CROCHETBENCH: CAN VISION-LANGUAGE MODELS MOVE FROM DESCRIBING TO DOING IN CROCHET DOMAIN?

Anonymous authors
Paper under double-blind review

ABSTRACT

We present CrochetBench, a benchmark for evaluating the ability of multimodal large language models to perform fine-grained, low-level procedural reasoning in the domain of crochet. Unlike prior benchmarks that focus on high-level description or visual question answering, CrochetBench shifts the emphasis from *describing* to *doing*: models are required to recognize stitches, select structurally appropriate instructions, and generate compilable crochet procedures. We adopt the *CrochetPARADE DSL* as our intermediate representation, enabling structural validation and functional evaluation via execution. The benchmark covers tasks including stitch classification, instruction grounding, and both natural language and image-to-DSL translation. Across all tasks, performance sharply declines as the evaluation shifts from surface-level similarity to executable correctness, exposing limitations in long-range symbolic reasoning and 3D-aware procedural synthesis. CrochetBench offers a new lens for assessing procedural competence in multimodal models and highlights the gap between surface-level understanding and executable precision in real-world creative domains.

1 INTRODUCTION

Procedural crafts such as crochet present a distinctive frontier for multimodal learning. Unlike traditional captioning or recipe datasets (Li et al., 2024; Hu et al., 2023; Mohbat & Zaki, 2024), crochet patterns intertwine three interdependent modalities: (i) **structured symbolic language**, where stitch abbreviations and counts define a precise grammar of construction; (ii) **long-form natural language**, which provides contextual guidance such as materials and sizing; and (iii) **visual evidence**, including photographs of completed objects and motif diagrams. Success requires not just alignment across modalities but step-wise reasoning that preserves *procedural fidelity*, making the challenge closer to *program synthesis* than generic description.

Crochet also offers a unique testbed for **3D-aware reasoning**. Each stitch encodes both local geometry and global connectivity, forming a topological structure that must be preserved across steps. Generating or interpreting patterns thus demands reasoning over how sequential operations accumulate into volumetric form. In effect, crochet couples symbolic instruction following with embodied spatial reasoning, cultivating abilities essential for domains where language must ground into physical tasks.

Despite the rapid growth of multimodal benchmarks (Fu et al., 2024; Li et al., 2023a; Zhang et al., 2025; Yue et al., 2024), existing datasets have largely focused on description or grounding. COCO (Lin et al., 2015) catalyzed captioning research, TextCaps (Sidorov et al., 2020) extended it to text-in-the-wild, and Recipe1M (Marin et al., 2019) explored cross-modal cooking instructions. While recipes also involve multi-step procedures, validating correctness typically requires real-world execution, making large-scale evaluation slow and resource-intensive. Crochet, by contrast, provides a symbolic domain where outputs can be automatically verified through DSL compilation, enabling scalable and efficient study of step-wise reasoning. For a more detailed survey, see Appendix A. Yet these benchmarks stop short of testing whether models can follow symbolic grammars, respect numerical and spatial constraints, and produce outputs that are *executable*. Current systems can describe, but not reliably *do*.

CrochetBench fills this gap by centering evaluation on **instructional fidelity**: can models not only recognize and generate, but also output step-wise, compilable instructions that respect symbolic, numerical, and topological structure? Each example in CrochetBench is a multimodal package—structured JSON metadata (stitch inventories and abbreviations), full-text procedures with rows/rounds and conditionals, and paired images of finished objects and motifs. Crucially, CrochetBench is paired with *CrochetPARADE* (Tassev, 2025), a domain-specific language (DSL) enabling executable evaluation, where natural language instructions are translated into compilable code enforcing geometric and topological coherence.

Our contributions are fourfold: (1) **CrochetBench**, the first executable benchmark for procedural crafts, unifying symbolic, textual, and visual modalities with evaluation protocols emphasizing procedural fidelity and 3D-aware reasoning; (2) a **comprehensive task suite** spanning recognition, comprehension, generation, and DSL translation; (3) integration of **CrochetPARADE into an executable pipeline**, enabling scalable, automated verification of outputs—unlike domains such as cooking, which require real-world execution—thereby shifting evaluation from surface similarity to procedural fidelity; and (4) **baseline analyses** of state-of-the-art VLMs/MLLMs, revealing systematic weaknesses including hallucinations, captioning bias, and structural artifacts.

Taken together, CrochetBench opens a new direction for multimodal research: moving beyond describing what we see, toward generating executable procedures that respect symbolic grammar, numerical accuracy, and topological coherence—paving the way for models that can reason in structured 3D spaces.

2 Dataset Description

CrochetBench is a large-scale, structured benchmark consisting of 6,085 crochet patterns spanning 55 distinct project categories. It was constructed by collecting publicly available patterns from the Yarnspirations website[1], a widely used repository for fiber arts. The raw patterns were originally formatted as PDF documents, which were parsed and normalized using a GPT-4o-mini-based conversion pipeline. This process extracted and standardized key fields including pattern metadata, materials, measurements, gauge, abbreviations, and full step-by-step instructions. Each entry was converted into a machine-readable JSON object with a consistent schema, and 98.77% of patterns include an associated product image, enabling multimodal supervision.

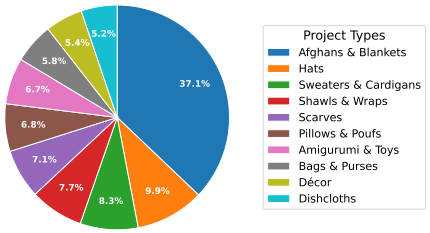

Figure 1: Distribution of the top-10 most common project types in **CrochetBench**.

The dataset supports diverse real-world crochet practices, with project types ranging from simple accessories to complex garments. Figure 1 lists the ten most common categories by frequency. The majority of patterns belong to a small number of dominant types—Afghans and Blankets alone account for over one-quarter of the dataset. More details can be found at Table **??**.

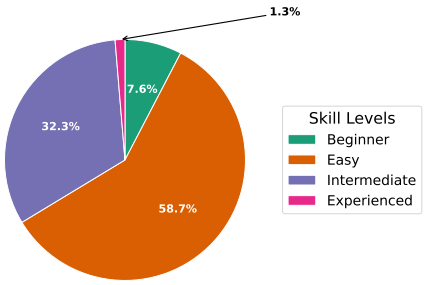

Figure 2: Skill level distribution across the **CrochetBench** dataset. Note that the "Experienced" slice (1.3%) is annotated externally due to its small size.

Each pattern is labeled with one of four primary skill levels—*beginner*, *easy*, *intermediate*, or *experienced*. This allows for stratified evaluation across complexity tiers. Figure 2 shows the skill level distribution, which is strongly skewed toward beginner-friendly content. Only one pattern (0.02%) is missing a skill level label. More detail can be found at Table **??**

[1] https://www.yarnspirations.com/collections/patterns

Instructional complexity varies substantially across patterns. The number of characters in each instruction ranges from 20 to over 30,000, with a mean of 3,216 and a median of 2,453. Abbreviation counts (i.e., unique stitch tokens per pattern) range from 1 to 31, with an average of 10.6. These statistics are summarized in Appendix B.1. We observe a clear correlation between skill level and instruction length: beginner patterns tend to be short and use fewer abbreviations, while experienced patterns are significantly longer and more symbolically dense.

