 adversaries are chasing the protagonist and are rewarded when catching it. The reward function of adversaries is:

$$r^{(t)}(s^{(t)}, a^{(t)}) = r_d^{(t)} + r_c^{(t)},$$

where $r_d^{(t)} = w_d \cdot (\|\mathbf{p}_i^{(t-1)} - \mathbf{p}_1\|_2 - \|\mathbf{p}_i^{(t)} - \mathbf{p}_1\|_2)$ with $w_d = 2.5$ and $r_c^{(t)} = 15$ if collision with protagonist happens. It is easy to see that $r_d^{(t)}$ encourages an adversary to chase the protagonist and $r_c^{(t)}$ encourages the collision. One thing to note is that if two adversaries collide, there is no punishment but will terminate the current episode and all the adversaries will be relaunched at new random positions.

Blocking: This paradigm has 3 adversarial agents trying to prevent the protagonist from reaching the goal position with the following reward function:

$$r^{(t)}(s^{(t)}, a^{(t)}) = (r_d^{(t)} - r_{d'}^{(t)}) + r_c^{(t)},$$

where $r_d^{(t)}, r_c^{(t)}$ are the same as in the chasing section and $r_{d'}^{(t)} = w_d \cdot (\|\mathbf{p}_1^{(t-1)} - \mathbf{p}_g\|_2 - \|\mathbf{p}_1^{(t)} - \mathbf{p}_g\|_2)$ with $w_d = 2.5$ encourages the protagonist to move towards the goal $\mathbf{p}_g$. It is easy to see that the component $r_{d'}^{(t)}$ encourages the adversaries to block the protagonist away from the goal.

Crossing: This paradigm is exactly the same as the chasing paradigm except the goal position is purposely put behind the 3 adversaries. The reason for designing this paradigm is that crossing is the one of the most common scenarios encountered by a human-centered robot. Many traditional approaches assume the humans follow social norms or reciprocal policy and should actively avoid colliding robot. However, in real applications, there could be some curious huamans (e.g., a kid in the grocery store) who tend to get close to the robot and that is the scenario we aim to model.

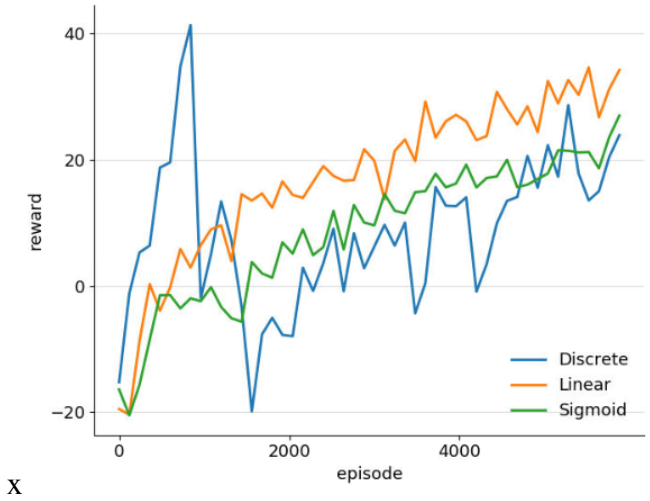
3.1.5. Experiment

We present experimental results to demonstrate the efficacy of the proposed method. In particular, we show that our method outperforms the existing state-of-the-art collision-free navigation methods in both benign and adversarial environments. We also deploy our learnt policy to a physical robot and demonstrate that the trained agent is directly applicable to real-world physical environment.

3.1.5.1. Computational Details. We conduct the multi-agent robots simulation in `Stage` (Vaughan, 2008) with the algorithm implemented in PyTorch(Paszke et al., 2017). The network architectures are the same as shown in Figure 3.2. The maximum linear speed of the mobile robot during training is 1m/s and the 2D LIDAR equipped on the robot senses 360 degree and 0-3.5m distance. We train collision-free policy on a server with 2 Xeon 8168 CPUs (48 cores), 1TB memory and 4 Nvidia GTX 2080 GPUs. The collision-free policy training takes 4 hours (about 2000 episodes) to converge to a robust solution in all adversarial paradigms.

3.1.5.2. Experimental Design. Evaluation Metrics: We consider the following metrics adopted from (Long et al., 2018): (i) *Success rate*: The proportion of trials that the protagonist successfully reaches the target position without any collision. (ii) *Extra time*: The difference between the average travel time of the protagonist over all testing cases and the lower bound of the travel time measured by robot moving straightly toward goals without adversarial interference. (iii) *Extra distance*: The difference between average travel distance over all testing cases and the travel distance measured by the protagonist moving straightly towards the goal without adversarial interference. (iv) *Average speed*: The average speed of the protagonist achieving goal.

Baseline Methods: We consider the following baseline methods: (i) ORCA (Van Den Berg et al., 2011), a centralized method that have a central sever controlling all agents. (ii) CADRL (Everett et al., 2018), an agent-level decentralized method with each agent accesses precise state information. The purpose of comparing with CADRL is to show that even with more crude input, our method achieves higher performance. (iii) DRLMACA (Long et al., 2018), a fully decentralized method in which a same policy is trained for all agents in several multi-agent environments.



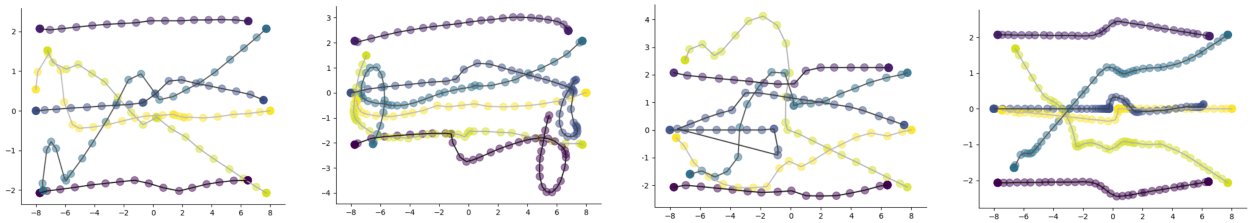
x

Figure 3.4. Protagonists' reward curves for collision avoidance policy learning. The learning curves of *Discrete*, *Linear*, *Sigmoid* path-following policy learning strategy converge around the reward 20.

3.1.5.3. Policy Training. Figure 3.4 evaluates the influence of different updating rules of τ (noted as *Discrete*, *Sigmoid* and *Linear*) on policy training. We see two dramatic drop down of the blue curve (discrete) corresponding to where adversaries change their aggressiveness intensity. Whenever τ increase 0.5 after each training epoch, it brings dramatic hardness to the environment. However, after several episodes, the protagonist becomes accustomed to the new switch and converges by the end. Unlike the bumpy curve from the *discrete* case, the other two reward curves change more smoothly with the *Linear*'s reward slightly outperforms the *sigmoid*'s. The reason could be the hardness of *sigmoid* starts increasing more rapidly than *linear* in the halftime which is more challenging for the protagonist to adapt. At the end, no matter in which way, all three methods converge to the same level.

3.1.5.4. Simulation Results. We compare our method with the baselines under both non-adversarial Figure 3.5 and adversarial environments Figure 3.3. All agents have identical size and speed same across different methods.

Non-adversarial Scenario: We place 6 robots in two groups facing each other. They're required to reach the positions their opposite agents located. We visualize the obtained trajectory plots in Figure 3.5. As expected, all methods achieve high success rate in this non-adversarial scenario.



(a) Our method (b) DRLMACA (c) CADRL (d) ORCA

Figure 3.5. The trajectory plots of 6 robots that are equally divided into two groups trying to reach the opposite positions. The interval between two adjacent footprints is 1s. ORCA moves extremely slow inferred by dense points in a trajectory but almost following the optimal path. CADRL moves fast while traveling longest. Some redundant trajectories shaped in circles are observed using DRLMACA. Our method achieves the best balance between time and travelling distance.

However, the CADRL and ORCA exploit global information which is not needed by our method. We observe that CADRL and ORCA chose completely opposite strategies for collision avoidance. CADRL is more aggressive with very high speed but takes much longer distance to reach the goal. In contrast, ORCA behaves extremely cautious and spend a large amount of time carefully calculating the shortest path. DRLMACA and our method achieve a better comprise of these two extremes with our method outperforms DRLMACA.

Also, from Figure 3.6, our method significantly outperforms the the remaining methods, with higher speed, shorter time and distance. We also see a gradient pattern within our three updating rules across all 4 histograms: the *discrete* performs best, followed by *linear* and *sigmoid*. It indicates that sharp intensity change brings more robustness to mobile robot.

Adversarial Scenario: Having demonstrated the superior performance of our method in normal setting, we now compare the policies learnt using our method with the three baselines in the adversarial scenarios described in Figure 3.3. In these experiments, the protagonist is controlled by each policy being compared while the adversaries are controlled by the adversarial policies obtained from adversarial training described in Section 3.1.5-C.

We present the comparison results in Figure 3.7. It is easy to see that our method outperforms all other baseline methods in the adversarial environments and behaves especially competitive in

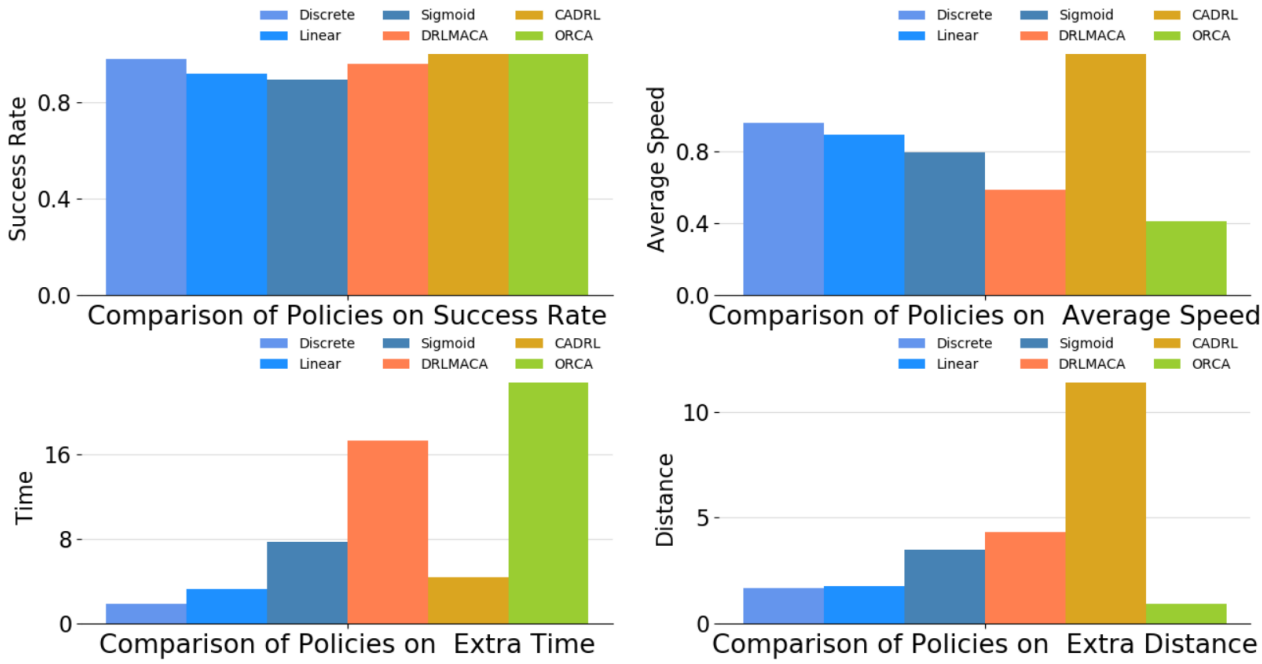


Figure 3.6. Performance metrics in non-adversarial scenario. We see that all methods achieve high success rate, but our methods performs better on extra time and extra distance.

the chasing paradigm. This is due to the fact that the other baseline methods rarely consider the scenario that the adversarial agents might move towards the protagnist from behind.

ORCA behaves badly as expected since its reciprocal assumption of t he adversarial agents is violated. In contrast, CADRL handles the adversarial environment pretty well since the protagonist has perfect knowledge of the adversaries' intents. Compared to the non-adversarial scenario, CADRL still achieves the highest speed but with a lower success rate. It's worth noting that DRLMACA has an abnormally high extra distance and travel time. It seems terribly frightened once the nearby agents deliberately move towards it rather than actively moving away.

3.1.5.5. Real-World Experiments. To test the transferability of our trained policy to physical robots in the real world, we deploy the policy trained using the *Discrete* updating rule on a mobile robot platform `Turtlebot3 waffle_pi`. This robot is equipped with LDS-01, a 2D LIDAR sensing 360 degrees with 0.12 to 3.5 meter range and a scan rate of 300 ± 10 rpm. The robot is controlled with the maximum speed 1m/s. The on-board computer of `waffle_pi` is `Raspberry Pi 3` with Robot Operating System (ROS) installed. Our control policy samples actions at a rate of 50Hz.

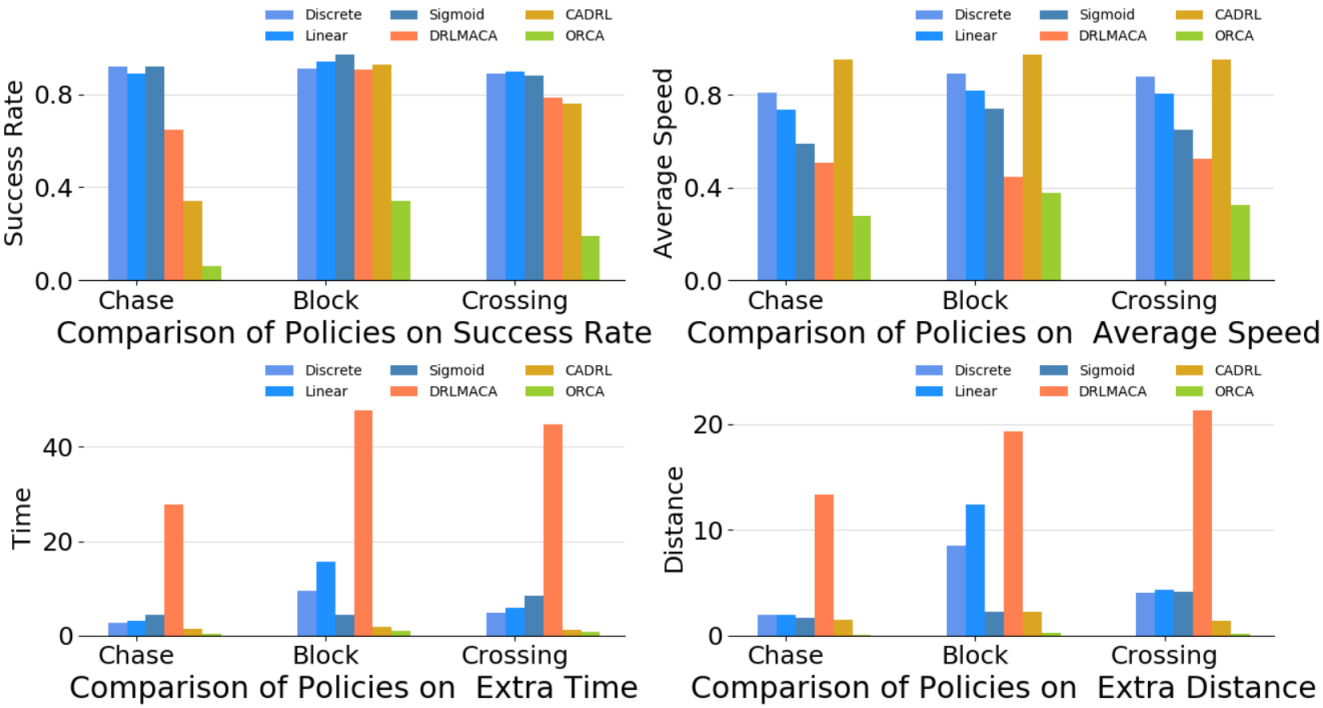


Figure 3.7. Performance metrics in adversarial scenarios. Our method performs the best in all four metrics while DRLMACA has abnormal behaviours when confronting adversarial agents.

In the real-world deployment, we first place the robot in an environment with dense pedestrians randomly moving around, then test the policy when several humans try to block it from different directions as shown in Figure 3.8. For the experiments, we randomly generate goal positions for the robot. Using our the learnt policy, the robot navigates very smoothly in many challenging scenarios, demonstrating the practical efficacy of our method.

3.1.6. Conclusion

We propose to learn robust collision-avoidance policies via Markov games by introducing multiple auxiliary agents to model a competing environment. The experiments show that our path-following adversarial policy learning strategy significantly improves the sample complexity and adaptivity of the trained primary agent (i.e., the robot). Promising experimental results on a physical robot also demonstrate that the superior performance of the learnt policy is generalized to new scenarios.

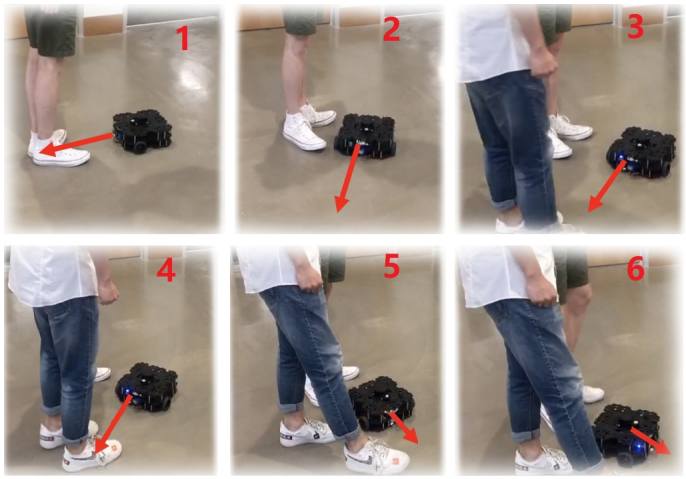


Figure 3.8. A real-world deployment. We deploy our collision avoidance policy on `turtlebot3`. The red arrows represent the moving directions of the robot. The result shows that when two pedestrians try to maliciously block the robot, our trained collision avoidance policy still find a collision-free path for the robot.

Future work will consider combining tree-based random search global planning algorithms with our learnt policy to navigate robots in crowded scenarios.

3.2. Switch Trajectory Transformer with Distributional Value Approximation for Multi-Task Reinforcement Learning

We propose SwitchTT, a multi-task extension to Trajectory Transformer but enhanced with two striking features: (i) exploiting a sparsely activated model to reduce computation cost in multi-task offline model learning and (ii) adopting a distributional trajectory value estimator that improves policy performance, especially in sparse reward settings. These two enhancements make SwitchTT suitable for solving multi-task offline reinforcement learning problems, where model capacity is critical for absorbing the vast quantities of knowledge available in the multi-task dataset. More specifically, SwitchTT exploits switch transformer model architecture for multi-task policy learning, allowing us to improve model capacity without proportional computation cost. Also, SwitchTT approximates the distribution rather than the expectation of trajectory value, mitigating the effects of the Monte-Carlo Value estimator suffering from poor sample complexity, especially in the sparse-reward setting. We evaluate our method using the suite of ten sparse-reward tasks from the

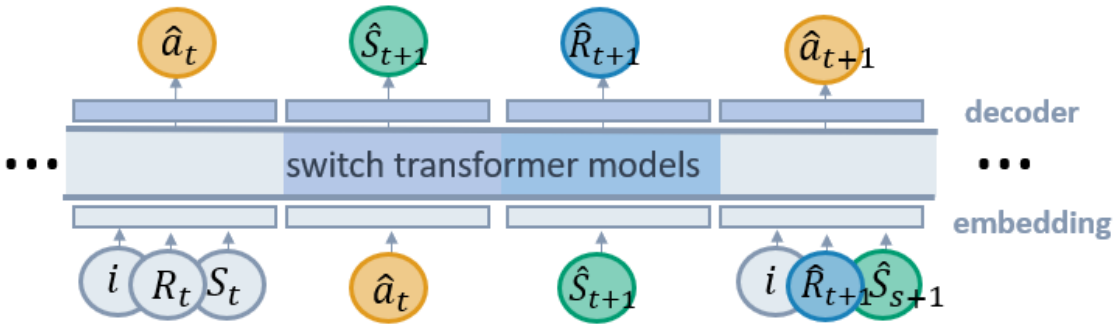


Figure 3.9. Switch Trajectory Transformer Architecture. Switch transformer models contain action transformer, dynamics transformer and return transformer. Task id, returns, states are fed into embedding layer and consequently in to the action transformer. The model predicts the following action and then provides it to the dynamics model for predicting the next state. The return transformer uses this information and predicts return; this process is repeated until the designed horizon length.

gym-minigrid environment. We show an improvement of 10% over Trajectory Transformer across 10-task learning and obtain up to 90% increase in offline model training speed. Our results also demonstrate the advantage of the switch transformer model for absorbing expert knowledge and the importance of value distribution in evaluating the trajectory.

