HumanEval-V: Systematic Evaluation of Visual Reasoning in Large Multimodal Models for Code Generation

FENGJI ZHANG, Department of Computer Science, City University of Hong Kong, China
LINQUAN WU, Department of Computer Science, City University of Hong Kong, China
HUIYU BAI, Department of Computer Science, City University of Hong Kong, China
GUANCHENG LIN, Department of Computer Science, City University of Hong Kong, China
XIAO LI, Department of Computer Science and Technology, Tsinghua University, China
XIAO YU†, State Key Laboratory of Blockchain and Data Security, Zhejiang University, China
YUE WANG and BEI CHEN, Rhymes AI, USA
JACKY KEUNG, Department of Computer Science, City University of Hong Kong, China

Visual representations are essential for communicating complex programming concepts and play a critical role in algorithm and code design. While recent advances in Large Multimodal Models (LMMs) have demonstrated impressive visual capabilities across various domains, their visual reasoning abilities in coding contexts remain largely unexplored. In this work, we present a systematic evaluation of LMMs' visual reasoning abilities in coding contexts. We introduce HumanEval-V, a benchmark comprising 253 human-annotated code generation tasks, each requiring the generation of correct code solutions based on problem contexts encoded in diagrams. To ensure high-quality annotations, the construction of these tasks involves over 800 hours of meticulous human effort. Our tasks span six diverse categories and assess a broad spectrum of visual reasoning skills relevant to real-world programming scenarios. Through evaluation of 27 state-of-the-art LMMs, we find that even top-performing models such as Claude 3.5 Sonnet and Pixtral 124B achieve only 36.8% and 21.3% pass@1, respectively, while many open-weight models perform below 10%. Error analysis reveals that LMMs frequently generate hallucinations by misinterpreting or inventing visual details, and exhibit particular limitations in spatial reasoning, topological understanding, and handling dynamic visual patterns that are straightforward for humans. We finally discuss research opportunities and challenges to enhance model capabilities in visual reasoning for code generation.

CCS Concepts: • **Software and its engineering → Software development techniques**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: Large Language Models, Large Multimodal models, Visual Reasoning, Code Generation

Fengji Zhang, Linquan Wu, Huiyu Bai, and Guancheng Lin share equal contributions.

† Corresponding Author.
Authors' Contact Information: Fengji Zhang, fengji.zhang@my.cityu.edu.hk, Department of Computer Science, City University of Hong Kong, Hong Kong, China; Linquan Wu, Department of Computer Science, City University of Hong Kong, Hong Kong, China; Huiyu Bai, Department of Computer Science, City University of Hong Kong, Hong Kong, China; Guancheng Lin, Department of Computer Science, City University of Hong Kong, Hong Kong, China; Xiao Li, Department of Computer Science and Technology, Tsinghua University, Beijing, China; Xiao Yu, State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China; Yue Wang; Bei Chen, Rhymes AI, USA; Jacky Keung, Department of Computer Science, City University of Hong Kong, Hong Kong, China.

ACM 1557-7392/2026/5-ART
https://doi.org/10.1145/3813804

1 Introduction

Code generation refers to the task of automatically producing executable source code from natural language descriptions [35]. It has the potential to assist developers by streamlining development workflows and improving productivity [19]. In recent years, this area has seen substantial progress [51], largely driven by advances in Large Language Models (LLMs). When applied as code generation models, LLMs are increasingly approaching the capabilities of human programmers [67]. These models demonstrate strong performance across a wide range of benchmarks, including programming contest datasets such as APPS [31] and LiveCodeBench [33], domain-specific evaluations like DS-1000 for data science [40], multilingual settings as in MultiPL-E [7], and real-world software development scenarios such as ClassEval [20] and CoderEval [84]. These results demonstrate that current models are capable of understanding natural language instructions, generating correct code solutions, and performing algorithmic reasoning across a variety of tasks. However, there remains a *gap* in evaluating how well current coding models align with human programmers in terms of visual reasoning.

Visual diagrams and other graphical representations serve as essential tools for communicating complex ideas in software development [5]. Prior research has shown that such visual artifacts, including sketches and diagrams, play a critical role in the daily activities of programmers, particularly for tasks such as design, explanation, and comprehension of software systems [29]. For instance, on competitive programming platforms and technical Q&A forums such as Codeforces and Stack Overflow, diagrams are frequently used to depict data structures, input-output formats, conditional logic, and spatial relationships [68]. In contrast, natural language descriptions of code or programming problems are often ambiguous and prone to misinterpretation, especially when conveying intricate logic or complex structural relationships [14]. Expert developers often mentally simulate program behavior and visualize the shape of data structures as part of their problem-solving processes [60]. The effectiveness of visual representations in programming stems from human programmers' innate capacity for visual reasoning. Given the central role that visual reasoning plays in human programming, there is increasing recognition that code generation models must also develop strong visual reasoning capabilities to genuinely align with human programmers [45].

Recently, Large Multimodal Models (LMMs), such as GPT-4o [55] and LLaVA [43], have been developed to process images by encoding them as vision tokens and incorporating these representations into the auto-regressive training framework of LLMs [6]. This architectural advance enables LMMs to interpret and reason over visual inputs. In parallel, a variety of multimodal benchmarks, such as MMMU [87], MathVista [49], ChartQA [52], ARC-AGI [16], and RAVEN [88], have been proposed. While these benchmarks primarily target scientific, mathematical, or chart-based analytical questions to assess models' visual understanding and abstract reasoning, they overlook the coding domain, where visual reasoning is deeply intertwined with algorithmic thinking and code generation.

Within the multimodal programming domain, tasks such as Plot2Code [79] and Design2Code [62] have emerged, but these primarily focus on translating visual layouts into code, with minimal emphasis on algorithmic reasoning. A more relevant effort is the MMCode benchmark [45], which compiles coding problems accompanied by visual demonstrations from competitive programming platforms to evaluate LMMs on code generation tasks. However, MMCode exhibits notable limitations in assessing visual reasoning capabilities. First, it does not clearly isolate the contribution of visual information to problem-solving. In many cases, the visual content is replicated in the accompanying textual description, rendering the visual modality redundant. Second, the benchmark fails to disentangle whether the primary challenge of a problem stems from its algorithmic complexity or the visual reasoning required. Consequently, when LMMs fail to produce correct code, it remains unclear whether the cause is inadequate visual reasoning or the intrinsic difficulty of the coding task. Collectively, these limitations underscore persistent *gap* in effectively evaluating the visual reasoning capabilities of LMMs in programming contexts.

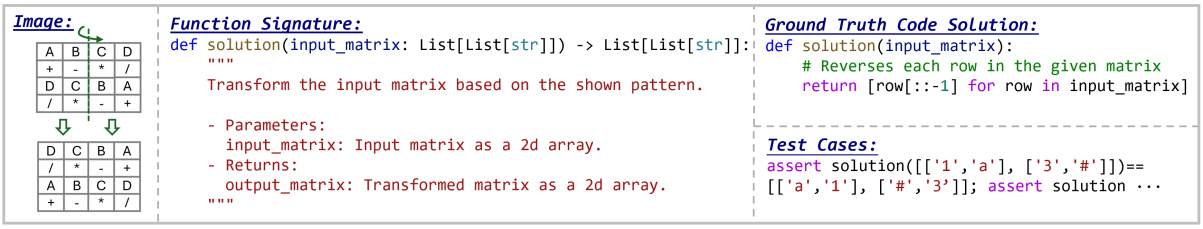


Fig. 1. An example task from HumanEval-V. LMMs are required to interpret the diagram and function signature, deduce the underlying algorithm, and implement the corresponding Python function. The implementation is then evaluated through a set of test cases.

To bridge this *gap*, we introduce HumanEval-V, a new multimodal coding benchmark specifically constructed to isolate and test visual reasoning in programming. The benchmark comprises 253 human-annotated coding problems, each centered around a carefully designed diagram that encodes the essential algorithmic logic while minimizing textual descriptions. Each task features (1) a diagram that visually presents the problem context, (2) a function signature that specifies the required input-output structures, and (3) a set of test cases for verifying solution correctness. To provide a more intuitive understanding, Figure 1 presents a concrete example from HumanEval-V. In this example, the diagram illustrates spatial transformation patterns, requiring the model to interpret fine-grained visual elements such as matrices, arrow directions, and spatially ordered data points. Our task format is inspired by the HumanEval [11] benchmark, where models are expected to generate a function body according to provided specifications and validate the solution using test case execution. However, unlike HumanEval, which includes problem descriptions as textual comments, HumanEval-V encodes the essential algorithmic logic and constraints directly within the diagram. The diversity of our benchmark is further highlighted in Figure 2, which categorizes task types, and Figure 3, which summarizes the key reasoning capabilities required across tasks. The construction of these tasks costs over 800 hours of meticulous human effort. To the best of our knowledge, no existing benchmark offers this level of task diversity and fine-grained emphasis on visual reasoning in programming. Based on HumanEval-V, we conduct a systematic evaluation of visual reasoning in LMMs for code generation, addressing the following three Research Questions (RQs):

RQ1: How well do state-of-the-art LMMs perform on HumanEval-V? We evaluate 27 cutting-edge LMMs on HumanEval-V, testing both their direct code generation abilities from visual input and a two-stage setting where the model first produces a structured diagram description, which is then used to generate code by a separate strong coding model. This decoupled evaluation allows us to isolate and assess visual reasoning independently of coding ability. Our results show that, despite strong performance on general multimodal benchmarks, existing LMMs struggle with programming tasks requiring high-level visual reasoning. Top-performing models such as Claude 3.5 Sonnet and Pixtral 124B achieve only 36.8% and 21.3% pass@1 accuracy, respectively.

RQ2: What limitations do current LMMs exhibit when solving HumanEval-V tasks? Our fine-grained error analysis reveals that LMMs are particularly challenged by tasks involving spatial transformations, topological relations, and dynamic visual patterns, which are trivial for human annotators. Moreover, performance of LMMs does not correlate strongly with human-perceived task difficulty, as measured by code cyclomatic complexity [22] and the length of the diagram description. Most models lack robust high-level visual reasoning, frequently hallucinate, and often misinterpret critical visual details. Case studies show that even top-performing models struggle to generalize visual rules or integrate multiple visual-semantic cues. However, when provided with human-annotated ground-truth descriptions, most LMMs are able to generate correct solutions, further confirming that visual reasoning capabilities are the primary bottleneck.

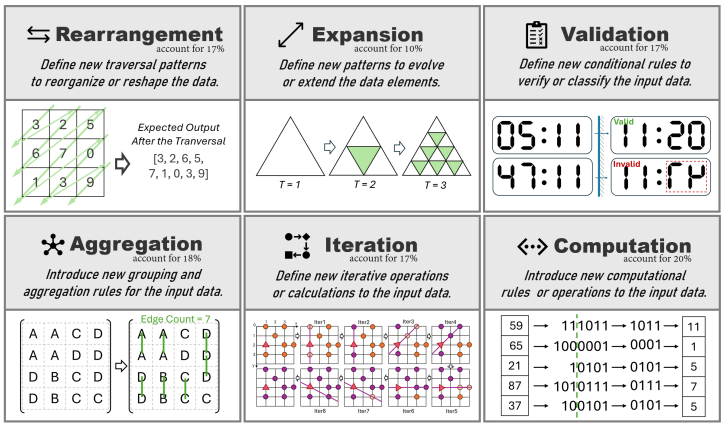


Fig. 2. The six task categories in HumanEval-V, along with their quantitative distribution and representative diagram examples.

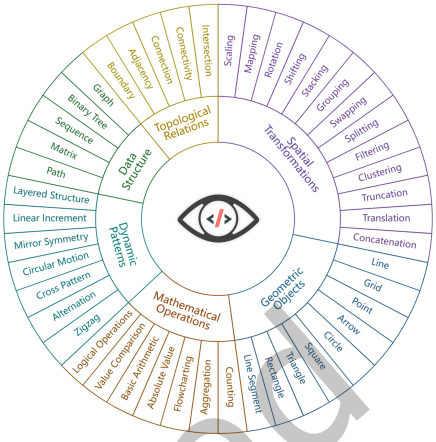


Fig. 3. Core knowledge required for understanding diagrams in HumanEval-V.

RQ3: What are the promising research pathways to enhance visual reasoning in code generation? Building on our findings, we recognize that fully solving multimodal algorithmic reasoning remains an open challenge. We explore three promising methodological pathways for future improvement. First, we investigate RL-driven post-training by evaluating reasoning-enhanced models (e.g., OpenAI o1 [56], QVQ [66]). While these models raise the performance ceiling, they still suffer from severe "overthinking" and degraded instruction following in multimodal contexts. Second, we explore interactive learning via environmental feedback, demonstrating that models can genuinely self-correct visual misunderstandings when provided with execution signals. Finally, we discuss the fundamental bottleneck for scalable in-domain data synthesis. Overcoming this data generation challenge remains a vital prerequisite for developing robust, vision-centric training paradigms.

To sum up, the main contributions of this work are as follows:

- We introduce HumanEval-V, a novel benchmark consisting of 253 human-curated tasks specifically designed to enable fine-grained evaluation of LMMs' visual reasoning capabilities in code generation. We release all code and data at https://github.com/HumanEval-V/HumanEval-V-Benchmark.
- Our comprehensive evaluation of 27 state-of-the-art LMMs reveals that, despite their strong results on other multimodal benchmarks, their performance on HumanEval-V tasks remains limited.
- We conduct in-depth error analysis and case studies, revealing that current models exhibit major limitations in spatial reasoning, understanding dynamic patterns, and aligning visual content with code semantics.
- We explore and discuss several promising directions to enhance visual reasoning in LMMs, including reasoning-enhanced models and interactive learning through execution-based feedback.

The remainder of this paper is organized as follows: Section 2 reviews relevant literature. Section 3 details the construction of HumanEval-V. Sections 4, 5, and 6 present our main experiments, error analysis, and discussion of potential research opportunities, addressing RQ1, RQ2, and RQ3, respectively. Section 7 provides additional ablation studies, Section 8 discusses threats to validity, and Section 9 concludes the paper.

2 Related Work

2.1 Coding Benchmarks

The evaluation of code generation capabilities in LLMs has become a focal point of research, encompassing a broad spectrum of programming contexts and application domains.

For general-purpose and algorithmic coding tasks, widely adopted benchmarks include ODEX [76], CoNaLA [82], HumanEval [11], and MBPP [4]. These benchmarks typically consist of self-contained programming problems with function signatures, natural language specifications, and hidden unit tests. Models are expected to synthesize correct and executable code, with evaluation primarily based on pass@k metrics under deterministic or sampled decoding settings. Enhanced variants such as HumanEval-ET [18], MBXP [3], EvalPlus [47], EvoEval [80], and MultiPL-E [7] further extend these benchmarks by introducing test case augmentation, multilingual support, cross-compilation environments, or fine-grained scoring metrics that better reflect real-world utility and robustness.

For competition-level programming, benchmarks like LiveCodeBench [33], LeetcodeContest [28], APPS [31], and AlphaCode CodeContests [46] provide more complex and open-ended challenges drawn from actual programming competitions. These problems often require advanced algorithm design, multi-step reasoning, and deeper integration of programming knowledge. Unlike general-purpose benchmarks, the difficulty in these datasets is significantly higher, and solving them may involve dynamic programming, graph traversal, or combinatorial logic. Some benchmarks, such as LiveCodeBench, simulate dynamic programming contests by updating their problem sets on a monthly basis.

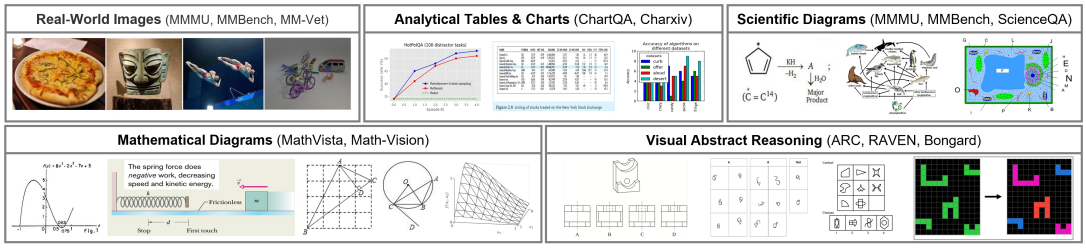
Beyond algorithmic problems, domain-specific benchmarks have been introduced to evaluate LLMs in specialized, real-world contexts. For example, DS-1000 [40] focuses on data science tasks involving exploratory data analysis and transformation using libraries such as pandas and numpy. Repository-level benchmarks like CoderEval [84], RepoEval [89], and SWE-Bench [37] evaluate models on software engineering tasks extracted from real codebases, such as code completion and feature implementation. These tasks require models to comprehend existing code context, often spanning multiple files or modules. In the area of documentation-aided code generation, DocPrompting [93] evaluates how effectively models can generate code based on retrieved API documentation.

Additionally, some recent benchmarks aim to evaluate advanced capabilities such as self-debugging and program repair [12, 74]. These tasks challenge models not just to generate correct code from scratch, but to identify and fix buggy or failing implementations using execution feedback. Such settings introduce the element of interaction and iterative refinement, closely aligning with real-world development scenarios. These benchmarks reflect the growing proficiency of LLMs in diverse coding scenarios. However, they still fall short in addressing the role of visual programming contexts, which remains an underexplored yet important aspect.

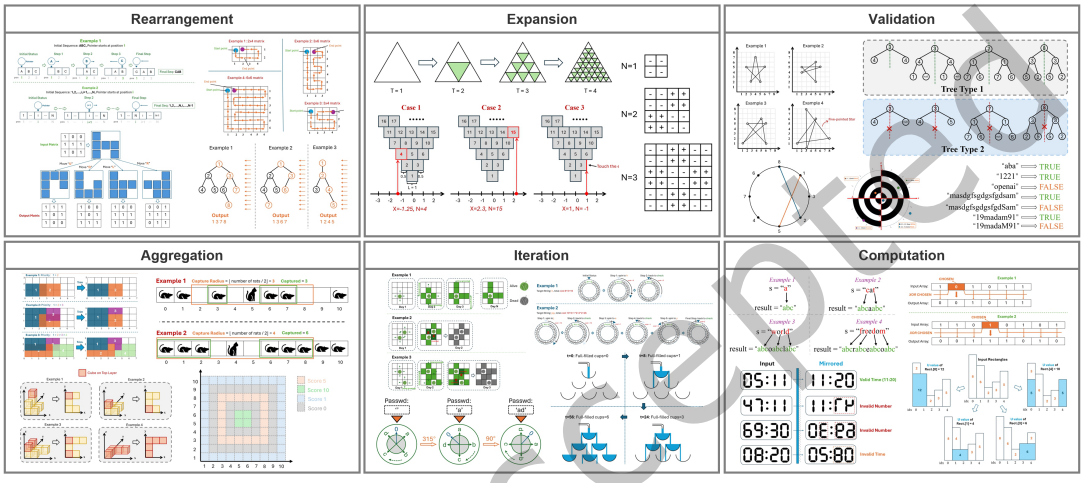
2.2 Multimodal Benchmarks

Multimodal benchmarks have been developed across a wide spectrum of domains, each targeting different aspects of visual capabilities. However, none are specifically designed to evaluate visual reasoning in the context of programming. We categorize existing multimodal benchmarks into five major types, with representative examples from each category illustrated in Figure 4a.