In addition to symbolic complexity, the dataset contains 3,143 abbreviation instances mapped to 789 unique standardized stitch tokens. This lexical mapping enables tasks such as vocabulary translation, sequence generation, and instruction validation. Beyond raw instructions, the structured schema also records rich metadata, including gauge, hook size, yarn weight, and measurements. A representative dataset entry is provided in Appendix B.1.

Overall, CrochetBench provides a rich resource for multimodal modeling, symbolic reasoning, and structure-aware generation. Its coverage across diverse categories and complexity levels enables broad benchmarking of both open-ended generation and instruction fidelity tasks.

	Total Patterns	**Image Coverage**	**Avg. Instr. Length**	**#Project Types**
CrochetBench	6,085	98.77%	3,216 characters	55

Table 1: Overall statistics of the CrochetBench dataset.

3 EXPERIMENT

We empirically evaluate CrochetBench by defining a set of structured tasks and benchmarking a diverse pool of multimodal large language models. Our experiments are designed to test models across progressively challenging stages, from low-level recognition to high-level executable synthesis.

Benchmark Tasks. Table 2 summarizes the four evaluation tasks in CrochetBench, which progress systematically from recognition to comprehension, generation, and ultimately executable synthesis. **Task A (Stitch Recognition)** evaluates a model's ability to detect symbolic primitives in crochet images, establishing the foundation for multimodal perception. **Task B (Instruction Selection)** requires models to align visual evidence with candidate textual instructions, thereby testing multimodal grounding and fine-grained comprehension. Unlike conventional description tasks, the candidates are procedural steps rather than captions, and correct selection often requires reasoning about how local steps contribute to the final product. **Task C (Instruction Generation)** advances from comprehension to open-ended production, challenging models to generate natural-language procedural instructions that are both perceptually grounded and linguistically faithful to domain conventions. Evaluation here emphasizes lexical and symbolic fidelity, but does not directly capture structural validity. **Task D (Instruction-to-DSL Translation)** addresses this gap by requiring models to output a compilable program in a domain-specific language (DSL). The step-level variant tests local semantic grounding, while the project-level variant demands global structural consistency across the entire pattern. Compilation-based evaluation directly measures executable faithfulness, ensuring that generated instructions are not only linguistically plausible but also structurally sound. Together, these tasks form a structured evaluation ladder that moves from perceptual recognition to programmatic execution, probing both low-level perception and high-level symbolic reasoning.

Model Selection. We evaluate a diverse set of vision–language models (VLMs) spanning both open-source and closed-source families. The open-source group includes Salesforce BLIP-2 Flan-T5 XL (3B), a perception-focused baseline widely used in image–text tasks; Google Gemma 3 (4B) and Qwen2-VL (7B), two recent models trained with large-scale multimodal alignment; and DeepSeek-VL (7B), a larger open-source model designed for stronger vision–language reasoning. On the closed-source side, we assess GPT-4o, Gemini 2.5 Flash-Lite, and Claude Sonnet 4, which represent state-of-the-art commercial VLMs that push the limits of multimodal reasoning. This

Table 2: Overview of benchmark tasks in CrochetBench. Tasks progress from recognition to comprehension, generation, and executable synthesis.

ID	Ability Tested	Task	Evaluation Metrics	Test Size
A	Recognition	Stitch Recognition	F1, Precision, Recall	6,009 (CrochetBench-A)
B	Comprehension	Instruction Selection	Accuracy	6,003 (CrochetBench-B)
C	Generation	Instruction Generation	BLEU, ROUGE, ChrF	6,009 (CrochetBench-C)
D	Formalization	Instr.-to-DSL (Step)	Valid Pattern Rate	119 (CrochetBench-D_{step})
		Instr.-to-DSL (Project)	Valid Pattern Rate	100 (CrochetBench-D_{proj})

selection spans perception-heavy baselines to general-purpose multimodal reasoning systems, enabling us to evaluate capabilities across both research-grade and production-grade settings.[2]

3.1 Task A: Stitch Recognition

The first task evaluates a model's ability to identify crochet stitch types from an image of a finished product. We construct **CrochetBench-A**, a subset of 6,009 examples from the full benchmark, where each product image is paired with ground-truth stitch annotations. These labels are derived from the official pattern instructions and normalized into a standardized set of stitch abbreviations (e.g., `sc`, `hdc`, `dc`) to ensure consistency across patterns. Unlike standard image classification, this is a *multi-label prediction problem*: multiple stitches may co-occur within the same image, often with subtle visual differences in texture and geometry. This task therefore probes fine-grained visual grounding of structured crochet semantics.

Evaluation. For each example, we compute overlap between the predicted and reference stitch sets. True Positives (TP) are stitches correctly predicted; False Positives (FP) are stitches predicted but not in the reference; and False Negatives (FN) are stitches in the reference but missed by the model. From these counts, we compute precision (fraction of correct predictions among all predictions), recall (fraction of ground-truth stitches recovered), and F1 score (harmonic mean). Metrics are averaged across examples to provide overall performance. This formulation rewards models that recover all present stitches while avoiding spurious predictions.

Accurate stitch recognition is foundational for the benchmark, as later tasks (e.g., instruction selection and instruction generation) depend on robust detection of stitch primitives.

Table 3: Evaluation results on the *Stitch Recognition* task. We report Precision, Recall, and F1. Best results are **bold**; second-best are underlined.

	Model	Size	Precision	Recall	F1
Open Source	Salesforce BLIP-2 Flan-T5 XL	3B	0.2953	0.2303	0.2250
	Google Gemma 3	4B	0.2054	0.1021	0.1265
	DeepSeek-VL	7B	0.5447	**0.7476**	0.6060
	Qwen2-VL	7B	0.5414	0.6974	0.5816
Closed Source	GPT-4o	–	0.6214	0.5939	0.5801
	Gemini 2.5 Flash-Lite	–	0.7449	0.4977	0.5683
	Claude Sonnet 4	–	**0.7861**	0.5312	**0.6094**

Results. Claude Sonnet 4 achieves the best overall F1 score, demonstrating strong precision in stitch recognition. Among open-source models, DeepSeek-VL performs best, with notably high recall, while Qwen2-VL offers a competitive balance. These results highlight the gap between commercial

[2]We use the term "VLM" broadly to include both traditional vision–language models (e.g., BLIP-2) and modern multimodal large language models (e.g., GPT-4o, Gemini, Claude).

VLMs and open-source alternatives, but also suggest that large open-source models are beginning to approach closed-source performance in fine-grained recognition.

3.2 TASK B: INSTRUCTION SELECTION

The second task evaluates whether a model can correctly align a finished crochet product image with its corresponding natural-language instruction. We construct **CrochetBench-B**, a subset of 6,003 examples, where each item consists of one correct instruction and three distractor instructions sampled from the same project category (e.g., hats, rugs). Distractors are carefully chosen to share structural motifs and vocabulary, thereby increasing difficulty and requiring fine-grained visual–textual alignment rather than reliance on superficial cues. The answer distribution across options is approximately uniform (A: 24.9%, B: 25.7%, C: 23.7%, D: 25.7%).

Evaluation. To enable scalable assessment, we formulate the task as a 4-way multiple-choice question (MCQ). The model must select one option (A–D), with exactly one correct answer. Predictions are extracted using a lightweight regex-based method that identifies explicit letter outputs (e.g., "A", "The answer is B"). If no parsable choice is found, the response is marked as unanswered. Accuracy is reported as the evaluation metric.

This task provides a controlled measure of visual grounding and semantic alignment between image content and procedural text, without requiring generative modeling. It highlights the challenge of distinguishing between subtle visual cues and domain-specific terminology, which is essential for bridging perception and structured instruction understanding.