3.2.1. Introduction

This paper studies the problem of multi-task offline reinforcement learning (RL). We first define a multi-task offline RL problem as learning a single policy that solves multiple tasks from previously collected data without online interaction with the environment. For example, suppose we want grocery robots to acquire a range of different behaviours (e.g. lift cans, pick up bowls and open closet). In that case, it is more practical to learn an extensive repertoire of behaviours using all previously collected datasets rather than learn each skill from scratch.

The large diversity of datasets collected in various tasks brings difficulty for traditional multi-task offline RL methods (Yu et al., 2021b,a; Kalashnikov et al., 2021b). Specifically, these methods emphasize transferring skill knowledge across related tasks and developing a sharing experience across different tasks. Such a data-sharing strategy makes learnt multi-task policy sensitive to data

distribution differences and relationships among tasks. The inherent conflict from task differences can harm the policy of at least some of the tasks, particularly when model parameters are shared among all tasks.

Recent offline RL works like Decision Transformer (Chen et al., 2021), and Trajectory Transformer (Janner et al., 2021b), abstracting RL as sequence modelling, demonstrate the capability of turning large datasets into powerful decision-making engines. Such modelling design benefits multi-task RL problem by serving a high-capacity model for handling task differences and absorbing vast knowledge in the collected diverse dataset, and also makes it possible for multi-task RL methods to adopt the associated advances (Fedus et al., 2021) in language modelling problems.

However, adopting such high-capacity sequential models for solving the multi-task RL problem poses three significant algorithmic challenges. The first is high computation cost and tall model capacity, which is critical for absorbing the vast knowledge available in a large heterogeneous dataset. The second one is sharing the same policy parameters across different tasks causing degraded performance over simple single-task training. The third is poor performance caused by the Monte Carlo value estimator, especially in sparse reward settings. Monte Carlo estimator suffers from poor sample complexity when online data collection is not allowed, and it is uninformative to guide the beam-search-based planning procedure.

To handle these challenges, we propose SwitchTT (Switch Trajectory Transformer), a multi-task extension to Trajectory Transformer but enhanced with two striking features. The high-level overview of SwitchTT is illustrated in Figure 3.9. First, unlike Trajectory Transformer and other traditional multi-task RL methods, reusing the same parameter for all input data, our method exploits a sparsely activated model for multi-task offline model training, with the sparsity coming from selecting different model parameters for each incoming example. Such a model allows us to perform efficient computation in high-capacity neural networks and improve parameter sharing in multi-task learning. Second, SwitchTT develops a trajectory-based distributional value estimator for learning the value distribution of trajectory instead of expected value. Such a distributional estimator enables us to

measure and utilize uncertainty around the reward, thus mitigating the effects of the Monte-Carlo Value estimator suffering from poor sample complexity, especially in sparse-reward setting, leading to better value estimate in an offline setting. We provide more details in Section 3.2.4.

This paper has two major contributions: (i) we exploit a sparsely activated model for multi-task model learning, which reduces the computation cost and improves multi-task learning performance. (ii) we develop a trajectory-based distributional value estimator to learn a better value estimator, improving offline reinforcement learning performance. The rest of the paper is organized as below. Section 3.2.3 introduces preliminaries. Section 3.2.4 describes the implementation of SwitchTT. Section 3.2.5 presents detailed experiment results to demonstrate advantage of SwtichTT. Section 3.2.2 introduces the related works and Section 3.2.8 concludes the paper with more discussion.

3.2.2. Related Works

Transformer for Reinforcement Learning. Decision Transformer (Chen et al., 2021) model Reinforcement Learning (RL) as a sequence modelling problem and matches or exceeds the performance of state-of-the-art model-free offline RL baselines. Based on this promising result, recent works draw upon the simplicity and scalability of the Transformer architecture to solve the reinforcement learning problem. These works can be divided into two kinds of approaches. The first is similar to Decision Transformer, which can be viewed as model-free RL at a high level. This suite of frameworks (Shang and Ryoo, 2021; Yang and Nachum, 2021) model the conditional distribution of actions given trajectory data. The other line of work (Janner et al., 2021a; Chen et al., 2022; Hafner et al., 2019) is viewed as model-based RL at a high level, which not only models the conditional distribution of action given trajectories but also model the state transition over the trajectory. The latter approach is more reliable in solving long-horizon sparse-reward tasks because the learned dynamics allow for future trajectories. The planning phase predicts the cumulative reward over a long horizon. Our method lies in the second line of work, which learns the state transition over trajectory data and utilizes the learned model to imagine the future trajectory.

Different from trajectory transformer, learning the value estimation by temporal difference learning, we improve the Monte Carlo value estimate by introducing distributional value learning. Instead of estimating the expected value return, we model the value learning as a categorical distribution and utilize the transformer to learn the distribution.

Model-based Offline Reinforcement Learning. Existing works have demonstrated the promise of model-based RL for offline learning. A command approach for model-based offline RL focuses on learning a dynamics model for uncertainty estimation and then optimizing the policy. For example, Model-based Offline Reinforcement learning (MOReL) firstly learns a pessimistic MDP from offline data using Gaussian dynamics model and then learns a policy for the learned MDP (Kidambi et al., 2020); Model-based Offline Policy Optimization (MOPO) estimates learned model error and penalizes rewards by such error to avoid distributional shift issues (Yu et al., 2020b); Offline Reinforcement Learning from images with Latent Space Models (LOMPO) extends MOPO to high-dimensional visual observation spaces (Rafailov et al., 2021). These approaches do not directly use the learned model to plan action sequences. MuZero combines a tree-based search with a learned model, using the learned model directly for policy and value improvement through online planning (Schrittwieser et al., 2020). MuZero Unplugged (Schrittwieser et al., 2021) extends MuZero to an offline-RL scenario and achieves state-of-the-art results. In contrast, our method utilizes sequence modelling tools to model the distribution of trajectories in the offline dataset. Such a high-capacity sequence model architecture provides a more reliable long-horizon predictor than a conventional dynamics model. It mitigates the effect of accumulated predictive error over a long horizon.

Multi-Task Reinforcement Learning. Multi-task Reinforcement Learning (RL) aims to learn a single policy that efficiently solves multiple skills. Prior works have made promising result but still face three major challenges, including optimization difficulties (Schaul et al., 2019; Hessel et al., 2019; Yu et al., 2020a), effective weight sharing for learning shared representations (Teh et al., 2017; Espeholt et al., 2018; Xu et al., 2020; D'Eramo et al., 2019; Sodhani et al., 2021; Stooke et al.,

2021) and sharing data across different tasks (Eysenbach et al., 2020; Kalashnikov et al., 2021a; Yu et al., 2021b). We study the challenge of effective weight sharing for learning shared representations in the multi-task offline RL setting. Current works focus on learning shared representation across different tasks, then apply traditional RL like computing policy gradient or learning value function to solve multiple tasks. In contrast, we abstract the multi-task RL problem as a sequence modelling problem and apply a high-capacity transformer model to solve the multi-task RL problem.

Transformer. Transformer (Vaswani et al., 2017) has been proposed as a novel model architecture to handle sequential input data in machine translation tasks. Since then, transformer-base pretrained language models like GPT-2/3 (Radford et al., 2019; Brown et al., 2020b), BERT (Devlin et al., 2018), XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019), T5 (Raffel et al., 2019), ALBERT (Lan et al., 2019), BART (Lewis et al., 2019) have achieved tremendous success in NLP because of their ability to learn language representations from large volumes of unlabeled text data and then transfer this knowledge to downstream tasks. In light of the above works, researchers are tempted to investigate the benefit of transformer models in improving reinforcement learning performance. The first line of work applies the transformer model to represent the component in standard RL algorithms, such as policy, models and value functions (Parisotto et al., 2020; Parisotto and Salakhutdinov, 2021). Instead of this, the second line of work (Chen et al., 2021; Janner et al., 2021b) abstracts RL as a sequence modelling problem and efficiently utilize the existing transformer framework widely used in language modelling to solve the RL problem. Intuitively, the latter approach is more influential since it can support the possibility that the advances in sequence models can directly be applied to the RL problem without relying on the RL algorithm framework. Motivated by the second approach, we model multi-task reinforcement learning problem as a multi-lingual task problem in Natural Language fields, which allows us to draw upon existing advanced transformer frameworks (Fedus et al., 2021; Lepikhin et al., 2020). Unlike previous works learning single task, our work makes efficient use of a high-capacity model to acquire multiple tasks inside a single model.

Mixture of Experts. The mixture-of-experts(MoEs) approach (Jordan and Jacobs, 1994; Jacobs et al., 1991) was proposed more than two decades ago to divide the problem space into homogeneous regions. Recent works on MoEs can be divided into two kinds of approaches. The first is to propose different types of architecture such as SVMs (Collobert et al., 2002), Bayesian Methods (Waterhouse et al., 1996) and Gaussian Processes (Tresp, 2001). While the above work considers mixture-of-experts as the whole model, the second kind of work use mixture-of-expert as a part of neural network. Recent work (Eigen et al., 2013) extends MoEs to use a gating network at each layer in a multilayer network, forming a Deep Mixture of Experts. Based on this idea, Shazeer (Shazeer et al., 2017) uses MoEs as a general-purpose neural network component and significantly advances state-of-the-art results on public language modelling data sets. GShard (Lepikhin et al., 2020) and Switch Transformer (Fedus et al., 2021) then adopt the MoEs into transformer model architecture to scale the model size efficiently. This dramatically reshaped the landscape of natural language processing research. It's intuitive that the latter approach efficiently scales the model capacity and improves model performance on complex problems since sub-problems in a complex problem require different expert solvers. Inspired by this novel architecture, we extend the MoEs to solve the problem of multi-task reinforcement learning. In our work, the MoEs layer consists of several feed-forward sub-networks, and a trainable gating network. The gating network determines a sparse combination of these experts to use for each task. The sub-networks here are considered as policy experts to solve the multi-task problem.

3.2.3. Preliminaries

Offline Reinforcement Learning. Here, we define the essential reinforcement learning (RL) concepts, following standard textbook definitions (Sutton and Barto, 2018). Reinforcement learning addresses the problem of learning to control a dynamical system in a general sense. The dynamical system is fully defined by a Markov decision process (MDP). The MDP is defined by the tuple $M = (\mathcal{S}, \mathcal{A}, r, P, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the continuous action space,

$r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and ρ_0 represents the initial state distribution. $P(s' \mid s, a)$ represents the transition probabilities, which specifies the probabilities of transition from the state s to s' under the action a. A trajectory is made up of a sequence of states, actions, and rewards: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$. The return of a trajectory at timestep t, $R_t = \sum_{t'=t}^{T} r_{t'}$ is the sum of future rewards from that timestep. The goal of RL is to find an optimal policy that maximizes the expected return $\mathbb{E}[\sum_{t=1}^{T} r_t]$ in a MDP. Different than online RL, which involves iteratively collecting experience by interacting with the environment, offline RL learns the optimal policy on fixed limited dataset $\mathcal{D} = \{(s_j.a_j.s'_j, r_j)\}_{j=1}^{N}$, consisting of trajectory rollouts of arbitrary policies.

Multi-Task Reinforcement Learning. The goal of multi-task RL is to find a optimal policy that maximizes expected return in multi-task Markov Decision Process (MDP), defined as $M = (\mathcal{S}, \mathcal{A}, \{r_i\}_{i=1}^{N}, P, \gamma, \rho_0)$, where $\{r_i\}_{i=1}^{N}$ is a finite set of task and others follow the definition in the offline RL subsection. Each task i represents a different reward function r_i but shares the dynamics P. In this work, we focus on multi-task offline RL setting, aiming to find a policy $\pi(a|s)$ that maximizes expected return over all the task: $\pi^*(a|s) = \text{argmax}_\pi \mathbb{E}_{i\sim[N]} \mathbb{E}_\pi[\sum_{t=1}^{T} r_i(s_t, a_t)]$, given a dataset $\mathcal{D} = \cup_{i=1}^{N} \mathcal{D}_i$ where $\mathcal{D}_i$ consists of experiences from task i.

Trajectory Transformer. Trajectory Transformer (Janner et al., 2021b) formulates offline RL as generic sequence modeling problem. The core of this approach is to use a Transformer architecture to model distributions over trajectories in the offline dataset and repurposing beam search as a planning algorithm to find the optimal action. Specifically, Trajectory Transformer augments each transition in the trajectory τ with reward-to-go $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$, then discretizes a trajectory τ with N-dimension states and M-dimensional actions into sequence of length $T(N + M + 1)$: $\tau = (..., s_t^1, s_t^2, ..., s_t^N, a_t^1, a_t^2, ..., a_t^M, r_t, ...), t = 0...T$. To model distribution over such trajectories, they use a smaller-scale GPT (Radford et al., 2018) architecture, parameterized as θ and induced conditional probabilities as P_θ, maximising the following objective:

$$L(\tau) = \sum_{t=0}^{T}(\sum_{i=1}^{N} \log P_\theta(s_t^i|s_t^{<i}, \tau_{<t}) + \log P_\theta(s_t^i|a_t, s_t, \tau_{<t}) + \sum_{j=1}^{N} \log P_\theta(a_t^i|s_t^{<i}, \tau_{<t})) \tag{3.2.1}$$

Then, beam search uses past trajectory as the input of the trained model and greedily selects the predicted trajectory τ with the highest reward. Other details of Trajectory Transformer are referred to in the original paper (Janner et al., 2021b). In this work, we extend the trajectory transformer with two enhanced features to solve multi-task RL. The first is to exploit switch transformer instead of naive transformer architecture, and the second is to adopt a distributional value estimator to guide the beam search. We refer to more details to Section 3.2.4.

Switch Transformer. Switch Transformer is a sparsely activated model designed to maximize the parameter count of a Transformer model in a simple and computationally efficient way. The critical difference is that instead of containing a single feed-forward neural network (FFN) in the original transformer, each switch layer has multiple FFNs known as an expert. More specifically, the switch layer consists of a set of n "expert networks" $E_1, \cdots, E_n$ and a "gating network" G, whose output is a sparse n-dimensional vector. The experts are themselves neural networks, each with their parameters. This layer takes a token representation x as an input and then routes it to the best determined top-k experts, selected from a set of $E_i(x)_{i=1}^n$. The router variable W_r produces logits $h(x) = W_r \cdot x$ which are normalized via a softmax distribution over the available n experts at that layer. The gate-value for expert i is given by:

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^n e^{h(x)_j}}. \tag{3.2.2}$$

The top-k gate values are selected for routing the token x. If $\mathcal{T}$ is the set of selected top-k indices then the output computation of the layer is the linearly weighted combination of each expert's

computation on the token by the gate value,

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x). \tag{3.2.3}$$

Instead of routing to k experts, we route input tokens to only a single expert. Switch Transformer shows this simplification preserves model quality, reduces routing computation and performs better. The benefit of the switch layer for solving reinforcement learning problems is two-fold: (1) Each Expert is considered a task expert, such as lifting cans, picking up bowls, and opening a closet. In this way, we can combine multiple expert knowledge inside a single policy to solve numerous tasks. (2) The gating network is considered a switch strategy, which measures the confidence of each expert and chooses the expert with the highest confidence to solve each task.

3.2.4. Method

This section presents SwitchTT in detail, a multi-task extension of the Trajectory Transformer. The details of the method is illustrated in Figure 3.10. Following Trajectory Transformer, SwitchTT model multi-task offline RL problem as sequence modelling problem and divides the algorithm into three phases (data collection, offline model training and planning). Since our model and planning strategy is nearly identical to the Trajectory Transformer, we briefly describe three phases and emphasize the critical difference: the model architecture of the switch transformer model and implementation of the distributional value estimator.

3.2.4.1. Data Collection. In the data collection phase, we collect a combined dataset $\mathcal{D} = \cup_{i=1}^{N} \mathcal{D}_i$ in task $1..N$ and transform the dataset into a sequential representation. Specifically, instead of using certain-level expert policy to interact with the environment, we utilize the online RL algorithm PPO (Schulman et al., 2017) to solve each task and collect the replay dataset $\mathcal{D}_i$. This adds exploratory trajectory into the dataset and increases the diversity of trajectory inside the dataset. Here, we study discrete RL tasks other than continuous tasks so it is unnecessary to discretize states and actions like Trajectory Transformer(Janner et al., 2021a). Trajectories from different tasks are combined into a

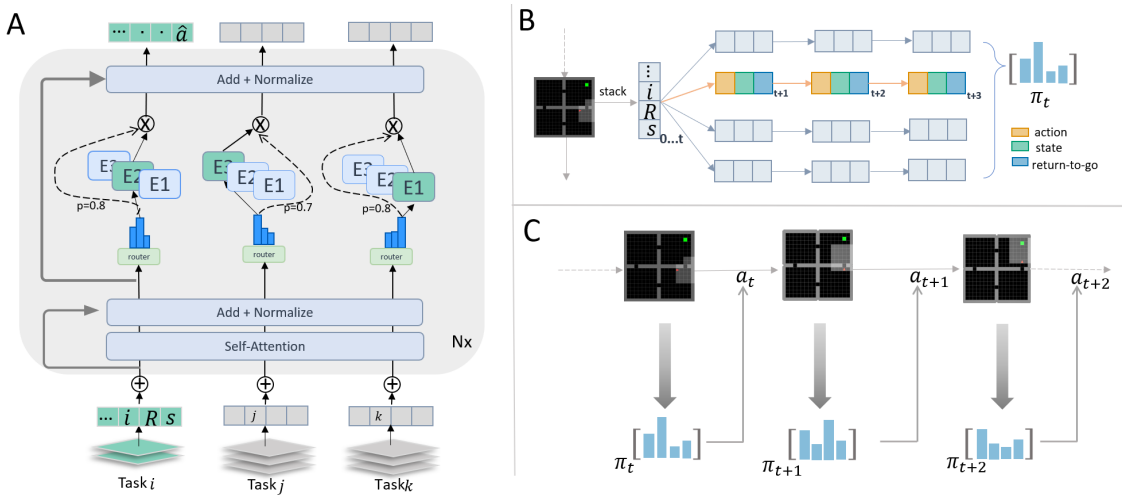


Figure 3.10. Training, planning, and action planner in the SwitchTT: (A) Represents the action transformer model training process. Here we diagram three task tokens being routed across different experts. At first, trajectories in the collected episodes are transformed into trajectory representation by adding task id and calculating return-to-go. Then it is fed as a token into a switch transformer model, which replaces the dense feed-forward network (FFN) layer present in the transformer with a sparse Switch FFN layer. The switch layer returns the output of selected experts multiplied by the router gate value. (B) is the planning process. Firstly all visited state, return-to-go, action and task id are stacked into a sequence $\{i, R_0, s_0, a_0, ...R_t, s_t\}$. Then the sequence is fed into switch transformer models to generate predicted four pairs of $\{a_t, s_{t+1}, R_{t+1}\}$. Stack again and generate only one pair until the designed length. Then this generates values of different chosen actions at the first generation. (C) The action planner. The planner is performed at each timestep as described in B. At each timestep, the highest column value action is sampled and executed to the environment.

single dataset $\mathcal{D}$ and each one is transformed into the trajectory representation following Decision Transformer (Chen et al., 2021). Then we feed the model with the returns-to-go $\widehat{R_t} = \sum_{t'=t}^{T} r_{t'}$. This leads to the following trajectory representation of task i:

$$\tau = (i, \widehat{R_0}, s_0, a_0, i, \widehat{R_1}, s_1, a_1, ..., i, \widehat{R_T}, s_T, a_T). \tag{3.2.4}$$

At test time, we feed the target return and first state (R_0, s_0) into the model and then get the desired action a_0 from the planning phase, depicted in the latter subsection. Here, R_0 represents the desired performance, usually the maximized cumulative reward of the task. s_0 is the first observed state from the environment. After executing the action, we receive (r_0, s_1) from the environment and

decrease the target return via equation $R_1 = R_0 - r_0$. Then we feed the current trajectory to the model and repeat until the episode terminates.

