NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report

Number: NU-CS-2026-14

March, 2026

StitchDraft: A Block-Based System for Visualizing Fit in Hand-Knit Garments

Diana Whealan

Abstract

Hand-knit patterns describe how to create a garment but do not explicitly describe the finished garment's shape or dimensions. From the pattern instructions, expressed as row-by-row stitch operations, knitters must infer the garment's final fit before knitting it. Existing tools for knitting visualization focus on stitch texture rather than whole-garment geometry. As a result, knitters are unable to translate hand-knit pattern instructions into visual representations of finished garments. This project introduces StitchDraft, a system that allows users to construct knitting programs from knitting pattern instructions using a block-based interface that contains commands that mirror pattern instructions and compile to a Python-based knitting domain-specific language (DSL). The system executes pattern commands, expands stitch operations, and computes a stitch layout that is rendered as a garment schematic. These schematics can be measured and overlaid on proportional body silhouettes to visualize garment fit. The system was evaluated by verifying stitch-count correctness, repeat expansion, marker behavior, garment construction operations such as stitches on hold and chart joins, and gauge-based measurement calculations. In addition, a real knitting pattern was translated into StitchDraft to compare the generated schematic with a finished garment. The results demonstrate that StitchDraft accurately interprets knitting instructions and generates meaningful garment schematics for evaluating garment geometry and fit prior to knitting.

Keywords

Knitting pattern visualization, garment schematic generation, block-based programming, domain-specific languages, garment fit visualization, visual programming, computational design tools

ABSTRACT

Hand-knit patterns describe how to create a garment but do not explicitly describe the finished garment's shape or dimensions. From the pattern instructions, expressed as row-by-row stitch operations, knitters must infer the garment's final fit before knitting it. Existing tools for knitting visualization focus on stitch texture rather than whole-garment geometry. As a result, knitters are unable to translate hand-knit pattern instructions into visual representations of finished garments. This project introduces StitchDraft, a system that allows users to construct knitting programs from knitting pattern instructions using a block-based interface that contains commands that mirror pattern instructions and compile to a Python-based knitting domain-specific language (DSL). The system executes pattern commands, expands stitch operations, and computes a stitch layout that is rendered as a garment schematic. These schematics can be measured and overlaid on proportional body silhouettes to visualize garment fit. The system was evaluated by verifying stitch-count correctness, repeat expansion, marker behavior, garment construction operations such as stitches on hold and chart joins, and gauge-based measurement calculations. In addition, a real knitting pattern was translated into StitchDraft to compare the generated schematic with a finished garment. The results demonstrate that StitchDraft accurately interprets knitting instructions and generates meaningful garment schematics for evaluating garment geometry and fit prior to knitting.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to the Northwestern Computer Science Department for giving me the opportunity to explore my interests and passions throughout the completion of my Master of Science degree. I would also like to thank Professors Dietrich Geisler and Connor Bain for serving as my advisors on this project and for guiding me toward tools that were directly integrated into this project. I would also like to thank Professor Haoqi Zhang for teaching me to self-direct research.

This project is not just a technical effort, but a true passion project that I would not have been able to do without the love and support of my family. When I felt that knitting was just a hobby, my family strongly encouraged me to pursue it as a true art and skill. I would like to thank my parents for always believing in the importance of my artistic pursuits alongside my academic goals. I would especially like to thank my fiancé James for always lifting me up when I was unsure of myself, and my dog Luna, who kept me company on the couch for many of the hours I spent working on this project.

Lastly, I would like to thank the knitting community. There are so many talented artists producing works that I believe deserve to be spotlighted. Knitting is not only an art form but is also a science. I hope that this project helps to highlight the technical knowledge, skill, and effort this craft encompasses.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION AND BACKGROUND

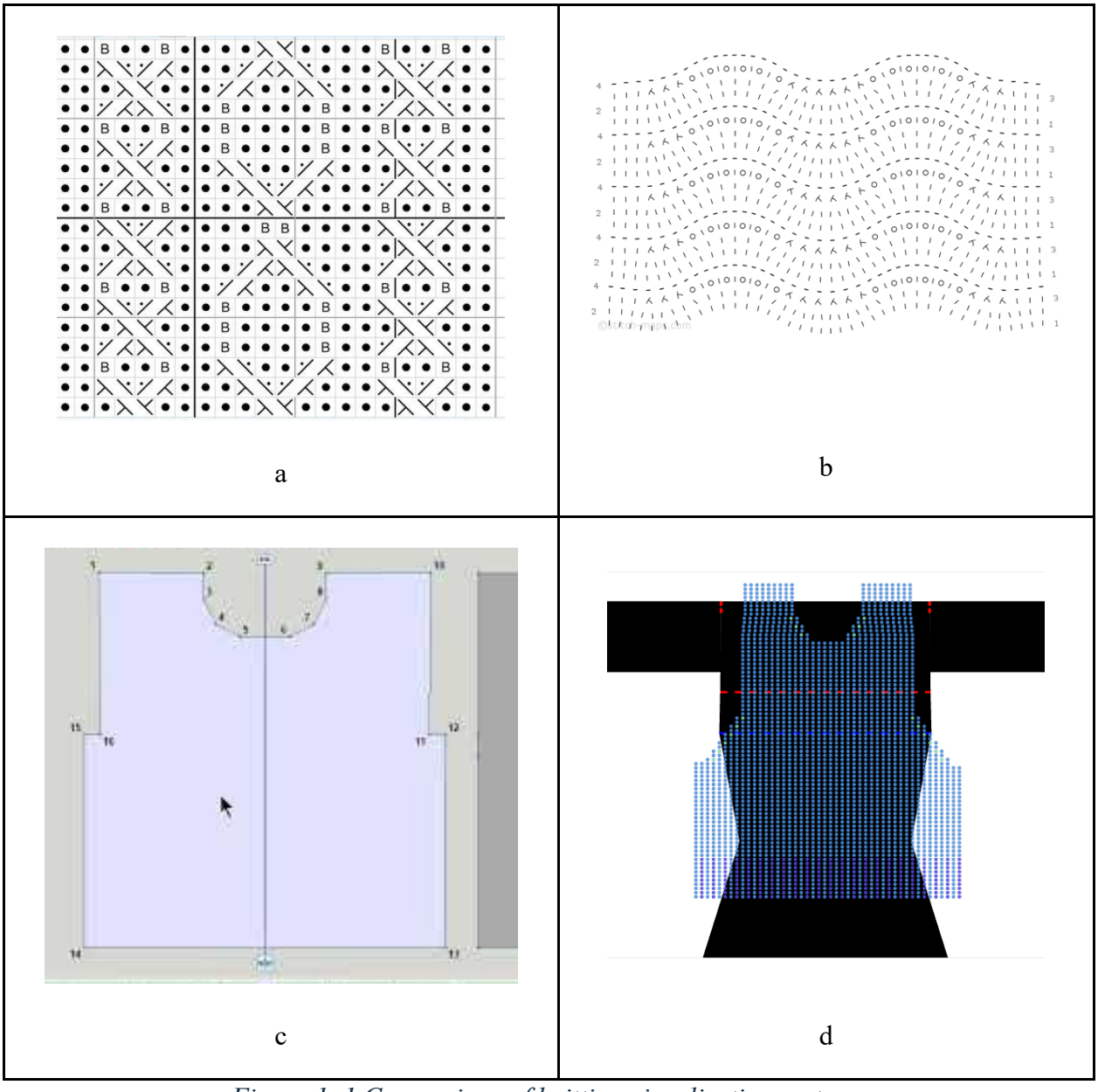
1.1 Introduction

In many disciplines, it is standard practice to create a visual rendering before physical implementation. Landscape architects use renderings to experiment with plant placements and spacing before installation, mechanical engineers use tools like CAD models to evaluate the dimensions and interaction of components before manufacturing prototypes, and even software engineers use diagrams and prototypes to reason about a system's structure before creating production-level code. All these tools, across varying fields, allow for trial and error, before committing time and resources that cannot be recovered. The practice of hand-knitting takes a lot of time and effort and the materials for a full-size sweater can easily cost more than $100.

Hand-knit patterns describe how to create a garment but do not explicitly describe the finished garment's shape or dimensions. From the pattern instructions, expressed as row-by-row stitch operations, knitters must infer the garment's shape, proportions, structure, and final fit before knitting it. When hand-knit designers create new patterns or translate a pattern to multiple sizes, they must adjust instructions without explicitly knowing the end visual and fit effects. This results in many cycles of knitting and unraveling for new designs, and a lack of complete understanding of how different size-based instructions look on the bodies of the intended size. A garment schematic provides a visual blueprint for a finished piece. It translates rows and stitches into real-world dimensions to show the final measurements and proportions to enable informed fit decisions. With this information, knitters and knitwear designers compare garment measurements to body measurements and to garments that fit well, to identify areas for modification. Schematics transform garment creation and sizing from guesswork to decision-making.