Table 4: Evaluation results on the *Instruction Selection* task (4-way multiple choice). We report Accuracy. Best results are **bold**; second-best are underlined.

	Model	Size	Accuracy
Open Source	Salesforce BLIP-2 Flan-T5 XL	3B	0.2562
	Google Gemma 3	4B	0.2494
	DeepSeek-VL	7B	0.2892
	Qwen2-VL	7B	0.4196
Closed Source	GPT-4o	–	**0.5811**
	Gemini 2.5 Flash-Lite	–	0.5563
	Claude Sonnet 4	–	0.5739

Results. GPT-4o achieves the highest accuracy, closely followed by Claude and Gemini, all of which substantially outperform open-source VLMs. Among open-source models, Qwen2-VL is the strongest, while BLIP-2 and Gemma remain near chance level, underscoring the difficulty of visually grounded instruction matching in this domain.

3.3 TASK C: INSTRUCTION GENERATION

The third task evaluates a model's ability to generate natural-language crochet instructions directly from an image of a finished product. We construct **CrochetBench-C**, a subset of 6,009 examples, where each product image is paired with its full pattern. Unlike classification or selection, this task requires producing a multi-step sequence that follows domain-specific syntax, ordering, and stitch logic. Outputs are expected to resemble real-world crochet patterns, written line by line (e.g., "Rnd 1: ch 4, 6 sc in ring"). This setting challenges models to translate visual evidence into coherent procedural text that preserves both symbolic accuracy and structural consistency.

Evaluation. We evaluate generation quality using standard text-generation metrics and a domain-specific structural measure. BLEU and ROUGE-L capture lexical and n-gram overlap with the reference instructions. ChrF, computed over character n-grams, provides a finer-grained signal of similarity and is particularly suited to stitch abbreviations. While these metrics measure surface-level overlap, they do not directly test whether the generated instructions are *structurally consistent* (e.g., balanced stitch counts, valid round progression). To address this limitation, we complement

Task C with **Task D**, where compilation-based evaluation of DSL translations directly assesses executable consistency. Together, these tasks allow us to probe both linguistic fidelity and procedural correctness.

Table 5: Evaluation results on the *Instruction Generation* task. We report BLEU, ROUGE-L, and ChrF. Higher is better. Best results are **bold**; second-best are underlined.

	Model	Size	BLEU	ROUGE-L	ChrF
Open Source	Salesforce BLIP-2 Flan-T5 XL	3B	0.0021	0.0926	9.32
	Google Gemma 3	4B	0.0010	0.0329	5.17
	Qwen2-VL	7B	0.0160	0.2084	15.76
	DeepSeek-VL	7B	0.0133	0.1968	18.12
Closed Source	GPT-4o	–	0.0333	0.2353	23.80
	Gemini 2.5 Flash-Lite	–	**0.0482**	**0.2583**	**30.20**
	Claude Sonnet 4	–	0.0331	0.2516	22.95

Results. Gemini achieves the strongest overall performance across all metrics, substantially outperforming other closed-source models. GPT-4o and Claude follow closely, while open-source models lag behind with significantly lower BLEU and ChrF scores. This gap highlights the difficulty of generating structurally faithful crochet instructions, which requires models to capture both visual details and domain-specific procedural logic.

3.4 Task D: Instruction-to-DSL Translation

We construct two subsets for Task D: **CrochetBench-D$_{\text{step}}$** (119 items) and **CrochetBench-D$_{\text{proj}}$** (100 items). For **CrochetBench-D$_{\text{step}}$**, we manually annotate the correct CrochetPARADE DSL for the *previous context* of each pattern (a prefix of NL (natural language) –DSL pairs). The model is then given this prefix along with the next natural-language instruction, and must predict the corresponding DSL line. Predictions are evaluated by checking compilation validity with the CrochetPARADE validator. For **CrochetBench-D$_{\text{proj}}$**, we adopt a *few-shot prompting setup* rather than full manual annotation. Annotators provide a single reference program in CrochetPARADE as context, which is paired with a new natural-language instruction and its product image. The model must then generate a complete DSL program, and outputs are assessed by whether they compile fully or partially using the validator.

Design Rationale. Crochet patterns admit many valid DSL realizations (e.g., alternative groupings or equivalent constructs), meaning that no single gold reference is canonical. A reference-based metric could unfairly penalize models that produce semantically correct but structurally different programs. By instead relying on the validator, CrochetBench shifts evaluation away from surface-level string matching toward *functional executability*—the central criterion for crochet synthesis. This design is consistent with other program synthesis benchmarks, such as semantic parsing and SQL generation, where execution accuracy is often preferred over exact string match.

3.4.1 Step-Level Translation

In the step-level setting, the model receives a prefix of NL–DSL pairs and must generate the DSL line corresponding to the next natural-language instruction. This setup reflects an incremental synthesis process in which correctness depends on maintaining stitch-level consistency across steps. Since crochet patterns are inherently stateful, earlier context is critical for resolving constructs such as increases, repeats, and turning chains. To capture progression through a pattern, we sample 52 early examples (steps 1–2), 34 mid examples (steps 3–4), and 33 late examples (steps 5–6). Step-level inputs are formatted as:

$$\text{Prefix (NL–DSL pairs)} + \text{Next NL instruction} \quad \rightarrow \quad \text{Next DSL line.}$$

This formulation allows models to generate locally plausible DSL steps, but global correctness ultimately requires stronger contextual reasoning.

Evaluation. We evaluate models using **Compilation Success Rate (CSR)**, defined as the proportion of generated DSL outputs that compile successfully with the CrochetPARADE validator. This validator-based metric is a key strength of CrochetBench: unlike reference-based string matching, CSR directly measures whether generated programs are *executable*, rewarding functional correctness rather than surface similarity. To better understand systematic errors, we also conduct fine-grained error analysis across four categories: (1) syntax structure errors, (2) stitch definition errors, (3) labeling and reference errors, and (4) structural or formatting issues. A detailed taxonomy with illustrative examples is provided in the Appendix.

CrochetBench further supports multimodal verification, as CrochetPARADE programs can be rendered into simulated product images. These renderings can be compared with ground-truth product images using pretrained vision–language models (e.g., CLIP), enabling evaluation of both structural validity and visual faithfulness. While current model performance leaves ample room for progress, the combination of validator-based execution checks and prospective image-grounded verification establishes CrochetBench as a uniquely rigorous testbed for structured multimodal reasoning.

Table 6: Verification results across models, grouped by category.

Model	CSR (%)	Undef.	Br.	MRef	Other	Tot. Err.
Open Source						
Salesforce BLIP-2 Flan-T5 XL	4.2	29.8	48.2	20.2	1.8	114
Google Gemma 3	3.4	26.1	10.4	63.5	–	115
DeepSeek-VL	32.8	36.2	38.8	11.2	13.8	80
Qwen2-VL	35.3	42.9	39.0	1.3	16.8	77
Closed Source						
GPT-4o	36.1	39.5	43.4	–	17.1	76
Gemini 2.5 Flash-Lite	41.2	51.4	21.4	5.7	21.5	70
Claude Sonnet 4	52.1	45.6	28.1	–	26.3	57

Results. Step-level translation is locally tractable: models often produce syntactically valid lines, but compilation failures reveal semantic underspecification. Claude achieves the highest valid rate (52.1%), while DeepSeek-VL and Qwen2-VL are the strongest among open-source models. Errors are dominated by syntax (brackets, references) and undefined stitches, underscoring the difficulty of maintaining state consistency even when local syntax is correct.