3.2.4.2. Training Phase. In the training phase, we train offline models modelling the trajectory distribution in the dataset $\mathcal{D}$ and use them for the planning phase. Instead of modelling the trajectory inside a single model like Trajectory Transformer, we train three separate models: (i) decision transformer θ, modelling the conditional distribution of actions given returns-to-go and states. (ii) dynamics transformer f, predicting state over past states and actions. (iii) return-to-go (RTG) transformer ϕ, a trajectory-based distributional value estimator, predicting value distribution of state-action trajectory. Here, the training procedure of each transformer is referred to in Chen et al. (2021). The training procedure of the three models is the same as the original Transformer model. The objective of the Decision Transformer model is to minimize the cross-entropy loss between predicted and true actions. We define the loss as a cross-entropy loss between predicted and true states for the Dynamics Transformer model. We also use switch transformer model architecture instead of transformer model and describe the distributional value estimator in detail.

We feed the last k timesteps into the decision transformer for a total of 4k tokens. For the dynamics transformer and return-to-go transformer, we provide previous k timesteps into them but only 3k tokens, including task id, state and actions. The output of the Dynamics Transformer is the next-timestep state, and the output of the return-to-go transformer is the distribution of the input trajectory's future return.

3.2.4.3. Model Architecture. We utilize a switch transformer to learn the distribution of the combined dataset. The model architecture is extended from GPT (Radford et al., 2018) model architecture. But we replace the dense feed-forward network (FFN) layer present in the transformer with a switch layer consisting of multiple FFN layers. Figure 3.10 illustrates the detail of the switch layer in the transformer models. When each token passes through this layer, it first passes through a router function and the router routes the token to a specific expert. Under the multi-task RL setting, we consider each expert as a policy model, and the router routes the observation from a task to the

highest-confidence expert. In our method, we implement a minor version of Switch Transformer architecture, and the detail of the switch layer is depicted in Switch Transformer (Fedus et al., 2021).

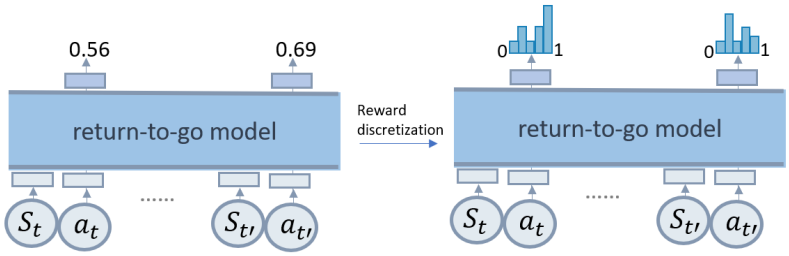


Figure 3.11. Overview of return-to-go transformer. Instead of predicting the expectation value of trajectory, return-to-go transformer discretise the reward and predicts the value distribution of the trajectory.

Trajectory-based Distributional Value Estimator. As shown in Figure 3.11, return-to-go transformer is a trajectory-based distributional value estimator that estimates the value distribution of the state-action trajectory instead of the expected return. Here, we model the distribution using a discrete distribution (categorical distribution) parameterized by $N \in \mathbb{N}$ and $V_{\text{min}}, V_{\text{max}} \in \mathbb{R}$. The support of such distribution is the set of atoms $\{z_i = V_{\text{min}} + i\triangle z : 0 \leqslant i < N\}$, $\triangle z = \frac{V_{\text{max}} - V_{\text{min}}}{N-1}$ and atom z_i probability is given by a parametric model $\phi : \mathcal{X} \to \mathbb{R}^N$, whose input is three-step state-action trajectory $x = \{s_t, a_t, s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}\}$ and output is the probability vector of atoms:

$$p_\phi^i(x) = \frac{e^{\phi(x)_i}}{\sum_j^N e^{\phi(x)_j}} \tag{3.2.5}$$

Then we project the value distribution learning onto a multiclass classification problem. Given a sample trajectory $\tau_t = \{\widehat{R}_j, s_j, a_j\}_{j=t}^{t+2}$ from dataset, we label the trajectory as class $c = \lfloor \widehat{R}_{t+3} - V_{\text{min}} \rfloor / \triangle z$ and use gradient descent to find parameter ϕ that minimize the cross-entropy loss between labeled class and discrete value distribution. In the planning phase, the output value of given trajectory x is the linearly weighted combination of each atoms' probability:

$$V_\phi(x) = \sum_{i=0}^{N-1} p_\phi^i(x) z_i. \tag{3.2.6}$$

Unlike TT predicting returnto-go(RTG) after immediate rewards, we predict RTG given sequence of (state, action), without relying on reward. This reduces RTG prediction error, which could be accumulated by reward prediction error in the planning phase due to the fact that prediction of RTG value can only depend on reward without state and action. Additionally, discretization is a case of distributional value estimator and distributional value estimator can be extended to learning continuous distribution.

The advantage of using distributional value learning is to learn a better value approximation. While the Monte Carlo value estimate suffers from poor sample complexity, especially in sparse-reward tasks, learning the distribution can preserve such effect. Specifically, with learned distribution of RTG, we can ignore RTG with low probability and calculated RTG based on high-probability value. Modeling distribution explicitly instead of only estimating the mean can lead to more knowledge for the agent. Recent work's results (Bellemare et al., 2017) show the usefulness of modeling the RTG distribution, leading to a much faster and more stable learning of the agent. Also, by leveraging the transformer model architecture, we learn the distribution in a simple self-supervised manner instead of dynamic programming as in the Distributional RL work (Bellemare et al., 2017).

Algorithm 2 SwitchTT Planner

Input: Transition model T of the Markov game

Initialize $\mathcal{T} = \{\}$, $\mathcal{A} = \{\theta(x)[i]\}_{i=0}^{c}$
for $i = 1$ **to** c **do**
    Sample i^{th} action candidate $a_1^i = \theta(x)[i]$
    Initialize trajectory $\tau_i = \{x, a_0^i\}$
    for $j = 1$ **to** h **do**
        Predict state $s_j^i = f(\tau_i)$
        Imagine trajectory $\tau_i = \tau_i \cup \{a_{j-1}^i, s_j^i\}$
        Sample action $a_j^i = \theta(x)[0]$
    Evaluate trajectory $\mathcal{T}[a_0^i] = V_\phi(\tau_i)$ with Eq. 3.2.6
return $\arg\max_{a \in \mathcal{A}} \mathcal{T}[a]$

3.2.4.4. Planning Phase. In the planning phase, we use beam search to generate optimal action with the highest estimated value based on the imagined sequence. We summarize this phase as Switch Planner and describe it in the Algorithm 2. The algorithm requires model (θ, f, ϕ) from training phase, hyperparameter candidate number c and horizon length h, current sequence x as input. Here, c determines the number of imagined trajectory, h defines the horizon length of each imagined trajectory, x is collected from past states, actions and rewards at timestep t: $x = \{R_i, s_i, a_i\}_{i=0}^{t}$. The planning phase in Figure 3.10 uses candidate number $c = 4$ and horizon length $h = 3$.

3.2.5. Experiment

In this section, we present the experimental results of our method and compare them with baselines methods in a multi-task setting. In particular, our experiment aims to answer the following question: (i) How well does our method perform on multi-task learning? (2) Does switch transformer speed up the training speeds of offline model learning. (3) Does switch transformer mitigate the effect of

degrading performance of multi-task learning over single-task learning? (4) Does the distributional value estimator improve multi-task performance?

3.2.5.1. Experiment Design. We evaluate our methods in 10 different tasks of the gym-mini-grid environment, including FourRooms, DoorKey, KeyCorridor, and the other seven tasks. We choose these tasks because they have sparse reward where by default, the agent only gets a positive reward when reaching the designed goal. This problem is difficult for policy learning because reward must be propagated from the beginning to the end of the episode when actions taken in the middle is skipped over. Also, these tasks contain different requirements, so the policy should be trained separately, although they have the same action and state space. For example, DoorKey is a simple sparse-reward problem; the Fourroms is a long-term credit assignment problem and KeyCorridor requires learning compositional tasks.

We compared our methods with Trajectory Transformer (TT), Decision Transformer (DT), Behavior Cloning. Our motivation for choosing these methods are: Our methods extend from Trajectory Transformer and are similar to Decision Trajectory, abstracting offline RL as a sequence modelling problem. Imitation learning is similar to our methods since it also uses supervised loss for training and planning from the trained models.

3.2.5.2. Performance in multi-task learning. Here, we firstly investigate the improvement of SwitchTT over TT and DT in multi-task learning. We collect a dataset across ten different tasks in a gym-mini-grid environment. The total amount of the dataset is 5 million timesteps combined dataset, 500k timesteps for each task. Combining all the data, we train DT, TT and Switch TT models and evaluate the trained models on the ten tasks. We report reward by running each task across 100 scenarios over 10 seeds. Secondly, we study the effect of the switch layer for multi-task learning. We compare SwitchTT with other baseline methods in three learning settings, 1-task learning, 3-task learning and 10 task learning. We report the performance of TT, DT, BC, which use the same model architecture and hyperparameters. For the beam width and planning horizon of TT, we refer the same value as the Fourroms Experiment in paper(Janner et al., 2021a). DT-Switch and

SwitchTT use the same model architecture and hyperparameters as TT, DT, BC, but replace the FFN layer with the same model-size switch layer. We use a context length of k=30 in both experiments in all trained transformer models. In the switch layer, we use the expert number of n=3 for 3-task learning and n=8 for 10-task learning.

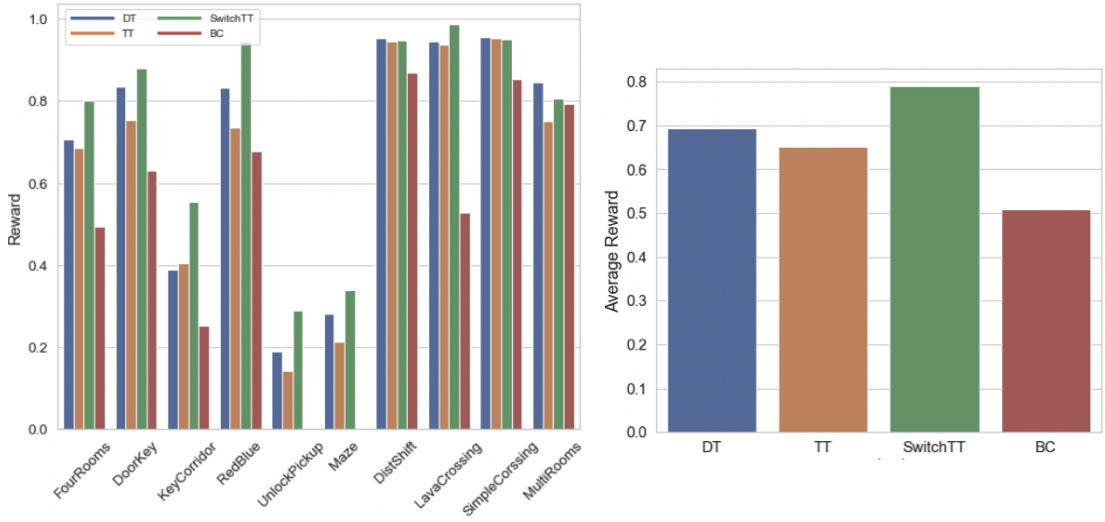


Figure 3.12. The left panel shows evaluation of Switch TT across 10 tasks. DT and TT train transformer models with FFN layer on the combined 10-task datasets. SwitchTT train transformer models with Switch layer on the same dataset. SwitchTT provides an improvement over 3 baseline methods across 80% of tasks. The right panel reports average reward of 10-task learning. We train offline models on the 10-task dataset, evaluate the performance on ten tasks, and report the average reward of 10 tasks in 10-task learning.

As shown in the left panel of Figure 3.12, SwitchTT outperforms DT, TT across 80% of tasks. In particular, SwithchTT improves about 15% performance over other methods in the FourRooms, DoorKey and Maze tasks, which require long-horizon planning ability. This highlights that the transformer models with switch layers learn a better-matched trajectory distribution in the dataset than FFN layers. Also, the distributional value estimator benefits SwitchTT by providing a more accurate value for estimating the imagined trajectories.

The right panel of Figure 3.12 shows that SwitchTT achieves the best result in 10-task learning. We observe that our methods improve about 10% over our base methods TT. These highlights switch

layers improve the offline model learning, and also the distributional value estimator also learns a better value in a sparse-reward setting.

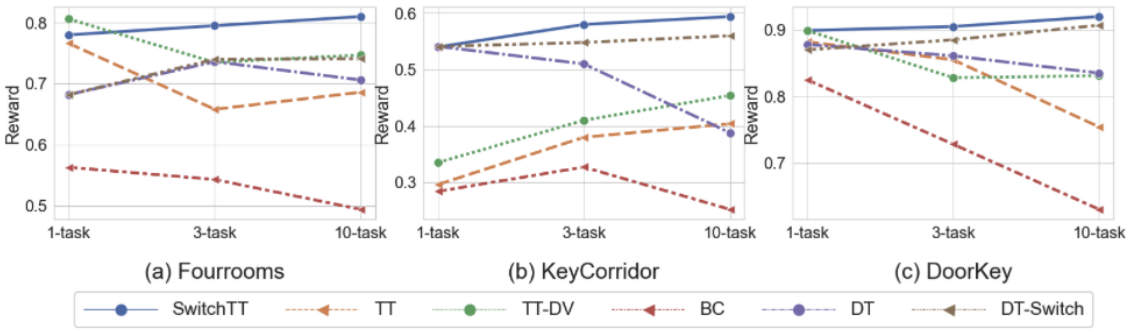


Figure 3.13. The effect of Switch Layer for multi-task learning. We plot the reward mean baseline methods in three multi-task learning settings. n-task learning in the table means trained datasets contain trajectories of n tasks. TT-DV stands for the Trajectory Transformer but is enhanced with a distributional value estimator. BC and DT-switch trains transformer models with switch layers.

We report the detail statics of Figure 3.13 in the appendix. Here, we highlight three key findings from Figure 3.13: (1) SwitchTT outperforms other baseline methods in 10-task learning. For the multi-task setting, SwitchTT can achieve the best results. (2) DT-Switch performs better in 3 multi-task learnings than DT, cooperating with the Switch Layer in the transformer model. DT-Switch improves the multi-task learning performance over DT using the FFN layer in the transformer model. We conclude that such a switch layer mitigates the degrading performance of multi-task learning over single-task learning. (3) TT-DV outperforms TT in 3 multi-task learning settings. The distributional Value estimator improves the performance of the TT. We conclude that our distributional value estimator provides better value estimation for TT and improves the performance of TT. (4) KeyCorridor's performance of method without using switcher layer improves from adding more task, but the other two task performance decreases. This infers that sharing all the same parameter among multi-task learning could improve the some single-task performance but also hurt other single task performance.

3.2.5.3. Computation Cost of Model Learning. Here, we design experiments to compare the computation cost of the Switch Transformer model with the baseline model in learning multi-task

dataset. In particular, we use PPO to collect the multi-task dataset in 10 gym-mini-grid tasks and train six models on the collected dataset. The six models are the Decision Transformer (DT) Model and DT with switch layer (DT-Switch) models with three different size, large, medium and small. Decision Transformer model is a torch-implemented mirror version of the GPT transformer model. DT-Switch is the same model but replaces the FFN layer with the switch layer. The head, layer, embedding size of small, medium, larger are $(4, 4, 64)$,$(8, 4, 64)$ and $(8, 4, 128)$. The expert number in the switch layer is 4. During the training process, we plot the training loss. We plot the performance on three environments, 100 scenarios for each environment.

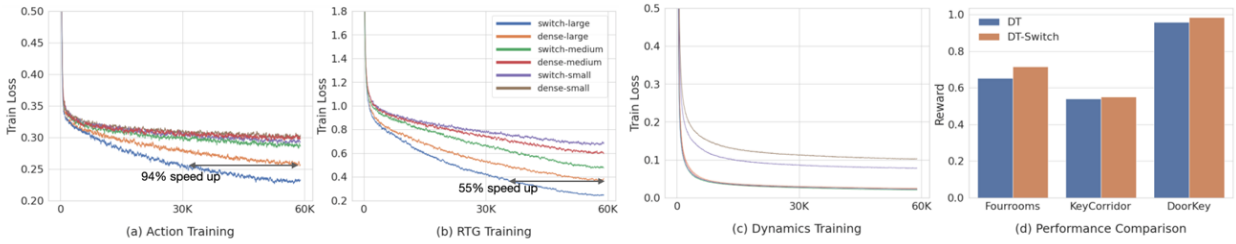


Figure 3.14. Comparison of time cost and performance between dense layer (FFN) and switch layers. Subfigure a, b, c plots the training loss of action model return-to-go model, dynamics model. Subfigure d plots the reward performance of DT and DT-Switch on three tasks.

Figure 3.14.a, 3.14.b, 3.14.c shows that transformer models with switch layer always converge to a lower train loss with faster speed. This means switch layers can reduce the computation cost, including training time and parameter size for multi-task model learning. Also, Figure 3.14 shows that DT-Switch outperforms DT on three tasks. This implies DT-switch mitigates the effect of overfitting the models and also learns a better distribution of trajectories in the multi-task dataset. This result demonstrates the advantage of such a sparsely activated layer in multi-task learning.

Effect of expert number. Table 3.1 compares the performance of different expert numbers in 10-task learning. Here we report the reward mean and variance in 3 difficult tasks, FourRooms, DoorKey and KeyCorridor. The model with four experts in the switch layer performs better than others. Notably, the model with higher expert numbers 16 performs worse than number 8. We conclude that the model with four experts performs best in 10-task learning.

Task	N=1	N=2	N=4	N=8	N=16
FourRooms	0.7056 ± 0.10	0.7056 ± 0.10	0.7413 ± 0.12	$\mathbf{0.8102 \pm 0.11}$	0.6605 ± 0.14
DoorKey	0.8348 ± 0.12	0.8612 ± 0.08	0.8667 ± 0.11	$\mathbf{0.9200 \pm 0.12}$	0.8344 ± 0.12
KeyCorridor	0.3882 ± 0.08	0.4127 ± 0.15	0.4276 ± 0.09	$\mathbf{0.5940 \pm 0.10}$	0.3182 ± 0.11

Table 3.1. Effect of Expert numbers in a 10-task learning setting. We report the reward mean and variance value across 10 different seeds in 10-task learning setting. The N represents the expert number of switch layer in the switch transformer models.

3.2.5.4. Performance of Distributional Value Estimator. Effect of Atom Number Here, we design experiments to compare the performance of different types of return-to-go transformer models. First, we implement four return-to-go transformer models, which discretize the reward into 11 atoms, 31atoms, 51 atoms, 101 atoms. We train SwitchTT in 10-task setting and report reward mean and variance of 3 tasks, FourRooms, DoorKey, KeyCorridor.

Atom	N=11	N=31	N=51	N=101
FourRooms	0.7742 ± 0.07	$\mathbf{0.8224 \pm 0.10}$	0.7704 ± 0.09	0.8102 ± 0.08
DoorKey	0.6863 ± 0.09	0.7457 ± 0.11	0.7583 ± 0.11	$\mathbf{0.9200 \pm 0.12}$
KeyCorridor	0.2295 ± 0.13	0.4122 ± 0.07	0.4568 ± 0.13	$\mathbf{0.5940 \pm 0.09}$

Table 3.2. Effect of Atom Number in the distributional value estimator. We report the reward mean and variance values across 10 different seeds in 10-task learning setting. Atom number stands for the number of discretizing the reward.

In Table 3.2, we study the effect of atom number in our distributional value estimator. We observe that the highest atom number has the highest performance in DoorKey and KeyCorridor tasks, which rely on an accurate value estimator. Specifically, DoorKey and KeyCorridor tasks need to consider move action and open door, pick keys, drop keys action while FourRooms task only requires move action. We conclude that increasing the atom number in the distributional value estimator improves the model performance in multi-task learning.