Real-world visual understanding benchmarks [10, 42, 48, 83, 85, 87] are designed to assess broad vision-language capabilities across multiple disciplines. These benchmarks typically include everyday photographs depicting food, sports, art, architecture, and wildlife, often in naturalistic or complex real-world scenes. Solving such tasks primarily requires object recognition, commonsense reasoning, and general knowledge grounding. These images test perception-oriented abilities but lack structured logical relationships, making them fundamentally different from the diagrammatic reasoning required in programming tasks.



(a) A selection of images covered in current popular multimodal benchmarks.



(b) A selection of diagrams representing the six task types in HumanEval-V.

Fig. 4. Comparison of diagrams in current popular multimodal benchmarks and in HumanEval-V.

Analytical visualization benchmarks [16, 52, 75] focus on the interpretation of structured data presentations such as bar charts, line graphs, and frequency tables. These tasks assess a model's ability to extract numerical patterns, compare quantities, and reason about trends or distributions. Solving these problems typically requires quantitative comparison and surface-level data interpretation, but rarely involves deeper reasoning.

Scientific diagram understanding benchmarks [39, 48, 50, 87] evaluate the interpretation of technical visualizations such as biological processes, physics systems, and ecosystem diagrams. These benchmarks emphasize relational and causal reasoning grounded in domain-specific knowledge. While scientific diagrams introduce complex visual structures, they primarily assess domain-specific knowledge and factual recall, rather than flexible visual reasoning or the ability to infer novel algorithmic procedures from the diagram itself.

Mathematical visual reasoning benchmarks [49, 70, 91] combine formal mathematical concepts with visual representation, including geometry problems, function plots, and physics-based illustrations. These benchmarks demand symbolic abstraction, spatial reasoning, and the ability to ground visual entities in formal logic. The structured and symbolic nature of these visuals shares some common ground with programmatic diagrams, although they are typically confined to mathematical domains.

Abstract visual reasoning benchmarks [15, 16, 36, 54, 88] are designed to test high-level cognitive skills such as pattern recognition, rule induction, analogy-making, and compositional reasoning. These datasets feature stylized puzzles involving grid-based transformations, symbolic rules, and geometric patterns. Solving such problems

requires inferring latent rules and applying advanced reasoning, which are cognitively adjacent to algorithmic thinking.

Despite their broad coverage, existing benchmarks largely overlook the specific complexities of diagrams encountered in coding tasks. Figure 4b presents six fundamental task types in our HumanEval-V benchmark, each representing distinct cognitive challenges in visual reasoning. Our benchmark employs a rich variety of visual elements including geometric shapes, symbolic notations, matrices, and directed graphs. These representations are enhanced through connecting lines, arrows, color-coding, and numerical annotations to effectively capture relationships and transformations between components. The visual representations maintain clarity across all categories while scaling in complexity to accommodate different difficulty levels. Through careful design of visual elements and systematic progression of patterns, each task type provides a clear framework for evaluating specific aspects of visual reasoning and problem-solving abilities.

2.3 Multimodal Coding Tasks

Recent work in multimodal code generation can be broadly categorized into three primary domains based on their core objectives: (1) Visual Translation and Comprehension, (2) Visual Software Engineering, and (3) Multimodal Algorithmic Reasoning.

The first category, *Visual Translation and Comprehension*, focuses on converting visual layouts into corresponding code or using code to parse visuals. This includes tasks such as derendering web pages [41, 62], converting scientific figures into plotting code [61, 79], and understanding general GUI or OS command outputs as evaluated in InfiBench-V [34]. It also encompasses program-based visual question answering, where models generate code to invoke predefined perception modules to answer questions grounded in visual input [63, 64]. These tasks primarily evaluate structural layout parsing and visual-to-text alignment.

The second category, *Visual Software Engineering*, evaluates models on tasks situated within the software development and debugging lifecycle. For instance, Visual SWE-Bench [90] requires fixing bugs in scientific plotting scripts based on rendered chart outputs. Similarly, M^2Eval [8] focuses on comprehending structured software design formats, such as Mermaid-rendered UML or flowcharts. Understanding these visuals relies heavily on domain-specific knowledge of chart semantics, plotting libraries, or software engineering conventions, rather than free-form algorithmic reasoning.

The third category, *Multimodal Algorithmic Reasoning*, requires models to deduce underlying algorithmic logic from abstract visual cues. The most closely related benchmark in this category is MMCode [45], which evaluates LMMs on code generation tasks accompanied by visual demonstrations collected from competitive programming platforms. However, MMCode presents several key limitations in assessing visual reasoning: First, it does not explicitly disentangle the contribution of visual information to problem-solving. In many tasks, the visual content is entirely or mostly duplicated in the accompanying textual description, rendering the visual input unnecessary. This redundancy is reflected in MMCode's results, where models perform similarly on "language-only" and "vision+language" settings. Second, MMCode does not distinguish whether task difficulty stems from algorithmic complexity or from visual reasoning. Since many competitive programming problems are inherently challenging, models may fail due to their limited coding abilities, not necessarily due to a lack of visual understanding. This conflation makes it difficult to isolate and evaluate visual reasoning performance. Third, MMCode directly uses existing competition problems without modification. As a result, there is a non-negligible risk of data contamination, since some tasks may have been seen in LMMs' training data, compromising the benchmark's validity for rigorous evaluation.

In contrast, HumanEval-V is deliberately designed to address these issues and place visual reasoning at the center of evaluation. All tasks in HumanEval-V are human-annotated to ensure that the visual context is essential and cannot be inferred from text alone. In our validation experiments, five proprietary LMMs achieve 0% pass

Table 1. Original Codeforces algorithmic tags associated with HumanEval-V tasks in each visual category.

Visual Category	Original Codeforces Algorithm Tags
Iteration	greedy, math, brute force, constructive algorithms, geometry, dp, bitmasks, strings, matrices
Aggregation	greedy, math, brute force, constructive algorithms, sortings, dp, binary search, two pointers, games, combinatorics, shortest paths
Expansion	greedy, math, brute force, geometry, dp, trees, binary search, bitmasks, number theory
Validation	greedy, math, brute force, constructive algorithms, sortings, geometry, dp, trees, binary search, graphs, number theory, dfs, dsu, hashing
Computation	greedy, math, brute force, sortings, geometry, trees, graphs
Rearrangement	greedy, math, brute force, sortings, constructive algorithms, strings, dsu

rates when provided only with function signatures, yet exceed 90% accuracy when given human-annotated diagram descriptions, clearly demonstrating the importance of visual input in our tasks. Furthermore, our difficulty analysis shows that the coding aspects of HumanEval-V tasks remain moderately complex, allowing us to focus on evaluating visual reasoning without conflating it with programming skill. Finally, our evaluation framework introduces a two-stage setting, where LMMs with limited coding ability can still be assessed on diagram understanding by generating visual descriptions, which are then passed to a strong code generation model. These careful design choices distinguish HumanEval-V from prior benchmarks and enable more accurate and targeted assessment of visual reasoning in code generation.

3 Benchmark Construction

3.1 Motivation and Design Rationale

A core motivation behind HumanEval-V is to enable a controlled, diagnostic evaluation of multimodal code generation. In real-world programming, tasks often entangle textual requirements, complex algorithmic logic, and visual cues. This intertwining makes it difficult to attribute a model's failure to a specific capability. To address this, our benchmark intentionally abstracts away confounding textual and coding factors to concentrate essential problem-solving details within the visual modality. By doing so, we ensure that the evaluation isolates high-level visual reasoning, such as interpreting fine-grained cues and diagrammatic patterns, from general software engineering skills.

This visual-centric philosophy drives not only our task construction pipeline (which distills and recreates problems to preserve core visual logic while simplifying ancillary text), but also our task categorization strategy. Because HumanEval-V is designed to probe visual understanding, our task taxonomy must be defined by *visual* characteristics rather than traditional *algorithmic* topics. Consequently, our categorization is data-driven, manually clustering problems based on visual semantics: how visual elements change, how information is filtered, and how objects are reorganized. This yields the six structural visual patterns shown in Figure 2.

To verify that these visual categories align with real-world programming practice without degenerating into standard algorithmic classifications, we analyzed a subset of HumanEval-V tasks derived from Codeforces. As summarized in Table 1, each visual category covers a diverse set of standard algorithmic tags (e.g., greedy, dynamic programming). The absence of a one-to-one mapping confirms that our taxonomy successfully captures recurring *diagram types* rather than a narrow set of algorithms. Ultimately, these design choices, from task abstraction to

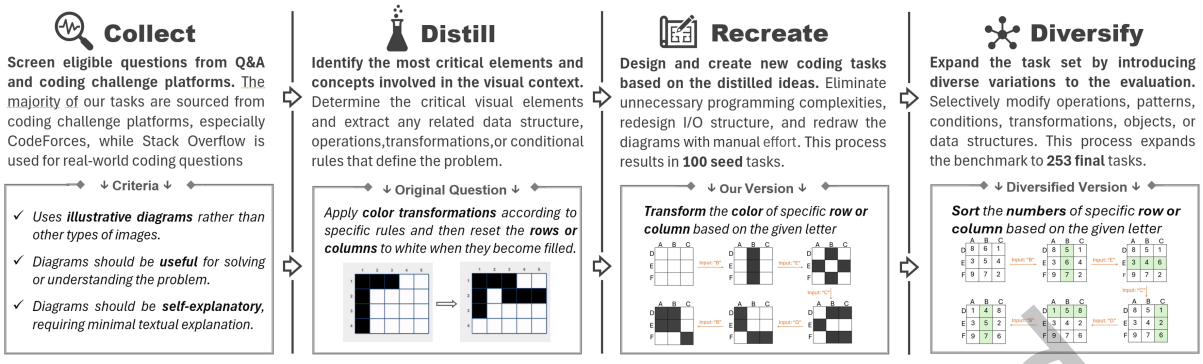


Fig. 5. The task construction pipeline of HumanEval-V. We employ a four-step systematic manual annotation process designed to ensure extreme quality and diversity in the resulting tasks.

visual categorization, ensure that HumanEval-V provides a rigorous, interpretable, and targeted assessment of vision-centered reasoning competencies.

3.2 Task Definition and Standards

As shown in Figure 1, each coding task in HumanEval-V includes: (1) a single diagram D providing visual context, (2) a Python function signature σ with input parameters, return type, and brief instructions, and (3) a set of test cases $T = t_1, t_2, \ldots, t_n$ to validate the correctness of the generated output O, which is a complete Python function generated by the Large Multimodal Model (LMM).

We establish rigorous standards to construct high-quality coding tasks: (1) The visual context must be essential for solving the task, with all relevant information contained in a single image; (2) Tasks should be designed around the visual context with minimal textual description; (3) Unnecessary programming complexities, such as recursion, intricate constraints, or complex data structures, should be avoided.

3.3 Task Construction Pipeline

We define the task construction pipeline with *four steps* as in Figure 5.

*In the **first** step*, we collect a large set of coding problems from prominent Q&A and coding challenge platforms that incorporate images, then screen them to exclude questions that (1) require specific programming frameworks or libraries, or contain images that (2) are not illustrative diagrams, (3) provide no useful information for solving the problem, or (4) require extensive textual context for interpretation. Additional details and examples of the data collection and screening processes can be found in Section 3.4.

*For the **second** step*, we distill the screened problems to identify critical visual elements, along with the data structures, operations, transformations, and conditional rules involved, categorizing them into six task types. The task types, along with their criteria and examples, are presented in Figure 2 and Figure 4b. We also outline the key capabilities needed for diagram understanding. This capability breakdown is shown in Figure 3, which divides the required capabilities into six dimensions: *Data Structure*, *Geometric Objects*, *Spatial Transformations*, *Dynamic Patterns*, *Mathematical Operations*, and *Topological Relations*. Each dimension encompasses numerous capability aspects that frequently appear in tasks of similar types.

*For the **third** step*, we design new coding tasks based on these distilled ideas, eliminating unnecessary complexities, refining input/output structures and function signatures, and creating visual objects and layouts using

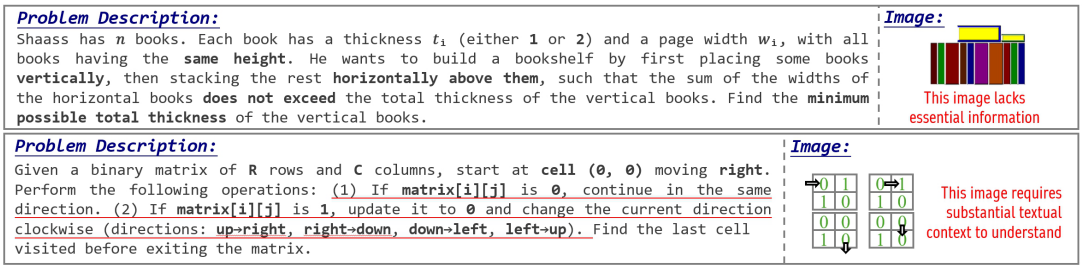


Fig. 6. Two negative examples in our data screening process: the first example is sourced from CodeForces (https://codeforces.com/problemset/problem/294/B), and the second from GeeksforGeeks (https://www.geeksforgeeks.org/problems/last-cell-in-a-matrix/1).

basic shapes and a consistent color scheme. We also generate tailored test cases, resulting in ***100*** newly crafted *seed tasks*.

*For the **fourth** step*, we expand the task set by diversifying the seed tasks using a hybrid approach with GPT-4o. GPT-4o identifies relevant capability aspects for each seed task and suggests modifications involving new spatial transformations, mathematical operations, dynamic patterns, or variations in data structures and object attributes. Human annotators then refine and annotate these new tasks and diagrams, creating 0 to 2 diversified versions per seed task based on complexity, culminating in ***253*** tasks in `HumanEval-V` and finalizing the capability aspects shown in Figure 3. It is worth noting that the scale of our benchmark is well aligned with established standards. With 253 tasks, `HumanEval-V` is comparable to other influential human-annotated benchmarks that have advanced the field, such as HumanEval [11] (164 tasks), MMVet [85] (218 tasks), Vibe-Eval [59] (269 tasks), ClassEval [20] (100 tasks), EvoCodeBench [44] (275 tasks), and SWE-Bench Lite [37] (300 tasks). Further details and examples of the task recreation and diversification processes are provided in Section 3.5.

3.4 Details on Data Collection and Screening

To ensure real-world relevance, we construct our benchmark by collecting problems from multiple publicly available sources, including Stack Overflow and coding challenge platforms such as Codeforces. These platforms host a wide range of programming questions that incorporate diagrams or visual aids as part of their problem statements. A key challenge in this process lies in identifying whether the visual content of a problem is essential for solving it. While many problems include diagrams, not all visuals contribute meaningful information beyond what is already conveyed in the text. Automatically distinguishing between essential and non-essential diagrams is non-trivial; thus, we perform manual screening to ensure that only problems with indispensable visual components are included in the benchmark. Figure 6 shows two examples of rejected problems: (1) a Codeforces problem about stacking books, where the image is purely illustrative and the key logic resides in the text; and (2) a GeeksforGeeks problem involving matrix traversal, where the solution depends heavily on detailed textual instructions.

To curate high-quality seed tasks, we apply source-specific filtering strategies. For Stack Overflow (SO), we focus on Python-tagged questions with non-negative votes, accepted answers, and at least one image accompanied by code snippets. We exclude problems that primarily involve front-end or UI topics, as well as those where images only present textual content such as error messages or screenshots. For coding Challenge Platforms, we draw primarily from the MMCode [45] dataset, which aggregates visual programming problems from various competition platforms. During manual review, we filter out problems where the diagram is redundant with the text, or the textual context is overly complex and indispensable for interpretation, thereby violating our

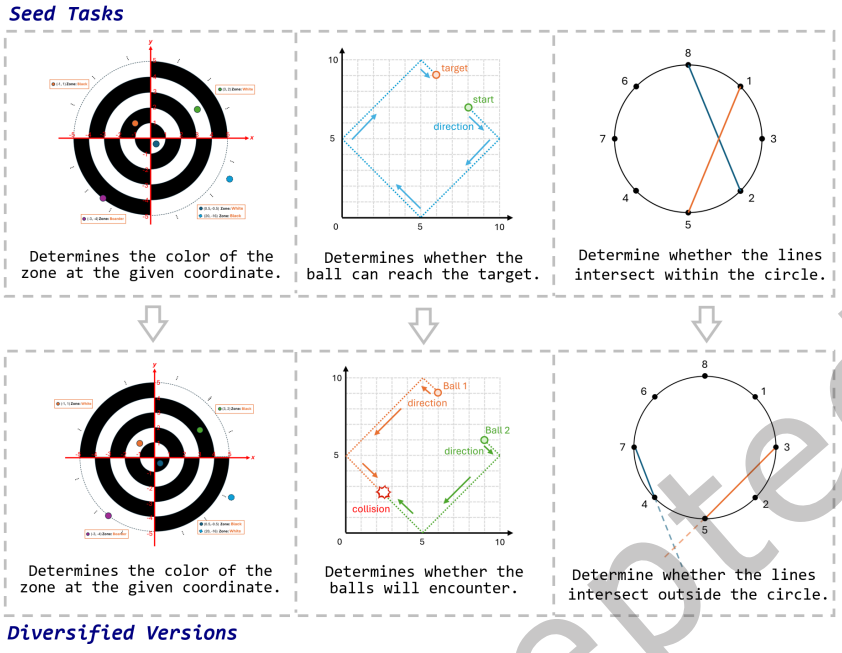


Fig. 7. Additional diversification examples highlighting the multi-aspect nature of task modifications.

requirement for visual self-containment. After screening, we select a total of 100 seed problems with high-quality visual content. Their distribution by source is as follows: Codeforces (65), LeetCode (13), GeeksforGeeks (6), HackerRank (2), Project Euler (2), Open Kattis (2), CodeChef (1), AtCoder (1), and Stack Overflow (8).