Existing tools do not bridge the gap between knitting instructions and garment schematics. Existing charting web tools, such as Stitch Fiddle [1], require users to draw charts manually and shade stitches individually, resulting in tedious, error-prone work. Stitch Fiddle charts are flat grids of stitches: Each stitch is placed in a single cell in the grid and serves as a visual sequence of stitches, equivalent to written instructions, rather than the geometry of the garment (Figure 1.1.a).

Knitting patterns already mirror code, as they function as a set of executable instructions. Prior work has explored this connection, including Knotty [2], a knitting domain-specific language (DSL) written in a format intended to be easy for humans to write and parse, but also highly structured. Knotty was created to generate knitting charts from written instructions and demonstrates that hand-knitting instructions can be represented computationally. However, Knotty was explicitly designed to create grid charts, like those in Stitch Fiddle, that represent stitch instructions, fabric texture, and colorwork. As with other gridded charts, Knotty does not model garment geometry or whole-garment construction.



a b c d

Figure 1. 1 Comparison of knitting visualization systems

Stitch geometry can be calculated, as evidenced by the web tool Stitch Maps [3]. Stitch Maps was designed to visually help knitters understand how the stitches of a lace pattern fit together. These charts are created using a DSL called “knitspeak” that is not confined to a grid-like structure and instead provides a more fluid and accurate representation of the finished product by showing each stitch and how it connects to the stitches above and below it. (Figure 1.1.b). Stitch Maps once again prove how DSLs can accurately translate knitting instructions into visual knitting representations. However, Stitch Maps and “knitspeak” were not intended for whole-garment visualization and were specifically for viewing lace-specific charts.

Garment design and pattern-generation software, such as Design a Knit 9 [4], helps designers create garment shapes digitally, define measurements and sizing, and generate row-by-row knitting instructions (Figure 1.1.c). While this tool is helpful in garment design, it uses an inverted workflow compared to this project. This system is not useful for knitters and designers working from an existing pattern with text instructions, as many knitters want to visualize a text pattern before starting, and designers often modify their own written patterns to create new designs. Furthermore, it is not helpful for designers who follow a less structured design process and use trial and error to write their pattern instructions.

To address these challenges, we introduce StitchDraft: a system that lets knitters design garments and visualize their 2-dimensional shaping through a block-based interface that compiles to a Python knitting DSL and generates garment schematics that can be overlaid on torso silhouettes of varying sizes. This system enables instruction-level garment construction, automatic schematic generation, visualization of fit relative to body measurements, and an exploration of size and modification decisions for both hand-knit pattern designers and knitters (Figure 1.1.d).

This work offers the following contributions:

1. A domain-specific language that models knitting patterns at the garment-construction level, supporting structural operations such as section joins, stitches placed on hold, and explicit stitch marker management.
2. A method for generating garment schematics directly from pattern instructions and overlaying the resulting schematic on proportional body measurements, enabling visualization of garment proportions and fit prior to knitting.

3. A rendering pipeline that incrementally updates knitting schematics as garment instructions are authored.
4. A visual programming interface that allows designers to construct garment logic through blocks that compile to the underlying knitting DSL.

1.2 Background

1.2.1 Understanding the Structure of Knitting Patterns

Hand-knit patterns are written as a sequence of rows or rounds. Each row or round will include a list of instructions that specifies a series of operations. Each operation tells the knitter how to manipulate the fabric. Knitting is defined as having “live” stitches, meaning loops on the needle that can be immediately knit. There are two basic stitches in knitting: knit and purl. Each stitch is considered the inverse of the other, meaning on one side, a knit stitch looks like a purl stitch. Increases can be constructed in many ways, but no matter the method, they create a new stitch in between two existing stitches. Decreases, like increases, can be done in different ways, but they always reduce the number of stitches. To bind off means to take stitches off the needle and secure them so they cannot unravel. These stitches are no longer “live.” Cast-ons, like increases, create new live stitches, but unlike increases, they are not only formed between other stitches. These new stitches can be added to the end of a row or round and can consist of any number of stitches. “Placing stitches on hold” is an operation that tells the knitter not to secure a set of stitches, but also not to work them until told to “pick up the stitches” again. Lastly, an operation can tell the knitter to join two sections together. This means that two separate pieces that were knit separately will now be knit together. Knitting is designated with a “Right Side” (RS) and a “Wrong Side” (WS) and is worked either flat or in the round. When working flat, rows alternate between the RS and WS. When working in the rounds, each round stays on the same side as the last. Pattern construction consists of a series of instructions that tell knitters each step they must take to construct the garment.

1.2.2 Stitch Counts and Garment Shaping

Within a knitting pattern, some instructions will include a series of increases, decreases, cast-ons, and cast-offs to shape the garment. All of these instructions result in adding or taking away stitches. In knitting, the number of stitches in a row, or the stitch count, determines how wide the piece is at that point. Changes in these counts shape the garment. As the knitter works increase stitches, the garment expands, and as they work decrease stitches, the garment decreases. The visual effect of these operations depends on their location within the row. Increases placed at the edges of a garment typically create outward expansion on the edge closest to the increase, while increases placed within the interior of the row distribute new stitches within the fabric, pushing stitches outward more evenly. Over successive rows, these operations produce the garment's geometric shaping.

1.2.3 Stitch Markers

Knitters often use a tool called a stitch marker (Appendix Figure A.1). Placing a stitch marker means placing a marker at a specified location to keep track of a position between stitches. Slipping a stitch marker simply means moving the marker from the right needle to the left as you continue working the stitches. These markers indicate structural positions in the pattern and can be used to segment rows/rounds of stitches. Using stitch markers, the pattern can denote an operation relative to the marker rather than in absolute positional terms. They often define the start of a round, shaping regions, pattern repeats, or garment sections.

1.2.4 Knitting Charts

Standard knitting charts represent stitch patterns using a grid, where each cell represents a stitch. These charts are mainly used to represent texture and stitch/color patterns (Appendix Figure A.2).

1.2.5 Garment Schematics

A schematic is a measurement diagram of the finished garment. It shows dimensions such as chest width, sleeve length, armhole depth, and shoulder width (Appendix Figure A.3). To

convert a stitch count to a schematic measurement, “gauge” is used. Knitting gauge refers to the tension and is the number of stitches and rows in a 4x4-inch (10x10cm) square of fabric. This determines the density of the fabric. Gauge values allow for automatic conversion between stitch and row counts into real measurements.

1.2.6 Domain-Specific Languages for Knitting

Knitting patterns include instructions typically written using a standard notation, and although intended for human interpretation, they resemble a domain-specific instruction language that defines operations applied sequentially. Some existing domain-specific languages (DSL) have been built based on this observation. A knitting domain-specific programming language is a specialized programming language designed to represent, generate, or automate knitting patterns for hand knitting or machine knitting. Knitting DSLs are generally designed to introduce automation and validation to reduce error-prone manual work.

1.2.7 Garment Fitting

Knitting patterns often include a fit recommendation. This includes information about which bust size corresponds to which letter or number size (e.g., S, M, L, or 1, 2, 3), and about how loose or fitted the garment is supposed to be. This amount of oversize or tightness is referred to as ease. Patterns with positive ease mean the final garment’s measurements will be larger than those of the body, and patterns with negative ease mean the final garment’s measurements will be smaller than those of the body and will produce a form-fitting look due to the fabric’s ability to stretch.

CHAPTER 2
SYSTEM DESIGN AND IMPLEMENTATION

2.1 System Overview

StitchDraft begins with the user assembling a knitting program using blocks in a browser-based web tool. The frontend then compiles the blocks into an intermediate representation (IR) composed of commands and pattern strings. The backend validates the IR and passes it to the Python engine to construct a geometrically accurate stitch layout comprising rows, nodes, and links. The frontend renders the resulting schematic and optionally overlays it on a proportional 2D torso SVG to visualize the garment fit.

2.1.1 System Architecture

StitchDraft's architecture comprises a block-based interface, a Python knitting DSL, a garment geometry engine, and a visualization layer (Figure 2.1). The block-based interface enables the user to build a knitting program using Blockly blocks [5]. This includes chart setup, rows/rounds of stitches, repeat rows/rounds, placing and removing stitch markers, placing stitches on hold, and joining charts. In StitchDraft a "Chart" block refers to a unique section of fabric and its construction commands. As the user edits the workspace in the frontend, the block program and pattern expressions are continuously compiled into a DSL-based IR that the backend can interpret, validate, and execute. The continuous compilation allows the user to preview incremental results and quickly find errors.