3.4.2 Project-Level Translation

In the project-level setting, the model is provided with the complete crochet instruction in natural language together with the corresponding product image, and must generate an entire CrochetPARADE program. This variant is globally self-contained but considerably more challenging than the step-level task: models must track stitch states over long horizons, resolve ambiguities in natural language, and produce code that is both syntactically valid and semantically aligned with the final design. Image grounding plays a crucial role in disambiguating constructs such as repeated motifs, symmetry, and termination conditions.

Evaluation. We assess model outputs using two complementary metrics. The first is **Compilation Success Rate (CSR)**, and the second is **Partial Executable Rate (PER)**, which measures the average fraction of a program that compiles successfully before failure. While CSR captures all-or-nothing executability, PER provides a finer-grained view of structural alignment, offering credit to models that generate correct prefixes even if the full program does not compile. Error types are categorized using the same taxonomy as in the step-level evaluation.

Results. Project-level translation remains highly challenging. Qwen2-VL achieves the best valid rate (21.0%) and strong PER, surpassing all closed-source systems. DeepSeek-VL demonstrates robust partial executability despite a lower valid rate. By contrast, GPT-4o, Gemini, and Claude achieve lower scores, highlighting that even state-of-the-art VLMs struggle with long-range structural consistency in executable synthesis. Error analysis (Table 8) reveals that closed-source models often fail due to label and reference inconsistencies, while open-source models more frequently exhibit undefined stitches and syntax errors.

Table 7: Project-level verification results. We report **Compilation Success Rate (CSR)** and **Partial Executable Rate (PER)**. Best scores are **bold**, second-best are underlined.

	Model	Size	CSR (%)	PER (%)
Open Source	Salesforce BLIP-2 Flan-T5 XL	3B	1.0	0.00
	Google Gemma 3	4B	1.6	5.29
	DeepSeek-VL	7B	8.1	**37.49**
	Qwen2-VL	7B	**21.0**	30.28
Closed Source	GPT-4o	–	4.0	2.76
	Gemini 2.5 Flash-Lite	–	4.0	5.67
	Claude Sonnet 4	–	5.0	8.16

Table 8: Project-level error type distribution across models (percentage). Abbreviations: Undef. (Undefined), Br. (Brackets), Lbl. (Labels Missing), Non-adj. (Non-adjacent), MRef (Multi-Reference), Other (Other/Runtime).

Model	Undef.	Br.	Lbl.	Non-adj.	MRef	Other
Open Source						
Salesforce BLIP-2 Flan-T5 XL	2.0	37.4	–	–	58.6	2.0
Google Gemma 3	39.3	14.8	11.5	8.2	14.8	11.5
DeepSeek-VL	36.3	45.1	4.4	11.0	–	3.3
Qwen2-VL	25.0	12.5	7.8	51.6	–	3.1
Closed Source						
GPT-4o	61.5	7.3	16.7	4.2	4.2	6.2
Gemini 2.5 Flash-Lite	51.1	18.1	9.6	8.5	4.3	8.6
Claude Sonnet 4	46.3	14.7	21.1	7.4	5.3	5.3

4 DISCUSSION

The results across CrochetBench highlight both the promise and current limitations of Visual language models in bridging perception, procedural reasoning, and executable synthesis. By structuring tasks in a progressive ladder, we expose clear gradients of difficulty: while contemporary models demonstrate competence in low-level recognition and mid-level comprehension, their performance declines substantially when asked to generate or formalize instructions into compilable domain-specific programs. This pattern underscores both methodological bottlenecks in multimodal grounding and fundamental challenges in symbolic reasoning over long-horizon structures.

In Task A, stitch recognition is feasible for both open- and closed-source models, though precision–recall tradeoffs differ. Claude Sonnet 4 prioritizes precision for a higher F1, while DeepSeek-VL favors recall via richer visual encoders. This reflects divergent inductive biases: commercial models better regularize spurious predictions, while open-source models overpredict to capture subtle textures. Crucially, this is a multi-label, fine-grained texture classification task with symbolic implications, distinguishing it from generic object recognition. Baselines like BLIP-2, designed for captioning, underperform due to insufficient symbolic grounding (Li et al., 2023b; Radford et al., 2021). Task B amplifies these challenges. Closed-source models, especially GPT-4o, outperform open models by a wide margin, underscoring the difficulty of aligning visual cues with procedural semantics. Distractor instructions are intentionally plausible, demanding reasoning over local–global coherence, not just token overlap (Hendricks et al., 2016; Agrawal et al., 2016; Miech et al., 2019). Qwen2-VL's competitive performance suggests that scale and pretraining diversity help, but commercial systems benefit from stronger instruction tuning and better vision–language alignment.

Task C exposes the steep drop in open-ended procedural generation. Gemini 2.5 leads across BLEU, ROUGE, and ChrF (Papineni et al., 2002; Lin, 2004; Popović, 2015), reflecting fluency in structured text. Yet absolute scores are low, and outputs often fail to conform to crochet-specific syntax. Open models hallucinate frequently, lacking exposure to domain-aligned distributions. This highlights a broader issue: fluency in general language modeling does not imply competence in structured, domain-specific generation (Miech et al., 2019; Alayrac et al., 2022).

Task D proves most demanding. Even top closed-source models rarely exceed 6% project-level compilation success, while Qwen2-VL unexpectedly achieves 21%, suggesting better symbolic generalization under execution constraints. Syntax errors (e.g., undefined stitches, malformed brackets) dominate in open models, while closed models produce syntactically valid but semantically inconsistent programs. These complementary failure modes hint at differing generalization paths: symbolic robustness versus lexical fluency. Notably, Task D also tests 3D-aware procedural fidelity—models must translate visual or textual cues into symbolic programs that unfold into spatially coherent structures, not just grammatically valid sequences.

Taken together, these results demonstrate that success in linguistic generation does not translate directly to executable synthesis. Performance decays sharply when evaluation moves from surface-level fidelity (BLEU, ROUGE) to structural validity (compilation), reinforcing the importance of execution-grounded metrics for procedural tasks. The relative strength of Qwen2-VL at the project level further suggests that progress may come not from scaling alone, but from architectural or training adjustments that better capture long-range dependencies and stateful operations. More broadly, CrochetBench exposes a critical frontier for multimodal reasoning: models must not only ground text in vision but also internalize procedural invariants that guarantee functional correctness.

However, our analysis must be contextualized within certain constraints. The benchmarks reflect standardized stitch sets and normalized instructions, which, while ensuring comparability, simplify the variability encountered in real-world crochet practice (e.g., designer-specific shorthand, unconventional repeats). Moreover, evaluation metrics in Task C rely on string overlap, which may underestimate semantically correct but lexically divergent outputs. Even the compilation-based metric in Task D, though stronger, cannot assess visual fidelity unless paired with image-render verification. Finally, the sample size for project-level DSL translation remains limited, and absolute validity rates are low, constraining fine-grained statistical comparisons.

5 Future Work

A central avenue for future research is advancing models that translate free-form natural language crochet instructions into the formal **CrochetPARADE DSL**. As a domain-specific programming language, CrochetPARADE not only enables executable verification of multimodal outputs but also positions crochet as a program synthesis problem, where compilers map symbolic grammars into machine-executable instructions, extending ideas from prior DSL work in domains such as knitting (Hofmann et al., 2023), graphics (Ellis et al., 2018), and robotics (Mu et al., 2024). This perspective connects naturally to **CAD/CAM integration**: industrial crochet and warp-knitting machines (e.g., COMEZ, Jakob Müller) already rely on pipelines from graphical design interfaces, to intermediate graph-based representations, to low-level machine code. CrochetPARADE could serve as a standardized intermediate layer in this workflow, bridging human-facing authoring tools with machine-facing execution systems (Khan et al., 2024).