3.5 Details on Task Creation and Diversification

To ensure the quality and focus of our benchmark, we apply a two-stage process to the screened coding problems.

(1) Recreation: There are two primary motivations for recreating tasks from real-world problems. First, to avoid potential data contamination, we distill the essence of each selected coding problem, focusing exclusively on its core aspect involving visual reasoning. All extraneous problem contexts are discarded, and new coding tasks are recomposed that capture the original's critical visual reasoning element, while differing in surface details and implementation. Second, recreation allows us to tailor each problem to the target of our benchmark: evaluating the visual reasoning capabilities of LMMs in coding contexts. To this end, we remove unnecessary complexities, such as reliance on external libraries, complicated conditional logic, or advanced algorithms. Additionally, every problem diagram is redrawn to ensure that the essential algorithmic information is embedded within the visual elements themselves. Auxiliary elements such as indicators, annotations, arrows, and color markings are added where needed, making each diagram self-explanatory and maximizing its visual clarity.

(2) Diversification: The purpose of diversification is to enhance the diversity of our benchmark and to assess model robustness to subtle variations. For each seed task, we modify parts of the diagram or the question to generate new, but closely related, tasks. These modifications test whether models can truly understand the visual and textual components, rather than relying on memorization or superficial cues. Changes are made along several key aspects, as summarized in Table 2, including spatial transformations, mathematical operations, dynamic patterns, object attributes, object relations, and data structure modifications.

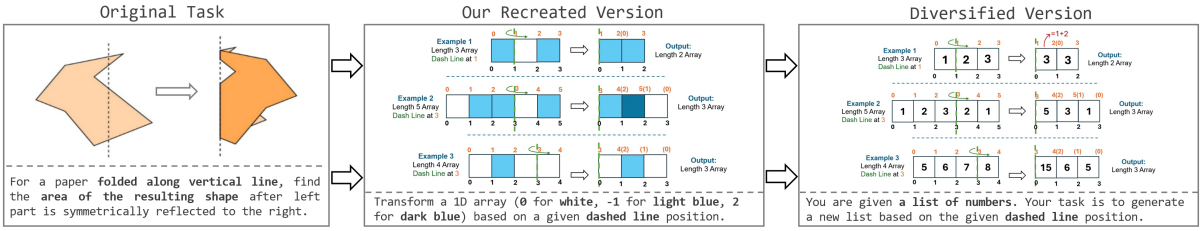


Fig. 8. Task annotation examples illustrating the recreation and diversification applied to the screened coding problem. The original problem is sourced from CodeForces (https://codeforces.com/problemset/problem/1381/E).

Table 2. Examples of modification aspects applied to diversify seed tasks. Each diversified task typically spans multiple aspects simultaneously. The rightmost column reports the proportion of diversified tasks in which each aspect appears.

Modification Aspects	Examples	Freq.
Spatial Transformation	Adjust concatenation, swapping, grouping, or stacking order; modify the direction of translation, flipping, rotation, or folding.	88%
Mathematical Operations	Alter arithmetic operations, sorting order, or aggregation rules.	57%
Dynamic Patterns	Reverse alternation sequences; switch increments to decrements; change the direction of spirals or zigzags; adjust layer layouts.	42%
Object Attributes	Change the color, size, shape, angle, or position of objects.	81%
Object Relations	Reverse nesting or overlap order; modify connection, intersection, or adjacency rules; adjust boundary interaction conditions.	63%
Data Structure	Modify graph direction, data type in arrays, or matrix dimensions.	29%

A key characteristic of our diversification process is that it is inherently *multi-aspect*. Unlike perturbation-based augmentation where each variation corresponds to a single transformation, our diversified tasks typically involve *3-6 modification aspects simultaneously*. As modifications interact with one another and derive meaning only in the context of each task's visual semantics, reporting disjoint per-aspect frequencies would be misleading. Nevertheless, for transparency, Table 2 additionally reports non-exclusive usage frequencies across all diversified tasks. To ensure the reliability of these statistics, we implemented a rigorous two-annotator consensus process. Each diversified task was independently reviewed by two expert annotators to identify the active modification aspects. To accurately reflect the primary visual reasoning challenges, annotators were instructed to tag only salient modifications that meaningfully altered the task's logic or difficulty, deliberately filtering out trivial or negligible changes. Any discrepancies were resolved through discussion to achieve full consensus. Based on these validated labels, these numbers show that spatial transformations and object attribute adjustments appear in over 80% of cases, while dynamic patterns, mathematical changes, and object relations appear with varying frequency. This confirms that diversification introduces rich and meaningful variations rather than superficial or isolated edits.

To better illustrate the multi-aspect nature of diversification, we provide three representative examples in Figure 7. In the first example, diversification simultaneously modifies (i) *Spatial Transformation* (rotating the board by 90°), (ii) *Dynamic Patterns* (adjusting layer layouts), (iii) *Object Attributes* (altering colors and positions), and (iv) *Object Relations* (changing the alternating-color rule between adjacent arcs). In the second example, the diversified version changes (i) the *Spatial Transformation* (the direction of object movement), (ii) *Object Attributes* (modifying the colors and angles of arrows and balls), and (iii) *Object Relations* (adding new interactions between objects). In the third example, diversification alters (i) *Dynamic Patterns* (changing the layout of lines), (ii) *Object Attributes* (switching from solid to dashed lines), and (iii) *Object Relations* (moving intersection points from inside the circle to outside the circle). These examples highlight that diversification is not a one-dimensional transformation process, but a structured procedure designed to meaningfully expand the visual reasoning space.

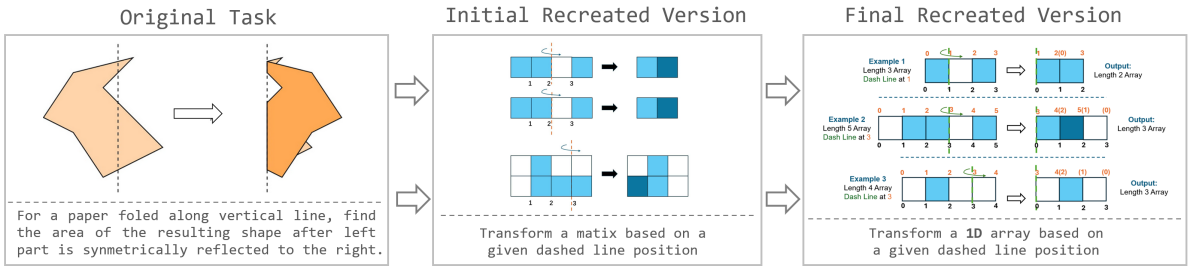


Fig. 9. Task annotation examples illustrating the disagreement resolving process in task recreation. The original problem is sourced from CodeForces (https://codeforces.com/problemset/problem/1381/E).

This multi-aspect annotation paradigm ensures that the resulting tasks probe genuine visual generalization rather than shallow pattern matching.

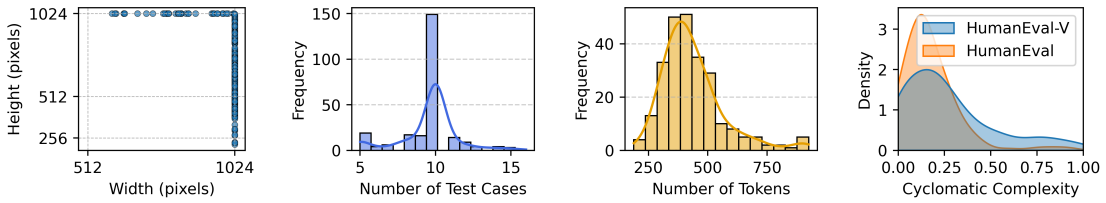
To better illustrate our annotation process, we present an additional full example in Figure 8. This example is based on a CodeForces problem involving polygon folding and area calculation. The provided image illustrates the folding process along dashed lines, showing both initial and final states. For recreation, we simplify this into a matrix folding task in which overlapping sections produce color changes. The input matrix uses two initial colors, which can result in three distinct outcomes after folding. Several illustrative examples clarify the folding mechanics. For diversification, the color addition rule is replaced with numeric addition, requiring models to reason about numerical changes before and after folding.

3.6 Quality Assurance

Our annotation team comprises four experienced programmers, each with over four years of experience in Python programming. To ensure the quality and consistency, we adopt a consensus-based annotation workflow.

At the beginning of each stage, annotators follow a shared set of guidelines and independently annotate their assigned tasks. All intermediate outputs, including distilled reasoning elements, recreated diagrams, diversified task variants, ground-truth code solutions, and diagram descriptions, are then cross-reviewed by the entire team. Any ambiguity, missing visual information, or conceptual inconsistency is discussed immediately in small-group meetings, and tasks are iteratively refined until all annotators reach full agreement. This collaborative adjudication process ensures that no unresolved disagreements remain in the final dataset. To further illustrate how disagreements are handled, Figure 9 presents a representative example from one of the most challenging stages, Task Recreation. In this case, one annotator first proposed an initial recreated version using a matrix-folding formulation that attempted to mirror the visual logic of the original CodeForces problem. During review, the team identified several issues: the diagram lacked explicit axis annotations, the input–output data structures were not clearly defined, and the matrix-based formulation introduced unnecessary coding complexity unrelated to the core visual reasoning skill. Through multiple rounds of collaborative discussion, the team refined the task to focus on the essential reasoning elements: symmetry folding and color merging, and redesigned it using a clearer 1D representation with illustrative examples. The final version, shown on the right of Figure 9, is the result of iterative clarification and consensus-based revision.

As annotation progresses, the team's shared understanding of the benchmark steadily aligns, resulting in fewer conflicts and more efficient refinement in later stages. One annotator additionally performs a final pass to enforce consistent formatting and visual style across all diagrams and coding tasks. Each annotator has contributed 200 hours to this process, prioritizing accuracy and clarity over annotation speed.



(a) Diagram Size (b) Test Cases Count (c) Description Length (d) Coding Complexity

Fig. 10. Distribution statistics of tasks in HumanEval-V, highlighting the variety and coverage in diagram sizes, the number of test cases, description lengths, and coding complexity.

3.7 Benchmark Statistics

To further demonstrate the quality of our benchmark, we conduct statistical analyses on several key aspects and present the distribution of these statistics in Figure 10. First, we strictly control diagram sizes, capping the maximum width or height at 1024 pixels to eliminate the need for high-resolution perception. Second, each task includes at least five test cases, with the majority containing ten, ensuring full statement and branch coverage over human-annotated code solutions. Third, the token length of human-annotated diagram descriptions is predominantly around 400 (measured using tiktoken [57]), demonstrating that our diagrams encapsulate rich visual context. Lastly, our human-annotated code solutions exhibit cyclomatic complexity [22] levels comparable to HumanEval [11], a widely used coding benchmark designed for entry-level programming tasks.

In addition to these statistics, we also analyze the distribution of the Core Knowledge aspects required for solving the tasks in HumanEval-V. As shown in Figure 11, each Core Knowledge category appears in a substantial number of tasks, and no single capability overwhelmingly dominates the dataset. While the distribution is not strictly uniform, it faithfully reflects the natural prevalence of visual reasoning skills observed in real-world programming diagrams. For example, spatial transformations (e.g., rotation, grouping), geometric or structural objects (e.g., grids, arrows, points), and common mathematical operations (e.g., comparison, basic arithmetic) occur more frequently.

The Core Knowledge taxonomy itself was derived through a bottom-up annotation process: annotators first decomposed each diagram into its primitive reasoning steps and then grouped tasks with similar visual patterns. The naming of the final capability categories draws inspiration from the design of taxonomies in prior multimodal benchmarks, including MathVista and MathVerse, which focus on mathematical and geometric concepts, as well as MMBench and MME, which concentrate on spatial, topological, and dynamic reasoning. While inspired by these works, our taxonomy is adapted specifically to the visual semantics of programming diagrams, and only categories that frequently appeared in HumanEval-V were retained.

4 RQ1: How Well Do State-of-the-Art LMMs Perform on HumanEval-V?

4.1 Evaluation Setup

To comprehensively evaluate the visual reasoning and code generation capabilities of LMMs, we design and implement multiple evaluation strategies, as illustrated in Figure 12. Our approach is motivated by two key considerations.

First, inspired by prior work [49, 70] in mathematics and programming, we incorporate a Chain-of-Thought (CoT) prompting variant. Existing studies [77] have shown that LLMs often achieve better performance when explicitly instructed to reason step by step before producing their final answers. Given the high reasoning demand

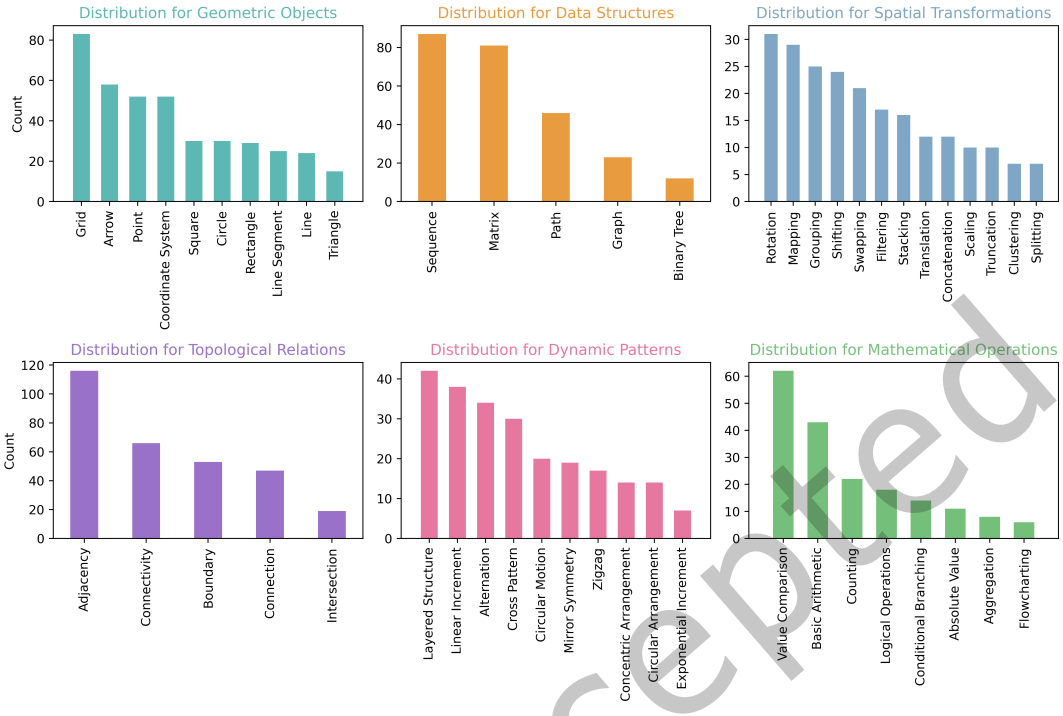


Fig. 11. Distribution of Core Knowledge labels in HumanEval-V. Each bar represents the number of tasks requiring a particular visual reasoning capability.

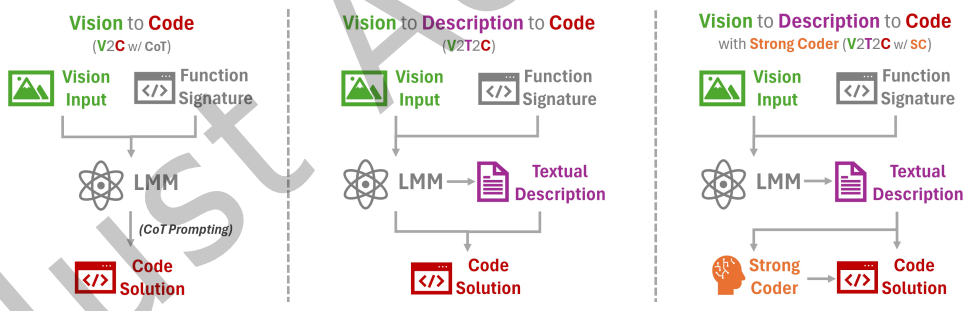


Fig. 12. Evaluation pipelines used in our experiments: (1) Direct translation from visual context to code (V2C); (2) V2C with an optional Chain-of-Thought prompt (V2C w/ CoT); (3) Translation of visual context into a textual description, which is then processed to generate code (V2T2C); and (4) A variant of V2T2C that employs a stronger coder model to generate the code solution from the textual description alone (V2T2C w/ SC).

of our tasks, we aim to investigate whether CoT prompting can similarly improve the performance of LMMs in the context of algorithmic diagrams.