Improvement over baseline Figure 3.13 shows the performance comparison between TT and TT-DV. We can see that the average performance of TT-DV in different multi-task settings outperforms TT. We conclude that the distributional value estimator improves the trajectory transformer performance by providing a more accurate value estimation of trajectories.

3.2.6. Additional Details

3.2.6.1. Overview of Three models. In the training phase, we train three three models over the dataset, shown in Figure 3.15. We feed the last K timesteps into Decision Transformer, for a total of 3K tokens. For the Dynamics Transformer and Reward Transformer, we feed last k timesteps into them but only 2k tokens, including state and actions. The output of the Dynamics Transformer is the next-timestep state and the output of reward transformer is distribution of reward-to-go value.

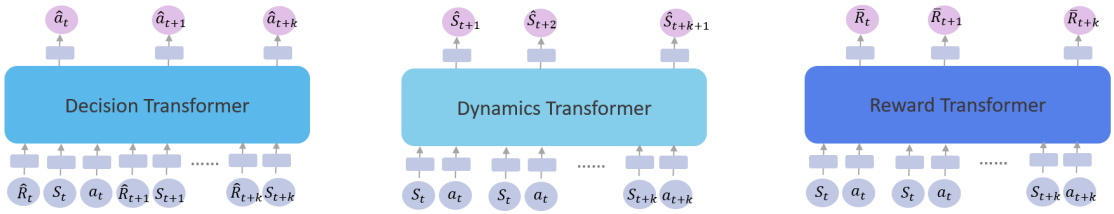


Figure 3.15. Overview of the three models. We train three models in the training phase. The Decision Transformer model is trained to predict action given past trajectory. The Dynamics Transformer model is trained to predict state given past states and past actions. The Reward Transformer model is trained to predict return-to-go of the past states and actions.

3.2.6.2. Experimental Results.

The effect of Switch Layer for multi-task learning.

3.2.6.3. Comparison on RTG models. In this section, we investigate the performance of distributional value-estimator, comparing with the mean value-estimator. We train these distributional RTG models and mean RTG models on the same dataset of minigrid fourrooms tasks. And implement greedy planner with same dynamics transformer model to solve fourroom tasks.

From Figure 3.16, we can see that distributional RTG model can estimate more accurate RTG value than the mean RTG models. This is due to the fact that learning distribution of RTG value provide more accurate information about relationship between trajectory and RTG values.

3.2.6.4. Trajectory Imagination Using Dynamics transformer. In this section, we investigate the performance of the Dynamics Transformer. We firstly train a dynamics model and then plan the optimal action based on imagined trajectories with optimal RTG values. In the planning phase,

	Method	1-task	3-task	10-task
FourRooms	SwitchTT	0.780 ± 0.11	0.795 ± 0.08	0.810 ± 0.11
	TT	0.766 ± 0.09	0.658 ± 0.09	0.686 ± 0.09
	TT-DV	0.806 ± 0.11	0.735 ± 0.09	0.747 ± 0.10
	BC	0.563 ± 0.15	0.543 ± 0.15	0.494 ± 0.12
	DT	0.682 ± 0.15	0.736 ± 0.09	0.706 ± 0.10
	DT-Switch	0.682 ± 0.15	0.740 ± 0.14	0.741 ± 0.12
DoorKey	SwitchTT	0.899 ± 0.04	0.905 ± 0.07	0.920 ± 0.12
	TT	0.882 ± 0.04	0.855 ± 0.07	0.754 ± 0.08
	TT-DV	0.899 ± 0.04	0.828 ± 0.05	0.831 ± 0.12
	BC	0.825 ± 0.08	0.728 ± 0.14	0.630 ± 0.08
	DT	0.878 ± 0.06	0.861 ± 0.07	0.835 ± 0.12
	DT-Switch	0.870 ± 0.06	0.885 ± 0.08	0.907 ± 0.11
KeyCorridor	SwitchTT	0.540 ± 0.08	0.580 ± 0.10	0.594 ± 0.10
	TT	0.296 ± 0.06	0.380 ± 0.08	0.404 ± 0.07
	TT-DV	0.335 ± 0.08	0.410 ± 0.06	0.454 ± 0.06
	BC	0.284 ± 0.15	0.327 ± 0.15	0.252 ± 0.11
	DT	0.541 ± 0.11	0.510 ± 0.08	0.388 ± 0.08
	DT-Switch	0.541 ± 0.11	0.548 ± 0.14	0.560 ± 0.09

Table 3.3. The effect of Switch Layer for multi-task learning. The table reports the detailed statics of Figure, including the reward mean and variance of baseline methods in three multi-task learning settings. n-task learning in the table means trained datasets contain trajectories of n tasks. TT-DV stands for the Trajectory Transformer but is enhanced with a distributional value estimator. BC and DT-switch trains transformer models with switch layers.

we imagine certain-horizon trajectories based on the candidate actions and choose the action with optimal RTG values, which is evaluated on the imagined trajectory.

To demonstrate the performance of Dynamics Transformer, we visualize the imagined trajectory using the tree-based planner with the configuration, where Candidate Number=1 and Plan Horizon=2. As shown in the Figure 3.17, the Dynamics performs very well.

3.2.7. Evaluation on MoEs-based models

This section evaluates whether the MoEs layer can improve the episode reward. Here, we use the trained Decision Transformer models with MoEs layers to interact with the gym-mini-grid FourRooms environments and compared the performance. Also, we evaluate the models on the modified continuous task. In the modified setting the agent gets positive reward when moving

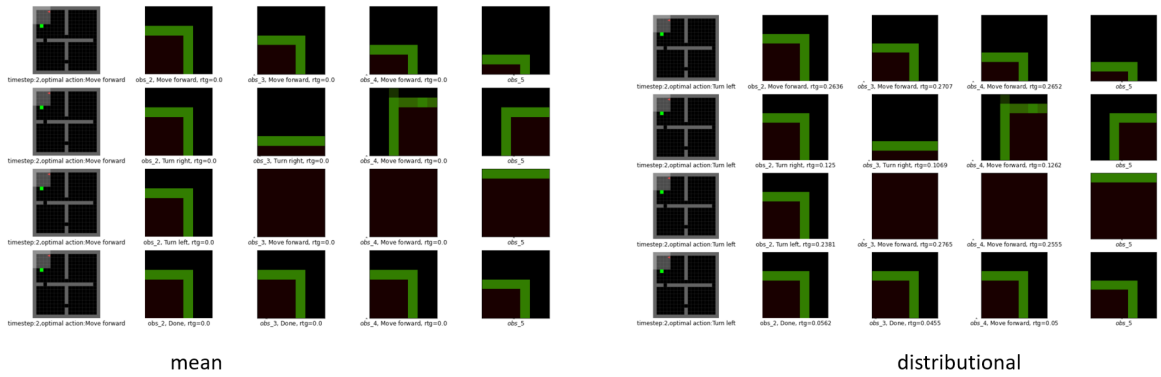


Figure 3.16. Illustration of Imagined Trajectories using Tree-based Planner. This illustrates two scenarios in the planning phase. Both panels show the agent's global view and imagined trajectories based on candidate actions at one time. The text below each global-view picture shows the timestep number and optimal action based on the imagined trajectories. The text below each local-view picture shows the current trajectory's timestep number, next action, and RTG value. The left panel shows that the value estimator, which only predicts the expectation of return-to-go value. The right panel shows the distributional value estimator's evaluation. The distributional estimator performs better than other estimators.

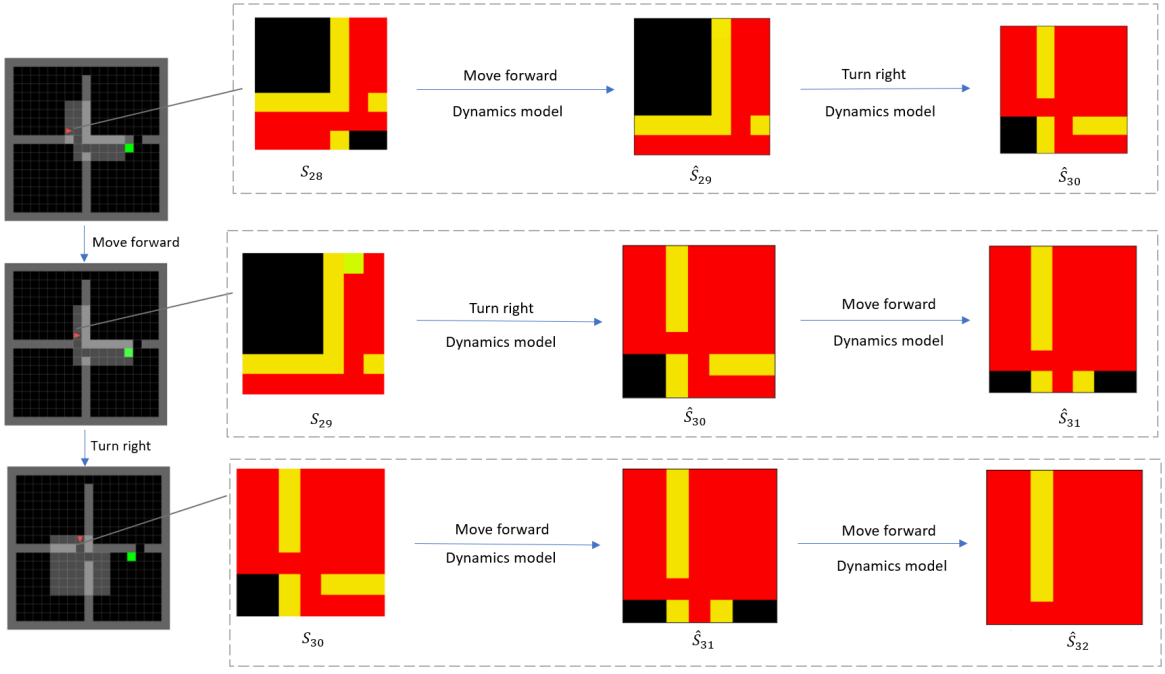
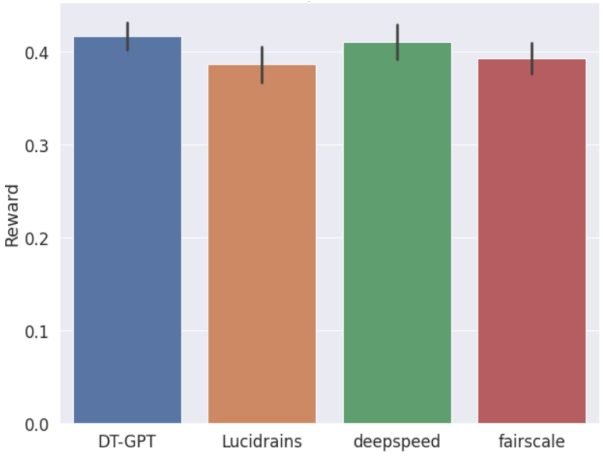
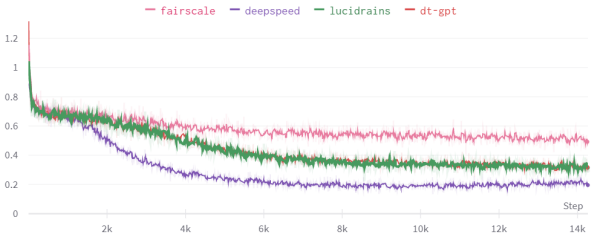


Figure 3.17. Illustration of Imagined Trajectories using Dynamics Transformer. The left grid shows the agent executes action in the Fourrooms environment. The right three panels shows the imagined trajectory, starting from the true observation from the environment and predicting the next observation according to the action.

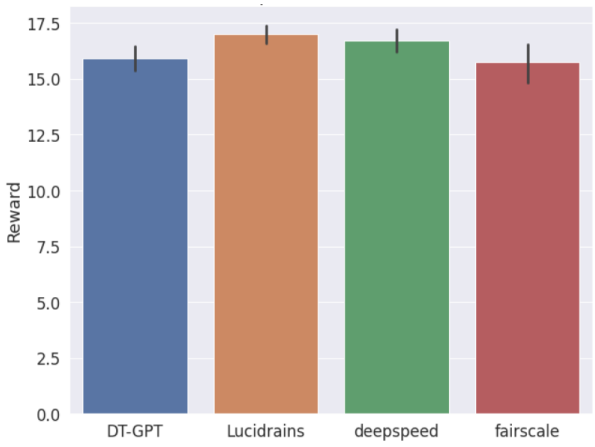
(a) Training loss (sparse-reward)



(b) Evaluation reward (sparse-reward)



(c) Training loss (continuous-reward)



(d) Evaluation reward (continuous-reward)

Figure 3.18. The training loss and evaluation reward in sparse-reward and continuous reward setting. The upper figures show the training loss and evaluation reward in a sparse-reward setting, while the lower figures show the continuous-reward setting. We report the training loss of each iteration in the training phase in (a) and (c). We also calculate the episode reward, averaging 1000 episode rewards over ten different seeds.

closer to the goal instead of only getting reward when reaching the goal. This modification is a strategy of reward shaping, which is an effective technique for incorporating domain knowledge into reinforcement learning and simplifies the problem to facilitate the study of dataset collection. In the evaluation phase, the desired target return is 1.0 in sparse-reward setting, and the desired target return is 25.0 in continuous reward setting.

We plot the evaluation reward in Figure 3.18. The upper figures show that the performance stay almost the same in sparse-reward setting even if the model has different training curves. This

implies that better training loss on the offline dataset doesn't lead to improved evaluation reward in the sparse-reward setting. We assume this result is caused by the long-term and sparse-reward feature in FourRooms task. The lower figures show that the MoEs integrated models have marginally higher evaluation reward than the original GPT model. This implies that the MoEs with lower train loss improve the evaluation reward in the continuous reward setting.

3.2.7.1. Overfitting on Dataset. This section investigates why the Decision Transformer (DT) model with lower train loss can not lead to higher evaluation reward. This problem is posted in the Section 3.2.7. Since DT models the conditional distribution of actions given returns-to-go and states, lower train loss essentially is indicative that this distribution is learned well. For checking generalization during training, we use techniques similar to supervised learning. Specifically, we split the collected dataset into two parts: 80% for the training phase and 20% for the validation phase. Then, we keep a validation dataset of trajectories and check the validation loss is decreasing along with training loss. Meanwhile, we review the evaluation reward during the training phase. Then we plot the training loss, validation loss and evaluation reward in the Figure 3.19

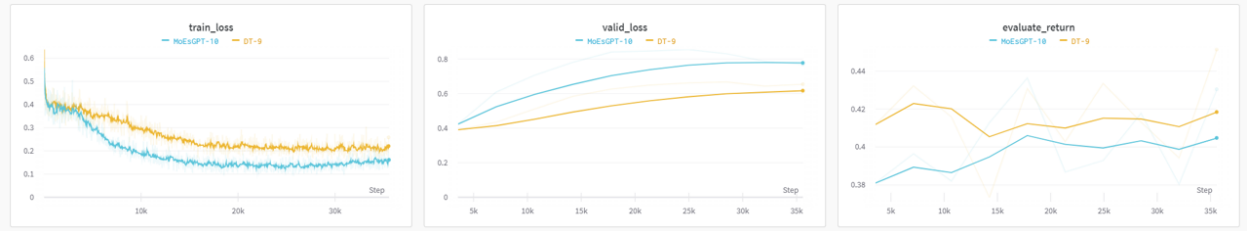


Figure 3.19. Comparison of train loss, validation loss, evaluation reward between different model architectures. The blue line represents the MoEs-based Decision Transformer, and the yellow line represents the transformer-based Decision Transformer. The training loss is collected during each iteration in the training phase of the gym-mini-grid four rooms task. The validation loss and evaluation reward are collected every epoch when training the model. In these figures, the MoEs-based one converges to a lower train loss than the other on the same collected dataset. But MoEs-based model shows a higher validation loss on the same validation dataset. It also has a lower evaluation reward on the four rooms task.

From Figure 3.19, we can see that MoEs-based DT converges to lower training loss compared with transformer-based DT since the model architecture is more complex and model size is more

significant. But the validation loss increases when train loss decreases. Also, MoEs-based DT goes to higher validation loss compared with transformer-based DT. This implies that the model overfits the training phase dataset and can not generalize well in the evaluation phase. The overfitting causes the model with lower train loss to perform worse in the evaluation phase. Also, from the subfigure (b) and subfigure (c) in the Figure 3.19, we can conclude that the model with lower validation can generalize better in the evaluation phase. To get higher validation loss and better performance in learning the task, we decrease the training epochs and reduced the model size in our experiment.

3.2.8. Conclusion

We propose SwitchTT, seeking to solve multi-task reinforcement learning via an advanced transformer model. Promising experimental results show that our method outperforms other offline RL methods. Future work will consider combined advanced tree search algorithms like Monte-Carlo Tree Search to improve the performance.

CHAPTER 4

Scaling laws in Embodied AI

This chapter establishes a predictive science for the development of next-generation embodied agents. As foundation models, particularly Large Language Models (LLMs), become central to high-level reasoning and planning, it is critical to understand how their capabilities scale with resources. We conduct a large-scale empirical study to derive scaling laws for LLM-driven decision-making in embodied tasks. These laws provide a principled framework for forecasting model performance based on factors like model size, data volume, and compute budget. By turning the art of model scaling into a predictable engineering discipline, these findings guide future research and investment, enabling the field to pursue more complex, generalist agents in a resource-efficient manner.

4.1. Observational Scaling Laws in LLM-based Embodied Decision Making

4.1.1. Abstract

We introduce an observational method to derive scaling laws for LLM performance in embodied decision-making tasks, predicting complex skills and quantifying the effects of interventions on LLMs. In contrast to conventional scaling research that trains multiple models from scratch at different scales, our approach bypasses new model training and instead leverages performance data from $\sim$ 140 publicly available LLMs of various scales. The challenge is that these models vary greatly in training efficiency and inherent capabilities – differences that are especially significant for embodied decision-making tasks requiring complex reasoning and structured outputs. To address this, we employ a generalized scaling law that represents model performance as a function of a low-dimensional capability space. This approach accounts for inter-family differences by modeling how efficiently each family's training translates into useful capabilities. In our experiment, we first validate

our scaling law on the Embodied Agent Interface (EAI) benchmark across 125 LLMs, confirming a predictive accuracy that represents at least a 50% improvement over traditional compute scaling laws. We then find that an LLM's decision-making ability is highly predictable—accurately forecasting the performance of larger models using data from those as small as 40B parameters—which allows us to quantify both the performance gap between simulation environments and the impact of structured decoding.

4.1.2. Introduction

Large Language Models (LLMs) power embodied agents that interpret goals, plan actions, and interact with dynamic environments. This progress presents a fundamental question: do classical scaling laws, which link model size to performance on language benchmarks, hold true for embodied tasks? These complex tasks demand structured action sequences and sequential decisions (Brohan et al., 2023; Ahn et al., 2022), a clear departure from simple text generation. Answering this question is critical to predict the capabilities of larger models and to guide the future development of embodied AI.

Prior work establishes systematic benchmarks to evaluate LLMs in embodied settings, measuring capabilities like goal interpretation, subgoal decomposition, and action sequencing (Li et al., 2024). These benchmarks reveal what current models can do, but they do not explain how performance scales. A clear relationship between compute, such as training FLOPs, and downstream embodied success remains unestablished. This disconnect is critical. Embodied tasks introduce challenges like environmental interaction and the sim-to-real gap, which are absent from the pure language domains where scaling laws are well-understood (Kaplan et al., 2020). The field currently lacks predictive models for embodied AI performance as a function of model scale, leaving a fundamental gap between capability evaluation and scaling science.

We introduce an observational scaling approach to predict the performance of LLMs in embodied tasks. Our method extends observational scaling laws (Ruan et al., 2024), which use a

low-dimensional representation of a model's capabilities to forecast success on complex downstream tasks. This technique allows us to build predictive scaling models without the need for costly, full-scale retraining, bridging the gap between embodied AI evaluation and scaling science.