Several technical directions arise from this framing. First, incorporating explicit state-tracking mechanisms—whether through memory-augmented architectures (Graves et al., 2016) or symbolic scaffolds (Nye et al., 2021)—could mitigate long-range inconsistencies in DSL translation. Second, multimodal pretraining enriched with procedural and topological domains (e.g., assembly instructions (Cao et al., 2023), instructional videos (Miech et al., 2019)) may narrow the gap between natural-language fluency and executable synthesis. Third, hybrid evaluation pipelines that combine compilation checks with visual render comparisons could more holistically assess structural and perceptual fidelity, building on metrics used in programmatic 3D generation (Jain et al., 2022; Paschalidou et al., 2021). Finally, CrochetBench offers a testbed for exploring neuro-symbolic integration, where neural perception is paired with symbolic reasoning to reconcile fine-grained visual cues with globally coherent program execution.

In sum, CrochetBench and CrochetPARADE highlight a pronounced frontier at the intersection of **multimodal learning, program synthesis, and digital fabrication**. Progress here may one day close the loop from human intent to automated textile production, advancing both structured multimodal reasoning and computational craft.

Ethics Statement

We acknowledge that the original crochet pattern PDFs are protected under copyright and therefore do not distribute raw files or full texts. Instead, we release only structured JSON annotations generated with GPT, reference URLs to the original sources, and our parsing and annotation scripts. The benchmark is provided strictly for non-commercial academic use. This approach enables reproducible research while respecting intellectual property and ensuring that our dataset serves as a tool for studying structured generation rather than redistributing creative works.

Reproducibility Statement

We have taken several steps to ensure the reproducibility of our results. All datasets, task templates, and evaluation procedures are documented in the main text and appendix. An anonymous repository containing the full source code, experiment scripts, and detailed reproduction instructions has been made publicly available at: `https://anonymous.4open.science/r/crochet-82E6/README.md`. This ensures that all reported results can be independently verified and extended by the research community.

References

Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, and Devi Parikh. Vqa: Visual question answering, 2016. URL https://arxiv.org/abs/1505.00468.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning, 2022. URL https://arxiv.org/abs/2204.14198.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot, 2017. URL https://arxiv.org/abs/1705.07962.

Yizhak Ben-Shabat, Xin Yu, Fatemeh Sadat Saleh, Dylan Campbell, Cristian Rodriguez-Opazo, Hongdong Li, and Stephen Gould. The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose, 2023. URL https://arxiv.org/abs/2007.00394.

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. Instruction mining: Instruction data selection for tuning large language models. *arXiv preprint arXiv:2307.06290*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention, 2017. URL https://arxiv.org/abs/1609.04938.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. Learning to infer graphics programs from hand-drawn images, 2018. URL https://arxiv.org/abs/1707.09627.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, and Rongrong Ji. Mme: A comprehensive evaluation benchmark for multimodal large language models, 2024. URL https://arxiv.org/abs/2306.13394.

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

Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations, 2016. URL https://arxiv.org/abs/1603.08507.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps, 2021. URL https://arxiv.org/abs/2105.09938.

Megan Hofmann, Lea Albaugh, Tongyan Wang, Jennifer Mankoff, and Scott E Hudson. Knitscript: A domain-specific scripting language for advanced machine knitting. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606789. URL https://doi.org/10.1145/3586183.3606789.

Yushi Hu, Hang Hua, Zhengyuan Yang, Weijia Shi, Noah A. Smith, and Jiebo Luo. Promptcap: Prompt-guided image captioning for vqa with gpt-3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2963–2975, October 2023.

Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 867–876, 2022.

Alexandre Kaspar, Tae-Hyun Oh, Liane Makatura, Petr Kellnhofer, Jacqueline Aslarus, and Wojciech Matusik. Neural inverse knitting: From images to manufacturing instructions, 2019. URL https://arxiv.org/abs/1902.02752.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad models from beginner-to-expert level text prompts, 2024. URL https://arxiv.org/abs/2409.17106.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*, 2023a.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023b. URL https://arxiv.org/abs/2301.12597.

Peiyu Li, Xiaobao Huang, Yijun Tian, and Nitesh V. Chawla. Cheffusion: Multimodal foundation model integrating recipe and food image generation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, pp. 3872–3876, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704369. doi: 10.1145/3627673.3679885. URL https://doi.org/10.1145/3627673.3679885.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81. Association for Computational Linguistics, 2004. URL https://aclanthology.org/W04-1013/.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen WU, Mingyi Yan, Zhengyuan Yang, Lijuan Wang, and Mike Zheng Shou. Videogui: A benchmark for gui automation from instructional videos, 2024. URL https://arxiv.org/abs/2406.10227.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images, 2019. URL https://arxiv.org/abs/1810.06553.

Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. Howto100m: Learning a text-video embedding by watching hundred million narrated video clips, 2019. URL https://arxiv.org/abs/1906.03327.

Fnu Mohbat and Mohammed J Zaki. Llava-chef: A multi-modal generative model for food recipes. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 1711–1721, 2024.

Yao Mu, Junting Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, Peize Sun, Haibao Yu, Chao Yang, Wenqi Shao, Wenhai Wang, Jifeng Dai, Yu Qiao, Mingyu Ding, and Ping Luo. Robocodex: Multimodal code generation for robotic behavior synthesis, 2024. URL https://arxiv.org/abs/2402.16117.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021. URL https://arxiv.org/abs/2112.00114.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002)*, pp. 311–318. Association for Computational Linguistics, 2002. URL https://aclanthology.org/P02-1040/.

Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks, 2021. URL https://arxiv.org/abs/2103.10429.

Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models, 2016. URL https://arxiv.org/abs/1505.04870.

Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT 2015)*, pp. 392–395. Association for Computational Linguistics, 2015. URL https://aclanthology.org/W15-3049/.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Klara Seitz, Patrick Rein, Jens Lincke, and Robert Hirschfeld. Digital crochet: Toward a visual language for pattern description. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2022, pp. 48–62, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450399098. doi: 10.1145/3563835.3567657. URL https://doi.org/10.1145/3563835.3567657.

Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. Assembly101: A large-scale multi-view video dataset for understanding procedural activities, 2022. URL https://arxiv.org/abs/2203.14712.

Oleksii Sidorov, Ronghang Hu, Marcus Rohrbach, and Amanpreet Singh. Textcaps: a dataset for image captioning with reading comprehension, 2020. URL https://arxiv.org/abs/2003.12462.

Svetlin Tassev. Crochetparade: Crochet pattern renderer, analyzer, and debugger, 2025. URL https://www.crochetparade.org/. Accessed: 2025-09-24.

Haiwan Wei, Yitian Yuan, Xiaohan Lan, Wei Ke, and Lin Ma. Instructionbench: An instructional video understanding benchmark, 2025. URL https://arxiv.org/abs/2504.05040.