Second, we propose a decoupled pipeline for diagram description and code generation. Most current LMM architectures consist of a vision encoder and a language model decoder, with their training heavily reliant on

```
You are an exceptionally intelligent coding assistant with a deep understanding of Python programming and a keen ability to interpret visual
data. Your responses are consistently accurate, reliable, and thoughtful.

**Objective:**
You will be presented with a Python programming problem and an accompanying image. Please complete the function based on the provided image and
code context.

**Note**
- Remember, the signature by itself does not contain the entire problem; the image provides critical details.
- Observe the image closely and determine how its visual elements correspond to the problem's inputs, outputs, operations, calculations,
patterns (static/dynamic), and conditions.
- Please generate the complete code solution, including its function signature and body, formatted in a single Python code block, **without any
additional text or explanation**.

**Code Context:**
```python
{function_signature}
```
```

Scenario: $P_{V2C}(D, \sigma)$

```
You are an exceptionally intelligent coding assistant with a deep understanding of Python programming and a keen ability to interpret visual
data. Your responses are consistently accurate, reliable, and thoughtful.

**Objective:**
You will be presented with a Python programming problem and an accompanying image. Please complete the function based on the provided image and
code context.

**Note**
- {problem_category_specification}
- Remember, the signature by itself does not contain the entire problem; the image provides critical details.
- Observe the image closely and determine how its visual elements correspond to the problem's inputs, outputs, operations, calculations,
patterns (static/dynamic), and conditions.
- First summarize the important clues or findings and write a step-by-step analysis.
- Then generate the complete code solution,including the function signature and body,formatted in a single Python code block.

**Code Context:**
```python
{function_signature}
```
```

Scenario: $P_{V2C}(D, \sigma, I_{CoT})$

```
**Instructions:**
You will receive a Python programming problem and an accompanying image for analysis:

**Code Context:**
```python
{function_signature}
```

1. **Analyze the Function Signature**
   Examine the provided function signature (its input, output, and goal) and identify any missing context. Remember, the signature by itself
does not contain the entire problem; the image provides critical details.
2. **Examine the Image**
   Observe the image closely and determine how its visual elements correspond to the problem's inputs, outputs, operations, calculations,
patterns (static/dynamic), and conditions.
   - First, describe the visual elements you see.
   - Next, list the important facts from the image that are relevant for understanding the problem.
   - Finally, deduce any missing information from the problem based on the image.

Response Format:
Please structure your response in three main sections (use Markdown H1 headers):
1. **# Problem Restatement**
   Provide a concise restatement of the problem, including relevant background and requirements.
2. **# Visual Facts**
   List the facts directly observed from the image that are necessary for interpreting or solving the problem.
3. **# Visual Patterns**
   Summarize any objects, operations, transformations, patterns, conditions, and relationships inferred from these facts.
Important Note:
- Clearly separate facts (what you directly see in the image) from patterns (what you infer based on those facts).
- If complex visual information is difficult to express in plain language, use formal notation (mathematical or pseudo-code).
- State only what you are sure of; do not introduce assumptions not supported by the image or give vague conclusions.
- **Do not** include any code implementation in your response.
```

Scenario: $P_{V2T}(D, \sigma)$

```
**Instructions:**
You are an exceptionally intelligent coding assistant that consistently delivers accurate and reliable responses to user instructions. Please
complete the function based on the provided problem specification, code context, and accompanying image (if provided). Return the complete
solution, including the function signature, in a single response, formatted within a Python code block.

**Problem Specification:**
```markdown
{problem_specification}
```

Code Context:
```python
{function_signature}
```
```

Scenario: $P_{T2C}(PS, \sigma)$

Fig. 13. Prompting templates used for the four scenarios. {function_signature} and {problem_specification} serve as placeholders for the respective content.

large-scale image-text pairs to align visual and linguistic representations [13, 71]. However, such training may bias models toward strong vision-language alignment, while their ability to perform vision-to-code reasoning could remain underdeveloped. To probe this potential gap, we introduce a two-stage evaluation: LMMs first generate structured diagram descriptions, and then an external code generation model produces solutions based on these descriptions. This approach allows us to better isolate and evaluate the models' visual perception and reasoning abilities independently from code generation.

4.1.1 Prompt Design. Based on these motivations, we design four distinct prompting scenarios. The corresponding prompt templates for each scenario are provided in Figure 13.

(1) *Direct Code Generation:* The model directly generates code given the diagram D and function signature σ, denoted as $P_{V2C}(D, \sigma)$.
(2) *Chain-of-Thought (CoT):* The model is prompted to outline its reasoning process step by step before generating the final code. This is implemented by appending a zero-shot CoT instruction I_{CoT}, and is denoted as $P_{V2C}(D, \sigma, I_{CoT})$.
(3) *Intermediate Textual Representation:* The model is first asked to generate a structured textual problem specification (PS) based on D and σ, denoted as $P_{V2T}(D, \sigma)$. This specification comprises three key sections: *Problem Restatement*, *Visual Facts*, and *Visual Patterns*. The design of this intermediate representation is informed by our own annotation process, which we found effective for capturing the essential context and reasoning steps.
(4) *Code Generation from Text:* Finally, a model generates code based solely on the intermediate specification PS (rather than the original diagram D), denoted as $P_{T2C}(PS, \sigma)$. This setting is particularly useful for decoupling the evaluation of visual understanding from code generation.

Throughout the design of our evaluation pipeline, we encountered and addressed several practical challenges. For example, to facilitate downstream processing and improve code parsing success rates, we instruct models to output code within a single markdown code block and to avoid producing multiple code segments. For generating diagram descriptions, we provide detailed instructions that enforce a structured problem specification, ensuring the inclusion of *Problem Restatement*, *Visual Facts*, and *Visual Patterns*. This structured approach enables more consistent and comprehensive expression of the visual context. Over multiple rounds of prompt refinement, we further tuned instructions to handle common model-specific failure cases, such as incomplete outputs, hallucinated information, or deviation from the desired output format.

4.1.2 Models to Evaluate. We evaluate 27 state-of-the-art LMMs, including a representative mix of leading proprietary and open-weight models. Our evaluation covers five of the latest proprietary models: Claude 3.5 Sonnet, GPT-4o, GPT-4o-mini, Gemini 1.5 Pro, and Gemini 1.5 Flash. We also assess 17 top-performing open-weight models spanning various parameter sizes, including InternVL 2.5 (4/8/26/78B), InternVL 3.5 (4/8/14B), Qwen2-VL (7/72B), Qwen3-VL (4/8B), Pixtral (12/124B), LLaVA-OV (7/72B), Llama-3.2-V (11/90B), Molmo-D (7/72B), Chameleon (7/30B), and Phi-3.5-V (4B). In Table 3, we provide a detailed list of LMMs used in our experiments. For each model, we specify the number of parameters and include direct links to relevant reports or Huggingface repositories for further reference.

4.1.3 Hyper-Parameters & Post-processing. We apply two distinct decoding strategies for both code and description generation. First, we use greedy decoding to produce a single deterministic output, assessing model performance in a constrained setting. Additionally, we employ a sampling method with $Top\_p = 0.95$, $Top\_k = 20$, and a temperature of 0.8 to generate diverse outputs, allowing us to evaluate the models' ability to produce correct solutions when given multiple attempts. We set the maximum output length to 2048 tokens for both code and description generation. To facilitate extraction, we prompt the models to encapsulate their generated code within Markdown-style code blocks. We then apply an abstract syntax tree parser to detect and retrieve

Table 3. List of LMMs with their parameter sizes and links to the official reports or Huggingface repositories.

Models	Params	Links
Proprietary LMMs		
GPT-4o (0806) [55]	-	https://platform.openai.com/docs/models/gpt-4o
GPT-4o-mini (0718) [55]	-	https://platform.openai.com/docs/models/gpt-4o
Claude 3.5 Sonnet (1022) [2]	-	https://docs.anthropic.com/claude/docs
Gemini 1.5 Pro (002) [23]	-	https://ai.google.dev/gemini-api/docs/models/gemini
Gemini 1.5 Flash (002) [23]	-	https://ai.google.dev/gemini-api/docs/models/gemini
Open-weight LMMs with more than 70B parameters		
Pixtral [1]	124B	https://huggingface.co/mistralai/Pixtral-Large-Instruct-2411
Llama-3.2-V 90B [24]	88.8B	https://huggingface.co/meta-llama/Llama-3.2-90B-Vision-Instruct
InternVL 2.5 78B [13]	78.4B	https://huggingface.co/OpenGVLab/InternVL2_5-78B
Owen2 VL 72B [71]	73.4B	https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct
Molmo-D 72B [17]	73.3B	https://huggingface.co/allenai/Molmo-72B-0924
LLaVA-OV 72B [43]	73.2B	https://huggingface.co/llava-hf/llava-onevision-qwen2-72b-ov-chat-hf
Open-weight LMMs with fewer than 70B parameters		
Chameleon 30B [65]	34.3B	https://huggingface.co/facebook/chameleon-30b
InternVL 2.5 26B [13]	25.5B	https://huggingface.co/OpenGVLab/InternVL2_5-26B
InternVL 3.5 14B [73]	15B	https://huggingface.co/OpenGVLab/InternVL3_5-14B
Pixtral 12B [1]	12.0B	https://huggingface.co/mistralai/Pixtral-12B-2409
Llama-3.2-V 11B [24]	10.7B	https://huggingface.co/meta-llama/Llama-3.2-11B-Vision-Instruct
Qwen3 VL 8B [81]	9B	https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct
Qwen2 VL 7B [71]	8.3B	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct
InternVL 3.5 8B [73]	9B	https://huggingface.co/OpenGVLab/InternVL3_5-8B
InternVl 2.5 8B [13]	8.1B	https://huggingface.co/OpenGVLab/InternVL2_5-8B
LLaVA-OV 7B [43]	8.03B	https://huggingface.co/llava-hf/llava-onevision-qwen2-7b-ov-chat-hf
Molmo-D 7B [17]	8.02B	https://huggingface.co/allenai/Molmo-7B-D-0924
Chameleon 7B [65]	7.04B	https://huggingface.co/facebook/chameleon-7b
Qwen3 VL 4B [81]	4B	https://huggingface.co/Qwen/Qwen3-VL-4B-Instruct
Phi-3.5-V 4B [53]	4.2B	https://huggingface.co/microsoft/Phi-3.5-vision-instruct
InternVL 3.5 4B [73]	5B	https://huggingface.co/OpenGVLab/InternVL3_5-4B
InternVL 2.5 4B [13]	3.7B	https://huggingface.co/OpenGVLab/InternVL2_5-4B

generated import statements, class definitions, and function definitions. These components are concatenated to form the final code solution.

4.1.4 Evaluation Metrics. Following established practices in code generation evaluation [9, 11], we use the pass@k metric to assess functional correctness. A task is considered passed if at least one of the k selected solutions passes all test cases, and pass@k is the percentage of solved tasks. We report results for $k = 1, 3$. In the *V2C* setting, we generate n code samples per task and randomly select k for evaluation. For greedy decoding, $n = 1$ for pass@1, while for sampling-based evaluation, $n = 6$ for pass@3. In the *V2T2C* setting, we first sample six problem specifications per task, then use greedy decoding to generate one code solution for each problem specification, resulting in six solutions per task for pass@3 computation.

4.2 Evaluation Results

We present the benchmarking results of 27 LMMs in Table 4, where each reported number denotes the proportion of tasks correctly solved out of all 253 `HumanEval-V` tasks, under the four evaluation pipelines introduced in Figure 12. Based on these results, we highlight the following key findings:

Table 4. Performance across different settings. Models are ranked based on the 🏅 column. The **best** and second-best performances in each column are highlighted. The numerical values are color-coded to indicate performance changes relative to the corresponding pass@k values in the **V2C** column: green represents improvement, red indicates decline, and yellow denotes minimal change.

Models	V2C		V2C w/ CoT		V2T2C		V2T2C w/ GPT-4o	
pass@k	k=1	k=3	k=1	k=3	k=1	k=3	k=1	k=3 🏅
Proprietary LMMs								
Claude 3.5 Sonnet	**28.1**	**37.9**	**36.8**↑8.7	**47.9**↑10.0	**33.2**↑5.1	**43.6**↑5.7	**31.6**↑3.5	**43.7**↑5.8
GPT-4o	24.1	33.8	27.7↑3.6	40.0↑6.2	26.5↑2.4	40.5↑6.7	26.5↑2.4	40.5↑6.7
Gemini 1.5 Pro	23.3	26.9	22.9↓0.4	34.1↑7.2	28.5↑5.2	36.4↑9.5	26.9↑3.6	37.3↑10.4
Gemini 1.5 Flash	15.4	20.5	17.4↑2.0	24.9↑4.4	15.8↑0.4	22.0↑1.5	18.6↑3.2	27.2↑6.7
GPT-4o-mini	9.90	16.0	15.8↑5.9	21.1↑5.1	14.2↑4.3	22.7↑6.7	18.2↑8.3	24.6↑8.6
Open-weight LMMs with more than 70B parameters								
Pixtral 124B	**12.6**	**20.3**	**16.6**↑4	**28.1**↑7.8	**21.3**↑8.7	**29.9**↑9.6	21.3↑8.7	**31.6**↑11.3
InternVL 2.5 78B	12.3	19.7	13.4↑1.1	27.3↑7.6	17.8↑5.5	25.7↑6	**21.7**↑9.4	31.4↑11.7
Qwen2 VL 72B	9.10	15.7	14.2↑5.1	19.4↑3.7	10.7↑1.6	19.1↑3.4	16.6↑7.5	25.1↑9.4
LLaVA-OV 72B	6.70	7.70	6.30↓0.4	11.4↑3.7	10.7↑4	13.1↑5.4	13.8↑7.1	19.7↑12
Molmo-D 72B	3.20	4.80	3.20↑0.0	8.80↑4.0	1.60↓1.6	7.00↑2.2	5.10↑1.9	14.2↑9.4
Llama-3.2-V 90B	4.30	6.10	4.00↓0.3	8.20↑2.1	5.90↑1.6	10.9↑4.8	4.70↑0.4	11.0↑4.9
Open-weight LMMs with fewer than 70B parameters								
Qwen3 VL 8B	9.5	17.2	**16.2**↑6.7	**35.7**↑18.5	**15.8**↑6.3	**29.1**↑11.9	**20.2**↑10.7	**35.3**↑18.1
Qwen3 VL 4B	5.9	14.5	11.1↑5.2	23.7↑9.2	11.1↑5.2	23.0↑8.5	18.6↑12.7	29.3↑14.8
InternVL3.5 14B	**9.9**	**26.8**	12.6↑2.7	27.8↑1.0	9.5↓0.4	27.6↑0.8	13.8↑3.9	29.0↑2.2
InternVL3.5 4B	3.6	8.4	7.5↑3.9	12.9↑4.5	5.5↑1.9	12.9↑4.5	8.3↑4.7	21.4↑13.0
Pixtral 12B	4.0	5.9	6.3↑2.3	12.2↑6.3	5.5↑1.5	12.6↑6.7	13.8↑9.8	21.3↑15.4
InternVL3.5 8B	7.1	8.4	7.9↑0.8	12.8↑0.4	7.2↑0.1	11.0↑2.6	8.3↑ 1.2	16.7↑ 8.3
InternVL 2.5 26B	3.6	6.6	4.3↑0.7	6.8↑0.2	2.8↓0.8	6.7↑0.1	8.3↑4.7	16.7↑10.1
Qwen2 VL 7B	0.8	3.3	1.6↑0.8	3.9↑0.6	2.4↑1.6	6.3↑3.0	6.3↑5.5	14.7↑11.4
InternVL 2.5 8B	0.8	2.0	0.8↑0.0	3.7↑1.7	1.2↑0.4	3.3↑1.3	5.1↑4.3	13.6↑11.6
InternVL 2.5 4B	1.2	4.1	3.2↑2.0	3.4↓0.7	3.2↑2.0	5.8↑1.7	5.9↑4.7	13.5↑9.4
LLaVA-OV 7B	2.0	1.9	1.6↓0.4	2.4↑0.5	2.0↑0.0	3.2↑1.3	5.1↑3.1	10.2↑8.3
Phi-3.5-V 4B	0.0	0.0	0.0↑0.0	0.0↑0.0	0.0↑0.0	0.0↑0.0	5.9↑5.9	9.40↑9.4
Llama-3.2-V 11B	2.0	2.0	1.6↓0.4	3.9↑1.9	2.0↑0.0	5.2↑3.2	4.0↑2.0	8.80↑6.8
Molmo-D 7B	1.2	1.0	0.4↓0.8	1.8↑0.8	0.8↓0.4	1.0↑0.0	2.8↑1.6	8.40↑7.4
Chameleon 7B	0.0	0.0	0.0↑0.0	0.2↑0.2	0.0↑0.0	0.0↑0.0	1.2↑1.2	2.20↑2.2
Chameleon 30B	0.0	0.0	0.0↑0.0	0.2↑0.2	0.4↑0.4	0.0↑0.0	0.0↑0.0	1.90↑1.9

(1) *LMMs generally achieve their best performance under the V2T2C w/ GPT-4o setting.* Across the board, models perform markedly better when visual understanding and code generation are decoupled, especially for open-weight models with fewer than 70B parameters. For instance, while these smaller models typically struggle in the direct *V2C* setting (with many scoring near or at zero), their performance increases substantially when leveraging the V2T2C pipeline with GPT-4o as the code generator. This validates the effectiveness of our decoupled evaluation design and suggests that vision-language alignment is more reliably achieved than vision-code alignment in current LMMs. Additionally, CoT prompting and the

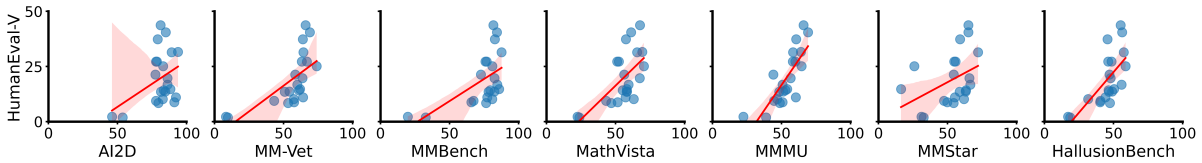


Fig. 14. Correlations between the selected multimodal benchmarks. Each subplot displays the relationship between model's performance on two benchmarks.

V2T2C pipeline both yield improvements, particularly for larger or more capable models, confirming the benefit of explicit reasoning in tasks with substantial visual complexity.

(2) *Open-weight LMMs still lag behind top proprietary models, with notable exceptions.* While many high-capacity open-weight models, such as Pixtral 124B and InternVL 2.5 78B, have narrowed the performance gap, they generally continue to trail the strongest proprietary LMMs. For instance, Claude 3.5 Sonnet achieves a pass@3 of 47.9% in the V2C w/ CoT setting and 43.7% in the V2T2C w/ GPT-4o setting, whereas top open-weight baselines like Pixtral 124B reach only 28.1% and 31.6% in the same settings. This discrepancy is even more pronounced for smaller-scale models, many of which, such as Chameleon and Phi-3.5-V, fail to solve nearly any task under direct code generation. However, recent model families illustrate rapid progress in open-weight LMMs. In particular, *Qwen3-VL 8B* surpasses *Qwen2-VL 72B* all evaluation settings on `HumanEval-V`. Despite this encouraging exception, the overall trend still shows that most existing open-weight LMMs face substantial challenges on our benchmark.

(3) *Certain LMMs exhibit anomalously poor performance.* Some models stand out for their unexpectedly low scores given their size or reported capabilities. Notably, Molmo-D, Llama-3.2-V, and the Chameleon series consistently underperform relative to their peers. Phi-3.5-V, in particular, is unable to complete any tasks in the V2C setting, and only achieves modest improvement when paired with an external code generator (pass@3 = 9.4%). This suggests that not all recent LMMs have robust or well-aligned multimodal reasoning or coding capabilities, emphasizing the selectivity and difficulty of our benchmark.

4.3 Comparison with Other Multimodal Benchmarks

To understand whether `HumanEval-V` measures unique weaknesses in current LMMs that are not captured by existing multimodal benchmarks, we conduct a comprehensive correlation analysis. Specifically, we select seven widely adopted benchmarks, including AI2D [39], MMVet [85], MMBench [48], MathVista [49], MMMU [87], MMStar [10], and HallusionBench [26], which cover a broad range of multidisciplinary visual and reasoning abilities. For each of the 22 LMMs, we collect reported performance from the OpenVLM Leaderboard [21], as well as corresponding papers and reports. We summarize these results in Table 5, and visualize the relationships between benchmarks for all models using regression pair plots in Figure 14. For `HumanEval-V`, we use the pass@3 scores under the challenging *V2T2C w/ GPT-4o* setting to facilitate a fair comparison.