The backend engine interprets the IR and expands the pattern expression into sequences of stitch operations, represented as rows of nodes. The stitch placement calculations are created from the expanded sequence to compute coordinates that reflect shaping, increases, and decreases, relative to the previous row, rather than a uniform grid. Once the stitch coordinates are calculated, the backend returns the stitch layout representation comprised of rows, markers, nodes, and links. The frontend renders the node-link stitch layout as a schematic and can overlay it onto a proportional torso silhouette generated by the backend (in inches) and aligned using a consistent units per inch mapping.

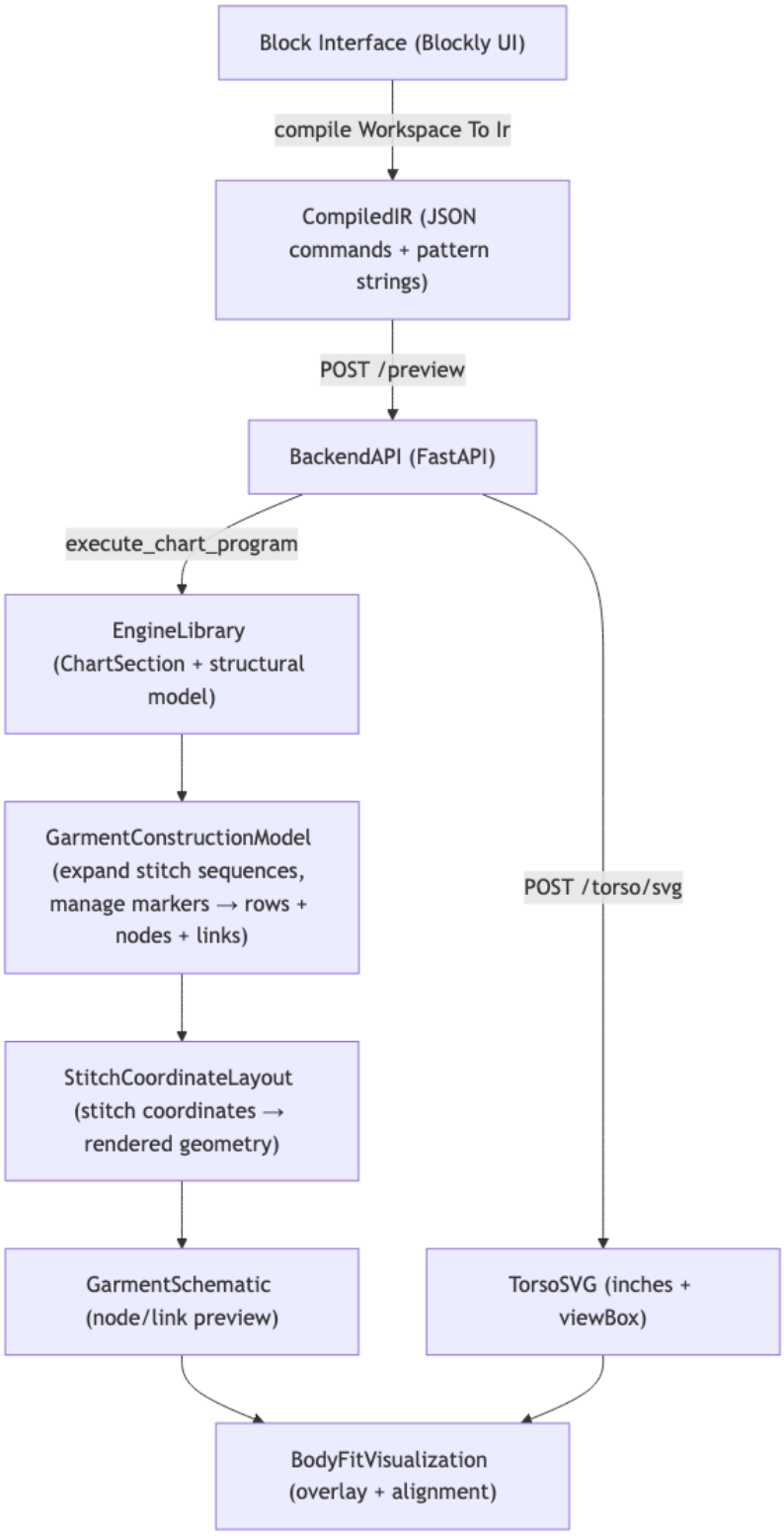


Figure 2. 1 StitchDraft system architecture end-to-end pipeline

2.1.2 User workflow

The user opens the StitchDraft web application and is presented with an interface that has a toolbox of blocks organized by category, an empty Blockly workspace, a preview panel for schematic visualizations, and a compile/error/warning panel for feedback. The toolbox contains 3 categories: “Chart setup,” “Stitch rows,” and “Structure & markers.” The user begins in the “Chart setup” category by dragging a “Chart” block into the workspace. This block defines the initial program context, including the chart’s name, the side on which the first worked row is (RS or WS), and the stitch gauge. The “Chart” block also contains a command area where the rest of the knitting program is assembled. To define the fabric's starting size, the user adds a “cast on start” block to establish the initial live stitches. This first row provides the base stitches from which subsequent rows or rounds are constructed.

To build the garment structure, the user adds blocks from the “Stitch rows” category inside the chart, such as “add row/round,” “repeat rows/rounds,” and “cast on additional” stitches. If the user chooses to use repeat rows/rounds, they will drag any number of pattern blocks into the repeat block command area. While building the garment, the user can also place stitch markers, place or pick up stitches on hold, and join charts together by dragging blocks from the “Structure & markers” category. Within each row, round, or pattern block, the user types stitch instructions in a text field. These instructions use the pattern-level DSL, with expressions like k2, p2, repeat(k1,p1), inc, sm. This process creates a hybrid authoring workflow in which blocks define the common pattern commands and the text DSL defines flexible stitch-level behavior.

As the user edits either the blocks or the pattern text, the frontend automatically recompiles the workspace into an IR. No explicit compile or run action is required. If the recomputation takes noticeable time, which occurs when rendering a large quantity of nodes, the system shows a loading state. The backend validates the IR and passes it to the Python engine. The engine parses and expands the pattern strings, checks stitch usage, and computes the stitch layout representation. In the current implementation, the stitch layout is recomputed from the beginning after each edit rather than updated only for affected rows, to ensure accuracy as row changes cascade to subsequent rows and charts that are joined. The preview panel updates automatically after each successful recomputation. The system also provides feedback in the form of execution errors, such as “Row would consume 11 but only 10 available,” parsing errors, such as “Unknown operation:

foo in token 'foo"', and warnings, such as "final repeat will be partial and the pattern may not align." This immediate feedback helps users identify invalid patterns and potentially unintentional results.

Then, as the user builds and modifies the garment program, they inspect the visual schematic to verify stitch layout and shaping behavior. The automatic visual updates provide a canvas for the user to iteratively refine the design and compare the preview against expectations. The user can drag a box over the schematic in the preview to view measurement information, the number of stitches across, the number of rows, and approximate physical dimensions. This allows the user to compare the schematic measurements to what was expected and record incremental stitch/row counts to use as checkpoints for garment construction. When desired, the user clicks "Show torso view" to open the body-based visualization. The generated schematic can then be overlaid on a proportional torso silhouette. The user may drag and move the schematic around on the torso. This helps the user assess how the garment will look on bodies of different sizes, and whether sizing or construction choices should be changed. At any point, the user can return to the workspace and modify any part of the program.

2.2 System Design

2.2.1. Block-Based Interface

Block-based programming environments are widely used as introductory programming tools because they reduce syntactic complexity and lower barriers to entry for novice programmers [6]. This interface was chosen to make the system accessible to the many knitters and designers who lack prior programming experience. Blocks reduce the need to memorize exact DSL syntax or repeatedly consult documentation. This is especially useful in the context of knitting, where the same operational instruction can be written differently across patterns. For example, a pattern may include "Place a marker to mark the beginning of a round" while another has "PM for BOR". While the operations in other knitting DSLs like Knotty [2] are easy for experienced knitters to read, writing the commands requires strict understanding of the language's documentation. Knotty is also written as a module or add-on for the programming language Racket, which requires prior coding knowledge. Additionally, StitchDraft's use of Blockly, Google's drag-and-drop functional block visual editor, enabled a browser-based interface and provided a structured visual

programming environment that emits events when the workspace changes, making it well-suited for a system that requires continuous compilation and previews.