Yejing Xie, Harold Mouchère, Foteini Simistira Liwicki, Sumit Rakesh, Rajkumar Saini, Masaki Nakagawa, Cuong Tuan Nguyen, and Thanh-Nghia Truong. Icdar 2023 crohme: Competition onnbsp;recognition ofnbsp;handwritten mathematical expressions. In *Document Analysis and Recognition - ICDAR 2023: 17th International Conference, San José, CA, USA, August 21–26, 2023, Proceedings, Part II*, pp. 553–565, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-41678-1. doi: 10.1007/978-3-031-41679-8_33. URL https://doi.org/10.1007/978-3-031-41679-8_33.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pp. 476–486. ACM, 2018. doi: https://doi.org/10.1145/3196398.3196408.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019. URL https://arxiv.org/abs/1809.08887.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9556–9567, 2024.

Zheyuan Zhang, Yiyang Li, Nhi Ha Lan Le, Zehong Wang, Tianyi Ma, Vincent Galassi, Keerthiram Murugesan, Nuno Moniz, Werner Geyer, Nitesh V Chawla, Chuxu Zhang, and Yanfang Ye. NGQA: A nutritional graph question answering benchmark for personalized health-aware nutritional reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5934–5966, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.296. URL https://aclanthology.org/2025.acl-long.296/.

Luowei Zhou, Chenliang Xu, and Jason J. Corso. Towards automatic learning of procedures from web instructional videos, 2017. URL https://arxiv.org/abs/1703.09788.

A Related Work

A.1 Multi-modal Datasets Beyond Captioning

Most multimodal benchmarks have centered on descriptive pairing of images and natural language. Large-scale resources such as COCO (Lin et al., 2015) and Flickr30k (Plummer et al., 2016) provide dense captions of everyday scenes, advancing vision–language representation learning. More recent datasets extend beyond captioning to procedural or instructional domains. Recipe1M+ aligns food images with ingredient lists and cooking steps (Marin et al., 2019), while instructional video corpora such as YouCook2 (Zhou et al., 2017) and HowTo100M (Miech et al., 2019) pair narrated demonstrations with visual segments. These resources emphasize semantic alignment but generally evaluate with retrieval- or similarity-based metrics.

Our benchmark departs from this paradigm by pairing images with *executable procedures*. Rather than asking models to generate a semantically similar description, we require them to synthesize a program (CrochetPARADE DSL) that can be rendered and structurally verified. This shift enables *functional evaluation*—akin to program synthesis—and reduces reliance on subjective similarity measures.

A.2 Bridging Visual Reasoning and Procedural Language

Procedural understanding benchmarks highlight the importance of sequential, state-dependent reasoning. Datasets such as Assembly101 (Sener et al., 2022) and IKEA-ASM (Ben-Shabat et al., 2023) capture human assembly activities, modeling dependencies across actions, objects, and preconditions. Instructional video benchmarks further test long-horizon understanding and error detection (Wei et al., 2025; Lin et al., 2024). Our task complements this line of work by grounding supervision not in temporally segmented actions but in *artifact-centric procedures*—finished crochet items paired with symbolic, stepwise instructions. This enables models to reason about topology, geometry, and sequential dependencies in a single unified representation.

A.3 Crafting and Domain-Specific Procedural Data

Closer to our domain, prior work has begun to explore fiber crafts. Seitz et al. introduced *Digital Crochet*, a visual, graph-based notation system for representing crochet patterns (Seitz et al., 2022). In knitting, Kaspar et al. developed *Neural Inverse Knitting*, mapping images of knitted patterns to machine instructions (Kaspar et al., 2019). These works underscore the feasibility of executable supervision in crafts, but remain limited in scale and scope. Our benchmark builds on this foundation by providing thousands of real, community-tested crochet patterns, along with a compiler and renderer for executable evaluation.

A.4 Relation to Code Benchmarks

The executable nature of CrochetPARADE connects it directly to program synthesis benchmarks. HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and APPS (Hendrycks et al., 2021) evaluate code generation by execution against unit tests. Similarly, Spider (Yu et al., 2019) and CoNaLa (Yin et al., 2018) frame natural language to code translation tasks. In vision, Im2LaTeX-100K (Deng et al., 2017) and pix2code (Beltramelli, 2017) evaluate image-to-program translation with render fidelity as the metric. Our tasks—ranging from text-to-DSL translation to image-to-stitch recognition—extend this paradigm to the crafting domain, introducing structured 2D/3D topology as part of the evaluation. This grounds multimodal learning in a setting where success requires both semantic alignment and structural correctness.

A.5 Domain-Specific Languages for Executable Evaluation

A key enabler of our benchmark is the use of a domain-specific language. Prior DSL-based benchmarks, such as Im2LaTeX and the CROHME competition on handwritten math recognition (Xie et al., 2023), demonstrate how symbolic formalisms enable reproducible rendering and structural evaluation. CrochetPARADE adopts this principle for fiber crafts: each pattern compiles into an

abstract syntax tree that can be rendered and tested, supporting metrics such as ExecPass@K, structural unit tests, and image–render similarity. This functional perspective moves beyond surface-level similarity to test whether a model's output *actually works*.

B Additional Dataset Statistics

B.1 Instruction Complexity by Skill Level

Skill Level	Avg. Length	Median Length	Avg. Abbr.	Count
Beginner	1,674	1,365	9.2	465
Easy	2,761	2,182	10.8	3,569
Intermediate	4,221	3,387	10.7	1,967
Experienced	7,689	6,729	9.8	80

Table 9: Instruction complexity by skill level. Length is measured in characters.

Example Dataset Entry

Table 10: Representative pattern entry from **CrochetBench**.

Field	Value
Pattern Name	SKULL TRICK OR TREAT BAG (TO CROCHET)
Skill Level	Intermediate
Project Type	Bags or Purses
Measurements	15 cm diameter $\times$ 15 cm high (excluding handle)
Gauge	13 sc and 14 rows = 10 cm
Materials	Lily® Sugar'n Cream (White, Black), 5 mm hook, cardboard
Image	`https://www.yarnspirations.com/cdn/shop/products/SCC0303-005314M.jpg`
Source	`input_file/Bags+Purses/SCC0303-005314M.pdf`
Instructions	(truncated for brevity)

B.2 Skill Level Distribution

B.2.1 Overall Distribution

Table 11 summarizes the overall distribution of skill levels across the CrochetBench dataset. The majority of patterns are labeled as *easy* (58.7%), followed by *intermediate* (32.3%). Only a small fraction are classified as *beginner* (7.6%) or *experienced* (1.3%).[3]

Table 11: Overall skill level distribution. Percentages are relative to all patterns with annotated skill levels.

Skill Level	Count	Percentage
Easy	3569	58.66%
Intermediate	1967	32.33%
Beginner	465	7.64%
Experienced	80	1.31%
Total	6084	100%

One pattern (0.02%) is missing an annotated skill level.

[3]Three additional rare labels were observed: `easy to intermediate` (1 pattern), `beginners` (1 pattern), and `beginner/easy` (1 pattern). Together they account for $< 0.1\%$ of the dataset.

B.2.2 Distribution by Project Type

We further break down skill levels by the top 10 most common project types. Results are shown in Table 12. In most categories, *easy* patterns dominate, typically ranging between 53–70%. *Intermediate* is the second most common, while *beginner* and *experienced* remain consistently low across categories.

Table 12: Skill level distribution by top 10 project types. Percentages are within each project category.