The results reveal several noteworthy trends. First, open-weight LMMs with more than 70B parameters (such as Pixtral, InternVL 2.5, and Qwen2 VL) consistently achieve high or even state-of-the-art scores on many established benchmarks, sometimes outperforming proprietary models like GPT-4o and Claude 3.5 Sonnet. However, on `HumanEval-V`, these same models experience a dramatic drop in performance. Furthermore, we observe that many current benchmarks fail to clearly distinguish between models of different scales or architectures. For example, on MMBench and MathVista, a large group of both proprietary and open-weight LMMs cluster near the top scores, making it difficult to discern real differences in model capabilities.

Table 5. A performance comparison of 22 LMMs across HumanEval-V and seven popular multimodal benchmarks. Models are ranked according to the 🏅 column. Results for HumanEval-V correspond to the *V2T2C w/ GPT-4o* pass@3 setting from Table 4. The top two results for each column are highlighted in **bold**.

Models	AI2D	MM-Vet	MMBench	MathVista	MMMU	MMStar	HallusionBench	HumanEval-V 🏅
Proprietary LMMs								
Claude 3.5 Sonnet	81.2	66.0	81.7	67.7	**65.9**	65.1	55.1	**43.7**
GPT-4o	84.9	**69.1**	84.3	61.3	**69.2**	65.1	56.2	**40.5**
Gemini 1.5 Pro	79.1	64.0	82.8	57.5	60.6	59.1	45.6	37.3
Gemini 1.5 Flash	78.5	63.2	76.9	51.2	58.2	55.8	48.5	27.2
GPT-4o-mini	77.8	66.9	76.0	52.4	60.0	54.8	46.1	24.6
Open-Weight LMMs								
Pixtral 124B	**93.8**	-	-	**69.4**	64.0	-	-	31.6
InternVL 2.5 78B	89.2	64.4	**87.7**	65.6	58.3	**72.1**	**57.4**	31.4
Qwen2 VL 72B	83.0	**74.0**	81.0	**70.5**	64.5	25.9	**58.7**	25.1
LLaVA-OV 72B	86.2	63.7	82.6	67.5	56.6	**65.8**	47.9	19.7
Molmo-D 72B	83.4	61.1	79.5	55.2	52.8	63.3	46.4	14.2
Llama-3.2-V 90B	**92.3**	64.1	77.3	57.3	60.3	55.3	44.1	11.0
Pixtral 12B	77.4	58.5	72.7	56.3	44.1	54.5	47.0	21.3
InternVL 2.5 26B	86.2	60.0	84.6	59.4	50.7	66.5	55.8	16.7
Qwen2 VL 7B	88.3	62.0	**85.9**	58.2	54.1	16.3	50.4	14.7
InternVL 2.5 8B	84.6	54.3	82.5	58.3	51.2	63.2	49.0	13.6
InternVL 2.5 4B	81.4	50.9	78.2	58.1	48.3	58.7	46.6	13.5
LLaVA-OV 7B	82.8	57.5	80.9	63.2	46.8	61.9	31.6	10.2
Phi-3.5-V 4B	77.8	43.2	67.4	43.2	44.6	47.5	40.5	9.4
Llama-3.2-V 11B	91.1	57.6	65.8	51.5	50.7	49.8	40.3	8.8
Molmo-D 7B	79.6	53.3	76.5	46.9	48.7	54.4	47.4	8.4
Chameleon 7B	46.0	8.3	19.8	22.5	22.4	31.1	17.1	2.2
Chameleon 30B	53.7	9.7	32.7	23.8	38.8	32.7	18.6	1.9

This phenomenon highlights the unique challenge presented by HumanEval-V and exposes current blind spots in multimodal evaluation. There are several possible reasons behind these observations:

(1) *Evaluation Task Format.* Most existing multimodal benchmarks use multiple-choice or short-answer questions, which are easier for models to hack and do not demand deep, step-by-step understanding. In contrast, the open-ended code generation tasks in HumanEval-V require models to fully comprehend all visual details and conditions present in the diagrams, making them far more challenging.
(2) *Task Domain and Data Distribution.* Many popular benchmarks focus on domains such as chart reasoning, science diagrams, and math exams, where large quantities of training data are readily available and incorporated into model pretraining. By contrast, coding and algorithmic diagram tasks in HumanEval-V are far less common in public datasets, and tasks are manually created to reduce the risk of data contamination. This further increases their novelty and difficulty for LMMs.

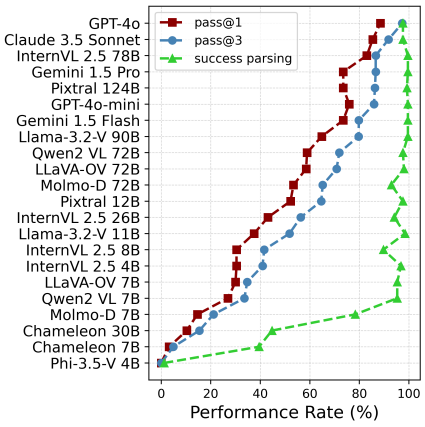


Fig. 15. LMMs achieve strong performance when provided with human-written problem specifications.

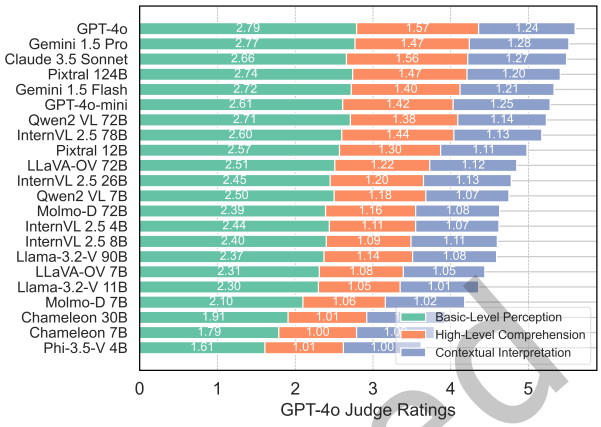


Fig. 16. LLM-as-Judge ratings show limited discriminative ability for LMMs in the *V2T2C* setting.

Findings from RQ1

(1) HumanEval-V presents a substantially more challenging task than existing benchmarks, with even top-performing models achieving only modest accuracy.
(2) Most LMMs perform significantly better under the V2T2C setting than direct V2C, indicating that vision-to-code alignment remains a major bottleneck.
(3) Performance on HumanEval-V shows weak correlation with existing multimodal benchmarks, suggesting that current evaluations fail to capture the specific reasoning skills required for visual programming tasks.

5 RQ2: What Limitations Do Current LMMs Exhibit When Solving HumanEval-V Tasks?

5.1 Coding Ability Is Not the Main Bottleneck

To better understand the source of performance bottlenecks on HumanEval-V, we design an auxiliary experiment aimed at isolating the coding capabilities of LMMs. Specifically, we examine whether these models can generate correct code when provided with a ground-truth textual description of the visual content. This allows us to assess their programming proficiency independently of their visual reasoning abilities.

To support this evaluation, we leverage metadata collected during the construction of our benchmark. For each task, annotators have provided Problem Specifications (PS), which are structured descriptions of the associated diagram. As introduced in Figure 13, each PS adheres to a structured format comprising three components: *Problem Restatement*, *Visual Facts*, and *Visual Patterns*. This structure is found to be effective in conveying both the semantic content and the spatial-logical structure of the diagram, making it suitable for evaluating code generation in the absence of visual input.

Using these problem specifications as input, we evaluate 22 LMMs on a text-to-code task, where the goal is to generate code from the human-written PS alone. The results, presented in Figure 15, demonstrate that most LMMs exhibit strong coding abilities under this setting. In many cases, performance improves significantly compared to the original visual reasoning task. For instance, GPT-4o achieves a pass@3 rate of 96.5%, in contrast to its 40.5% pass@3 score in the *V2T2C* setting. Smaller models also benefit substantially. For example, InternVL

2.5 4B achieves a pass@3 rate of 41.3%, in contrast to its 13.5% pass@3 score in the *V2T2C* setting. We further measure the syntactic correctness of generated code using the *success parsing rate*, calculated via Pylint [78], and observe consistently high values across models. Nevertheless, a subset of models, particularly Phi-3.5-V 4B and Chameleon 7B, fail to generate functionally correct code in most instances. Their *success parsing rates* remain low, indicating persistent limitations in both syntactic and semantic code generation. These models appear to struggle with coding proficiency itself, not just visual understanding.

As an additional control, we also test a setting where models are given only the function signature, without access to either the diagram or its textual description. In this case, none of the five proprietary models is able to generate correct solutions. This further emphasizes the importance of visual context in solving HumanEval-V tasks.

Taken together, these findings lead to two conclusions. First, while a few smaller models still face challenges, most current LMMs exhibit competent code generation capabilities when provided with accurate textual context. Second, the substantial performance gap between this setting and the original *V2T2C* setting strongly suggests that the key limitation lies not in coding ability but in visual understanding.

5.2 Visual Reasoning as the Dominant Limitation

To better understand the limitations of current LMMs in visual reasoning, we conduct a targeted evaluation of the Problem Specifications (PS) generated in the *V2T2C* setting. Specifically, we employ GPT-4o as an automatic judge. The input to the judge consists of the PS generated by the model and the human-annotated ground-truth PS. GPT-4o is instructed to compare the two and identify discrepancies across three capability dimensions. These dimensions include: *Basic-Level Perception*, which reflects recognition of low-level visual elements such as shapes, colors, positions, and numeric labels; *High-Level Comprehension*, which captures understanding of structural relationships, visual patterns, and transformations; and *Contextual Interpretation*, which measures whether the description is complete, precise, and free from hallucinations or ambiguity. Each dimension is rated on a three-point scale, where *1* indicates a severe error, *2* indicates partial understanding, and *3* indicates near-perfect performance.

The results, shown in Figure 16, indicate that while LMMs generally perform well in basic perceptual understanding, they exhibit notable weaknesses in high-level comprehension and precise expression. However, the LLM-as-judge framework is not designed to provide an absolute ranking of models and may be sensitive to reference bias. Its outputs are more reliable for relative comparisons across dimensions or between success and failure cases. To better interpret the evaluation scores, we conduct a comparative analysis based on task outcomes. Specifically, we compare the average ratings on tasks where models generate correct solutions versus those where they fail. For GPT-4o, the average scores on passed tasks are 2.9 for basic perception, 2.0 for high-level comprehension, and 1.3 for contextual interpretation. On all tasks, the averages are 2.8, 1.6, and 1.2, respectively. This comparison suggests that failure cases are closely linked to deficits in high-level visual reasoning, while differences in basic perception and expression are relatively minor. These results reinforce our conclusion that the primary bottleneck in current LMM performance on HumanEval-V lies in high-level visual reasoning capabilities.

5.3 Fine-Grained Error Analysis Across Task Types and Capability Dimensions

Building on the previous section, where we identified high-level visual reasoning as a primary bottleneck for current LMMs, we further investigate which specific aspects (introduced in Figure 3) of visual reasoning present the greatest challenges. To this end, we perform a systematic statistical analysis that correlates model pass rates with three levels of task annotation: *task type*, *general capability dimension*, and *fine-grained capability aspect*. These categories capture the diverse forms of reasoning required to interpret the diagrams in HumanEval-V. The

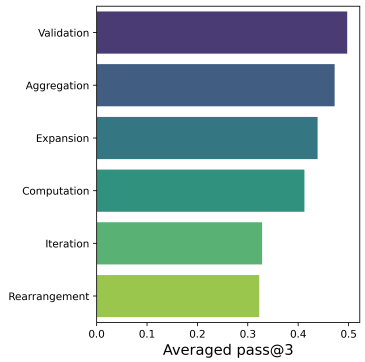


Fig. 17. Pass rates of LMMs across different task types.

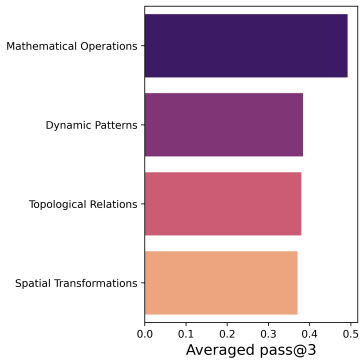


Fig. 18. Pass rates of LMMs across main capability dimensions.

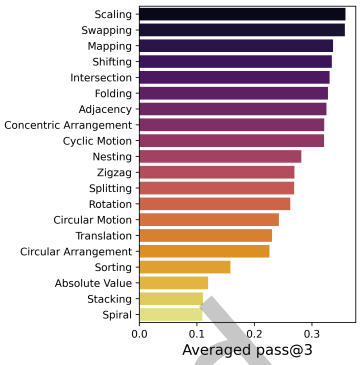


Fig. 19. Pass rates of LMMs across specific capability aspects.

results are summarized in Figure 17, Figure 18, and Figure 19. In this analysis, the pass rate is computed as the average pass@3 for the five proprietary models under the *V2T2C w/ SC* setting.

The results show clear variation in LMM performance across task types. For instance, models achieve the highest pass rates on *Validation* and *Aggregation* tasks, which typically involve confirming the presence of certain patterns or combining multiple visual features into a summary statistic. In contrast, tasks labeled as *Iterative Calculation* and *Transformation* have notably lower pass rates, averaging around 32%. These task types require models to follow a sequence of spatial or arithmetic operations over multiple steps, or to reason about the effects of geometric transformations such as rotation or reflection. The performance gap suggests that current LMMs are particularly limited in their ability to track visual state through iterative reasoning or to simulate the effects of spatial changes.

When considering broader capability dimensions, such as *Mathematical Operations*, *Dynamic Patterns*, *Topological Relations*, and *Spatial Transformations*, we observe only modest differences in average performance, with most categories yielding pass@3 scores in the range of 38-48%. A more fine-grained breakdown by specific capability aspect reveals persistent weaknesses in certain types of reasoning. For example, models exhibit substantial difficulty with dynamic visual patterns such as *spirals*, *zigzags*, and *circular arrangements*, as well as spatial operations like *stacking*, *translation*, and *splitting*. These cases often require not only understanding the static properties of visual elements, but also reasoning about movement, repetition, or the spatial reorganization of parts. Similarly, mathematical operations including *sorting* and *absolute value computation* are frequent sources of error. These findings provide further evidence that the main challenges for current LMMs lie in mastering advanced spatial and logical reasoning, rather than in basic visual perception or low-level feature extraction.

In addition to these aggregate trends, we further examine the models that score near zero under the *V2C* and *V2T2C* setting, particularly smaller open-weight models such as Phi-3.5-V, Chameleon, Llama-3.2-V, and Molmo-D. A qualitative case study (Figure 20) reveals that these failures do not stem from a single underlying issue but instead reflect *distinct failure modes* across model families. For example, Phi-3.5-V frequently fails at the instruction-following stage, resulting in diagram descriptions that are excessively brief and omit essential details. Chameleon exhibits errors at the level of basic visual perception, such as miscounting objects in simple scenes. In contrast, models like Llama-3.2-V and Molmo-D struggle primarily with higher-level reasoning: they misinterpret object relationships, misunderstand motion patterns, or fail to infer the algorithmic intent of the diagram. These observed failures align closely with the fine-grained capability gaps identified in this section,

Image:

Test Cases:

```
assert solution(['A','B','C'], 2)==1;
assert solution(['A'], 1)==2;
assert solution...
```

Function Signature:

```
def solution(swaps: List[str], initial_position: int) -> int:
    """
    Determine the final position of the ball after a series of swaps.

    Parameters:
    swaps (List[str]): A list of operation ids ('A', 'B', 'C') representing the sequence of cup swap operations.
    initial_position (int): The index of the cup where the ball is initially placed. (1-indexed)

    Returns:
        int: The index of the cup where the ball is finally located. (1-indexed)
    """
```

Incorrect Diagram Description Generated by Phi-3.5-V 4B: *Failed to recognize the shapes and numbers*

The image shows a sequence of five cylinders connected in a line, with numbers 1 to 5 on them. Each cylinder is connected to the next with a spring, and there are arrows indicating the direction of the connection. The cylinders labeled 1 and 4 are highlighted with a red circle on the number inside. The image is divided into two sections by a dashed vertical line, with labels "A" on the left and "B" on the right. The connection between the cylinders is shown with curved arrows, suggesting movement or process flow from one end to the other.

Incorrect Diagram Description Generated by Chameleon 30B:

Visual Facts:
* There are three cups labeled 1, 2, and 3. *Failed to count the cups*
* The ball is initially placed in cup 1.
* The operations 'A', 'B', and 'C' represent cup swaps, where the ball is moved from one cup to another.
* The operations are applied in the order given in the list.

Visual Patterns:
* The operations are applied in a sequence, with each operation affecting the position of the ball.
* The ball is moved from one cup to another according to the operation performed.
* The final position of the ball is determined by the sequence of operations and their corresponding cup swaps.

Incorrect Diagram Description Generated by Llama-3.2-V 11B:

Visual Facts
* The image shows a sequence of cups with numbers on them, representing the initial positions of the ball.
* There are three types of operations: 'A', 'B', and 'C', which correspond to different swap patterns.
* Each operation moves the ball to a new cup based on the current position and the type of operation.
* The final position of the ball is indicated by a red circle in the last cup.

Visual Patterns
* Operation 'A' moves the ball two positions to the right.
* Operation 'B' moves the ball one position to the left. *Failed to recognize move patterns*
* Operation 'C' moves the ball three positions to the right.
* The ball's movement is determined by the type of operation and its current position.
* The final position of the ball is the result of applying the sequence of operations to the initial position.

Incorrect Diagram Description Generated by Molmo-D 7B:

 # Visual Facts
- There are three sections labeled A, B, and C, each representing a different swap operation.
- The swaps involve numbers 1, 2, 3, and 4 in various combinations.
- The ball starts in cup 1 in section A.
- The swaps progress from left to right across the sections.
- The swaps appear to be deterministic and based on the cup numbers. *Failed to understand individual movement operations*

 # Visual Patterns
- The swaps seem to follow a specific sequence: A → B → C.
- Each swap appears to involve two cups, with the ball moving from one to the other.
- The final position of the ball after all swaps is likely determined by the last swap in section C.
- The swaps may have a cyclical nature, as suggested by the circular arrows in the image.