Our work starts with the observation of general scaling laws across LM families that relate downstream performance to training measures. We test if this relationship extends to complex embodied skills. An agent's success in goal interpretation, for example, understanding "bring me the red cup from the kitchen," depends on core skills like natural language understanding and commonsense reasoning. We posit that a model's downstream performance is a function of a low-dimensional space of such capabilities. Model families differ only in the efficiency with which they convert training compute into these capabilities. This relationship implies a log-linear trend from capabilities to downstream performance across all model families, and a log-linear trend from training compute to capabilities within each specific family.

This observational approach provides key advantages. First, it enables the study of scaling behavior without retraining models. Second, it combines models from heterogeneous families with different scaling properties, such as LLaMA (Dubey et al., 2024; Touvron et al., 2023a,b), and StarCoder (Lozhkov et al., 2024b,a). This allows an analysis of different scaling strategies and their impact on downstream performance and algorithmic interventions.

We validate these scaling laws on the Embodied Agent Interface (EAI) benchmark using 125 open LLMs from 28 model families. We demonstrate our method's utility in three settings: predicting emergent capabilities, quantifying simulation gaps, and measuring the effect of structured outputs. First, we predict the performance of models larger than 40B parameters using data from models smaller than 40B. Second, we use the scaling laws to quantify the performance gap between different simulation environments. Third, we quantify the effect of structured outputs and find they degrade the model's decision-making performance.

Our contributions are twofold. First, we introduce an observational scaling framework that unifies scaling laws for embodied tasks. This framework predicts decision-making performance as

a function of model capabilities and scale. Second, using our framework on the EAI benchmark, we quantify the performance degradation from structured outputs and measure the gap between simulation environments.

The paper proceeds as follows. Section 2 reviews related work. Section 3 formulates our problem. Section 4 presents our method and Section 5 details our experiments. We conclude in Section 6.

4.1.3. Related Works

Embodied benchmarks such as VirtualHome, ALFRED, BEHAVIOR-1K, TEACh, and Habitat evaluate whether agents can map goals and observations into machine-executable action–state sequences that achieve task goals, enabling step- and goal-level verification of decision making (Puig et al., 2018; Shridhar et al., 2020; Li et al., 2023; Padmakumar et al., 2022; Savva et al., 2019). The Embodied Agent Interface (EAI) formalizes this setting by standardizing four LLM decision-making modules (goal interpretation, subgoal decomposition, action sequencing, transition modeling), specifying I/O formats, and adding fine-grained error taxonomies that support modular, diagnostic evaluation (Li et al., 2024). Building on these interfaces, researchers either (i) improve LLM performance via prompting and planning—e.g., Chain-of-Thought and ReAct—or affordance-grounded planning for robotics (SayCan), or (ii) extend toward VLA policies that couple vision, language, and action for robot control (RT-2; OpenVLA) (Wei et al., 2022b; Yao et al., 2023; Ichter et al., 2023; Zitkovich et al., 2023; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**). Unlike work that augments algorithms or expands tasks, our focus is to analyze scaling behavior of LLMs within EAI, linking standardized upstream capabilities (reasoning, coding, math) to downstream embodied performance via observational scaling principles (Ruan et al., 2024).

Scaling laws fall into two main categories: compute-based scaling laws and downstream performance scaling laws. Standard scaling laws (Bahri et al., 2021; Henighan et al., 2020; Hernandez et al., 2021; Hestness et al., 2017; Hoffmann et al., 2022; Kaplan et al., 2020; Muennighoff et al.,

2024), which are compute-based scaling laws, are typically expressed as power-law relationships between a model's cross-entropy loss L and compute-scale measures. In this context, "compute scale" refers to training resources such as the number of training FLOPs (C), model parameters (N), and training tokens (D). Compute-based scaling laws characterize pretraining behavior within a single model family, linking upstream performance to controllable quantities like training compute. In contrast, downstream performance scaling laws (Abnar et al., 2021; Hernandez et al., 2021; Tay et al., 2023; Caballero et al., 2022; Ghorbani et al., 2021; Henighan et al., 2020) analyze scaling across model families, connecting benchmark results to compute-related metrics (e.g., model size (Brown et al., 2020a)) or predicting its performance due to appearing rapid "emergence" (Ganguli et al., 2022; Suzgun et al., 2022; Wei et al., 2022a). Specifically, Researchers (Finnveden, 2020; Owen, 2024) have explored both linear and sigmoidal functional forms to extrapolate downstream performance from pretraining loss or compute measures. Chen et al. (2024) introduced a two-stage approach—first predicting pretraining loss from compute, then mapping that loss to downstream performance—even when using models from different families with varying compute-efficiencies. On the theory front, Arora and Goyal (2023) and Ruan et al. (2024) derive theories characterizing how performance on complex skills of LMs can be derived as a composition of base skills. Drawing from these downstream scaling insights including observation scaling laws (Ruan et al., 2024), we aim to identify scaling patterns between embodied decision-making performance and conventional benchmark metrics, emphasizing empirical observation rather than compute-driven modeling.

Corelation between benchmarks have been investigated in numerous works. Specifically, extensive research has explored the relationship between the out-of-distribution performance and in-distribution performance of machine learning models (Miller et al., 2021; Recht et al., 2018, 2019; Taori et al., 2020; Yadav and Bottou, 2019). In NLP and LM evaluations, Qiu et al. (2018) and Torregrossa et al. (2020) found that multiple evaluation metrics for word embeddings are highly correlated, while Liu et al. (2021) observed robust correlations across question-answering benchmarks. Perlitz et al. (2023) and Polo et al. (2024) further noted that performance is strongly

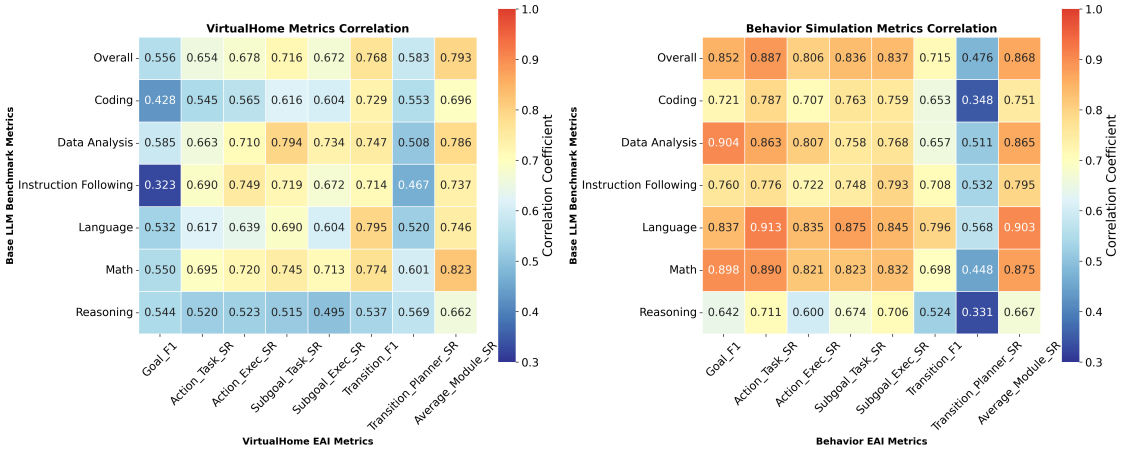


Figure 4.1. Pearson correlation heatmap between LiveBench 08-31-2024 and EAI tasks. Strongest alignments are observed between GSM8K and AS, and HumanEval and TM. Descripency found between two simulations.

correlated across samples of different LM benchmarks, enabling the design of more efficient evaluation suites. Beyond empirical observations, studies have identified compact latent structures driving performance across tasks. Ilić (2023) demonstrated that a single latent factor accounts for 85% of performance variance on the Open LLM Leaderboard (Beeching et al., 2023) and GLUE benchmark (Wang et al., 2018). Burnell et al. (2023) similarly uncovered that three factors explain 82% of variation on the HELM benchmark (Liang et al., 2022). These findings align with cross-task consistency seen elsewhere: for example, MixEval (Ni et al., 2024) combines diverse benchmark queries and achieves a high ranking correlation (Pearson 0.96) with human-composed Chatbot Arena (Chiang et al., 2024), demonstrating coherence between aggregated benchmarks and human judgment. In Figure 4.1, our work observes benchmark correlations (the highest is 91.3%) between LiveBench(White et al., 2025) and EAI, and finds the gap between simulations. This leads to formulating simulation-aware scaling predictions based on benchmark performance.

4.1.4. Problem Formulation

We formulate scaling laws within the Embodied Agent Interface (EAI) benchmark (Li et al., 2024). Our objective is to determine if a smooth scaling relationship exists between general-purpose LLM capabilities and the specialized skills of Goal Interpretation and Action Sequencing.

4.1.4.1. Embodied Agent Interface. EAI is a benchmark for embodied decision-making. It uses Linear Temporal Logic (LTL) as a formal language to represent goals and plans, enabling a precise evaluation of an agent's ability to understand instructions and generate action sequences. For a comprehensive overview of the framework's formalisms, including its state representation and LTL semantics, we refer the reader to the original EAI paper. In this work, we focus specifically on evaluating the Goal Interpretation and Action Sequencing modules.

Goal Interpretation module $\mathcal{G}$ translates a natural language instruction l_g into a formal LTL goal g, given an initial state s_0. The **Input-Output** is $\mathcal{G} : (s_0, l_g) \to g$. Its performance is measured by an F_1 **set-matching score** between the generated goal $\widehat{g}$ and the ground truth.

Action Sequencing module Q generates an action sequence $\overline{a}$ to achieve a given LTL goal g from a state s_0. The **Input-Output** is $Q : (s_0, g) \to \overline{a}$. It is evaluated on two metrics: **Trajectory Feasibility** (whether the sequence $\overline{a}$ is executable in a simulator) and **Goal Satisfaction** (whether the resulting trajectory achieves g).

4.1.4.2. Problem formulation. Let E_m be the normalized performance metric for a given model m on a specific EAI evaluation task. We focus on key indicators of embodied competence, such as the `task_success_rate` or `execution_success_rate` for the Action Sequencing module, and the `all_f1` score for the Goal Interpretation module.

Let C_m be the model's m training compute in FLOPs, N_m be parameter size, and D_m be the pretraining token size. Following Kaplan et al. (2020), we estimate C_m using the approximation $C_m \approx 6N_mD_m$. This allows us to connect the concrete properties of a model to its performance.

Recent studies (Finnveden, 2020; Owen, 2024) have found that a predictable scaling relationship holds for models within a single architectural family (e.g., Llama, Gemma, or Qwen). They observe

a sigmoidal relationship between training compute and task error, formally expressed using a generalized linear model with a logistic link function (σ^{-1}):

$$\sigma^{-1}(E_m) \approx \lambda_f \log(C_m) + \mu_f. \tag{4.1.1}$$

Here, λ_f and μ_f are constants that are determined empirically.

Our primary goal is to generalize this relationship to better quantify the scaling laws of the EAI benchmark, potentially finding a more universal framework that holds across different model families. A successful generalization would allow for more robust performance forecasting.

4.1.5. Method: Observational Scaling Laws

Our work builds on the Obscaling (Ruan et al., 2024), a framework for creating a universal scaling model for diverse language models, including those with unknown training compute. This approach allows us to move beyond the family-specific limitations discussed previously and pursue a more generalizable law.

Hypothesis 1 (Universal Performance Model) The core hypothesis is that we can predict a model's (m) performance on a complex task (measured by error $E_m \in \mathbb{R}$) using a universal linear model based on a latent low-dimensional capability vector $S_m \in \mathbb{R}^K$:

$$\sigma^{-1}(E_m) \approx \beta^\top S_m + \alpha. \tag{4.1.2}$$

Here, S_m is the capability vector for model m in a K-dimensional space, σ is the logistic function, $\beta \in \mathbb{R}^K$ is a universal weight vector that maps capabilities to performance, and $\alpha \in \mathbb{R}$ is a scalar bias.

Hypothesis 2 (Latent Capability Projection) Then, we hypothesize that a model's latent capability, S_m, is a linear projection of its benchmark performance vector, $B_m \in \mathbb{R}^T$. To compute

this capability vector, we apply a projection matrix $\gamma \in \mathbb{R}^{K \times T}$ such that

$$S_m := \gamma B_m. \tag{4.1.3}$$

We derive this matrix by applying Principal Component Analysis (PCA) to the performance vectors of all models. The rows of γ consist of the top K principal components, as exemplified in Figure 4.2b.

Hypothesis 3 (Log-linear Capability Scaling) Next, if we assume that within a specific model family f, capability grows log-linearly with compute (Equation 4.1.4), then substituting this into Equation 4.1.2 recovers the familiar family-specific scaling law (Equation 4.1.1):

$$S_m \approx \theta_f \log(C_m) + \nu_f \tag{4.1.4}$$

$$\sigma^{-1}(E_m) \approx w_f \log(C_m) + b_f. \tag{4.1.5}$$

Here, $\theta_f \in \mathbb{R}^K$ is a family-specific vector, $\nu_f \in \mathbb{R}^K$ is a bias vector, $w_f = \beta^\top \theta_f$ and $b_f = \beta^\top \nu_f + \alpha$. Thus, Equation 4.1.5 is consistent with Equation 4.1.1

Fitting Observational Scaling Laws We begin with a set of LMs $\mathcal{M}$, and four quantities for each model $m \in \mathcal{M}$: its compute measure FLOPs C_m, its vector of benchmark scores B_m, and its performance on a complex task E_m. From this data, we estimate the scaling relationship through a multi-stage procedure.

Firstly, we estimate the capability vectors S_m via fitting PCA on B_m, and then find the universal parameters β^* and α^* by minimizing the squared error for the relationship defined in Equation 4.1.2:

$$(h^*, \beta^*, \alpha^*) = \underset{h,\beta,\alpha}{\operatorname{argmin}} \sum_{m \in \mathcal{M}} \|(E_m) - h\sigma^(\beta^\top S_m + \alpha)\|^2. \tag{4.1.6}$$

where $\beta \in \mathbb{R}^K$, $\alpha \in \mathbb{R}$ are regression weights and bias. $h \in [0, 1]$ is the sigmoid scale and it results in $h^* = 1$ in most experiments. This defines a scalar capability score $P_m := (\beta^*)^\top \widehat{S}_m + \alpha^*$ for any model.

Secondly, we determine the coefficients w_f^* and b_f^* for the scaling law described in Equation 4.1.5. Specifically, we select a reference family (e.g. llama-2) f, and then fit another linear regression using only the models from the reference family f:

$$(w_f^*, b_f^*) = \underset{w_f, b_f}{\operatorname{argmin}} \sum_{m \in f} \|P_m - (w_f \log(C_m) + b_f)\|^2, \tag{4.1.7}$$

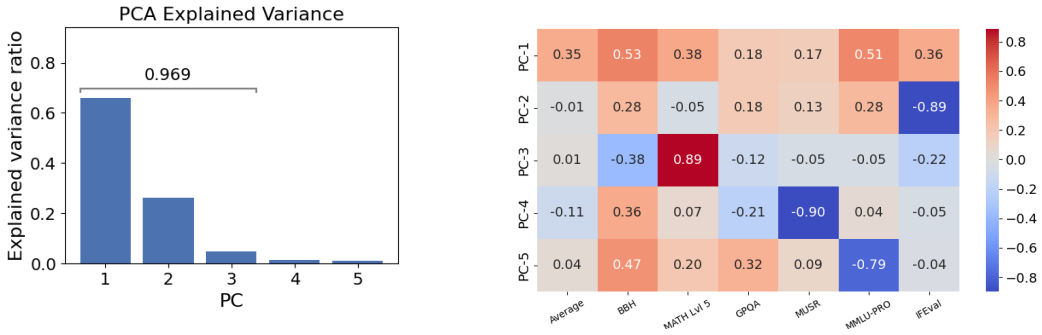
This mapping allows us to convert any model's capability score P_m into an intuitive metric—the f-equivalent FLOPs, $\widetilde{C}_{m,f}$—by inverting the relation: $\log\left(\widetilde{C}_{m,f}\right) := (P_m - b_f^*)/w_f^*$. This provides a single, compute-anchored axis for comparing all models.

4.1.6. Experiments

Our experimental evaluation proceeds in four stages. First, we verify the core assumptions underlying our proposed observational scaling laws. Second, we validate the laws by fitting them to LLM performance on the EAI benchmark. Third, we apply the validated laws to identify a "simulation gap" between distinct EAI environments. Finally, we demonstrate the practical utility of our findings for model intervention by quantifying the performance impact of structured decoding.

Experimental setup We evaluate 124 open-source LLMs (listed in Tables 4.1 and 4.2) using the llama-factory (Zheng et al., 2024) with vLLM backend (Kwon et al., 2023) for efficient inference. We measure the target performance metric (E_m) on two tasks from the EAI benchmark (Li et al., 2024): Action Sequencing (task and execution success rates) and Goal Interpretation (F1 score). To establish a general capability measure (S), we gather scores from the OpenLLM Leaderboard (Beeching et al., 2023) on benchmarks testing reasoning (e.g., BBH, MATH), instruction following (IFEval), and expert knowledge (e.g., MMLU-PRO). We then apply Principal Component Analysis (PCA) with $K = 3$ to these general scores to derive our final capability measure S.

4.1.6.1. Valiation of Assumption on Observation scaling laws. We validate two assumptions including Hypothesis 0 and Hypothesis 1, since we use different metrics than the original paper (Ruan et al., 2024).



(a) PCA explained variance

(b) Principle component weights

Figure 4.2. Just a few capability dimensions explain most variability on a diverse range of standard LM benchmarks. We find that (a) the benchmark-model matrix is low-dimensional with the top 3 PCs explaining $\sim 97\%$ of the variance and (b) the PCs are interpretable: PC-1, PC-2, and PC-3 emphasize LMs' general, instruction-following, mathematical reasoning capabilities, respectively.

Hypothesis 0 and 1 posit that a low-dimensional latent variable can effectively represent model performance. To extract this variable, we apply Principal Component Analysis (PCA with $K = 5$) to the full suite of benchmark metrics (B). We define the resulting components as the "principal capability" (PC) measures, S (see (Ruan et al., 2024) for additional details).

Our analysis validates this low-rank assumption. As shown in Figure 4.2a, the top three PCs capture approximately 97% of the total variance, with the first PC alone accounting for nearly 70%. Furthermore, these PCs are highly interpretable (Figure 4.2b). **PC-1** represents a **"general capability"**, **PC-2** corresponds to **"instruction following"**, and **PC-3** reflects **"mathematical reasoning"**. This evidence indicates that the complex LM capabilities covered by our benchmarks can be expressed as a linear combination of a few fundamental principal capabilities S.

Hypothesis 3 proposes a linear relationship between the principal capability measures (S) and log-scale training compute (C). We use the first principal component (**PC-1**) to represent a model's capability S. We estimate the training compute C by collecting the model parameter count (N) and pretraining token size (D) from technical reports and project pages, then approximating the training FLOPs as $C \approx 6ND$.