In StitchDraft's block-based interface, common knitting operations are represented directly as blocks (Figure 2.2). These operations have consistent meaning and are commonly found across patterns and therefore map naturally to individual blocks. These include chart creation, cast-on operations, row and round construction, repeat rows/rounds, marker placement, placing/removing stitches on hold, and garment joins. Knitting patterns consist of sequential steps that describe how the fabric evolves over time. As the blocks are linked together within an established "Chart" block, they form an ordered sequence of construction steps that mirrors the process of constructing a written pattern or a knitted garment.

Several blocks support common garment-construction operations that are not represented in existing charting tools. These examples include blocks for operations such as placing stitches on hold (removing stitches currently being worked without binding them off), returning stitches that were placed on hold to the needle (making them live again), and joining two charts. By modeling these operations directly, the block interface supports the construction of complete garments rather than isolated stitch charts.

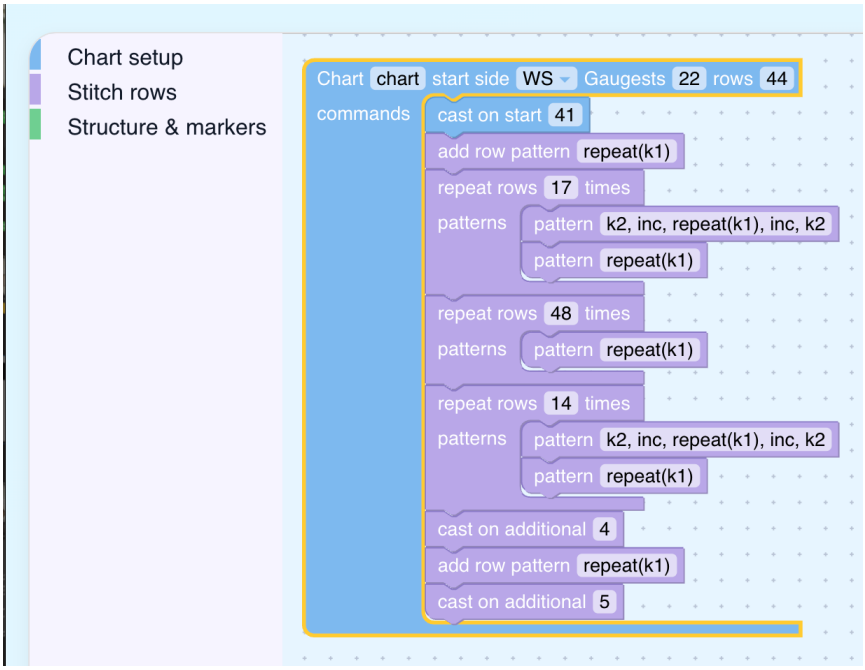


Figure 2. 2 StitchDraft Blockly UI

2.2.2 Knitting DSL

StitchDraft introduces a domain-specific language (DSL) as an intermediate representation (IR) between the block-based interface and the backend engine that supports calculating stitch positioning and eventual schematic geometry. Through the Blockly interface, the user constructs programs by linking blocks together and writing pattern text fields, and these instructions are compiled into a structured representation that the backend can interpret, validate, and execute.

StitchDraft's DSL has two layers: a command-level language and a pattern-level language. The command level specifies which operations occur and in what order to capture common pattern commands in garment construction, while the pattern-level isolates the flexible, individual stitch semantics, allowing the geometry engine to reason about stitch counts and shaping. The system uses a sequence of op-tagged operations, such as `add row` and `repeat rounds`, that describe how a garment is built over time. Some commands embed a textual mini-language in a designated `pattern` field, such as "k2, inc, repeat(k1, p1), sm, work est" that specifies how to work each stitch within a row or round using tokens such as k, p, inc, dec, bo, co, pm, sm, rm, repeat(...), and "work as established". The pattern-string DSL is parsed, validated, and expanded by the backend to produce a per-stitch sequence with geometrically accurate stitch coordinates.

The DSL was designed as a hybrid of block commands and pattern strings. The block commands reduce the user's ability to input malformed commands that are common across knitting patterns but often written with slight variations. Additionally, the blocks reduce the amount of syntax the user must learn. However, StitchDraft could not rely solely on the command-level language; pattern sequences can be very long and unique and building with blocks would become tedious and unintuitive. Therefore, StitchDraft still offers string inputs for the pattern-level language, creating a smaller set of language syntax for the user to learn.

2.2.3 Pattern Execution Engine

When the backend receives a list of "ChartPrograms", each with a list of "KnittingCommands", from the intermediate representation (IR), it iterates through each Chart and executes each command by calling the appropriate method in the backend Python engine. Commands like `add row` and `add round` carry a `pattern` string, like "k2, inc, repeat(k1)". The engine tracks information such as available stitch count, side, markers, last-row side, and whether it is round to

correctly expand the pattern string. This expansion tokenizes the strings and resolves “repeat()” and “work est” using all the available context, such that the result is a pure list of stitch types (k, p, inc, dec) that can be converted into nodes. The engine then uses the side context, Right Side (RS) or Wrong Side (WS), to determine if the expanded sequence of stitches should be reversed before it is appended. This is done because rows worked on the wrong side (WS) look reversed on the right side (RS), and StitchDraft renders the fabric from the RS perspective.

The context that allows the rows to be expanded is available because the knitting state is maintained as rows are processed. The engine maintains the number of stitches currently on the needle by always tracking the number of stitches in the last row. When a join occurs, the last row becomes a joint row of the last row of the chart, visually on the left, followed by the chart on the right. The backend also maintains what exact stitches were in the previous row, to be used when the user writes “work established.” This allows the program to copy the exact stitch type from the last worked row and maintain repeating stitch patterns. Along with the number of stitches and stitch types, the engine also tracks the side of the last row that was worked on. This way, the engine knows a new row will be the opposite side of the last, and a new round will be the same side as the last.

The backend engine manages the current state of the stitch markers, both relative to the RS and WS, to allow the computational function of stitch markers to serve the same function as the physical stitch markers. When a row is expanded, the engine uses these markers to split the pattern into segments. As each segment is processed, marker positions are moved to maintain their actual index when stitches are added or removed from a row. Stitches on hold are saved in the engine by storing current unworked/unconsumed stitches under a name given by the user. The engine maintains the actual set of nodes so the next row does not work these stitches, and so the stitches can be placed back on the needle.

2.2.4 Stitch Layout

StitchDraft calculates stitch positions within the engine from the gauge and the positions of the stitches in the previous row. At a high level, the x position of a stitch is derived from the previous row of stitches, then recentered, and the y position is determined by the row index and the row height. The user enters the gauge, and the engine calculates distance of each stitch within a row and the distance between each row. In knitting, increases and decreases produce different visual

results based on where the shaping stitch occurs in the row (Figure 2.3). To capture this and produce stitch layouts that more accurately represent how the fabric will look, the engine uses anchors to indicate where the shaping occurred, then computes and "spreads" the row of stitches so it remains consistent in width while preserving topology. Newly cast-on stitches extend past the last stitch worked.

When displaying the stitches, the engine processes rows worked on the WS to flip the stitch types and display what they look like on the RS. This means purl stitches worked on the WS are rendered as knit stitches and knit stitches worked on the WS are rendered as purl stitches. When two charts are joined, the stitches in the chart positioned to the right are shifted right in their x coordinates so its last row sits to the right of the left chart's last row, and the stitches' y coordinates are adjusted so the last rows line up vertically.

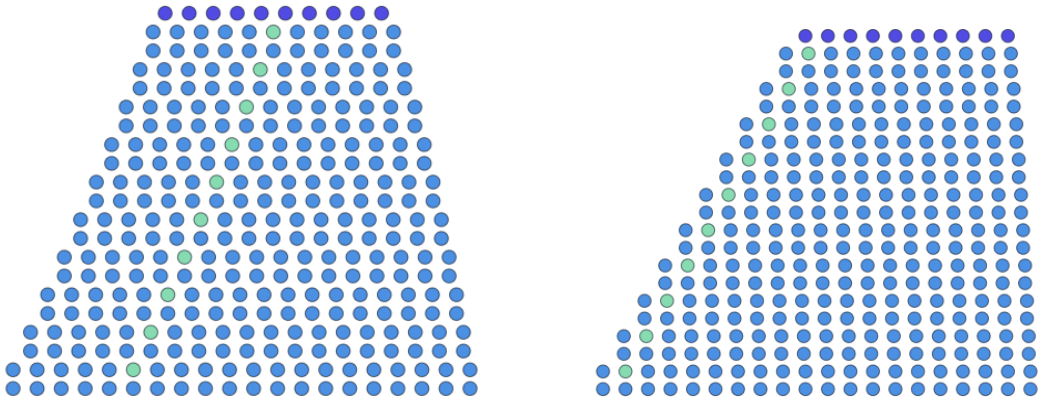


Figure 2. 3 Comparison of increase stitch placement. Left: increases worked 5 stitches from the left edge. Right: increases worked 1 stitch from the left edge

2.2.5 Final Measurement and Fit Visualization

Once all the nodes are placed at the correct x-y coordinates, the visual schematic forms organically. However, the goal of the schematic is also to expose measurements that are important for the finished garment. To accomplish this, StitchDraft uses a drag-and-measure tool to highlight groups of stitches and gather information such as how many stitches span the box, how many rows are in the box, and the dimensions of the stitches within the box. All of this can be calculated from stitch count and gauge. Additionally, since all parts of the schematic's dimensions correspond to real-

world dimensions, StitchDraft allows the schematic to be overlaid onto a 2D torso silhouette of various sizes (Figure 2.4).