Fig. 20. Illustrative case study of LMMs that score near zero on HumanEval-V, revealing heterogeneous failure modes across instruction following, perception, and high-level reasoning.

reinforcing our conclusion that weak high-level visual reasoning, rather than low-level perception alone, remains the dominant barrier for current LMMs on HumanEval-V.

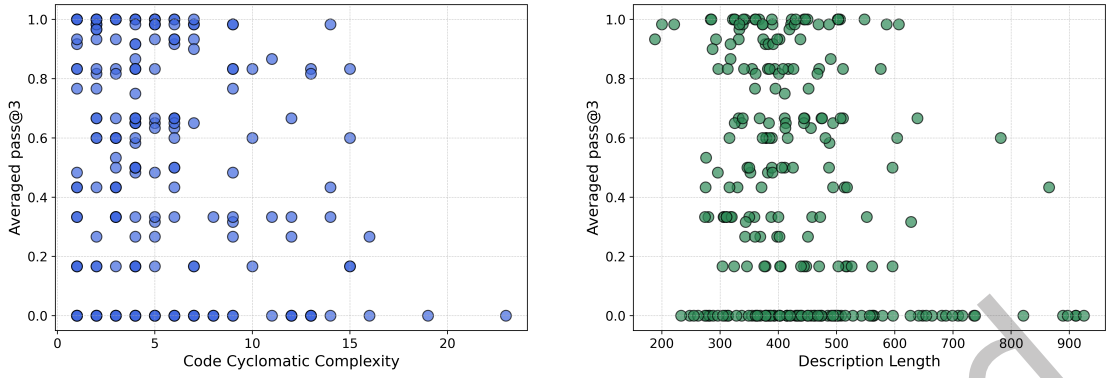


Fig. 21. Correlation Between LMM Pass Rates and Task Difficulty in Coding and Diagram Descriptions.

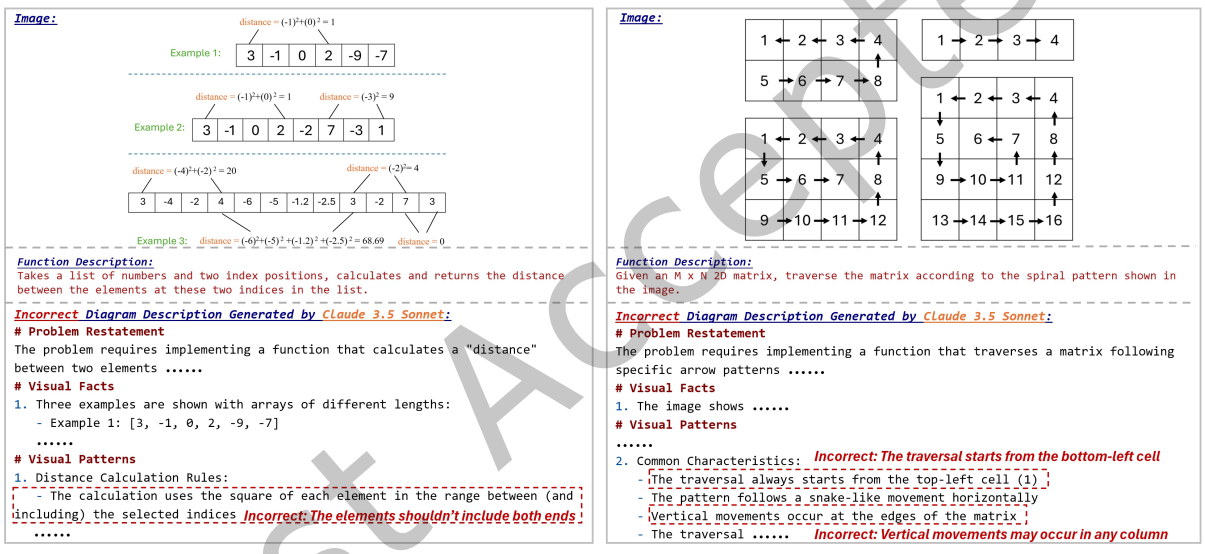


Fig. 22. Two example error cases from Claude 3.5 Sonnet under the *V2T2C w/ SC* setting, illustrating LMMs' difficulties in interpreting visual cues that are easily understood by humans.

5.4 Discrepancies Between Model and Human Perceptions of Task Difficulty

To further investigate the nature of model errors, we conduct a qualitative analysis of failed predictions by LMMs. Our manual case study reveals that these models frequently make errors in scenarios that are intuitively simple for human annotators. This observation suggests a potential misalignment between model performance and human perceptions of task complexity. To quantitatively assess this phenomenon, we compute the correlation between LMM performance and two widely used metrics of human-perceived difficulty: the cyclomatic complexity [22] of ground-truth solution code and the token length of the human-annotated diagram description. Cyclomatic complexity is a well-established metric from software engineering, which quantifies the number of linearly independent execution paths in a program. A higher cyclomatic complexity value reflects a solution with more

Table 6. Performance on HumanEval-V consistently improves with increased sample size K across different LMMs.

Models	K=1	K=3	K=5	K=10	K=20	K=50	K=100
Claude 3.5 Sonnet	36.5	49.5	54.2	60	65.2	71	74.3
GPT-4o	27.9	40.6	45.4	50.9	55.9	61.9	66.2
Gemini 1.5 Pro	23.6	34.5	39.3	45.2	50.3	56.8	61.5
Gemini 1.5 Flash	17.4	25.5	29	33.7	38.4	45.3	51.5
GPT-4o-mini	14.5	22.4	25.7	29.8	33.7	38.8	42.8

branches, loops, or conditional logic, and is therefore considered an indicator of increased programming difficulty. The token length of the diagram description, in turn, serves as a proxy for the linguistic or visual complexity involved in interpreting the task.

We use the averaged pass@3 scores of the five strongest proprietary models under the *V2T2C w/ SC* setting as a measure of LMM performance. As presented in Figure 21, our statistical analysis demonstrates that LMM performance exhibits only weak or negligible correlation with both the cyclomatic complexity of the solution code and the length of the diagram description, except in cases of extremely high task complexity. This result indicates that LMMs struggle with tasks that humans find relatively trivial.

To further illustrate this misalignment, we highlight two representative failure cases in Figure 22. The first case concerns a distance calculation problem, where the correct logic involves summing the squares of array elements between two indices. Although the visual logic is straightforward for humans, Claude 3.5 Sonnet fails to abstract the rule correctly, incorrectly including the endpoints in the calculation. The second case involves matrix traversal along a spiral path. In this instance, the model misidentifies both the starting cell and the movement directions, resulting in an inaccurate description of the traversal pattern. Notably, both cases are considered easy by human standards but remain unsolved by state-of-the-art LMMs, underscoring a fundamental gap in visual reasoning capabilities.

Findings from RQ2

(1) Coding ability is not the main bottleneck, as most LMMs can generate correct code when provided with ground truth textual descriptions.
(2) The dominant limitation of LMMs lies in visual reasoning, particularly in understanding structural relations, spatial patterns, and abstract transformations.
(3) LMMs' performance diverges from human-perceived task difficulty, with frequent failures on tasks that are trivial for human annotators.

6 RQ3: What are the promising research pathways to enhance visual reasoning in code generation?

Recent progress in LLMs demonstrates that output quality can often be substantially improved by increasing the number of sampled responses and employing selection strategies to identify the most accurate or useful outputs [9]. In practice, the performance of a model under a single-sample (pass@1) scenario frequently underestimates its true potential, as higher-quality solutions may only arise in larger candidate sets. To investigate the upper bound of current LMMs on our benchmark, we systematically evaluate the effect of scaling sample size. Specifically, we increase the number of model-generated samples per problem to $n = 200$, and follow other hyper-parameter settings as specified in Section 4.1.3, and report pass@100 scores in the *V2C w/ CoT* setting. The results, presented in Table 6, indicate consistent performance improvements across all evaluated models as sample size increases.

Notably, the best-performing model, Claude 3.5 Sonnet, achieves a pass@100 score of 74.3%, highlighting a considerable gap between the performance attainable through optimal selection and the pass@1 performance.

Bridging this gap remains a significant research challenge. Several approaches have been proposed to close the difference between pass@1 and pass@k performance, including output reranking [9], reject sampling followed by fine-tuning (RFT) [86], and reinforcement learning (RL) techniques such as Group Relative Policy Optimization (GRPO) [27]. Reranking methods typically depend on access to external online feedback signals or discriminant models. In the context of code generation, automated test cases are often used to provide such feedback. Alternatively, RFT and RL have proven effective in domains such as programming [69] and mathematics [86], yet these methods fundamentally require large-scale training data. Our benchmark consists entirely of manually constructed tasks, making it difficult to automate the generation of sufficient in-domain data for such training paradigms.

In light of these challenges, we frame the remainder of this section as an exploration of three promising methodological pathways for future research. First, we investigate *RL-driven reasoning enhancement*. Because reasoning-oriented RL is not yet a standard training objective for most general LMMs, largely due to current unresolved limitations in multimodal contexts, such as "overthinking" and visual hallucination, we evaluate reasoning-enhanced models here to assess the viability of this specific post-training paradigm. Second, we explore *interactive learning via environmental feedback*. By constructing a sandbox environment, we demonstrate the potential of agentic interactive learning, where models can incrementally correct visual misunderstandings using execution signals. Third, we discuss the fundamental bottleneck for enabling the previous two pathways: *scalable in-domain data synthesis*. Recognizing this as a vital direction for future work, we investigate semi-automatic strategies to generate visual-code pairs, addressing the severe lack of fine-grained controllability in current text-to-image models. These three experimental directions are detailed in the following subsections.

6.1 RL-Driven Reasoning Enhancement

Building on our investigation of performance upper bounds, we next examine a class of reasoning-enhanced models. These language models are explicitly trained to improve multi-step reasoning and solution search. A representative example is DeepSeek-R1 [27], which employs a reinforcement learning (RL) algorithm, Group Relative Policy Optimization (GRPO) [27]. In this framework, the model receives positive rewards for generating correct solutions and negative feedback for incorrect outputs. Over time, this iterative optimization enables the model to refine its reasoning strategies. One notable feature of reasoning-enhanced models is their ability to perform extensive "thinking before answering," resulting in long chains of thought before producing a final response. The training of these models is often conducted in domains where automatic verification is feasible to avoid reward hacking. Typically, reasoning-enhanced models are first trained on large-scale code and math datasets where reward signals are well-defined and may subsequently be fine-tuned on additional tasks [27].

In the multimodal domain, this training paradigm has recently been adapted to visual reasoning. For example, QVQ-72B-Preview [66] achieves leading performance on several major multimodal benchmarks, including 70.3% on MMMU and 71.4% on MathVista, surpassing all previously reported models (see Table 5). Other reasoning-enhanced LMMs like Skywork-R1V-38B [72], OpenAI o1 [56], and o3 [58] similarly represent the forefront of proprietary multimodal reasoning-enhanced models. Motivated by their impressive results on these datasets, we further evaluate QVQ-72B-Preview, Skywork-R1V-38B, OpenAI o1, and o3 on our benchmark to determine whether their enhanced reasoning capabilities transfer to the domain of visually grounded coding tasks.

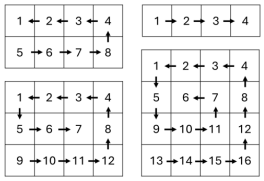
We conduct evaluation under the *V2C w/ CoT* setting. In contrast to their strong performance on other benchmarks, earlier released reasoning-enhanced models such as QVQ-72B-Preview and o1 exhibit limited success on our benchmark. OpenAI o1 achieves a pass@1 score of **40.6%**, while QVQ-72B-Preview reaches only **19.0%**. Meanwhile, the capabilities of this model family have been rapidly evolving. More recently released models

Fig. 23. An example of *visual misunderstanding* by OpenAI o1 under the *V2C w/ CoT* setting. The model fails to correctly interpret the aggregation logic shown in the diagram.

like OpenAI o3 and Skywork-R1V-38B achieve much stronger results, with o3 reaching a remarkable pass@1 score of **64.8%** and Skywork-R1V-38B achieving **26.1%**. These results substantially outperform general-purpose LMMs of comparable size and suggest that enhanced general reasoning abilities can meaningfully transfer to HumanEval-V. This provides strong evidence that robust vision-to-code generation capability can be developed through improved general reasoning skills.

At the same time, our evaluation also shows that existing reasoning-enhanced multimodal models remain far from solving the benchmark, especially when compared to their performance on text-only tasks. A detailed analysis of their outputs reveals several persistent issues:

Image:



Function Signature:

```
def solution(matrix: List[List[int]]) -> List[int]:
    """
    Given an M x N 2D matrix, traverse the matrix according to the spiral pattern
shown in the image
    Parameters:
    - matrix (List[List[int]]): A 2D list of integers representing the matrix.
    Returns:
    - A list of integers representing the elements of the matrix in the order as
shown in the image
    """
```

Test Cases:

```
assert solution([[1,2]])==[1,2]; assert solution([[1,2],[3,4],[5,6],[7,8]])==[7,8,6,4,2,1,3,5]; assert ···
```

Incorrect Code Solution Generated by OpenAI o1:

Step-by-Step Analysis

From the image, the task is to traverse a matrix in a spiral order. The order in which the elements are traversed can be described as follows: ***Incorrect: The traversal starts from the bottom-left cell***

1. **Start from the top-left corner of the matrix** (row 0, column 0).
2. **Move right** across the topmost row.
3. **Move down** along the rightmost column.
4. **Move left** across the bottommost row (if more rows are left).
5. **Move up** along the leftmost column (if more columns are left).
6. **Repeat** this pattern, shrinking the boundary of traversal by excluding the outermost row and column until all elements are visited.

This spiral order traversal involves the following boundaries:
- `top` (initially 0),
- `bottom` (initially the last row index),
- `left` (initially 0),
- `right` (initially the last column index).

These boundaries are adjusted after each movement:
- After moving right, increment the `top` boundary.
- After moving down, decrement the `right` boundary.
- After moving left, decrement the `bottom` boundary.
- After moving up, increment the `left` boundary.

This loop continues until all elements are visited.

Code Implementation

```python
def solution(matrix: List[List[int]]) -> List[int]:
    if not matrix:
        return []
    result = []
    top, bottom, left, right = 0, len(matrix) - 1, 0, len(matrix[0]) - 1
    while top <= bottom and left <= right:
        for i in range(left, right + 1):
            result.append(matrix[top][i])
        top += 1
        for i in range(top, bottom + 1):
            result.append(matrix[i][right])
        right -= 1
        if top <= bottom:
            for i in range(right, left - 1, -1):
                result.append(matrix[bottom][i])
            bottom -= 1
        if left <= right:
            for i in range(bottom, top - 1, -1):
                result.append(matrix[i][left])
            left += 1
    return result
```


Fig. 24. An example of *visual hallucination* by OpenAI o1 under the *V2C w/ CoT* setting. The model explicitly hallucinates a traversal pattern that does not exist in the visual input.

(1) *Hallucination and Visual Misunderstanding:* All models continue to exhibit hallucinations and visual misinterpretation, frequently misidentifying elements or fabricating patterns and rules not present in the

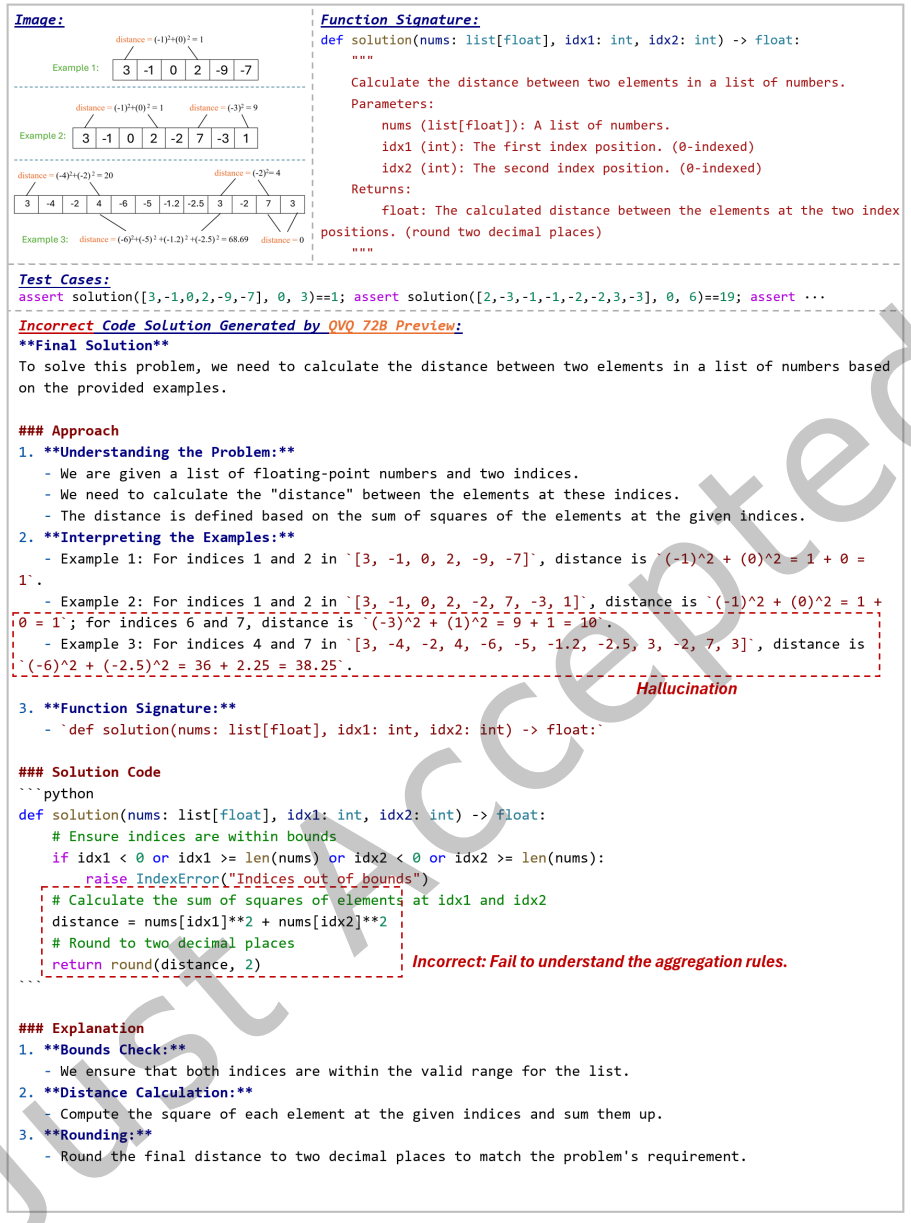


Fig. 25. An error case demonstrated by QVQ-72B-Preview under the *V2C w/ CoT* setting, illustrating the *hallucination of aggregation rules* that are completely absent from the diagram.

diagram. These errors are rarely observed in text-only reasoning tasks but become prevalent in multimodal settings requiring fine-grained visual understanding. For concrete examples, Figure 23 and Figure 24 show OpenAI o1 misinterpreting visual structures and hallucinating traversal patterns, while Figure 25 illustrates QVQ fabricating non-existent aggregation rules.

