EmbedCopilot-Bench: Evaluating Vision-Language Models for Hardware-Aware Embedded System Development

Dongsheng Yuan, Yimo Deng, and Huangxun Chen*

Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
dyuan987@connect.hkust-gz.edu.cn, dengemo.neu@gmail.com,
huangxunchen@hkust-gz.edu.cn
Code: https://github.com/X-EASys/EmbedCopilot-Bench
Dataset: https://huggingface.co/datasets/X-EASys/EmbedCopilot-Bench

Abstract. We introduce EmbedCopilot-Bench, a multimodal benchmark designed to evaluate large vision-language models (LVLMs) as assistants for embedded development. Compared with pure software programming, embedded programming inherently involves hardware, making it well-suited to benefit from LVLMs' visual capabilities. However, to the best of our knowledge, no existing benchmark comprehensively evaluates how effectively LVLMs can serve as embedded copilots. Our benchmark is developed to fill this gap. Built on real-world embedded development videos, we constructed 216 annotated multimodal QA triplets spanning heterogeneous hardware platforms and peripheral modules, covering hardware operation, software configuration, and code generation tasks. To assess model performance, we pair a rubric-driven LLM-as-a-Judge protocol with execution-centric evaluation in Wokwi platform and hardware-in-the-loop Execution Success Rate (ESR), jointly capturing semantic quality and functional correctness. Experiments on a range of closed- and open-source LVLMs show that visual context significantly boosts performance, especially for hardware-related tasks, but that models remain far from reliable embedded copilots. ESR analyses and a pin-localization case study reveal frequent near-miss failures caused by brittle grounding and limited structured reasoning over board layouts. EmbedCopilot-Bench provides a challenging, realistic testbed for future methods that more tightly integrate perception, circuit understanding, and executable code synthesis for real-world embedded development.

Keywords: Multimodal benchmark · Embedded systems

1 Introduction

LLMs have dramatically advanced the automation of complex reasoning and problem-solving across diverse domains. Recent advances in large vision-language

* Corresponding author

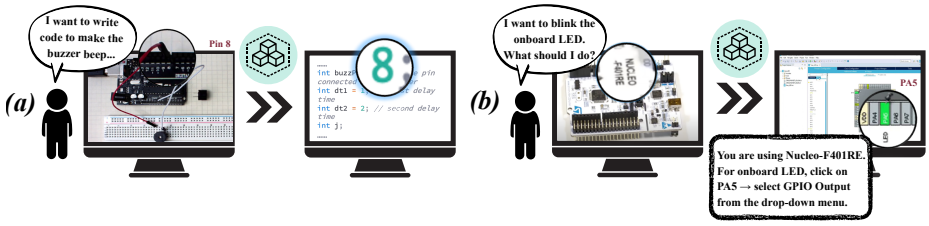

Fig. 1: Example EmbedCopilot-Bench tasks. (a) Arduino buzzer: from a wiring photo and a short text goal, the LVLM must infer the buzzer pin and generate Arduino code. (b) STM32 Nucleo LED: from a board image and IDE view, the LVLM must recognize the board, locate the onboard LED pin, and output the UI configuration steps. Both require joint hardware recognition, pin-level grounding, and code/UI reasoning.

models (LVLMs) aim to further empower LLMs with visual understanding capabilities. In response, a growing body of benchmarks has emerged to probe different aspects of LVLM capabilities. These benchmarks cover various visual tasks, including visual recognition [18, 20], Optical Character Recognition (OCR) [14, 34], solving diagram-illustrated mathematical problems [36, 44, 56], and multi-disciplinary figure-based reasoning [51, 52].