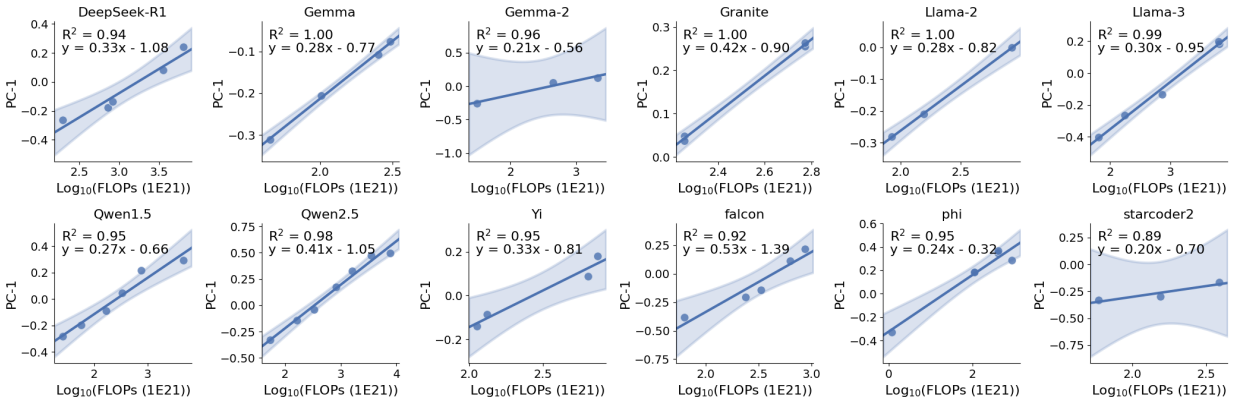
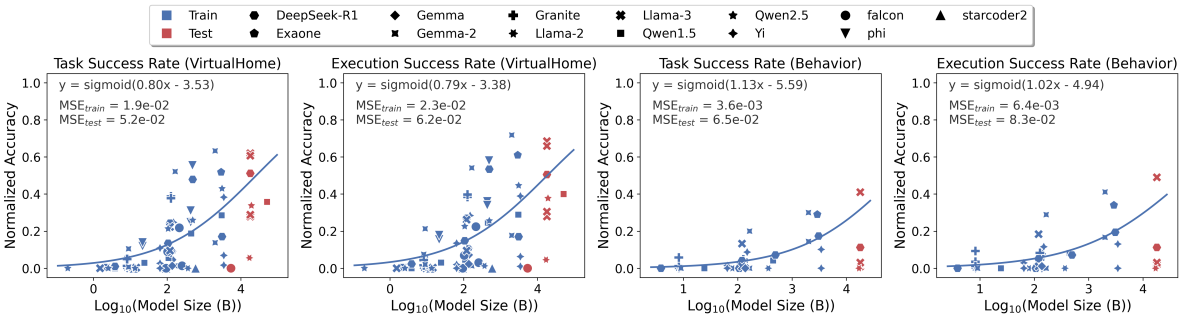


Figure 4.3. The extracted PC measures linearly correlate with log-compute within each model family. The linearity generally holds for various model families.

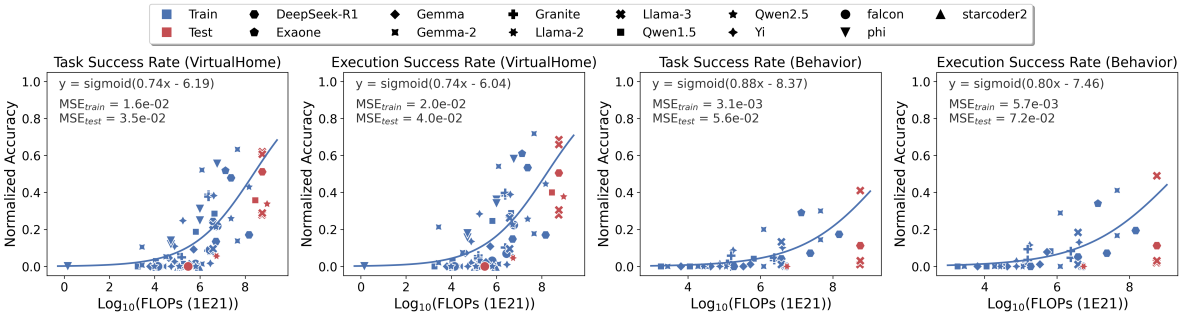
Figure 4.3 validates this log-linear relationship within specific model families. We find a strong correlation where the PC-1 measure scales linearly with log-training FLOPs, achieving an $R^2 > 0.89$. This trend holds across diverse model architectures, including distilled models like DeepSeek-R1 (AI, 2025) and code-focused models like StarCoder2 (Lozhkov et al., 2024a). The relationship also extends to lower-ranked components such as PC-2 and PC-3 (Figures 4.8 and 4.9). This empirical evidence supports our hypothesis in Equations 4.1.3 and 4.1.4, which state that different model families convert compute into capabilities at varying efficiencies within a shared capability space.

4.1.6.2. Validating observational scaling laws. Our objective is to validate that observational scaling laws can predict the performance of large language models on unseen, higher-capability models.

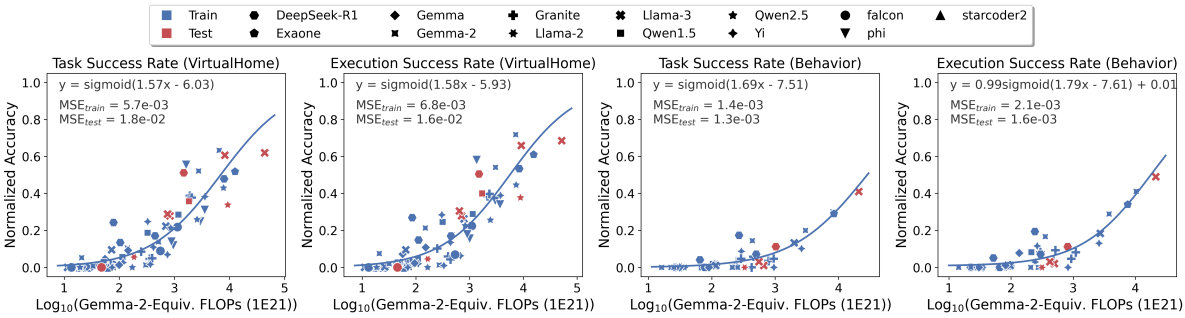
Experiment setup As a preprocessing step, we filter the dataset for data quality. We exclude any model entry where: (1) the reported task performance is zero, as this often indicates a failure (e.g. maximum token length ¡ input token length) in the evaluation setup rather than a measure of capability, or (2) a benchmark score is not available. To test the extrapolative capabilities of the proposed scaling laws, we use a held-out test set. We partition the available models into a training set and a test set based on parameter count. Train set contains models with fewer than 40 billion parameters, and test set contains the other. Additionally, preprocessing transformations and scaling



(a) Model size based scaling law



(b) Training FLOPS based scaling law



(c) Observational scaling law

Figure 4.4. Scaling curves of action sequencing on Virtualhome and Behavior. We compare three scaling laws: (a) Model Size, (b) Training FLOPs, and (c) our proposed Observational scaling law. Each plot shows the training data ($< 40B$ parameters, blue circles), the held-out test data ($\geqslant 40B$ parameters, red crosses), and the fitted sigmoid curve. The reported Mean Squared Error (MSE) on the train and test sets shows that the observational scaling law consistently achieves the lowest test MSE, indicating its superior ability to extrapolate performance to larger, more capable models.

law parameters (e.g., Equations 4.1.7 4.1.6) are fitted on the training data only. These transformations are then applied to the test set to prevent information leakage and evaluate generalization.

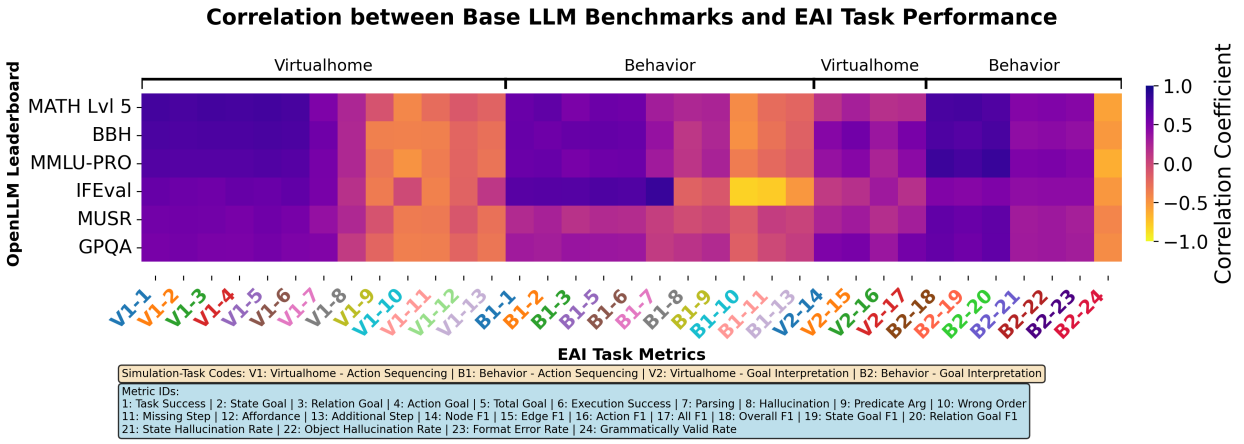


Figure 4.5. Correlation heatmap between EAI task metrics (x-axis) and base LLM benchmarks from the OpenLLM Leaderboard (y-axis). Dark purple cells indicate positive correlation, and light yellow cells indicate negative correlation. The plot reveals that different EAI tasks and environments draw on different foundational capabilities, such as mathematical reasoning (`MATH Lvl 5`) for `Virtualhome` and instruction following (`IFEval`) for `Behavior`.

Baselines: We compare the observational scaling law against two baselines, fitted by Eqation 4.1.1: (i)Model Size Scaling: A power-law fit based on the number of model parameters. (ii) Training FLOPs Scaling: A power-law fit based on the estimated floating-point operations used for training.

We present the results in Figure 4.4. Our method achieves the lowest Mean Squared Error (MSE) on the held-out test set of models with $\geqslant 40B$ parameters. For instance, on the Task Success Rate (Behavior) metric, the observational law yields a test MSE of $1.3 \times 10^{-}3$. This is more than an order of magnitude better than both the Model Size baseline ($6.5 \times 10^{-}2$) and the Training FLOPs baseline ($5.6 \times 10^{-}2$). A similar trend holds for the Execution Success Rate (Behavior) task.

4.1.6.3. Quantifying scaling gap between simulations. Interpreting Scaling Law Coefficients

Following the validation in Section 4.1.6.2, we analyze the fitted regression coefficients from our observational scaling law to understand the relative difficulty of the tasks.

We observe two key trends: (i) For both the `Virtualhome` and `Behavior` environments, the regression weight (w_f^*) for `execution success rate` is consistently larger than the weight for `task success rate`. This is consistent with the logical constraint that a successful task execution

is a stricter, and therefore more difficult, condition to satisfy than a plan that is merely executable. (ii) When comparing the two environments for the same task, `Behavior` exhibits a higher regression weight (w_f^*) but a lower bias (b_f^*) than `Virtualhome`. This suggests that the `Behavior` simulation presents a higher initial barrier to effective performance (lower bias), but that performance scales more steeply with increasing model capability (higher weight) once a baseline of competence is achieved.

Correlation with Foundational LLM Capabilities To contextualize the skills required by our EAI benchmarks, we compute the correlation between EAI task performance and scores from the OpenLLM Leaderboard. We present the results in the heatmap in Figure 4.5. The analysis reveals that task performance in different simulation environments is associated with distinct underlying LLM capabilities. Specifically, Action Sequencing performance in `Virtualhome` shows a strong positive correlation with mathematical reasoning benchmarks (`MATH Lvl 5`). In contrast, the same task in the `Behavior` environment correlates most strongly with instruction following capabilities (`IFEval`). This suggests that `Virtualhome` may test a model's logical planning and reasoning abilities more heavily, while `Behavior` emphasizes the precise interpretation and execution of commands. Furthermore, we note that specific error metrics within our benchmark, such as `Missing Step` (V1-11) and `Affordance` (V1-12), show weak to no correlation with any of the general OpenLLM benchmarks. This indicates that standard LLM evaluations do not adequately measure a model's proficiency in these crucial aspects of embodied planning, highlighting a potential gap in existing evaluation practices.

4.1.6.4. The Impact of Structured Decoding. In this section, we investigate the impact of enforcing structured outputs on the performance of LLMs in EAI tasks. While compelling models to generate plans in a specific format like JSON guarantees syntactic correctness and eliminates parsing failures, it is unclear whether this constraint helps or hinders the model's underlying reasoning and planning capabilities.

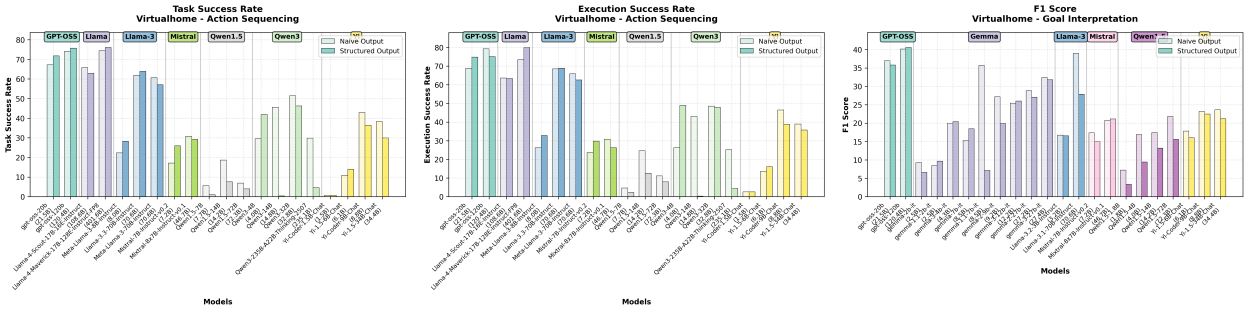
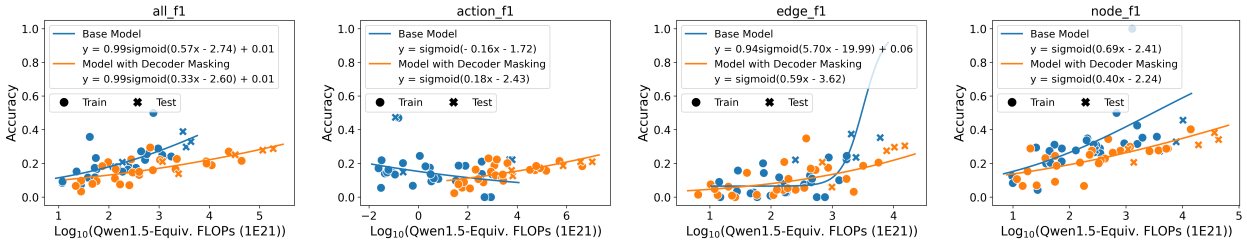


Figure 4.6. Comparison of observational scaling laws for standard generation (Base Model) versus structured decoding (Model with Decoder Masking) on the Virtualhome goal interpretation task. The plots show performance on action sequencing and goal interpretation tasks on Virtualhome.



(a) Overall F1 score (b) Action F1 (c) Edge F1 (d) Node F1

Figure 4.7. Comparison of observational scaling laws for standard generation (Base Model) versus structured decoding (Model with Decoder Masking) on the Virtualhome goal interpretation task. The plots show performance on four different F1 metrics as a function of model scale. While overall performance is comparable, structured decoding significantly degrades scaling performance on granular sub-tasks, particularly for Edge F1, suggesting that output constraints can hinder the learning of complex relational structures.

To quantify this trade-off, we designed a controlled experiment to measure performance differences between standard and constrained decoding. Using vLLM (Kwon et al., 2023) as our inference backend, we evaluated a suite of models on our EAI benchmarks under two conditions: (1) with standard, unconstrained text generation, and (2) with structured decoding enabled via Xgrammar(Dong et al., 2024) to enforce a strict JSON output schema. The performance in both conditions was measured using the primary success metrics from our benchmark to isolate the effect of the decoding constraint. We report results in Figures 4.7 4.10

A direct model-by-model comparison in Figure 4.6 reveals that the impact of structured decoding is not uniform and can be difficult to predict. For instance, on the `Action Sequencing` task,

the constraint improves the `Task Success Rate` for capable models like `Llama-3-70B`, but it harms the performance of others like `Yi-1.5-6B`. Similarly, for `Goal Interpretation`, structured decoding hurts the performance of `GPT-4` and `Mixtral-8x7B`, yet provides a notable benefit to models such as `Yi-1.5-34B` and `Phi-3-mini-128k`.

As shown in Figure 4.7, our results for the Virtualhome goal interpretation task reveal that forcing a structured output consistently hurts model performance. For the main `Overall F1` score, models with the output constraint always performed slightly worse than the regular models, even though both improved at a similar rate as they scaled up. This performance gap was much larger on more detailed sub-tasks. For example, on `Edge classification F1`, the regular model's performance improved more than four times faster than the constrained model's (a scaling slope of w=0.79 vs. w=0.19). This suggests that while structured decoding guarantees a clean output format, these strict rules prevent the model from fully learning the complex relationships needed for the planning task.

4.1.7. Additional Experimental Details

4.1.7.1. Experiment Model Information.

4.1.8. PC measures linearly correlated wit log-compute measures

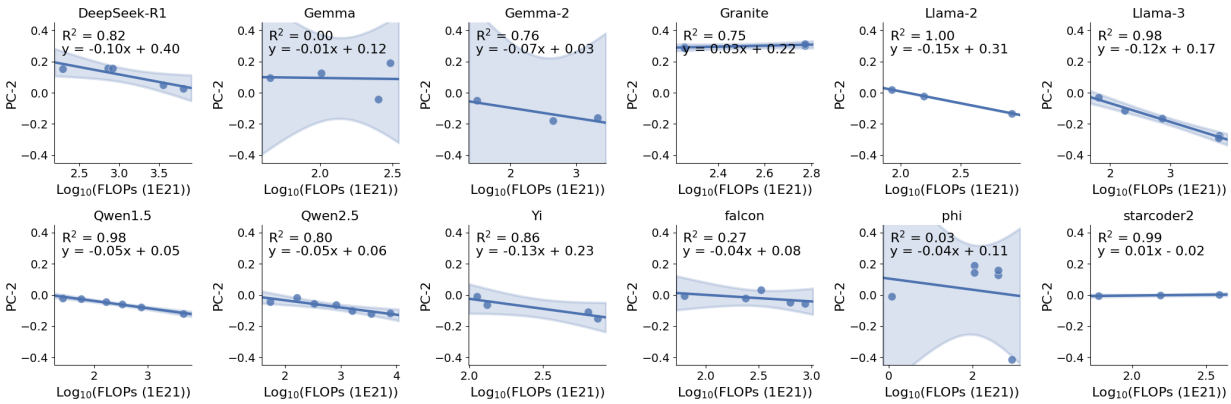


Figure 4.8. The extracted PC measures linearly correlate with log-compute within each model family. The linearity generally holds for various model families.

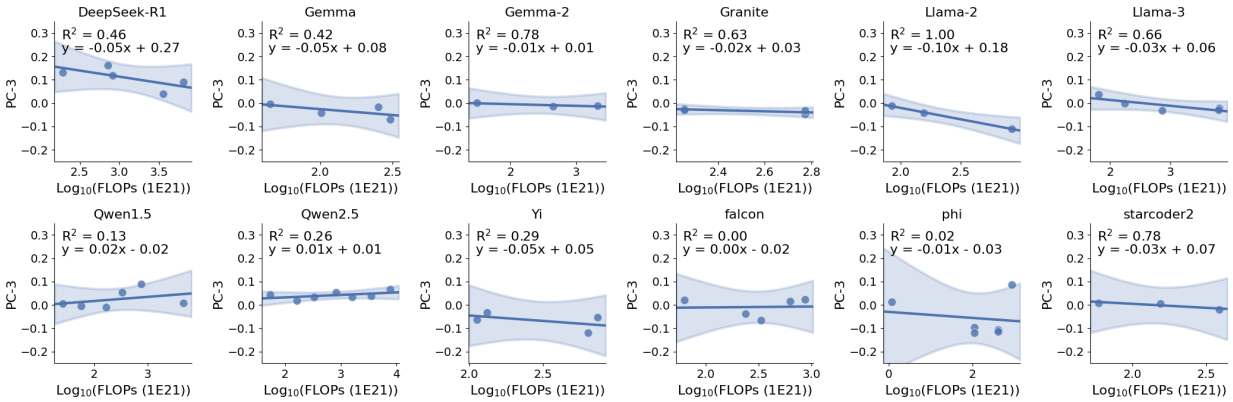


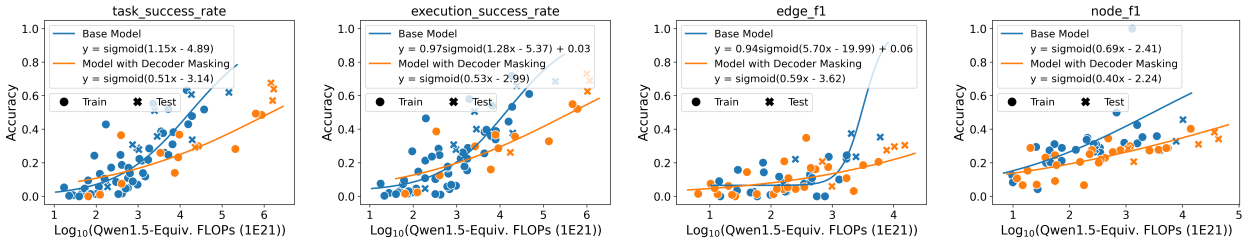
Figure 4.9. The extracted PC measures linearly correlate with log-compute within each model family. The linearity generally holds for various model families.