StitchDraft's torso is created in the backend where an SVG generator turns either standard sizes or custom measurements into a 2D silhouette SVG. The torso is built in inches and uses a schematic overlay because both are mapped to the same unit system, thereby maintaining their relative size. Lastly, StitchDraft allows the torso silhouette to be modified based on the user's desired ease. If the user intends for a positive ease, the torso will expand horizontally by the specified amount, and if the user intends for negative ease, the torso will shrink horizontally. This way, the user can see if the dimensions of the garment line up with their intended ease.

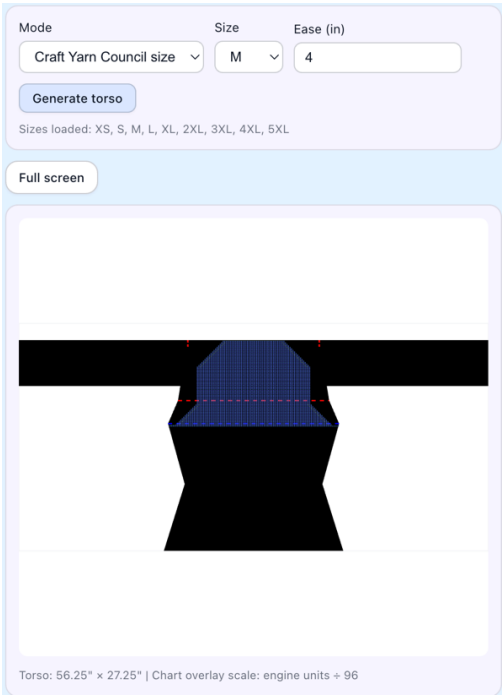


Figure 2. 4 StitchDraft schematic overlaid on a standard size M with 4 inches of positive ease

2.3 Implementation

2.3.1 Technologies used

The single-page application frontend of StitchDraft was built with React, TypeScript, and Vite to render a Blockly workspace for pattern authoring and schematic generation, a node-link stitch layout, and an SVG-based Torso overlay view. Google's drag-and-drop functional block visual editor, Blockly, was used as the visual programming interface to define custom knitting blocks,

organize them into toolbox categories, and convert them into a JSON IR that the backend understands. The intermediate representation (IR) was built using TypeScript and Pydantic. StitchDraft's front end defines a TypeScript interface for "KnittingIR" and "ChartProgram" commands. The backend uses Pydantic, a data validation and settings management library for Python, to define matching models and a discriminated union, a type-safe pattern, so the IR could be validated on input. FastAPI was used as the backend API. The API `POST /preview` endpoint takes the IR, runs it through the backend engine, and returns a schematic preview. The API registers `GET /torso/sizes` and `POST /torso/svg` for torso SVG services. The API also handles join dependency ordering before executing charts. StitchDraft used Python for the engine library and torso generator, to implement the DSL interpreter, maintain knitting state during pattern execution, and compute stitch geometry and proportional body measurements for schematic overlays.

2.3.2 DSL Implementation

The StitchDraft DSL consists of two layers: a command-level command language and a pattern-level stitch language. The command-level DSL represents garment construction as a sequence of typed commands, while the pattern-level DSL describes the stitch operations performed within an individual row or round. Both layers are represented as structured data, not executable code, and are interpreted by the backend engine.

The command-level DSL defines common operations used in knitting patterns. Each chart is represented as a program consisting of metadata and an ordered sequence of commands. These commands represent operations such as casting on stitches, adding rows or rounds, repeating rows or rounds, placing markers, managing stitches on hold, and joining charts. On the frontend, the DSL is defined using TypeScript union types that outline the allowed command operations and their parameters. Each command contains an operation identifier (op) and the fields required for that operation. For example, the cast-on command may specify the number of stitches to cast on, an add row may specify the pattern string to work in that row, or a place marker may specify the position to place the marker. All of the charts built in the workspace are grouped together within the intermediate representation (IR) object and form the complete DSL program transmitted to the backend.

To write the DSL, the user creates pattern programs using Blockly, a structured editor for constructing the DSL program. Each block corresponds to a specific DSL command, such as `add row` or `repeat rounds`. A compilation step traverses the block structure, extracts block parameters, and constructs the corresponding command objects, turning the block workspace into the DSL IR. The result is serialized as a JSON and sent to the backend for execution. This compilation ensures that the visual block program is represented in a way that adheres to the DSL grammar.

The backend uses Pydantic models that mirror the frontend TypeScript types to validate the incoming DSL program. Each DSL command corresponds to a Python model. The complete program is validated to ensure that the commands contain the correct fields and data types before execution. Once validated, each DSL command is interpreted in order by the backend engine. As each command is processed, the engine updates the internal knitting state and maintains the changing state of the knitted fabric. Because the DSL is represented as structured data and not executable code, the engine only executes predefined operations.

Row and round commands express stitch instructions using the pattern-level DSL. These pattern strings describe stitch operations using tokens such as knit (k), purl (p), increase (inc), decrease (dec), bind-off (bo), and marker operations (pm, sm, rm). The backend engine includes a pattern parser that expands these pattern expressions into stitch-by-stitch sequences based on the engine's current knitting state, including the working side of the fabric, marker position, and number of available stitches. Expanding the pattern can also result in marker updates tracked by the engine. The expanded pattern is used to calculate the visual position of each stitch, thereby determining the schematic's overall layout.

2.3.3 Layout and Coordinate System

StitchDraft’s backend engine converts expanded knitting instructions into a spatial representation of stitches that together create a schematic. The layout algorithm combines gauge-derived spacing, topological relationships between stitches, and a normalization step to compute coordinates for each stitch node.

The engine contains an internal coordinate system in which 96 units correspond to one physical inch, aligning with the common SVG and CSS assumption of 96 pixels per inch. This

convention allows the engine to represent stitch coordinates in a scale that corresponds directly with physical measurements. The torso generator, used for fit visualization, is also defined in inches and is converted by the frontend using a simple scaling factor:

$$x_{inches} = \frac{x_{engine}}{96}, \quad y_{inches} = \frac{y_{engine}}{96}$$

This shared scale allows stitch layouts and torso silhouettes to be rendered within the same spatial coordinate system, enabling garment-to-body overlays.

Knitting gauge is defined as the number of stitches and rows within a 4-inch square of fabric. StitchDraft uses these values to compute the horizontal spacing between stitches and the vertical distance between rows. The engine computes spacing values in engine units:

$$\text{stitch spacing} = \frac{4 \times 96}{sts}$$

$$\text{row height} = \frac{4 \times 96}{rows}$$

These formulas transform physical stitch and row dimensions into the coordinate system using the known conversion of 96 engine units to one inch. The row height is used to place rows vertically. All stitches in the same row share the same vertical coordinate, and as additional rows are constructed, they stack vertically using the spacing determined from the gauge. Given a row index *r*, with base of 1, the vertical coordinate of that row is:

$$y = (r - 1) \times row\ height$$

Horizontal placements of stitches are calculated using an anchor-based layout system that maps each stitch in a new row to the stitch it was produced from in the previous row. When a row is created, the engine computes the initial horizontal positions, called anchors. The anchors represent the horizontal location of each stitch, based solely on its relationship to and placement of stitches in the previous row. For right-side (RS) rows, stitches are processed from the leftmost stitch of the previous row to the rightmost stitch. For wrong-side (WS) rows, the interpretation is mirrored, and when looking at the fabric from the RS, the stitches appear to be worked from right to left. To calculate the correct stitch positions, the engine will flip the order of stitches in a WS

row in flat knitting so both sides can be treated as if they are flowing left to right. The first cast on row places stitches evenly spaced using the stitch spacing calculation. For subsequent rows, different stitch operations produce different anchor behaviors. Regular stitches, such as knit or purl, have anchors equal to the horizontal coordinate of the stitch that it consumes in the previous row. Increases don't consume a stitch from the previous row but instead are formed between two existing stitches in the previous row. Increase stitches have an anchor equal to the midpoint of the last consumed stitch in the previous row and the next stitch to be consumed in the previous row:

$$x = \frac{x_a + x_b}{2}$$

where x_a is the x coordinate of the last consumed stitch, and x_b is the x coordinate of the next stitch to be consumed. Decrease stitches consume two stitches from the previous row, merge them into one, and have anchors equal to the midpoint of the consumed stitches:

$$x_i = \frac{x_c + x_{c+1}}{2}$$

where xc and xc+1 are the two adjacent stitches in the previous row that are being consumed to produce one stitch in the current row.