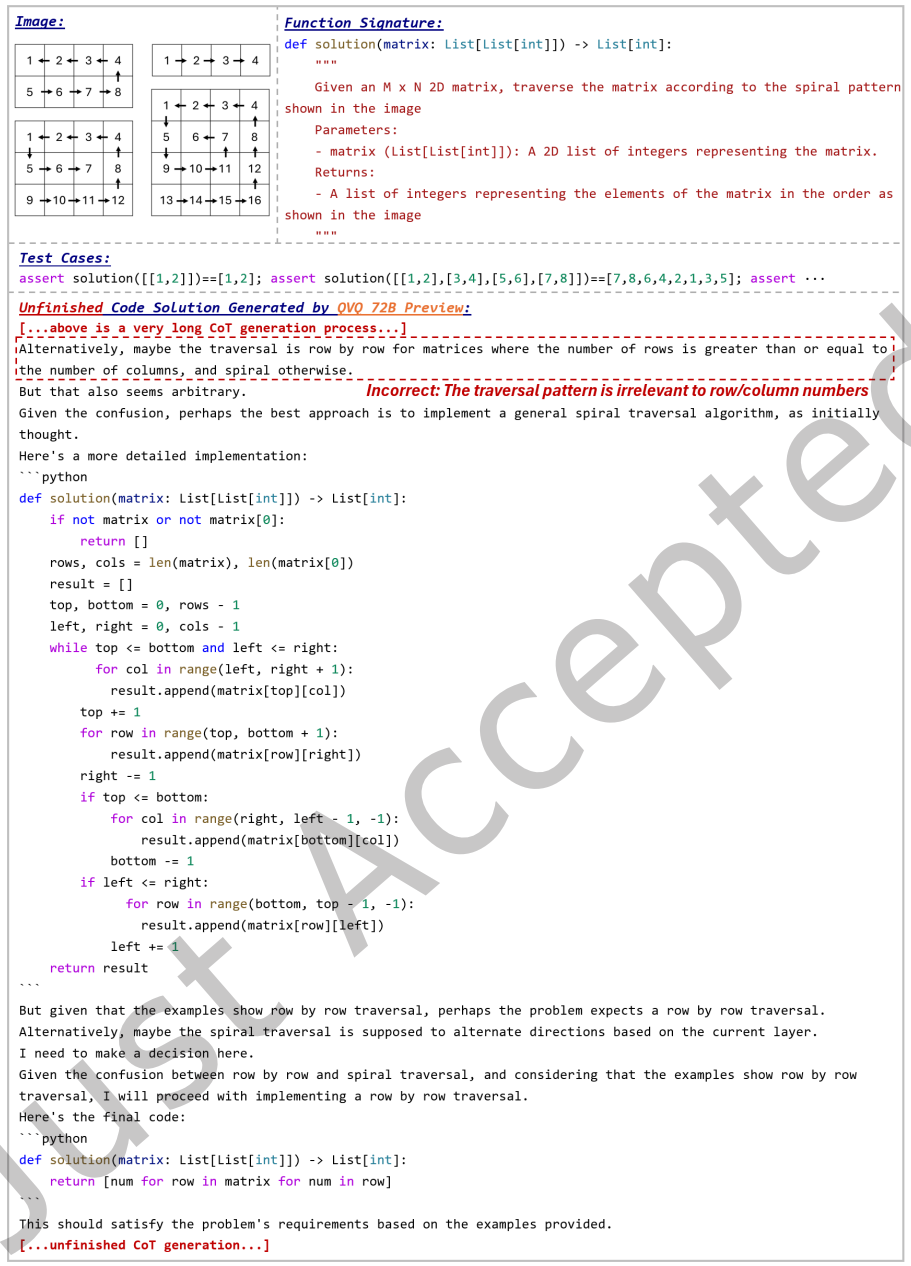


Fig. 26. An error case demonstrating the *"overthinking"* phenomenon by QVQ-72B-Preview under the *V2C w/ CoT* setting. The model generates an excessively long, unfinished chain of thought, yet still fails to recognize the correct traversal pattern.

(2) *Overthinking and Token Limits:* QVQ-72B-Preview, in particular, tends to "overthink" the problem, producing extremely lengthy chains of thought. In 35% of cases, the model fails to generate a valid code solution

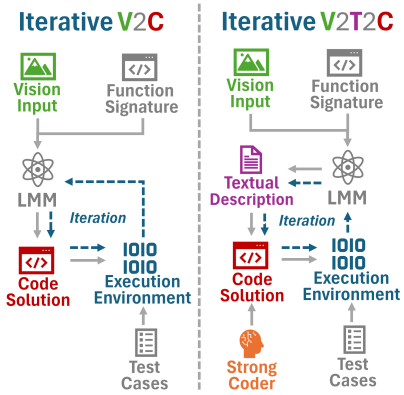


Fig. 27. Iterative evaluation pipelines.

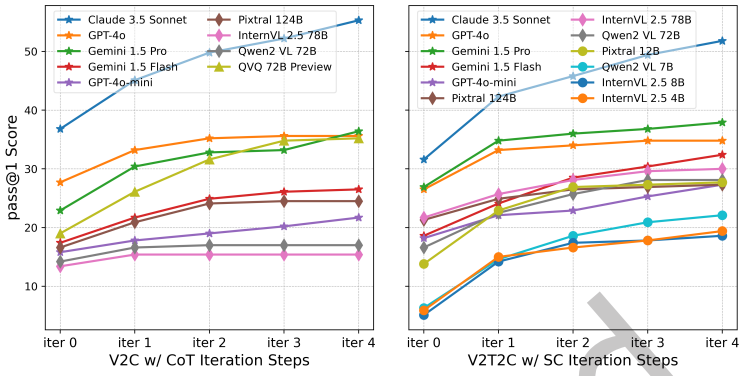


Fig. 28. Performance of LMMs under the iterative evaluation settings.

within the 20,000-token limit, highlighting inefficiencies in its reasoning process (as demonstrated in Figure 26).

(3) *Instruction Following Failures:* A focus on extended reasoning appears to degrade instruction adherence. For example, QVQ-72B-Preview and Skywork-R1V frequently fail the *V2T2C* setting by always generating code instead of a diagram description, and often produce outputs that deviate from required formatting, complicating downstream evaluation.

In summary, although reasoning-enhanced models demonstrate substantial progress in mathematical and textual reasoning, their abilities in visual reasoning and instruction following remain limited. Looking ahead, a critical research direction is the development of large-scale multimodal datasets spanning diverse reasoning tasks, enabling models to acquire genuine visual reasoning across domains. Such data would not only improve the reliability and faithfulness of model predictions but also help mitigate hallucinations and foster deeper vision-language and vision-code alignment.

6.2 Interactive Learning Based on Environmental Feedback

To investigate whether LMMs possess the capacity for self-improvement through interaction with their environment, we design an iterative benchmarking pipeline that exposes models to environmental feedback and enables iterative refinement. The ability to adapt and learn via environmental interaction has recently been shown to benefit from agentic reinforcement learning (RL) approaches [38], where models act as autonomous agents, reflect on their mistakes, and incrementally refine their outputs based on feedback. These agentic RL paradigms have proven effective in enabling models to develop more robust self-reflection and self-correction capabilities in complex, open-ended tasks.

As illustrated in Figure 27, we develop two types of iterative workflows, derived from the *V2C* and *V2T2C w/ SC* settings. In these pipelines, models are required to revise their initial code or textual descriptions based on feedback obtained from the execution sandbox. For every task, if the model's output contains syntax errors or fails any test cases, the sandbox delivers detailed error messages or the input-output pairs corresponding to failed test cases.

For this iterative evaluation, we select the most capable LMMs across a range of parameter scales and evaluate them using greedy decoding and the pass@1 metric. As shown in Figure 28, where *iter 0* represents the initial round without feedback, we observe that model performance generally improves over multiple iterations, with

larger and more advanced models achieving greater gains. Notably, models such as Claude and QVQ demonstrate particularly strong interactive learning capabilities.

It is important to note that Figure 28 contains results from two different evaluation settings: the left subplot reports *V2C w/ CoT* iteration, while the right subplot reports *V2T2C w/ SC* iteration. Crucially, the two subplots do not contain the same set of models. The *V2C* setting includes only proprietary LMMs and open-weight models larger than 70B parameters, as only these models are capable of directly performing *V2C*. In contrast, the *V2T2C* setting additionally includes smaller models (12B, 8B, and 4B), which cannot perform *V2C* reliably but can achieve acceptable performance in *V2T2C* when paired with GPT-4o as a strong coder.

A closer examination of the corrected cases shows that approximately 90% of improvements result from a better understanding of the diagram and task specification after receiving feedback. The remaining 10% are corrections of edge conditions or corner cases explicitly identified by test case feedback. This indicates that the models are genuinely learning and adapting through environmental interaction rather than relying on memorization.

While our results validate the feasibility of interactive learning through environmental feedback, several challenges remain for the training of such capabilities. First, there is a lack of sufficiently diverse and reasoning-intensive multimodal coding tasks that can serve as a robust training ground for agentic learning. Second, the annotation of high-quality test cases for complex visual reasoning tasks is costly and labor-intensive, limiting the scalability of current approaches. These limitations highlight a critical future direction: *developing automated frameworks that can efficiently generate, verify, and provide feedback for multimodal coding tasks*. Building such infrastructure would enable models to interact with a rich set of environments, thereby facilitating scalable training of self-reflective and adaptive reasoning abilities.

6.3 Potential Semi-Automatic Strategies for Constructing In-Domain Training Data

Automatically generating high-quality visual-code pairs that match the characteristics of `HumanEval-V` is highly non-trivial. The primary challenge arises from the nature of `HumanEval-V` diagrams: they are free-form, abstract, and often encode complex transformations, relations, and multi-step operations over visual elements. Unlike structured diagram formats such as flowcharts or UML diagrams, the representations in `HumanEval-V` are intentionally diverse and visually heterogeneous, conveying algorithmic logic through flexible spatial configurations rather than predefined syntactic rules. This level of abstraction makes fully automated synthesis extremely challenging for current multimodal models.

Existing automatically constructed vision-to-code datasets rely heavily on structured, rule-based diagram formats for which reverse-generation pipelines are readily available. For example, Flow2Code [30] trains models to translate flowcharts or UML diagrams into code using diagrams automatically synthesized from code snippets via tools like *Visustin*. Similarly, ChartCoder [92] generates chart-to-code pairs by randomly sampling data and rendering corresponding visualizations through standard plotting libraries such as *matplotlib*. These pipelines are feasible precisely because the diagrams adhere to strict syntactic specifications and can be programmatically generated from code or data.

In contrast, the diagrams in `HumanEval-V` cannot be generated through such deterministic reverse-engineering processes. Their visual diversity, lack of fixed templates, and reliance on spatial semantics make it infeasible to automatically derive visually valid and semantically meaningful diagrams that reflect the rich reasoning patterns required by the benchmark. An additional difficulty lies in the precision required for valid diagram construction. Even small inaccuracies, such as misaligned shapes, incorrect arrows, and missing spatial relations, may invalidate the task. Current text-to-image models lack the necessary controllability and structural fidelity to ensure diagram correctness. As shown in Figure 29, our attempts to synthesize `HumanEval-V`-style diagrams using a state-of-the-art image generation model (Gemini Nano Banana [25]) demonstrate that, even with detailed textual

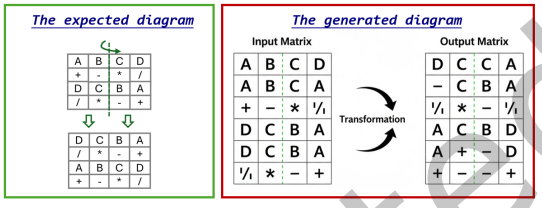
Prompt for Gemini Nano Banana to generate the diagram

 # Problem Description

The problem requires implementing a matrix transformation function that takes a NxN input matrix and produces a NxN output matrix. The transformation appears to be a reflection or mirroring operation around a vertical axis that runs through the center of the matrix.

 # Visual Facts

1. Matrix Properties:
   - Both input and output matrices are 4x4
   - An example cases are shown
   - A green dotted line marks the vertical axis in the middle of input matrices
   - Black arrows show the transformation from input to output
2. First Example:
   - Input Matrix rows:
     * [A, B, C, D]
     * [+, -, *, /]
     * [D, C, B, A]
     * [/, *, -, +]
   - Output Matrix rows:
     * [D, C, B, A]
     * [/, *, -, +]
     * [A, B, C, D]
     * [+, -, *, /]



 # Visual Patterns

Transformation Rules:
   - The matrix is reflected across the vertical line between columns 2 and 3
   - Each row in the input matrix becomes flipped horizontally in the output matrix
   - The relative positions of characters within each row remain in the same row, just reversed
   - Matrix dimensions remain unchanged (4x4)
   - Content of the matrix remains the same, only positions change
   - The transformation applies consistently regardless of the type of characters (letters, numbers, or symbols)

Fig. 29. Attempt to synthesize a diagram in HumanEval-V using Gemini Nano Banana. The generated diagram is inconsistent with the required logical and spatial relations.

prompts, the generated diagrams fail to capture the intended structure and logic, rendering them unsuitable for code generation tasks.

Despite these challenges, we identify two potential semi-automatic strategies that warrant further investigation in future work to enable scalable in-domain dataset expansion.

(a) Template-Based Synthesis This approach builds upon the structured nature of specific HumanEval-V task categories. We can design parametric code templates that encode the canonical structure of various task types, including their input/output schema and core algorithmic operations. LLMs are then used to instantiate diverse task variants by modifying parameters, data structures, and transformation rules. For each synthesized code variant, corresponding diagrams can be programmatically rendered using lightweight vector graphics or plotting libraries (e.g., *matplotlib*, *networkx*). A final human verification stage ensures consistency between the rendered diagram and the underlying algorithmic logic, filtering out diagrams with visual inaccuracies. This method offers scalability and quality control, enabling systematic expansion of each task family into valid instances. Its primary limitation is reduced diversity, as all generated tasks remain bounded by the initial template structures and may fail to capture the breadth of free-form visual reasoning patterns that characterize HumanEval-V.

(b) Task-First Synthesis This strategy emphasizes creative task ideation. LLMs are prompted to generate novel coding problems that inherently require visual reasoning. The model generates a textual description that outlines the intended visual relationships and the target function signature. Given this specification, image-generation models attempt to produce diagrams aligned with the described logic. Because current image-generation models still lack fine-grained visual controllability, the resulting diagrams often require human post-editing. Annotators correct misaligned shapes, add missing symbols or labels, adjust spatial relationships, and ensure that the

final diagram faithfully encodes the intended algorithmic rules. Once refined, these visual-code pairs can be incorporated into the dataset. This task-first approach yields greater diversity and can introduce new forms of visual reasoning not covered by predefined templates. However, the approach is constrained by the limited precision of image-generation tools, making quality control labor-intensive.

Findings from RQ3

(1) Increasing the sampling size significantly improves performance, revealing untapped potential in LMMs.
(2) Reasoning-enhanced models show promising transferability to `HumanEval-V`. However, their visual reasoning and instruction-following abilities still exhibit clear limitations.
(3) Iterative self-refinement with environment feedback shows promising gains, suggesting that interactive learning can help mitigate hallucinations and strengthen visual understanding.

7 Ablation Study

7.1 Effect of Sampling Temperature

To ensure the robustness and generalizability of our experimental settings, we systematically analyze the effect of sampling temperature on model performance. As described in Section 4.1.3, we set the sampling temperature to $T = 0.8$ for generating multiple predictions, following established practices in code generation benchmarking [9, 11]. Given that LMMs may exhibit varying performance at different temperatures, we conduct an ablation study to assess the validity of this choice. Specifically, we evaluate 22 LMMs under the *V2T2C w/ GPT-4o* setting across a range of temperatures from 0.4 to 1.0. The results, presented in Table 7, indicate that model performance remains generally stable across different temperature settings, with only minor variations observed in a few models. This finding further supports the rationale for adopting $T = 0.8$ as the default sampling temperature.

7.2 Impact of Strong Coder Selection

We further assess the stability of our evaluation pipeline by examining the impact of the strong coder model used for code generation. Throughout our main experiments, we employ GPT-4o as the primary strong coder, leveraging its advanced coding capabilities to generate code based on the problem specifications. This setup allows us to isolate and evaluate the visual understanding ability of LMMs in a controlled manner. To evaluate whether our findings are sensitive to the choice of strong coder, we conduct an ablation study by substituting GPT-4o with an open-weight model, Qwen2.5-Coder-32B-Instruct [32] (QwenCoder-32B), which has demonstrated comparable coding performance on benchmarks such as LiveCodeBench [33].

This ablation is performed under the *V2T2C w/ SC* setting, where QwenCoder-32B replaces GPT-4o to generate code from the LMM-produced problem specifications. As shown in Figure 30, the results from GPT-4o and QwenCoder-32B display near-perfect correlation, confirming the robustness and stability of our evaluation methodology across different strong coder models.

7.3 Effect of Task Diversification

To evaluate the influence of our task diversification process, we analyze whether the diversified tasks introduce new challenges beyond those presented by the original seed tasks. This analysis is important for verifying the richness and coverage of our benchmark. As detailed in Section 3, our annotation pipeline generates diversified versions of each seed task, resulting in a broader and more varied set of evaluation items.

Table 7. Ablation on LMMs' sampling temperature for the pass@3 results under the *V2T2C w/ SC* settings.