4.1.9. Detail correlation value between OpenLLM leaderboard and EAI skills

4.1.9.1. Impact of Structuring output.

4.1.10. Conclusion

This paper presents an observational method for deriving scaling laws in embodied decision-making, leveraging a large set of public LLMs to avoid costly training. Our generalized scaling law maps performance to a low-dimensional capability space, effectively modeling diverse model families. Validated on the EAI benchmark, our method shows high predictive accuracy, significantly improving on traditional compute-based laws. This framework provides a cost-effective way to forecast model performance, quantify the effects of interventions like structured decoding, and measure simulation gaps. Future work can extend this approach to build a unified model of the simulation gap and to quantify the effects of more complex LM interventions.



(a) Overall F1 score (b) Action F1 (c) Edge F1 (d) Node F1

Figure 4.10. Comparison of observational scaling laws for standard generation (Base Model) versus structured decoding (Model with Decoder Masking) on the Virtualhome Action Sequencing task.

Table 4.1. Model summary (part 1 of 3). Models sorted by family then name; OpenLLM metric = non-NA ‘Average‘.

Model	Family	Size (B)	Tokens (T)	FLOPs (1E21)	OpenLLM metric
Baichuan-7B	Baichuan	—	1.20	—	No
Baichuan2-7B-Base	Baichuan	7	2.60	109.20	No
Baichuan2-7B-Chat	Baichuan	7	2.60	109.20	No
DeepSeek-V3	DeepSeek	684.5	14.80	60783.60	No
deepseek-coder-1.3b-base	DeepSeek-Coder	1.3	2.00	15.60	No
deepseek-coder-1.3b-instruct	DeepSeek-Coder	1.3	2.00	15.60	No
deepseek-coder-33b-base	DeepSeek-Coder	33.3	2.00	396.00	No
deepseek-coder-33b-instruct	DeepSeek-Coder	33.3	2.00	399.60	No
deepseek-coder-6.7b-base	DeepSeek-Coder	6.7	2.00	80.40	No
deepseek-coder-6.7b-instruct	DeepSeek-Coder	6.7	2.00	80.40	No
deepseek-coder-7b-base-v1.5	DeepSeek-Coder	6.9	2.00	82.80	No
deepseek-coder-7b-instruct-v1.5	DeepSeek-Coder	6.9	2.00	82.80	No
DeepSeek-R1	DeepSeek-R1	684.5	14.80	60783.60	No
DeepSeek-R1-Distill-Llama-70B	DeepSeek-R1	70.6	15.00	6354.00	Yes
DeepSeek-R1-Distill-Llama-8B	DeepSeek-R1	8	15.00	720.00	Yes
DeepSeek-R1-Distill-Qwen-1.5B	DeepSeek-R1	1.8	18.00	194.40	Yes
DeepSeek-R1-Distill-Qwen-14B	DeepSeek-R1	14.8	18.00	1598.40	Yes
DeepSeek-R1-Distill-Qwen-32B	DeepSeek-R1	32.8	18.00	3542.40	Yes
DeepSeek-R1-Distill-Qwen-7B	DeepSeek-R1	7.6	18.00	820.80	Yes
EXAONE-3.5-32B-Instruct	Exaone	32	6.50	1248.00	Yes
EXAONE-Deep-32B	Exaone	32	6.50	1248.00	No
gpt-oss-120b	GPT-OSS	120.4	—	—	No
gpt-oss-20b	GPT-OSS	21.5	—	—	No
gemma-1.1-2b-it	Gemma	2.5	3.00	45.00	Yes
gemma-1.1-7b-it	Gemma	8.5	6.00	306.00	Yes
gemma-7b	Gemma	8.5	6.00	252.00	Yes
gemma-7b-it	Gemma	8.5	2.00	102.00	Yes
gemma-2-27b	Gemma-2	27.2	13.00	2121.60	Yes
gemma-2-27b-it	Gemma-2	27.2	13.00	2121.60	Yes
gemma-2-2b	Gemma-2	2.6	2.00	31.20	Yes
gemma-2-2b-it	Gemma-2	2.6	2.00	31.20	Yes
gemma-2-9b	Gemma-2	9.2	8.00	441.60	Yes
gemma-2-9b-it	Gemma-2	9.2	8.00	441.60	Yes
gemma-2b	Gemma-2	2.5	6.00	72.00	Yes
gemma-2b-it	Gemma-2	2.5	6.00	90.00	Yes
gemma-3-12b-it	Gemma-3	12.2	12.00	878.40	No
gemma-3-12b-pt	Gemma-3	12.2	12.00	878.40	No
gemma-3-27b-it	Gemma-3	27.4	14.00	2301.60	No
gemma-3-4b-it	Gemma-3	4.3	4.00	103.20	No
gemma-3-4b-pt	Gemma-3	4.3	4.00	103.20	No
granite-3.1-2b-base	Granite	2.5	12.00	180.00	Yes
granite-3.1-2b-instruct	Granite	2.5	12.00	180.00	Yes
granite-3.1-8b-base	Granite	8.2	12.00	590.40	Yes
granite-3.1-8b-instruct	Granite	8.2	12.00	590.40	Yes
granite-3.2-2b-instruct	Granite	2.5	12.00	180.00	Yes
granite-3.2-8b-instruct	Granite	8.2	12.00	590.40	Yes
granite-3.3-2b-base	Granite	2.5	12.00	180.00	No
granite-3.3-2b-instruct	Granite	2.5	12.00	180.00	No
granite-3.3-8b-base	Granite	8.2	12.00	590.40	No
granite-3.3-8b-instruct	Granite	8.2	12.00	590.40	No

Table 4.2. Model summary (part 2 of 3). Models sorted by family then name; OpenLLM metric = non-NA ‘Average‘.

Model	Family	Size (B)	Tokens (T)	FLOPs (1E21)	OpenLLM metric
Kimi-K2-Instruct	Kimi	1000	15.50	93000.00	No
Llama-4-Maverick-17B-128E-Instruct-FP8	Llama	401.6	22.00	53011.20	No
Llama-4-Scout-17B-16E-Instruct	Llama	108.6	40.00	26064.00	No
llama3_8B_o4-mini-2025-04-16	Llama	—	—	—	No
Llama-2-13b-hf	Llama-2	13	2.00	156.00	Yes
Llama-2-70b-hf	Llama-2	69	2.00	840.00	Yes
Llama-2-7b-hf	Llama-2	6.7	2.00	84.00	Yes
Llama-3.1-70B	Llama-3	70.6	15.00	6354.00	Yes
Llama-3.2-1B	Llama-3	1.2	9.00	64.80	Yes
Llama-3.2-3B	Llama-3	3.2	9.00	172.80	Yes
Llama-3.3-70B-Instruct	Llama-3	70.6	15.00	6354.00	Yes
Meta-Llama-3-70B	Llama-3	70.6	15.00	6300.00	Yes
Meta-Llama-3-70B-Instruct	Llama-3	70.6	15.00	6354.00	Yes
Meta-Llama-3-8B	Llama-3	8	15.00	720.00	Yes
Meta-Llama-3-8B-Instruct	Llama-3	8	15.00	720.00	Yes
Mistral-7B-Instruct-v0.2	Mistral	7.2	—	—	Yes
Mixtral-8x7B-Instruct-v0.1	Mistral	46.7	—	—	Yes
Qwen-14B	Qwen	14.2	3.00	252.00	No
Qwen-72B	Qwen	72.3	3.00	1296.00	No
Qwen-7B	Qwen	7.7	2.40	100.80	No
Qwen1.5-1.8B	Qwen1.5	1.8	2.40	25.92	Yes
Qwen1.5-110B	Qwen1.5	111.2	7.00	4670.40	Yes
Qwen1.5-14B	Qwen1.5	14.2	4.00	336.00	Yes
Qwen1.5-32B	Qwen1.5	32.5	4.00	768.00	Yes
Qwen1.5-4B	Qwen1.5	4	2.40	57.60	Yes
Qwen1.5-72B	Qwen1.5	72.3	3.00	1296.00	No
Qwen1.5-7B	Qwen1.5	7.7	4.00	168.00	Yes
Qwen2.5-0.5B	Qwen2.5	0.5	18.00	54.00	Yes
Qwen2.5-1.5B	Qwen2.5	1.5	18.00	162.00	Yes
Qwen2.5-14B	Qwen2.5	14.8	18.00	1598.40	Yes
Qwen2.5-32B	Qwen2.5	32.8	18.00	3542.40	Yes
Qwen2.5-3B	Qwen2.5	3.1	18.00	334.80	Yes
Qwen2.5-72B	Qwen2.5	72.7	18.00	7851.60	Yes
Qwen2.5-7B	Qwen2.5	7.6	18.00	820.80	Yes
Qwen3-0.6B	Qwen3	0.8	36.00	172.80	No
Qwen3-1.7B	Qwen3	2	36.00	432.00	No
Qwen3-14B	Qwen3	14.8	36.00	3196.80	No
Qwen3-235B-A22B-Thinking-2507	Qwen3	235.1	36.00	50781.60	No
Qwen3-32B	Qwen3	32.8	36.00	7084.80	No
Qwen3-4B	Qwen3	4	36.00	864.00	No
Qwen3-8B	Qwen3	8.2	36.00	1771.20	No
Yi-1.5-34B	Yi	34.4	3.60	743.04	Yes
Yi-1.5-34B-Chat	Yi	34.4	3.60	743.04	Yes
Yi-1.5-6B	Yi	6.1	3.60	131.76	Yes
Yi-1.5-6B-Chat	Yi	6.1	3.60	131.76	Yes
Yi-1.5-9B	Yi	8.8	3.60	190.08	Yes
Yi-34B	Yi	34.4	3.10	639.84	Yes
Yi-6B	Yi	6.1	3.10	113.46	Yes
Yi-Coder-1.5B	Yi	1.5	2.40	21.60	No
Yi-Coder-1.5B-Chat	Yi	1.5	2.40	21.60	No

Table 4.3. Model summary (part 3 of 3). Models sorted by family then name; OpenLLM metric = non-NA 'Average'.

Model	Family	Size (B)	Tokens (T)	FLOPs (1E21)	OpenLLM metric
Yi-Coder-9B	Yi	8.8	2.40	126.72	No
Yi-Coder-9B-Chat	Yi	8.8	2.40	126.72	Yes
Falcon3-10B-Base	falcon	10.3	14.00	865.20	Yes
Falcon3-7B-Base	falcon	7.5	14.00	630.00	Yes
falcon-11B	falcon	11.1	5.00	333.00	Yes
falcon-40b	falcon	41.8	1.00	240.00	Yes
falcon-7b	falcon	7.2	1.50	63.00	Yes
gpt-4.1-2025-04-14	gpt-4.1-2025-04-14	—	—	—	No
gpt-4.1-mini-2025-04-14	gpt-4.1-mini-2025-04-14	—	—	—	No
gpt-4.1-nano-2025-04-14	gpt-4.1-nano-2025-04-14	—	—	—	No
o4-mini-2025-04-16	o4-mini-2025-04-16	—	—	—	No
Phi-3-medium-128k-instruct	phi	14	4.80	403.20	Yes
Phi-3-medium-4k-instruct	phi	14	4.80	403.20	Yes
Phi-3-mini-128k-instruct	phi	3.8	4.90	111.72	Yes
Phi-3-mini-4k-instruct	phi	3.8	4.90	111.72	Yes
phi-1_5	phi	1.4	0.15	1.17	Yes
phi-4	phi	14.7	9.80	864.36	Yes
starcoderbase	starcoder	15.5	1.00	93.00	No
starcoderbase-1b	starcoder	15.5	1.00	6.00	No
starcoderbase-3b	starcoder	15.5	1.00	18.00	No
starcoderbase-7b	starcoder	15.5	1.00	42.00	No
starcoder2-15b	starcoder2	16	4.30	387.00	Yes
starcoder2-3b	starcoder2	3	3.30	59.40	Yes
starcoder2-7b	starcoder2	7.2	3.70	155.40	Yes

Table 4.4. Correlation between Base LLM Benchmarks and Virtualhome Action Sequencing Task Performance. Bold values indicate strong correlations ($|r| \geqslant 0.7$), italic values indicate moderate correlations ($0.5 \leqslant |r| < 0.7$).

EAI Task Metrics	GPQA	MUSR	IFEval	MMLU-PRO	BBH	MATH Lvl 5
Task Success	*0.525*	*0.558*	*0.618*	**0.714**	**0.754**	**0.782**
State Goal	*0.554*	*0.564*	*0.589*	**0.706**	**0.742**	**0.761**
Relation Goal	*0.521*	*0.558*	*0.577*	**0.707**	**0.743**	**0.783**
Action Goal	*0.514*	*0.531*	*0.622*	**0.704**	**0.746**	**0.779**
Total Goal	*0.552*	*0.571*	*0.608*	**0.725**	**0.764**	**0.792**
Execution Success	*0.507*	*0.532*	*0.648*	**0.701**	**0.747**	**0.773**
Parsing	0.480	0.382	*0.530*	*0.535*	*0.574*	0.496
Hallucination	0.081	0.212	0.152	0.217	0.217	0.227
Predicate Arg	-0.204	-0.085	-0.345	-0.308	-0.382	-0.080
Wrong Order	-0.369	-0.350	-0.016	-0.485	-0.388	-0.417
Missing Step	-0.360	-0.329	-0.386	-0.355	-0.380	-0.255
Affordance	-0.198	-0.102	-0.189	-0.199	-0.205	-0.122
Additional Step	-0.317	-0.286	0.112	-0.304	-0.277	-0.190

Table 4.5. Correlation between Base LLM Benchmarks and Behavior Action Sequencing Task Performance. Bold values indicate strong correlations ($|r| \geqslant 0.7$), italic values indicate moderate correlations ($0.5 \leqslant |r| < 0.7$).

EAI Task Metrics	GPQA	MUSR	IFEval	MMLU-PRO	BBH	MATH Lvl 5
Task Success	0.311	0.203	*0.689*	*0.601*	*0.613*	*0.604*
State Goal	0.282	0.264	**0.702**	*0.613*	*0.581*	*0.649*
Relation Goal	0.355	0.148	**0.709**	*0.526*	*0.620*	*0.510*
Total Goal	0.340	0.196	**0.740**	*0.578*	*0.629*	*0.589*
Execution Success	0.333	0.184	**0.726**	*0.587*	*0.615*	*0.574*
Parsing	0.081	0.062	**0.838**	0.307	0.372	0.295
Hallucination	0.137	-0.035	-0.187	0.120	0.111	0.225
Predicate Arg	0.201	0.016	-0.112	0.256	0.220	0.247
Wrong Order	-0.171	-0.131	**-0.816**	-0.356	-0.469	-0.444
Missing Step	-0.010	0.069	**-0.783**	-0.229	-0.296	-0.254
Additional Step	0.063	0.027	*-0.515*	-0.147	-0.180	-0.208

Table 4.6. Correlation between Base LLM Benchmarks and Virtualhome Goal Interpretation Task Performance. Bold values indicate strong correlations ($|r| \geqslant 0.7$), italic values indicate moderate correlations ($0.5 \leqslant |r| < 0.7$).

EAI Task Metrics	GPQA	MUSR	IFEval	MMLU-PRO	BBH	MATH Lvl 5
Node F1	0.495	0.237	0.087	0.372	0.459	0.136
Edge F1	*0.531*	0.305	0.163	0.456	*0.567*	0.274
Action F1	0.339	0.175	0.314	0.250	0.343	0.163
All F1	*0.554*	0.289	0.162	0.427	*0.526*	0.187

Table 4.7. Correlation between Base LLM Benchmarks and Behavior Goal Interpretation Task Performance. Bold values indicate strong correlations ($|r| \geqslant 0.7$), italic values indicate moderate correlations ($0.5 \leqslant |r| < 0.7$).

EAI Task Metrics	GPQA	MUSR	IFEval	MMLU-PRO	BBH	MATH Lvl 5
Overall F1	*0.627*	*0.642*	0.484	**0.821**	**0.742**	**0.761**
State Goal F1	*0.576*	*0.605*	0.459	**0.777**	**0.707**	**0.780**
Relation Goal F1	*0.646*	*0.652*	*0.500*	**0.837**	**0.765**	**0.727**
State Hallucinati...	0.309	0.300	0.402	0.493	0.403	0.466
Object Hallucinat...	0.325	0.324	0.420	*0.511*	0.422	0.490
Format Error Rate	0.295	0.280	0.419	0.473	0.384	0.460
Grammatically Val...	-0.446	-0.412	*-0.515*	*-0.634*	*-0.518*	*-0.574*

CHAPTER 5

Conclusion

This thesis confronted a central challenge in the pursuit of general-purpose embodied intelligence: the need for a practical and economically viable path to scale. We argued that the primary obstacle is not only a single missing algorithm, but also a systemic one, rooted in the prohibitive costs of real-world interaction and the lack of a cohesive methodology for managing large-scale research and development. The core thesis of this work is that overcoming this requires a holistic solution that co-designs the foundational infrastructure, the learning models, and the scientific principles that guide their development.

To this end, we presented a three-pillar contribution. In Chapter 2, we constructed the foundational infrastructure for scalable research with EMS®and RoboFlow, systems that automate and ensure the reproducibility of large-scale experimentation. On this foundation, Chapter 3 introduced efficient and safe models, including a Markov game formulation for guaranteed collision-free navigation and the Switch Trajectory Transformer, which leverages a Mixture-of-Experts architecture to enable multi-task learning without prohibitive computational costs. Finally, Chapter 4 established the guiding principles for this new scale of research by deriving empirical scaling laws for LLM-driven agents, transforming the development process from an art into a predictable engineering discipline. These contributions are not independent; they form an integrated stack where the infrastructure enables the development of the models, and the scaling laws provide the scientific map to guide that development efficiently.

The broader impact of this work is a blueprint for the future of embodied AI research. By demonstrating an end-to-end system that addresses bottlenecks in computation, data, and predictability, this thesis provides a template for escaping the cycle of bespoke, one-off projects. It

helps shift the field's focus from asking "can a robot perform a task?" to "what is the most efficient system to teach a robot thousands of tasks"

While this thesis provides a significant step forward, it also illuminates the path for future work. The scaling laws presented here, while powerful, should be extended to new modalities and more complex, long-horizon tasks. The data flywheel, now largely automated, can be made more intelligent by developing algorithms for automatic curriculum generation and failure identification. The ultimate goal remains a single, unified world model for robotics.

References

Mlflow. https://mlflow.org/.

Google anthos. cloud.google.com/anthos.

Aws outposts. aws.amazon.com/outposts, a.

Aws robomaker. aws.amazon.com/robomaker, b.

Aws batch. https://aws.amazon.com/batch, c.

Azure arc. azure.microsoft.com/services/azure-arc.

Codalab worksheets. worksheets.codalab.org.

Confluent. www.confluent.io.

Docker. https://www.docker.com/.

Fetch robotics. fetchrobotics.com.

Formant robotics. formant.io.

Kubeflow. https://www.kubeflow.org/, a.

K8s batch. kubernetes.io/docs/concepts/workloads/controllers/job, b.

Kubernetes. https://kubernetes.io/, c.

Pywren. pywren.io.

Trifacta. www.trifacta.com.

Tarek Abdelzaher, Brian Blum, Qing Cao, Yong Chen, David Evans, Jemin George, Selvin George, Lin Gu, Tian He, Sudha Krishnamurthy, et al. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 582–589. IEEE, 2004.

Samira Abnar, Mostafa Dehghani, Behnam Neyshabur, and Hanie Sedghi. Exploring the limits of large scale pre-training. *arXiv preprint arXiv:2110.02095*, 2021.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgeniy Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

DeepSeek AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025.