The anchors may produce uneven spacing because some stitches are placed between the x-coordinates of stitches in the previous row. Each stitch occupies about the same amount of space, as determined by the stitch spacing, so uneven spacing must be redistributed to normalize stitch spacing within the row while preserving the overall horizontal position implied by the anchors. First, to normalize the anchors, the average anchor position is computed:

$$c = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} a_i$$

where a_i are the anchor coordinates. Next, a uniformly spaced row is generated and centered at c:

$$x_i = c + s\left(i - \frac{\mathrm{n\text{-}1}}{2}\right)$$

where *s* is equal to *stitch spacing*. The result is a uniformly spaced row of stitches centered at the position determined by the anchors. The anchoring and normalization process preserves the garment's true shape rather than relying on a grid-based chart.

2.3.4 Rendering and Visualization

The backend produces a node-link stitch layout of spatial positions for stitches that together form a schematic of the knitted fabric, which serves as a single source of truth and is rendered by the frontend. Additionally, the torso overlay visualization, which places the generated schematic onto a proportional body silhouette, is also supported. The stitch positions remain consistent across both views because the visualizations use the same node data and coordinate system.

The backend engine produces an object containing structures that store stitch nodes and their relationships. Each node represents a stitch and includes a stitch id, a stitch type, a row, and spatial coordinates expressed in units where 96 units correspond to one physical inch. The backend serializes this data for the frontend. The API returns this information as a JSON containing node coordinates, link relationships, row metadata, stitch counts, markers, and validation messages, and the frontend renders the schematic directly from the provided node positions.

In the schematic view, the frontend renders stitch nodes inside an SVG element. The SVG coordinate system is aligned with the backend engine coordinate system so that one unit in the SVG viewBox equals one engine unit. To frame the schematic, the frontend computes a bounding box of all the nodes by finding the min and max x coordinates and min and max y coordinates. Then, padding is added to ensure the full schematic is visible. Each stitch is drawn as a circle positioned at the x-y coordinates determined by the engine. Stitch appearance is determined by stitch type: knit and cast-on stitches are blue, purl stitches are purple, increases are green, decreases are orange, and bind-offs are dark grey. The circles are drawn inversely proportional to stitch spacing. For high-gauge, meaning many sts/rows per 4-inch square, the engine calculates a smaller spacing and the node radii and padding decrease. For low-gauge charts, radii and padding increase so nodes don’t look tiny.

The schematic view enables two primary interactions: panning, which allows the entire schematic to be dragged, and measuring, in which the user can draw a rectangular region to

compute stitch measurements in centimeters and inches within it. When panning is enabled, the rendered schematic is converted into a snapshot image, displayed as a single SVG image element, and placed inside a translated group element, allowing the user to drag the schematic within the viewport while preserving the coordinate system. Creating a snapshot allows the schematic to be moved quickly, rather than translating each node individually, and provides a more responsive interface when placement control is necessary. The measuring tool allows users to estimate garment dimensions in different regions of the schematic. When the user draws a highlighted rectangular region over the schematic, the system identifies all nodes within the selection and computes the number of rows, the maximum number of stitches per row, and the physical width and height of the selected region. The physical dimensions of the highlighted box are calculated using the conversion factor of engine units to inches:

$$width_{inches} = \frac{width_{units}}{96}$$

$$height_{inches} = \frac{height_{units}}{96}$$

The panning functionality was created to support garment fit visualization and overlay onto a two-dimensional torso silhouette. The backend system constructs the torso as an SVG, expressed in physical inches using body measurements such as torso width, height, and shoulder width, either from standard measurements or from user-entered values. The torso generator uses these measurements to compute a key set of reference points to outline the figure's shoulders, chest, waist, hips, and arms. The points are connected using Bézier curves and straight segments. Because the torso and the schematic use the same unit system, the schematic can overlay onto the torso, and the comparison between the two remains meaningful.

2.3.5 System Integration

The StitchDraft system follows a client-server architecture, in which the frontend handles user interaction to build knitting programs and visualizations, while the backend executes the knitting programs and generates the structured data used for rendering.

The system contains a frontend application implemented in React and TypeScript, the backend API server is implemented in Python using FastAPI, and the backend engine is

implemented as a Python library. The frontend communicates with the backend through HTTP requests that include JSON data. The backend API communicates with the backend engine through in-process function calls, as the engine is imported directly as a Python library.

The frontend automatically compiles and converts the block structure into the IR format when modifications are made to the Blockly workspace. A reactive effect then monitors changes to the compiled program, and when the program is valid, the IR is sent to the backend via the `POST /preview` HTTP endpoint. Preview requests are debounced so that the backend is queried after a short delay following user input, preventing excessive network requests during rapid editing. If the engine successfully processes the program, the returned data is stored in the application and is passed to the visualization components. If the program fails, an error message is shown in the interface. The backend engine calculates all node and link data, and rendering is handled purely in the presentation layer. When a user requests a torso overlay, the frontend sends a request to `POST /torso/svg` and displays the result in the application. The frontend interacts with the API endpoints through a dedicated client module that encapsulates HTTP requests and ensures that the frontend and backend share consistent data types.

The backend API validates preview requests using Pydantic models that represent the IR structure, ensuring that the command types and parameters conform to the DSL specification before execution. Additionally, the join operations are analyzed to create a directed dependency graph to check for cycles. If a cycle is detected, the request is rejected, and an error message is shown to the user describing the problematic join command. After the backend engine executes, the API returns the assembled preview as a JSON. The backend engine is integrated as a standalone Python library, independent of the web application. DSL commands are executed sequentially through the library's services. Because the engine operates independently of the API and frontend, it can be tested and reused in other environments.

CHAPTER 3
SYSTEM EVALUATION

3.1 Evaluation Goals

StitchDraft was evaluated according to three main questions:

1. Can StitchDraft correctly expand instruction-level knitting patterns into explicit stitch sequences while maintaining accurate stitch counts?
2. Can StitchDraft represent garment construction operations that modify stitch structure, such as stitches on hold and chart joins?
3. Can StitchDraft generate meaningful garment schematics from stitch structure and gauge?

These questions evaluate the pattern expansion state, the system's ability to represent structural garment operations, and the conversion of stitch layouts into correct spatial schematics.

3.2 Pattern Interpretation and Pattern Expansion

3.2.1 Stitch Count Correctness

StitchDraft supports four types of stitches: knit, purl, increase, and decrease. Knit and purl stitches both consume one stitch and produce one stitch. Increase stitches consume zero stitches and produce one stitch, and decrease stitches consume 2 stitches and produce one stitch. Each increase adds a stitch to the expected stitch count, and each decrease removes a stitch from the expected stitch count.

Pattern	Expected Stitches	System Result	Schematic
`cast on start` 10 `add row pattern` k10	10	10	

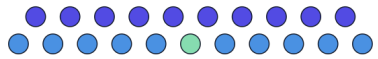
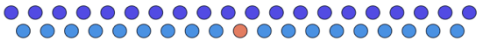
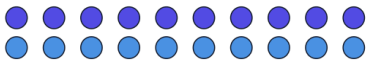
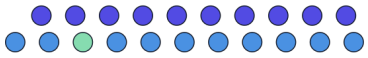
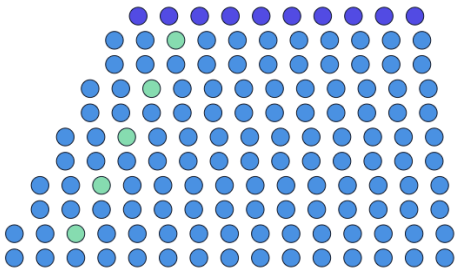
`cast on start` 10 `add row pattern` k5, inc, k5	11	11	
`cast on start` 20 `add row pattern` k9, dec, k9	19	19	
`cast on start` 10 `add row pattern` repeat(k1)	10	10	
`cast on start` 10 `add row pattern` k2, inc, repeat(k1)	11	11	
`cast on start` 20 `add row pattern` k2, dec, repeat(k1)	19	19	
`cast on start` 10 `repeat rows` 5 `times patterns` [k2, inc, repeat(k1)] [repeat(p1)]	15	15	

Table 3. 1 Evaluation of stitch count

3.2.2 Repeat Expansion

Repeat expansion involves repeating a section of stitches as many times as fits within the number of existing stitches in a row or row segment. Only full repeats are included, meaning if a row's

stitch count is not a multiple of the repeat stitch count, the entire row will not automatically be filled. The number of unworked stitches will be equal to the total number of stitches modulo the number of stitches in the repeat.