Project Type	Easy	Intermediate	Beginner	Experienced
Afghans & Blankets	56.1%	35.3%	7.0%	1.5%
Hats	61.3%	27.8%	10.1%	0.7%
Sweaters & Cardigans	56.6%	35.9%	5.0%	2.5%
Shawls & Wraps	52.7%	41.8%	4.2%	1.2%
Scarves	63.2%	20.7%	16.1%	–
Pillows & Poufs	70.0%	22.9%	6.5%	0.7%
Amigurumi & Toys	64.0%	33.2%	2.1%	0.7%
Bags & Purses	53.8%	39.0%	6.8%	0.4%
Décor	58.4%	33.3%	6.5%	1.7%
Dishcloths	62.6%	27.5%	9.9%	–

Overall, the predominance of *easy* patterns reflects the accessibility of crochet as a craft and aligns with the goal of many project types to cater to a wide audience. The relative scarcity of *experienced*-level patterns suggests that most published resources emphasize broad usability rather than advanced expertise.

B.3 Pattern Complexity Analysis

B.3.1 Instruction Length Statistics

We first analyze the distribution of instruction lengths, measured in raw character counts. As shown in Table 13, the average instruction length is over 3,200 characters, while the median is substantially lower at 2,453 characters, reflecting a long-tailed distribution. The most complex patterns extend beyond 30,000 characters, while some very short patterns are as small as 20 characters.

Table 13: Instruction length statistics (in characters).

Statistic	Value
Average	3216.0
Median	2453.0
Min	20
Max	30634
25th percentile	1511.8
75th percentile	4136.2
90th percentile	6403.9

Out of 6,085 total patterns, 6,084 (99.98%) contain full instructions.

B.3.2 Abbreviation Statistics

Abbreviations, such as `sc`, `dc`, and `hdc`, are a distinctive element of crochet instructions. Table 14 reports abbreviation counts across all patterns. Most patterns contain about 10 abbreviations, with values ranging from 1 to 31.

Table 14: Abbreviation count statistics.

Statistic	Value
Average	10.6
Median	10.0
Min	1
Max	31

B.3.3 Complexity by Skill Level

Instruction length correlates with the designated skill level. As shown in Table 15, beginner-level patterns average under 2,000 characters, while intermediate patterns extend to over 4,200. Experienced patterns are the longest, averaging 7,689 characters. Rare categories such as `easy to intermediate` skew extremely long due to outliers.

Table 15: Instruction length and abbreviation counts by skill level.

Skill Level	Avg. Length	Median Length	Avg. Abbr.	Count
Easy to intermediate	13812.0	13812.0	21.0	1
Experienced	7689.4	6729.0	9.8	80
Intermediate	4221.3	3387.0	10.7	1967
Easy	2760.7	2182.0	10.8	3569
Beginner	1673.9	1365.0	9.2	465
Beginners	1633.0	1633.0	11.0	1
Beginner/Easy	1063.0	1063.0	–	1

B.3.4 Most and Least Complex Project Types

Finally, we identify the most complex and simplest project types by average instruction length. Tables 16 and 17 list the top 10 categories. Garments such as dresses, vests, pants, and tunics are the most demanding, with average instructions exceeding 5,800 characters. By contrast, smaller accessories such as cowls, washcloths, scarves, and headbands are substantially shorter, typically under 2,000 characters.

Table 16: Top 10 most complex project types (by average instruction length).

Project Type	Avg. Length	Median	Count
Dresses	6484.9	5799.0	34
Vests	6032.0	5193.5	64
Pants	5866.7	5409.0	11
Tunics	5850.4	5832.0	29
Sets	5625.5	4847.0	111
Sweaters & Cardigans	5429.2	5113.0	357
Amigurumi & Toys	5322.4	4505.0	286
Jackets	5311.9	4831.0	31
Onesies & Rompers	5263.4	5181.0	5
Aprons	4467.8	4494.0	11

Taken together, these results highlight strong alignment between project type, designated skill level, and instruction length. Garment-oriented projects require substantially longer and more complex instructions, while accessories and small decorative items remain simple and concise.

Table 17: Top 10 simplest project types (by average instruction length).

Project Type	Avg. Length	Median	Count
Cowls	1288.3	956.5	154
Washcloths & Mitts	1502.5	1420.0	28
Scarves	1567.3	1221.0	304
Headbands	1617.5	1475.5	38
Dishcloths	1688.4	1571.0	222
Coasters	1750.3	1625.0	26
Booties	1921.9	1938.5	24
Jewelry	1960.3	1549.0	55
Super Scarves	2007.6	1213.0	13
Tech Accessories	2011.1	2099.0	13

C PROMPTS

C.1 TASK A: STITCH RECOGNITION PROMPT

This task evaluates a model's ability to identify stitches present in a crochet product image.

Stitch Recognition Prompt (Rendered Example)

SYSTEM PROMPT You are a crochet stitch expert.
Given an image of a crochet product, identify all stitches that appear.
Requirements:
- Use only standard U.S. crochet abbreviations
(e.g., sc, hdc, dc, tr, ch, sl st, pop, etc.).
- Output must be a comma-separated list of abbreviations.
- Do not include explanations, extra text, or formatting beyond the list."""

USER PROMPT Look at this crochet product image and list the stitches used.
[Image]

C.2 TASK B: INSTRUCTION SELECTION PROMPT

This task evaluates a model's ability to choose the correct instructions from multiple-choice options.

Instruction Selection Prompt (Rendered Example)

SYSTEM PROMPT
You are a crochet expert. Your task is to determine which of the given options (A, B, C, or D) contains the correct crochet instructions for the image shown."""

USER PROMPT
Look at this crochet image and choose which option best matches the instructions for making it.
[Image]
Options: {options text}
Choose exactly ONE option. Your answer should be only one letter: A, B, C, or D.

C.3 TASK C: INSTRUCTION GENERATION PROMPT

This task evaluates a model's ability to generate complete crochet instructions from an image.

Instruction Generation Prompt (Rendered Example)

SYSTEM PROMPT
You are a professional crochet pattern writer. Examine the image of the finished crochet product carefully. Write a complete set of crochet instructions in the standard style used in published patterns.
Requirements:
- Use standard abbreviations: sc (single crochet), hdc (half double crochet), dc (double crochet), tr (treble), ch (chain), sl st (slip stitch), rep (repeat).
- Organize the instructions row by row or round by round (e.g., "Rnd 1: ...", "Row 2: ...").
- If color changes are visible in the image, include them in the pattern.
- Keep the instructions concise and precise, as if for experienced crocheters.
- Output only the crochet pattern. Do not add any explanations, commentary, or extra text.

USER PROMPT
Generate step-by-step crochet instructions for this image.
[Image]

C.4 Task D (Step-level): NL → DSL Translation Prompt

This task evaluates whether a model can translate a single natural language instruction into exactly one line of compilable **CrochetPARADE** DSL code.

Step-level NL → DSL Translation Prompt (Rendered Example)

SYSTEM PROMPT
You are a crochet compiler. Translate the next instruction NL into one line of CrochetPARADE DSL.
Use consistent naming and syntax.
Important rules for translations:
1. Make sure your output ONLY contains the DSL code, nothing else.
2. Use the previous examples to understand the pattern of translation.
3. Be consistent in naming conventions with the examples.
4. Your output should be exactly one line of DSL code.

USER PROMPT
Now translate the NL into DSL:
NL:
DSL:

C.5 Task D (Project-Level): NL → DSL Translation Prompt

This task evaluates whether a model can convert natural language crochet instructions (with optional images) into compilable CrochetPARADE DSL code.