Robin Amsters and Peter Slaets. Turtlebot 3 as a robotics education platform. In *International Conference on Robotics in Education (RiE)*, pages 170–181. Springer, 2019.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Cédric Arisdakessian, Sean B Cleveland, and Mahdi Belcaid. Metaflow— mics: Scalable and reproducible nextflow pipelines for the analysis of microbiome marker data. In *Practice and Experience in Advanced Research Computing*, pages 120–124. 2020.

Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936*, 2023.

Rajesh Arumugam, Vikas Reddy Enti, Liu Bingbing, Wu Xiaojun, Krishnamoorthy Baskaran, Foong Foo Kong, A Senthil Kumar, Kang Dee Meng, and Goh Wai Kit. Davinci: A cloud computing framework for service robots. In *2010 IEEE international conference on robotics and automation*, pages 3084–3089. IEEE, 2010.

Tyler WH Backman and Thomas Girke. systempiper: Ngs workflow and report generation environment. *BMC bioinformatics*, 17(1):388, 2016.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *arXiv preprint arXiv:2102.06701*, 2021.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

Quentin Bateux, Eric Marchand, Jürgen Leitner, François Chaumette, and Peter Corke. Training deep neural networks for visual servoing. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3307–3314. IEEE, 2018.

Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2023.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemys law Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Homanga Bharadhwaj, Zihan Wang, Yoshua Bengio, and Liam Paull. A data-efficient framework for training and sim-to-real transfer of navigation policies. *arXiv preprint arXiv:1810.04871*, 2018.

Ekaba Bisong. Kubeflow and kubeflow pipelines. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 671–685. Springer, 2019a.

Ekaba Bisong. An overview of google cloud platform services. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 7–10, 2019b.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*.

Barry W Boehm. A spiral model of software development and enhancement. *Computer*, 21(5): 61–72, 1988.

Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

Amanda Bouman, Muhammad Fadhil Ginting, Nikhilesh Alatur, Matteo Palieri, David D Fan, Thomas Touma, Torkom Pailevanian, Sung-Kyun Kim, Kyohei Otsu, Joel Burdick, et al. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2518–2525. IEEE, 2020.

Radhia Bouziane, Labib Sadek Terrissa, Soheyb Ayad, Jean-Franois Brethe, and Okba Kazar. A web services based solution for the nao robot in cloud robotics environment. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0809–0814. IEEE, 2017.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgeniy Chebotar, Joseph D'Amico, Sudeep Dasari, Byron David, Kurt D'Souza, Chuyuan Fu, Sagi Gleichman, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020a.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020b.

Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Ryan Burnell, Han Hao, Andrew RA Conway, and Jose Hernandez Orallo. Revealing the structure of language model capabilities. *arXiv preprint arXiv:2306.10062*, 2023.

Ethan Caballero, Kshitij Gupta, Irina Rish, and David Krueger. Broken neural scaling laws. *arXiv preprint arXiv:2210.14891*, 2022.

Philippe Casgrain, Brian Ning, and Sebastian Jaimungal. Deep q-learning for nash equilibria: Nash-dqn. *arXiv preprint arXiv:1904.10554*, 2019.

Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. *arXiv preprint arXiv:1903.00070*, 2019.

Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJgJDAVKvB.

Chang Chen, Jaesik Yoon, Yi-Fu Wu, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models, 2022. URL https://openreview.net/forum?id=s3K0arSRl4d.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

Yangyi Chen, Binxuan Huang, Yifan Gao, Zhengyang Wang, Jingfeng Yang, and Heng Ji. Scaling laws for predicting downstream performance in llms. *arXiv preprint arXiv:2410.08527*, 2024.

Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2347–2354.

IEEE, 2015.

Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of svms for very large scale problems. *Neural computation*, 14(5):1105–1114, 2002.

André M Comeau, Gavin M Douglas, and Morgan GI Langille. Microbiome helper: a custom and streamlined workflow for microbiome research. *MSystems*, 2(1), 2017.

Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.

Stampfer Dennis, Lotz Alex, Lutz Matthias, and Schlegel Christian. The smartmdsd toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software. 2016.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2019.

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.

Rosen Diankov. Automated construction of robotic manipulation programs. 2010.

Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Coumans Erwin and Bai Yunfei. Pybullet a python module for physics simulation for games. *PyBullet*, 2016.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059. IEEE, 2018.

Benjamin Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. *arXiv preprint arXiv:2002.11089*, 2020.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 538–547, 2019.

Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *International Conference on Robotics and Automation (ICRA)*, 2018.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

Lukas Finnveden. Extrapolating gpt-n performance. https://www.lesswrong.com/posts/k2SNji3jXaLGhBeYP/extrapolating-gpt-n-performance, 2020. Accessed: 2024-05-07.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

Anthony Francis, Aleksandra Faust, Hao-Tien Lewis Chiang, Jasmine Hsu, J Chase Kew, Marek Fiser, and Tsang-Wei Edward Lee. Long-range indoor navigation with prm-rl. *arXiv preprint arXiv:1902.09458*, 2019.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *arXiv preprint arXiv:1405.5848*, 2014.

Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, 2020.

Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5973–5979. IEEE, 2021.

Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova Dassarma, Dawn Drain, Nelson Elhage, et al. Predictability and surprise in large generative models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1747–1764, 2022.

Mitch Garnaat. boto documentation. 2018.

Raúl Garreta, Guillermo Moncecchi, Trent Hauck, and Gavin Hackeling. *Scikit-learn: machine learning simplified: implement scikit-learn into every step of the data science pipeline*. Packt Publishing Ltd, 2017.

Behrooz Ghorbani, Orhan Firat, Markus Freitag, Ankur Bapna, Maxim Krikun, Xavier Garcia, Ciprian Chelba, and Colin Cherry. Scaling laws for neural machine translation. *arXiv preprint arXiv:2109.07740*, 2021.

Pablo Gil, Iván Maza, Anıbal Ollero, and P Marrón. Data centric middleware for the integration of wireless sensor networks and mobile robots. In *proc. 7th Conference on Mobile Robots and Competitions, ROBOTICA*. Citeseer, 2007.

Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615*, 2019.

Ken Goldberg and Ben Kehoe. Cloud robotics and automation: A survey of related work. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5*, 2013.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international*

conference on robotics and automation (ICRA), pages 3389–3396. IEEE, 2017.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2616–2625, 2017.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Daan Hobbelen, Tomas De Boer, and Martijn Wisse. System overview of bipedal robots flame and tulip: Tailor-made for limit cycle walking. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 2486–2491. IEEE, 2008.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. *arXiv preprint arXiv:2006.14480*, 2020.

Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

Gaoping Huang, Pawan S Rao, Meng-Han Wu, Xun Qian, Shimon Y Nof, Karthik Ramani, and Alexander J Quinn. Vipo: Spatial-visual programming with functions for robot-iot workflows. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

Dominique Hunziker, Mohanarajah Gajamohan, Markus Waibel, and Raffaello D'Andrea. Rapyuta: The roboearth cloud engine. In *2013 IEEE international conference on robotics and automation*, pages 438–444. IEEE, 2013.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Vıctor Mayoral-Vilches, Hugo Zhan, Derek Xu, Ramtin Ghassemi, John Kubiatowicz, et al. Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2. *arXiv preprint arXiv:2205.09778*, 2022.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulić, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, Dec 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Robert Ilijason. Getting started with databricks. In *Beginning Apache Spark Using Azure Databricks*, pages 27–38. Springer, 2020.

David Ilić. Unveiling the general intelligence factor in language models: A psychometric approach. *arXiv preprint arXiv:2310.11616*, 2023.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in Neural Information Processing Systems*, 34, 2021a.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. *arXiv preprint arXiv:2106.02039*, 2021b.

Daniel Jiang, Emmanuel Ekwedike, and Han Liu. Feedback-based tree search for reinforcement learning. In *International conference on machine learning*, pages 2284–2293. PMLR, 2018.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Ameet V Joshi. Amazon's machine learning toolkit: Sagemaker. In *Machine Learning and Artificial Intelligence*, pages 233–243. Springer, 2020a.

Ameet V Joshi. Azure machine learning. In *Machine Learning and Artificial Intelligence*, pages 207–220. Springer, 2020b.

HW Jun, HJ Kim, and BH Lee. Goal-driven navigation for non-holonomic multi-robot system by learning collision. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1758–1764. IEEE, 2019.

Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5129–5136. IEEE, 2018.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021a.

Dmitry Kalashnikov, Jake Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Scaling up multi-task robotic reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021b.

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.

Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13(5):43–51, 2009.

Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on automation science and engineering*, 12(2):398–409, 2015.

Farzad Khodadadi, Rodrigo N Calheiros, and Rajkumar Buyya. A data-centric framework for development and deployment of internet of things applications in clouds. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6. IEEE, 2015.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246.

Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. Human-centered robot navigation—towards a harmoniously human–robot coexisting environment. *IEEE Transactions on Robotics*, 27(1):99–112, 2010.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Alex X Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration. *arXiv preprint arXiv:1703.11000*, 2017.

Yin Lei, Zhou Fengyu, Wang Yugang, Yuan Xianfeng, Zhao Yang, and Chen Zhumin. Design of a cloud robotics visual platform. In *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, pages 1039–1043. IEEE, 2016.

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. *arXiv preprint arXiv:2006.16668*, 2020.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R. Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In Karen Liu, Dana

Kulić, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 80–93. PMLR, Dec 2023. URL https://proceedings.mlr.press/v205/li23a.html.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Fei-Fei Li, Jiayuan Mao, and Jiajun Wu. Embodied agent interface: Benchmarking llms for embodied decision making. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/b631da756d1573c24c9ba9c702fde5a9-Abstract-Datasets_and_Benchmarks_Track.html.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 85, 2017.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, Ken Goldberg, et al. Fogros: An adaptive framework for automating fog robotics deployment. *arXiv preprint arXiv:2108.11355*, 2021.

Qinjie Lin, Han Liu, and Biswa Sengupta. Switch trajectory transformer with distributional value approximation for multi-task reinforcement learning. *arXiv preprint arXiv:2203.07413*, 2022a.

Qinjie Lin, Guo Ye, Jiayi Wang, and Han Liu. Roboflow: a data-centric workflow management system for developing ai-enhanced robots. In *Conference on Robot Learning*, pages 1789–1794. PMLR, 2022b.

Qinjie Lin, Guo Ye, and Han Liu. Ems®: A massive computational experiment management system towards data-driven robotics. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9068–9075. IEEE, 2023.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

Michael L Littman. Friend-or-foe q-learning in general-sum games. In *Conference on Artificial Intelligence (AAAI)*, 2001.

Nelson F Liu, Tony Lee, Robin Jia, and Percy Liang. Do question answering modeling improvements hold across benchmarks? *arXiv preprint arXiv:2102.01065*, 2021.

Yi Liu and Yuchun Xu. Summary of cloud robot research. In *2019 25th International Conference on Automation and Computing (ICAC)*, pages 1–5. IEEE, 2019.

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.

Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259. IEEE, 2018.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Armel Zebaze, Chenghao Mou, Christopher Akiki, Marco Zocca, Chenghao Yang, Gabriel Villalobos, and BigCode-Project. StarCoder 2 and The Stack v2: The Next Generation, 2024a.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024b.

Yunlian Lyu, Yimin Shi, and Xianggang Zhang. Improving target-driven visual navigation with attention on 3d spatial relationships. *Neural Processing Letters*, pages 1–20, 2022.

Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1957–1964. IEEE, 2016.

Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

Michael Mathieu, Sherjil Ozair, Srivatsan Srinivasan, Caglar Gulcehre, Shangtong Zhang, Ray Jiang, Tom Le Paine, Konrad Zolna, Richard Powell, Julian Schrittwieser, et al. Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021.

Oier Mees and Wolfram Burgard. Language-conditioned policy learning for long-horizon robot manipulation tasks.

Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache

spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

John P Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *International conference on machine learning*, pages 7721–7735. PMLR, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Middleware for robotics: A survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. Ieee, 2008.

Hatef Monajemi, David L Donoho, and Victoria Stodden. Making massive computational experiments painless. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2368–2373. IEEE, 2016.

Hatef Monajemi, Riccardo Murri, Eric Jonas, Percy Liang, Victoria Stodden, and David L Donoho. Ambitious data science can be painless. *arXiv preprint arXiv:1901.08705*, 2019.

André Monteiro, Cláudio Teixeira, and Joaquim Sousa Pinto. Sky computing: Exploring the aggregated cloud resources—part ii. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2014.

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.

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A Raffel. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

Jinjie Ni, Fuzhao Xue, Xiang Yue, Yuntian Deng, Mahir Shah, Kabir Jain, Graham Neubig, and Yang You. Mixeval: Deriving wisdom of the crowd from llm benchmark mixtures. *arXiv preprint arXiv:2406.06565*, 2024.

Kenji Nishimiya and Yuta Imai. Serverless architecture for service robot management system. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11379–11385. IEEE, 2021.

OpenAI. ChatGPT (august 7, 2025 version) [large language model]. https://chat.openai.com, 2025. Used via prompts in Appendix; see methodology section for prompt wording and generated responses.

David Owen. How predictable is language model benchmark performance? *arXiv preprint arXiv:2401.04757*, 2024.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tür. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025. AAAI Press, 2022. URL https://ojs.aaai.org/index.php/AAAI/article/view/20097.

Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. Risk averse robust adversarial reinforcement learning. *arXiv preprint arXiv:1904.00511*, 2019.

Haotian Pang, Han Liu, Robert J Vanderbei, and Tuo Zhao. Parametric simplex method for sparse learning. In *Advances in Neural Information Processing Systems*, pages 188–197, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Efficient transformers in reinforcement learning using actor-learner distillation. *arXiv preprint arXiv:2104.01655*, 2021.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

Yotam Perlitz, Elron Bandel, Ariel Gera, Ofir Arviv, Liat Ein-Dor, Eyal Shnarch, Noam Slonim, Michal Shmueli-Scheuer, and Leshem Choshen. Efficient benchmarking (of language models). *arXiv preprint arXiv:2308.11696*, 2023.

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: Evaluating llms with fewer examples. *arXiv preprint arXiv:2402.14992*, 2024.

Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, 2000.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8494–8502, June 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Puig_VirtualHome_Simulating_Household_CVPR_2018_paper.html.

Yuanyuan Qiu, Hongzheng Li, Shen Li, Yingdi Jiang, Renfen Hu, and Lijiao Yang. Revisiting correlations between intrinsic and extrinsic evaluations of word embeddings. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data: 17th China National Conference, CCL 2018, and 6th International Symposium, NLP-NABD 2018, Changsha, China, October 19–21, 2018, Proceedings 17*, pages 209–221. Springer, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. Offline reinforcement learning from images with latent space models. In *Learning for Dynamics and Control*, pages 1154–1168. PMLR, 2021.

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. *arXiv preprint arXiv:1910.10683*, 2019.

Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*, 2018.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International conference on machine learning*, pages 5389–5400. PMLR, 2019.

Sekou L Remy and M Brian Blake. Distributed service-oriented robotics. *IEEE Internet Computing*, 15(2):70–74, 2011.

Adam Richie-Halford and Ariel Rokem. Cloudknot: A python library to run your existing code on aws batch. In *Proceedings of the 17th python in science conference*, pages 8–14, 2018.

Robert Riener, Lars Lünenburger, and Gery Colombo. Human-centered robotics applied to gait training and assessment. *Journal of Rehabilitation Research & Development*, 43(5), 2006.

Fetch Robotics. Fetch research robot.

Kay Römer, Oliver Kasten, and Friedemann Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. URL https://neurips.cc/virtual/2024/poster/95350.

Olimpiya Saha and Prithviraj Dasgupta. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, 7(3):47, 2018.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019. ISSN 2377-3766.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9339–9347, 2019. doi: 10.1109/ICCV.2019.00943. URL https://openaccess.thecvf.com/content_ICCV_2019/html/Savva_Habitat_A_Platform_for_Embodied_AI_Research_ICCV_2019_paper.html.

Gigi Sayfan. *Mastering kubernetes*. Packt Publishing Ltd, 2017.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, Vladlen Koltun, and Kayvon Fatahalian. Large batch simulation for deep reinforcement learning. *arXiv preprint arXiv:2103.07013*, 2021.

Subarna Shakya et al. Survey on cloud based robotics architecture challenges and applications. *Journal of Ubiquitous Computing and Communication Technologies (UCCT)*, 2(01):10–18, 2020.

Jinghuan Shang and Michael S Ryoo. Starformer: Transformer with state-action-reward representations. *arXiv preprint arXiv:2110.06206*, 2021.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10740–10749, June 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Shridhar_ALFRED_A_Benchmark_for_Interpreting_Grounded_Instructions_for_Everyday_Tasks_CVPR_2020_paper.html.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. *arXiv preprint arXiv:2102.06177*, 2021.

Joseph H Solomon, Mark A Locascio, and Mitra JZ Hartmann. Linear reactive control for efficient 2d and 3d bipedal walking over rough terrain. *Adaptive Behavior*, 21(1):29–46, 2013.

Jake M Stauffer. *A Smart and Interactive Edge-Cloud Big Data System*. PhD thesis, Purdue University Graduate School, 2022.

Elias Stengel-Eskin, Andrew Hundt, Zhuohong He, Aditya Murali, Nakul Gopalan, Matthew Gombolay, and Gregory Hager. Guiding multi-step rearrangement tasks with natural language instructions. In *Conference on Robot Learning*, pages 1486–1501. PMLR, 2022.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR, 2021.

Robin Strudel, Alexander Pashevich, Igor Kalevatykh, Ivan Laptev, Josef Sivic, and Cordelia Schmid. Learning to combine primitive skills: A step towards versatile robotic manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4637–4643. IEEE,

2020.

Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj, and Mahbube Ardani. Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments. *arXiv preprint arXiv:2005.13857*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Mirac Suzgun, Nathan Scales, Nathanael Sch"arli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1111–1117. IEEE, 2018.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. *Advances in Neural Information Processing Systems*, 33:18583–18599, 2020.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling? In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

AlphaStar Team. Alphastar: mastering the real-time strategy game starcraft ii. *DeepMind blog*, 24, 2019.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

François Torregrossa, Vincent Claveau, Nihel Kooli, Guillaume Gravier, and Robin Allesiardo. On the correlation of word embedding evaluation metrics. In *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, pages 4789–4797, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 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, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Volker Tresp. Mixtures of gaussian processes. *Advances in neural information processing systems*, pages 654–660, 2001.

Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706, 2020.

Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

Dizan Vasquez, Billy Okal, and Kai O Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison. In *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

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*, pages 5998–6008, 2017.

Richard Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2):189–208, 2008.

Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Häussermann, Rob Janssen, JMM Montiel, Alexander Perzylo, et al. Roboearth. *IEEE Robotics & Automation Magazine*, 18(2):69–82, 2011.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2018.

Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Advances in neural information processing systems*, pages 1603–1610, 2003.

Steve Waterhouse, David MacKay, Tony Robinson, et al. Bayesian methods for mixtures of experts. *Advances in neural information processing systems*, pages 351–357, 1996.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Michael Weinberg and Jeffrey S Rosenschein. Best-response multiagent learning in non-stationary environments. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 506–513. IEEE Computer Society, 2004.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-free LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization revisited: A case study for quadrupedal locomotion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4955–4961. IEEE, 2021.

Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. Production machine learning pipelines: Empirical analysis and optimization opportunities. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2639–2652, 2021.

Zhiyuan Xu, Kun Wu, Zhengping Che, Jian Tang, and Jieping Ye. Knowledge transfer in multi-task deep reinforcement learning for continuous control. *arXiv preprint arXiv:2010.07494*, 2020.

Chhavi Yadav and Léon Bottou. Cold case: The lost mnist digits. *Advances in neural information processing systems*, 32, 2019.

Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. *arXiv preprint arXiv:2102.05815*, 2021.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Guo Ye, Qinjie Lin, Tzung-Han Juang, and Han Liu. Collision-free navigation of human-centered robots via markov games. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11338–11344. IEEE, 2020.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020a.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020b.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Chelsea Finn, Sergey Levine, and Karol Hausman. Data sharing without rewards in multi-task offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021a.

Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021b.

Yang Yu, Bhaskar Krishnamachari, and Viktor K Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE network*, 18(1):15–21, 2004.

Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Başar. Fully decentralized multi-agent reinforcement learning with networked agents. *arXiv preprint arXiv:1802.08757*, 2018.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.