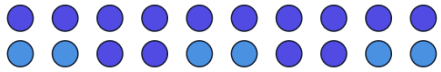
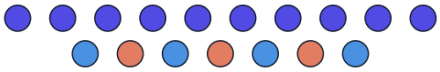
Pattern	Expected Row/Round	System Result	Schematic
`cast on start` 10 `add row pattern` repeat(k2,p2), k2	k, k, p, p, k, k, p, p, k, k	k, k, p, p, k, k, p, p, k, k	
`cast on start` 10 `add row pattern` k1, repeat(dec, k1)	k, dec, k, dec, k, dec, k	k, dec, k, dec, k, dec, k	

Table 3. 2 Evaluation of repeat expansion

3.2.3 Marker Segmentation

Markers segment a row of stitches and provide a natural stopping point. If a series of stitches is written before a slip-marker ("sm"), these stitches would appear physically before the marker's index. Increases and decreases affect the marker's indexed position as they would in physical knitting. If increases occur before a stitch marker, there is an additional stitch to the left of the marker and the marker's RS index is shifted up by one; if an increase occurs after a marker, the RS index is unaffected but the index on the WS is shifted up one. For decreases the indexes shift down.

Pattern	Expected Row/Round	System Result	Schematic
`cast on start` 10 `Place marker side` RS 4 `Place marker side` RS 6	k, k, k, k, inc, k, k, inc, k, k, k, k	k, k, k, k, inc, k, k, inc, k, k, k, k	

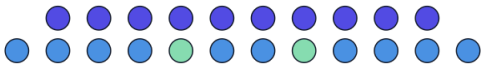
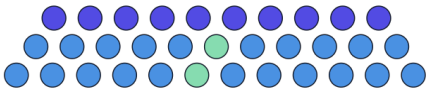
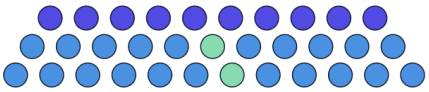
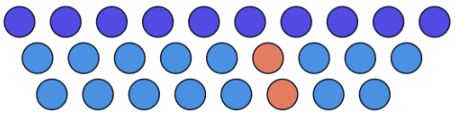
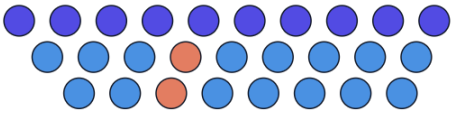
`add row pattern` repeat(k1), inc, sm, repeat(k1), sm, inc, repeat(k1)			
`cast on start` 10 `Place marker side` RS 5 `repeat rounds 2 times patterns` [repeat(k1), sm, inc, repeat(k1)]	[k, k, k, k, k, inc, k, k, k, k, k] [k, k, k, k, k, inc, k, k, k, k, k, k]	[k, k, k, k, k, inc, k, k, k, k, k] [k, k, k, k, k, inc, k, k, k, k, k, k]	
`cast on start` 10 `Place marker side` RS 5 `repeat rounds 2 times patterns` [repeat(k1), inc, sm, repeat(k1)]	[k, k, k, k, k, inc, k, k, k, k, k] [k, k, k, k, k, k, inc, k, k, k, k, k]	[k, k, k, k, k, inc, k, k, k, k, k] [k, k, k, k, k, k, inc, k, k, k, k, k]	
`cast on start` 10 `Place marker side` RS 5 `repeat rounds 2 times patterns` [repeat(k1), sm, dec, repeat(k1)]	[k, k, k, k, k, dec, k, k, k] [k, k, k, k, k, dec, k, k]	[k, k, k, k, k, dec, k, k, k] [k, k, k, k, k, dec, k, k]	
`cast on start` 10 `Place marker side` RS 5 `repeat rounds 2 times patterns` [repeat(k1), dec, sm, repeat(k1)]	[k, k, k, dec, k, k, k, k, k] [k, k, dec, k, k, k, k, k]	[k, k, k, dec, k, k, k, k, k] [k, k, dec, k, k, k, k, k]	

Table 3. 3 Evaluation of marker segmentation

3.3 Structural Stitch Management

3.3.1 Stitches on Hold

Placing stitches on hold means they can be excluded from the row without binding them off. When stitches are taken off hold and restored to the needle they can be worked again. When stitches are placed back on the needle, the user can decide what side is worked first. Multiple groups of stitches can be placed on hold at the same time using different name identifiers.

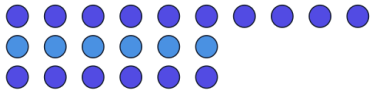
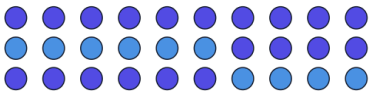
Pattern	Expected Rows/Round	System Result	Schematic
`cast on start` 10 `add row pattern` k6 `place on hold name` right `add row pattern` repeat(k1)	[k,k,k,k,k,k] [k,k,k,k,k,k]	[k,k,k,k,k,k] [k,k,k,k,k,k]	
`cast on start` 10 `add row pattern` k6 `place on hold name` right `add row pattern` repeat(k1) `place on hold name` left `place on needle from hold` right `join side` RS `add row pattern` repeat(p1) `add row pattern` repeat(p1)	[p,p,p,p] [p,p,p,p]	[p,p,p,p] [p,p,p,p]	

Table 3. 4 Evaluation of placing stitches on hold and on needles

3.3.2 Chart Joins

Two charts can be joined, just like two pieces of knitted fabric can be joined. The joined charts will contain the sum of stitches from the last row of both charts. Charts are given names that can be used to identify which chart is being joined. That chart containing the join operation will be visually on the left, as the yarn is still connected to this chart and will knit these stitches first, and the chart joined by name will be visually on the right.

Pattern	Expected	System	Schematic
Chart 1: `cast on start` 10 `repeat rows 2 times patterns` [repeat(k1)] Chart 2: `cast on start` 10 `repeat rows 2 times patterns` [repeat(k1), inc, k1] `add row pattern` repeat(k1) `cast on additional` 1 `join chart` Chart 1 `work right with pattern` repeat(k1)	23 sts Increases on left	23 sts Increases on left	

Table 3. 5 Evaluation of joining charts

3.4 Schematic Geometry Validation

3.4.1 Gauge-Based Measurement Validation

The gauge determines how close together the stitches are horizontally and vertically. Higher gauge numbers correspond to more stitches in one inch, and lower gauge numbers result in fewer stitches per inch. The system measuring tool can be dragged over the rendered schematic to produce measurements.

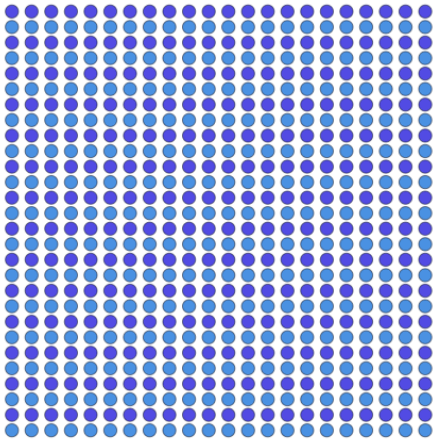
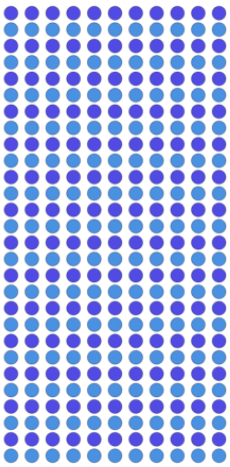
Gauge	Sample Dimensions	Expected Dimensions	System result	Schematic
22 sts x 28 rows	22 sts x 28 rows	4 in x 4 in	4 in x 4 in	
11 sts x 14 rows	11 sts x 28 rows	4 in x 8 in	4 in x 8 in	

Table 3. 6 Evaluation of Gauge-Based Measurement

3.4.2 Effect of Increase and Decrease Placement

Increases and decreases in different positions produce different shaping effects, even when they yield the same stitch counts. Increases and decreases closer to one edge have a greater effect on that edge, while shaping stitches closer to the center cause stitches to spread out on both sides.