NL → DSL Translation Prompt (Rendered Example)

SYSTEM PROMPT
You are a professional crochet pattern writer. Convert instructions + images into compilable CrochetPARADE DSL code. Output only the DSL code. No explanations, commentary, or extra text.
Example 1:
"image path": `https://www.yarnspirations.com/cdn/shop/files/BRC0116-035467M.jpg`,

INSTRUCTIONS
Note: Join with sl st to first sc at end of each rnd.

Ch 2.
Rnd 1: 6 sc in 2nd ch from hook. Join. (6 sc)
Rnd 2: Ch 1. 2 sc in each sc around. Join. (12 sc)
Rnd 3: Ch 1. (2 sc in next sc, 1 sc in next sc) repeat around. End with 1 sc. Join. (18 sc)
Rnd 4: Ch 1. (2 sc in next sc, 1 sc in each of next 2 sc) repeat. End with 1 sc in last 2 sc. Join. (24 sc)
Rnd 5: Ch 1. Sc in each sc around. Join. (24 sc)
Rnd 6: Ch 1. (2 sc in next sc, 1 sc in each of next 3 sc) repeat. End with 1 sc in last 3 sc. Join. (30 sc)
Rnds 7–8: Repeat Rnd 5 (sc in each sc). Join. (30 sc each round)
Rnd 9: Ch 1. **Working in back loops only**: (2 sc in next sc, 1 sc in each of next 2 sc) repeat. End with 1 sc in last 2 sc. Join. (40 sc)
Rnd 10: Ch 1. Sc in each sc around (both loops). Join. (40 sc)
Rnd 11: Ch 1. (2 sc in next sc, 1 sc in each of next 3 sc) repeat. End with 1 sc in last 3 sc. Join. (50 sc)
Finish: Fasten off.

DSL
¶ch.B
¶sc@B.A,5sc@B,ss@A
¶ch.A,sk,6sc2inc,ss@A
¶ch.A,sk,[sc2inc,sc]*6,ss@A
¶ch.A,sk,[sc2inc,2sc]*6,ss@A
¶ch.A,sk,24sc,ss@A
¶ch.A,sk,[sc2inc,3sc]*6,ss@A
¶[ch.A,sk,30sc,ss@A
¶[]*2
¶ch.A,sk,[scbl,scbl@[@],2scbl]*10,ss@A
¶ch.A,sk,40sc,ss@A
¶ch.A,sk,[sc2inc,3sc]*10,ss@A

USER PROMPT
Now generate DSL code for the following:

[Image]

[Instructions]
Rnd 1: Ch 2, 6 sc in ring
Rnd 2: 2 sc in each (12)
Rnd 3: [Sc, sc, inc] around (16)
Rnd 4: [Tr, sc] repeat around

[DSL]

D CrochetPARADE: Pattern Renderer, Analyzer, and Debugger

CrochetPARADE (short for *Crochet Pattern Renderer, Analyzer, and Debugger*) is an interactive platform that enables users to author, visualize, test, and export crochet patterns in both 2D and 3D (Tassev, 2025). By combining a custom pattern grammar with simulation and rendering tools, CrochetPARADE addresses common issues of ambiguity, correctness, and interpretability in textual crochet instructions.[4]

Core Capabilities.

[4] https://www.crochetparade.org/

- **Interactive authoring and rendering.** Users write pattern instructions in the CrochetPARADE grammar and then invoke a "calculate" operation to convert those instructions into a virtual model. The system supports both 2D and 3D views, along with interactive controls such as zoom, rotation, and stitch highlighting.
- **Validation and debugging.** CrochetPARADE parses the input, checks for syntactic and consistency errors (e.g., mismatched stitch counts, impossible attachments), and flags over- or under-stretched stitches.
- **Export and interoperability.** From a rendered pattern, users can export:
  - A standard crochet chart (SVG) with conventional stitch symbols and labeled stitch connections.
  - A 3D model (GLTF format) for integration into external tools such as Blender.
  - The underlying pattern instructions text (in the CrochetPARADE grammar), ensuring reproducibility and sharing.

Design Ideals and Rationale. CrochetPARADE is built to meet several design goals: (i) *unambiguous precision*, where the grammar is far more strict than free-form natural language, reducing interpretive errors; (ii) *local computation*, since all parsing, simulation, and rendering occur client-side in the browser with no user instructions sent to a central server; and (iii) *open source extensibility*, as the platform is released under GPLv3, with the grammar manual provided under a Creative Commons BY-NC-SA license.

Role in Our Work. Within the context of CrochetBench, CrochetPARADE provides a rigorous target representation: model predictions can be compiled into CrochetPARADE instructions, validated for syntactic and structural correctness, and then visualized or executed. This enables evaluation beyond surface-level metrics (e.g., BLEU, ROUGE) toward *executor correctness*—whether a generated pattern is valid, renderable, and stitch-balanced.

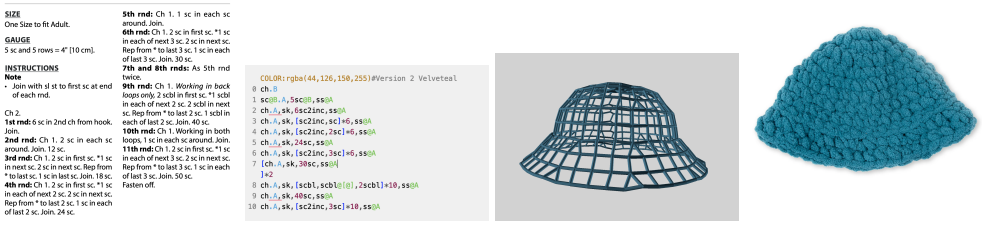

Figure 3: Example of the CrochetBench translation pipeline. (Left) Natural language crochet instructions from the dataset. (Second) Automatically translated into CrochetPARADE DSL, a formal stitch grammar. (Third) Mesh rendering generated from the DSL. (Right) Target crocheted item image provided in the dataset. This pipeline enables direct text-to-image consistency checks, automated validation, and future training of NL $\rightarrow$ DSL models, analogous to text-to-code generation.

E DSL Error Taxonomy

To better understand failure cases in Task D, we extend the validator's error analysis with detailed subcategories and examples. Errors fall into four major groups:

1. Syntax Structure Errors

Unbalanced Brackets. Missing opening/closing parentheses or brackets.

Examples

```
Unbalanced brackets:  (sc,hc5,sltr)infl)
```

Multiple References Without Parentheses. Improper formatting of references.

Example

```
Multiple references defined without parenthesis:
(21ch),turn
sk,(20sc)
(2ndrow):Ch1.(1scbl)ineachchtoendofrow.Turn
```

2. STITCH DEFINITION ERRORS

Stitch Not Found. Undefined stitch types not in the dictionary.

Examples

`ch1`, `ch3`, `scfp`, `hdc_bar`

3. LABELING AND REFERENCE ERRORS

Label Not Found. Reference to a non-existent label.

Example

```
Label not found:  C
```

Reusing Labels Incorrectly. Same label used for non-adjacent stitches.

Example

```
Cannot use same label over non-adjacent stitches.
Consider using different labels.
```

4. STRUCTURAL AND FORMATTING ISSUES

Turning Errors. Misplaced turning commands.

Example

```
Turning can happen only at the end of a row.
```

Variable Naming Issues. Conflicts between variable names and stitch names.

Example

```
Error:  variable name matches stitch name.  For
example, $ch=0$ cannot be used since 'ch' is a stitch
name.
```

Runtime Errors. Low-level parsing failures from the JavaScript compiler.

Examples

```
Cannot read properties of null (reading '0')
Cannot use 'in' operator to search for 'attach_id' in
NaN
```