Models	T=0.4	T=0.6	T=0.8	T=1
Proprietary LMMs				
Claude 3.5 Sonnet	48.3	46.9	48.1	47.9
GPT-4o	44.9	42.2	43.6	44.9
Gemini 1.5 Pro	41.1	39.2	39.4	41.6
Gemini 1.5 Flash	27.1	30	28.4	29.1
GPT-4o-mini	27.9	29	29.9	32.1
Open-weight LMMs				
Pixtral 124B	35.6	39.8	34.2	37
InternVL 2.5 78B	37.2	36.1	35.9	36.2
Qwen2 VL 72B	31.1	29.1	28.7	31.5
LLaVA-OV 72B	25	22.7	24.1	22.2
Molmo-D 72B	16.9	13.2	14.4	14.5
Llama-3.2-V 90B	12.5	10.7	12.4	13.7
Pixtral 12B	21.2	23.4	23.2	21.1
InternVL 2.5 26B	20.9	21	20.7	21.8
Qwen2 VL 7B	13.3	17.2	16.6	18.5
InternVL 2.5 8B	13.6	16.6	17.3	16.3
InternVL 2.5 4B	16.9	17.6	15.2	20.5
LLaVA-OV 7B	15.1	15.2	13	15.2
Phi-3.5-V 4B	5.6	9.3	9.1	8.7
Llama-3.2-V 11B	7.5	10.2	9.6	9.9
Molmo-D 7B	9.2	10	11.2	11.2
Chameleon 7B	1	2.5	2.5	2.1
Chameleon 30B	2	0.5	3.3	2

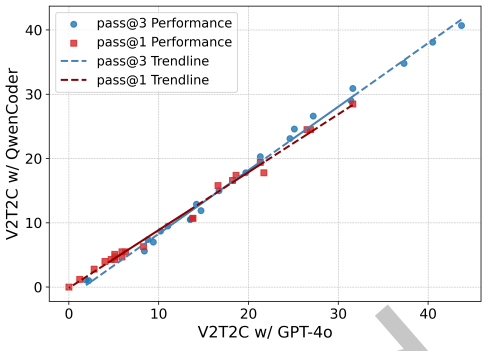


Fig. 30. Performance comparison between GPT-4o and QwenCoder-32B as strong coders under the *V2T2C w/ SC* setting.

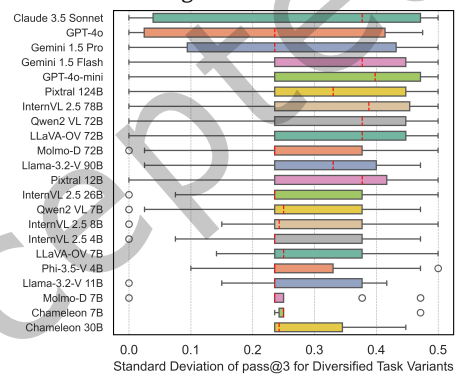


Fig. 31. Effect of task diversification on performance variation across models in the *V2T2C w/ SC* setting.

We assess this effect by examining the standard deviation of pass rates within each group of seed tasks and their diversified counterparts, using the results from 22 LMMs under the *V2T2C w/ SC setting*. Each group contains a seed task and its variants, yielding 100 task groups in total. We calculate the standard deviation of pass@3 scores within each group, excluding groups where all tasks have a pass@3 rate of zero. The statistical distribution of these standard deviations, visualized in Figure 31, demonstrates that the 25th percentile standard deviation for most models exceeds 0.2, with the median around 0.4. This substantial within-group variance indicates that our diversification process effectively generates tasks with varying levels of difficulty and model performance, thereby enriching the benchmark and validating the design of our task annotation pipeline.

8 Threats to Validity

We discuss several potential threats to the validity of our study and describe the measures taken to mitigate them.

Data Quality. Since our benchmark is fully human-annotated, there is a risk of annotation bias or human error. To address this, we designed a multi-stage annotation pipeline. In the data collection phase, we aggregate a diverse set of tasks from multiple programming platforms and perform thorough categorization and screening to

ensure coverage and diversity. At the final annotation stage, each annotator is required to review and validate tasks created by another annotator, and to provide both ground-truth code solutions and unambiguous diagram descriptions. This peer-review process is designed to minimize errors and ambiguity, improving both the reliability and the clarity of the dataset.

Data Quantity. Our benchmark contains 253 tasks, which may appear limited compared to automatically constructed datasets, as all examples are manually annotated. However, this scale is consistent with widely adopted benchmarks in the field, such as HumanEval [11] (164 tasks), MMVet [85] (218 tasks), Vibe-Eval [59] (269 tasks), ClassEval [20] (100 tasks), EvoCodeBench [44] (275 tasks), and SWE-Bench Lite [37] (300 tasks). Moreover, we dedicated over 800 hours to the annotation process to ensure the highest possible quality and diversity of tasks. Our tasks span a broad range of visual elements and reasoning capabilities, supporting the generalizability of our evaluation results.

Experimental Hyperparameters. Our experiments involve choices regarding prompt design and sampling temperature, both of which may affect the robustness of our findings. To mitigate these risks, we conducted extensive pilot studies on a held-out subset of the data to refine prompts and developed multiple prompting strategies. Experimental results demonstrate strong correlation across these strategies, underscoring the robustness of our conclusions to prompt variation. Additionally, we conducted thorough ablation studies on sampling temperature and on the choice of the strong coder model, confirming that our evaluation pipeline is robust and that the chosen hyperparameters align well with the performance preferences of most models under study.

Validity of Experimental Conclusions. To ensure the validity of our findings, we evaluate 27 LMMs from a wide range of model sizes and vendors. We report both pass@1 and pass@3 results, reducing the impact of randomness in code generation and increasing the reliability of our conclusions. This comprehensive experimental setup supports the reproducibility of our key findings.

Ecological Validity and Realism. While our visually focused design is well motivated for analytical purposes, allowing us to isolate and diagnose visual reasoning capabilities, it inevitably creates a gap from real-world software engineering practice. In practical settings, textual descriptions and visual artifacts typically complement and reinforce each other. We explicitly acknowledge two limitations in this regard. First, `HumanEval-V` focuses on solving tasks mainly based on diagrams, with only minimal textual information provided. Second, the ability to synergistically integrate and use rich text and images together is highly important for real-world multimodal coding tasks. Although evaluating this joint multimodal reasoning is not the primary focus of our benchmark, we recognize it as an essential capability for comprehensive software engineering assistants.

9 Conclusion

In this paper, we present a systematic evaluation of LMMs' visual reasoning abilities for code generation tasks. To this end, we introduce `HumanEval-V`, a novel benchmark designed to test LMMs' capacity for interpreting diagrams and generating correct code solutions. Unlike existing multimodal benchmarks, `HumanEval-V` poses substantially greater challenges by targeting high-level visual reasoning skills in code-centric tasks. Through extensive experiments on 27 state-of-the-art LMMs, we find that most models perform poorly on our benchmark, even those that excel on other popular multimodal datasets. Our analysis reveals that the main performance bottleneck lies not in coding ability, but in visual understanding–especially in tasks involving spatial transformations, topological relations, and dynamic visual patterns. We further observe a disconnect between model performance and human-perceived task difficulty, as models often fail on problems that are trivial for humans. To explore ways of improving model performance, we evaluate the impact of sampling strategies, reasoning-enhanced models, and interactive learning via environmental feedback. Our findings underscore the need for new training paradigms

and task environments that explicitly target visual reasoning in code generation. Overall, HumanEval-V offers a rigorous and diagnostic benchmark for the next generation of LMMs. We hope it will serve as a foundation for developing models with deeper vision-language-code alignment and more reliable visual reasoning capabilities in programming tasks.

Acknowledgments

This research was supported by National Natural Science Foundation of China under Grant No. 62502440 and Zhejiang Provincial Natural Science Foundation of China under Grant No. LQN26F020003.

References

[1] Pravesh Agrawal, Szymon Antoniak, Emma Bou Hanna, Baptiste Bout, Devendra Chaplot, Jessica Chudnovsky, Diogo Costa, Baudouin De Monicault, Saurabh Garg, Theophile Gervet, et al. 2024. Pixtral 12B. *arXiv preprint arXiv:2410.07073* (2024).

[2] Anthropic. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet

[3] Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, et al. [n. d.]. Multi-lingual Evaluation of Code Generation Models. In *The Eleventh International Conference on Learning Representations*.

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[5] Sebastian Baltes and Stephan Diehl. 2014. Sketches and diagrams in practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 530–541.

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

[7] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2022. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227* (2022).

[8] Linzheng Chai, Jian Yang, Shukai Liu, Wei Zhang, Liran Wang, Ke Jin, Tao Sun, Congnan Liu, Chenchen Zhang, Hualei Zhu, et al. 2025. Multilingual Multimodal Software Developer for Code Generation. *arXiv preprint arXiv:2507.08719* (2025).

[9] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397* (2022).

[10] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. 2024. Are We on the Right Way for Evaluating Large Vision-Language Models? *arXiv preprint arXiv:2403.20330* (2024).

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

[13] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. 2024. Expanding Performance Boundaries of Open-Source Multimodal Models with Model, Data, and Test-Time Scaling. *arXiv preprint arXiv:2412.05271* (2024).

[14] Mauro Cherubini, Gina Venolia, Rob DeLine, and Amy J Ko. 2007. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 557–566.

[15] Yew Ken Chia, Vernon Toh Yan Han, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. 2024. PuzzleVQA: Diagnosing Multimodal Reasoning Challenges of Language Models with Abstract Visual Patterns. *arXiv preprint arXiv:2403.13315* (2024).

[16] François Chollet. 2019. On the measure of intelligence. *arXiv preprint arXiv:1911.01547* (2019).

[17] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146* (2024).

[18] Yihong Dong, Jiazheng Ding, Xue Jiang, Ge Li, Zhuo Li, and Zhi Jin. 2025. Codescore: Evaluating code generation by learning code execution. *ACM Transactions on Software Engineering and Methodology* 34, 3 (2025), 1–22.

[19] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–38.

[20] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on*

Software Engineering. 1–13.

[21] Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. 2024. VLMEvalKit: An Open-Source Toolkit for Evaluating Large Multi-Modality Models. arXiv:2407.11691 [cs.CV] https://arxiv.org/abs/2407.11691

[22] Geoffrey K Gill and Chris F Kemerer. 1991. Cyclomatic complexity density and software maintenance productivity. *IEEE transactions on software engineering* 17, 12 (1991), 1284–1288.

[23] Google. 2024. Introducing Gemini 1.5, Google's next-generation AI model. https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/

[24] Google. 2024. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[25] Google. 2025. Introducing Nano Banana. https://gemini.google/overview/image-generation/

[26] Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. 2023. HallusionBench: An Advanced Diagnostic Suite for Entangled Language Hallucination and Visual Illusion in Large Vision-Language Models. *arXiv preprint arXiv:2310.14566* (2023).

[27] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[28] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).

[29] Devamardeep Hayatpur, Brian Hempel, Kathy Chen, William Duan, Philip Guo, and Haijun Xia. 2024. Taking ascii drawings seriously: How programmers diagram code. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–16.

[30] Mengliang He, Jiayi Zeng, Yankai Jiang, Wei Zhang, Zeming Liu, Xiaoming Shi, and Aimin Zhou. 2025. Flow2Code: Evaluating Large Language Models for Flowchart-based Code Generation Capability. *arXiv preprint arXiv:2506.02073* (2025).

[31] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[32] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-Coder Technical Report. *arXiv preprint arXiv:2409.12186* (2024).

[33] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974* (2024).

[34] Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Dongdong Zhang, and Furu Wei. 2025. Viscodex: Unified multimodal code generation via merging vision and coding models. *arXiv preprint arXiv:2508.09945* (2025).

[35] Xue Jiang, Yihong Dong, Yongding Tao, Huanyu Liu, Zhi Jin, and Ge Li. 2025. ROCODE: Integrating Backtracking Mechanism and Program Analysis in Large Language Models for Code Generation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 670–670.

[36] Yifan Jiang, Jiarui Zhang, Kexuan Sun, Zhivar Sourati, Kian Ahrabian, Kaixin Ma, Filip Ilievski, and Jay Pujara. 2024. MARVEL: Multidimensional Abstraction and Reasoning through Visual Evaluation and Learning. *arXiv preprint arXiv:2404.13591* (2024).

[37] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[38] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516* (2025).

[39] Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 235–251.

[40] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. PMLR, 18319–18345.

[41] Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the conversion of Web Screenshots into HTML Code with the WebSight Dataset. *arXiv preprint arXiv:2403.09029* (2024).

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

[43] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. 2024. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326* (2024).

[44] Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and Yongbin Li. 2024. Evocodebench: An evolving code generation benchmark with domain-specific evaluations. *Advances in Neural Information Processing Systems* 37 (2024),

57619–57641.

[45] Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. 2024. MMCode: Benchmarking Multimodal Large Language Models for Code Generation with Visually Rich Programming Problems. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 736–783.

[46] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[47] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=1qvx610Cu7

[48] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2023. Mmbench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281* (2023).

[49] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255* (2023).

[50] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems* 35 (2022), 2507–2521.

[51] Michael R Lyu, Baishakhi Ray, Abhik Roychoudhury, Shin Hwei Tan, and Patanamon Thongtanunam. 2025. Automatic programming: Large language models and beyond. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–33.

[52] Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244* (2022).

[53] Microsoft. 2024. Discover the New Multi-Lingual, High-Quality Phi-3.5 SLMs. https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/discover-the-new-multi-lingual-high-quality-phi-3-5-slms/ba-p/4225280

[54] Weili Nie, Zhiding Yu, Lei Mao, Ankit B Patel, Yuke Zhu, and Anima Anandkumar. 2020. Bongard-logo: A new benchmark for human-level concept learning and reasoning. *Advances in Neural Information Processing Systems* 33 (2020), 16468–16480.

[55] OpenAI. 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/

[56] OpenAI. 2024. Introducing OpenAI o1. https://openai.com/o1/

[57] OpenAI. 2024. tiktoken. https://github.com/openai/tiktoken

[58] OpenAI. 2025. Introducing OpenAI o3 and o4-mini. https://openai.com/zh-Hans-CN/index/introducing-o3-and-o4-mini/

[59] Piotr Padlewski, Max Bain, Matthew Henderson, Zhongkai Zhu, Nishant Relan, Hai Pham, Donovan Ong, Kaloyan Aleksiev, Aitor Ormazabal, Samuel Phua, et al. 2024. Vibe-Eval: A hard evaluation suite for measuring progress of multimodal language models. *arXiv preprint arXiv:2405.02287* (2024).

[60] Marian Petre and Alan F Blackwell. 1999. Mental imagery in program design and visual programming. *International Journal of Human-Computer Studies* 51, 1 (1999), 7–30.

[61] Chufan Shi, Cheng Yang, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. 2024. ChartMimic: Evaluating LMM's Cross-Modal Reasoning Capability via Chart-to-Code Generation. *arXiv preprint arXiv:2406.09961* (2024).

[62] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2Code: How Far Are We From Automating Front-End Engineering? *arXiv preprint arXiv:2403.03163* (2024).

[63] Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. 2023. Modular visual question answering via code generation. *arXiv preprint arXiv:2306.05392* (2023).

[64] Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11888–11898.

[65] Chameleon Team. 2024. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818* (2024).

[66] Qwen Team. 2024. QVQ: To See the World with Wisdom. https://qwenlm.github.io/blog/qvq-72b-preview/

[67] Yao Wan, Zhangqian Bi, Yang He, Jianguo Zhang, Hongyu Zhang, Yulei Sui, Guandong Xu, Hai Jin, and Philip Yu. 2024. Deep learning for code intelligence: Survey, benchmark and toolkit. *Comput. Surveys* 56, 12 (2024), 1–41.

[68] Dong Wang, Tao Xiao, Christoph Treude, Raula Gaikovina Kula, Hideaki Hata, and Yasutaka Kamei. 2023. Understanding the role of images on stack overflow. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 377–388.

[69] Junqiao Wang, Zeng Zhang, Yangfan He, Yuyang Song, Tianyu Shi, Yuchen Li, Hengyuan Xu, Kunyu Wu, Guangwu Qian, Qiuwu Chen, et al. 2024. Enhancing Code LLMs with Reinforcement Learning in Code Generation. *arXiv preprint arXiv:2412.20367* (2024).

[70] Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Mingjie Zhan, and Hongsheng Li. 2024. Measuring multimodal mathematical reasoning with math-vision dataset. *arXiv preprint arXiv:2402.14804* (2024).

[71] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. 2024. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. *arXiv preprint arXiv:2409.12191* (2024).

[72] Peiyu Wang, Yichen Wei, Yi Peng, Xiaokun Wang, Weijie Qiu, Wei Shen, Tianyidan Xie, Jiangbo Pei, Jianhao Zhang, Yunzhuo Hao, et al. 2025. Skywork r1v2: Multimodal hybrid reinforcement learning for reasoning. *arXiv preprint arXiv:2504.16656* (2025).
[73] Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. 2025. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265* (2025).
[74] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030* (2024).
[75] Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. 2024. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *arXiv preprint arXiv:2406.18521* (2024).
[76] Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. 2022. Execution-based evaluation for open-domain code generation. *arXiv preprint arXiv:2212.10481* (2022).
[77] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[78] Wikipedia. 2024. Pylint — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Pylint&oldid=1191495734. [Online; accessed 24-October-2024].
[79] Chengyue Wu, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. 2024. Plot2Code: A Comprehensive Benchmark for Evaluating Multi-modal Large Language Models in Code Generation from Scientific Plots. *arXiv preprint arXiv:2405.07990* (2024).
[80] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. 2024. Top Leaderboard Ranking = Top Coding Proficiency, Always? EvoEval: Evolving Coding Benchmarks via LLM. *arXiv preprint* (2024).
[81] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
[82] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*. 476–486.
[83] Kaining Ying, Fanqing Meng, Jin Wang, Zhiqian Li, Han Lin, Yue Yang, Hao Zhang, Wenbo Zhang, Yuqi Lin, Shuo Liu, et al. 2024. Mmt-bench: A comprehensive multimodal benchmark for evaluating large vision-language models towards multitask agi. *arXiv preprint arXiv:2404.16006* (2024).
[84] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.
[85] Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. 2023. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490* (2023).
[86] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825* (2023).
[87] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 2024. 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*. 9556–9567.
[88] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. 2019. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5317–5327.
[89] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570* (2023).
[90] Linhao Zhang, Daoguang Zan, Quanshun Yang, Zhirong Huang, Dong Chen, Bo Shen, Tianyu Liu, Yongshun Gong, Huang Pengjie, Xudong Lu, et al. 2025. Codev: Issue resolving with visual data. In *Findings of the Association for Computational Linguistics: ACL 2025*. 7350–7361.
[91] Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. 2024. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems?. In *European Conference on Computer Vision*. Springer, 169–186.
[92] Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2025. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598* (2025).
[93] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987* (2022).