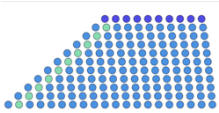
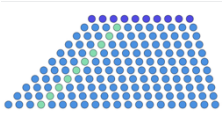
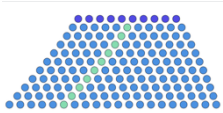
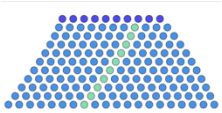
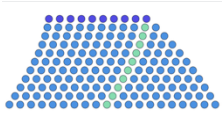
Increases 1 stitch from edge	Increases 3 stitches from edge	Increases 5 stitches from edge	Increases 7 stitches from edge	Increases 9 stitches from edge
				

Table 3. 7 Comparison of increase stitches placed in different positions, starting with 10 stitches and 10 total increases

3.4.3 Torso Overlay Alignment

The knitting pattern "Holiday Slipover" [7] (see Figure 3.1) was translated into a StitchDraft schematic (see Figure 3.2). The built size is XS, and the overlaid torso is also a standard XS. The images of the garment closely resemble the shape and fit shown in the schematic.

Figure 3. 1 Finished garment from the Petite Knit Holiday Slipover

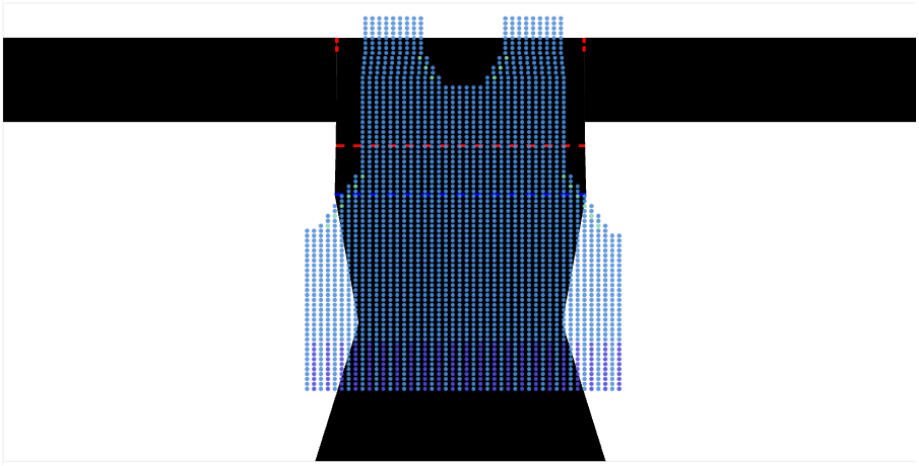


Figure 3. 2 StitchDraft 2D schematic of the Holiday Slipover in size XS on torso size XS

3.5 Comparison With Existing Tools

StitchDraft is, to our knowledge, the only pattern-driven system capable of constructing a full garment schematic directly from pattern instructions.

Feature	StitchDraft	Stitch Fiddle	Knotty	StitchMaps	Design-a-Knit
Pattern-driven modeling	✓	✗	partial	✓	✗
Stitches on hold	✓	✗	✗	✗	✗
Chart joins	✓	✗	✗	✗	✗
Automatic schematic generation (including measurements)	✓	✗	✗	✗	✓
Body overlay visualization	✓	✗	✗	✗	✓

Table 3. 8 StitchDraft vs existing tool feature comparison

CHAPTER 4
CONCLUSION

4.1 Results

This project evaluated StitchDraft's ability to interpret instruction-level knitting patterns and create structured garment schematics. The evaluation of the tool examined whether the system correctly expands pattern instructions and maintains accurate stitch counts, whether it can represent structural garment operations, such as stitches placed on hold and chart joins, and whether the resulting stitch structure can be converted into meaningful schematic geometry. This evaluation showed that StitchDraft correctly interprets pattern instructions and maintains consistent stitch counts across rows. Pattern expansion experiments confirmed that increases, decreases, repeats, and marker placement produce the expected stitch sequences, further indicating that the pattern-level DSL and expansion reliably translate pattern instructions into specific stitch operations.

The system also successfully modeled garment construction operations. Examples, including stitches placed on hold and placed on the needle, and chart joins, showed the engine could maintain multiple stitch relationships and combine them when needed. This confirmed that the command-level DSL and backend engine can represent these commonly found garment construction patterns. Finally, the generated schematics were shown to represent the expected garment dimensions derived from the stitch counts and gauge, and the expected shape derived from the shaping stitch placements and the backend engine's coordinate geometry. The final schematic's comparison with the actual physical garment demonstrated that StitchDraft can translate instructions written in a hand-knit pattern into a structured, accurate garment schematic and visualization.

4.2 Future work

StitchDraft was built with a limited library of stitches and commands to show a proof of concept. This leaves ample room for other stitches or techniques that knitters may want to visualize in their schematics. These include stitches such as brioche increases, which consume one stitch and produce three; cables, which rotate stitch positions to create a visual twist; and colorwork, where different stitches are knit with different-colored yarn to produce a design. Additionally, other

shaping techniques, such as German short rows, that do not involve increases or decreases, could be added. While StitchDraft contains geometric calculations that find accurate x-coordinates, it treats y-coordinates as fixed, incrementing them based on row height. In reality, some knitting stitch combinations can cause the stitches to curve up and down, and further geometry calculations could be explored. Knitting patterns also often include edge stitches that help form collars and armholes that are not currently supported by StitchDraft. The project could be extended to allow for knitting directions other than down to allow for such additions.

Currently, StitchDraft can be cloned from GitHub and run locally. However, to make the tool accessible to the knitting community, this tool could be hosted on a live webpage. To expand the user experience, StitchDraft could connect to a database, allowing users to log in and save schematics in their own libraries. Lastly, there is work that has been done in the space of 3D modeling knitted fabrics. StitchDraft could eventually be extended to offer 3D schematics and 3D garment body form try-ons.

REFERENCES

[1] Stitch Fiddle, “Stitch Fiddle: Create knitting and crochet charts,” https://www.stitchfiddle.com/, accessed Mar. 2026.

[2] T. Price, “Knotty: A domain-specific language for knitting charts,” https://github.com/t0mpr1c3/knotty, accessed Mar. 2026.

[3] Stitch Maps, “Stitch Maps: knitting charts drawn without grids,” https://stitch-maps.com/, accessed Mar. 2026.

[4] Soft Byte Ltd., “DesignaKnit knitting software,” https://www.designaknit.com/, accessed Mar. 2026.

[5] Google, “Blockly,” https://developers.google.com/blockly. Accessed: Mar. 2026.

[6] M. Resnick et al., “Scratch: Programming for all,” Communications of the ACM, vol. 52, no. 11, pp. 60–67, 2009.

[7] PetiteKnit, "Holiday Slipover," knitting pattern, https://www.petiteknit.com/en/products/holiday-slipover, accessed Mar. 2026.

APPENDIX

Figure A. 1 Stitch marker

Figure A. 2 Grid-based chart

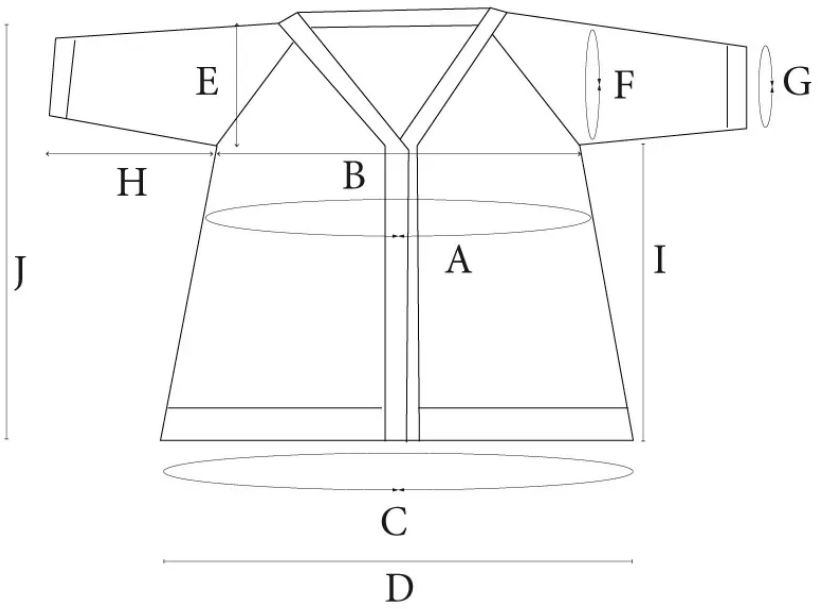


Figure A. 3 Garment Schematic