Naturally, we also see the opportunity for LVLMs to enhance developer efficiency. Most current code copilot products [3, 5, 10, 15] focus on the textual modality, where users express intent in text and the system generates code snippets. Recent works explore incorporating visual modality, for example, generating code from an existing plot [46] or from an algorithmic problem-solving figure [28]. However, for embedded-system development, little effort has been made to systematically leverage LVLMs' capabilities. Compared with pure software programming, embedded programming has additional demands. Typical workflows span multiple intertwined stages: reasoning about hardware wiring, interpreting pin labels and schematics, configuring software environments, and finally generating and debugging low-level code to be flashed onto microcontrollers. Visual capabilities could greatly assist this process. As illustrated in Fig. 1, consider a beginner who has connected a buzzer to an Arduino Uno and wants the device to emit a periodic beep. The user provides an image of the current wiring state plus a brief natural-language description, querying an LVLM to produce a complete solution: correctly identifying the board and buzzer, inferring which GPIO pin the buzzer is attached to, and generating Arduino code that drives the buzzer via the appropriate pin configuration and timing logic.

To the best of our knowledge, no existing benchmark comprehensively evaluates how effectively LVLMs serve as a copilot for embedded development. In fact, the seemingly simple task shown in Fig. 1 requires LVLMs to jointly exhibit OCR, fine-grained localization, pin/module identification, causal reasoning about circuits, and reliable code generation. To systematically evaluate such capabilities, we propose EmbedCopilot-Bench, a benchmark designed to assess LVLMs for assisting embedded development.

To construct a representative benchmark that aligns with real-world usage, we extensively searched for step-by-step embedded development videos across various online platforms. As shown in Fig. 2 (a) and (b), our data covers more than five widely used embedded platforms (*e.g.*, Arduino, STM32, ESP32) together with a broad suite of peripheral modules (*e.g.*, HC-SR04 ultrasonic sensors, character LCDs, and servos). Based on the curated videos, we designed a rigorous procedure (as shown in Fig. 3) to derive comprehensive multimodal QA triplets, each annotated with capability labels spanning hardware operation, software configuration, and code generation, delivering fine-grained coverage of typical embedded development workflows. The tasks span from entry-level exercises to senior-level projects, enabling controlled evaluation of LVLMs across varying levels of difficulty and hardware complexity.

Besides the dataset, EmbedCopilot-Bench also features effective evaluation protocols. The answers in our QA triplets are inherently heterogeneous and open-ended, including low-level code snippets (Fig. 1(a)), fine-grained UI instructions (*e.g.*, "Click on `PA5` → select `GPIO_Output` from the dropdown menu") as in Fig. 1(b). Thus, we develop a hybrid evaluation protocol: for open-ended outputs, we adopt LLMs as automatic judges, drawing inspiration from recent practices [11, 13, 33, 54]. We propose an LLM-based evaluator that scores model outputs using a rubric tailored to embedded-device tutorials, with its reliability calibrated via human audits. Additionally, for code outputs, we implement strict and deterministic evaluation by executing the generated programs both in a Wokwi emulator and on physical hardware, and report the ESR.

Contributions. Our contributions are as follows:

- **A hardware-software-code benchmark for LVLM-based embedded development.** To the best of our knowledge, EmbedCopilot-Bench is the first integrated benchmark designed to systematically evaluate LVLMs' capabilities in assisting realistic embedded development. It covers comprehensive aspects of hardware operation, software configuration, and code generation, spanning diverse development platforms and functional modules.
- **An evaluation methodology combining LLM-as-a-Judge and rigorous execution assessment.** We design a rubric-driven, LLM-based evaluator for open-ended outputs and validate its reliability through human expert auditing. For code outputs, we implement an execution-based evaluator to objectively measure ESR. This strikes a balance between efficiently assessing semantic correctness and precisely verifying functional deployment validity.
- **Comprehensive analysis of LVLMs as embedded copilots.** We benchmark a diverse range of LVLMs supporting multi-image input: (i) closed-source models including Claude-Sonnet-4.5 [4], GPT-5 [40], GPT-4o [21], and Gemini-2.5-Pro [17]; and (ii) open-source models including Qwen2.5-VL [8], Qwen3-VL [7], InternVL3 [55], Mistral-Small-3 [38], LLaVA-OneVision [25], and Gemma-3 [43]. We comprehensively examine their strengths and weaknesses across hardware, software, and code-related tasks to clarify whether current LVLMs can effectively serve as embedded copilots. Our results highlight substantial performance gains from visual context input, persistent gaps

between closed- and open-source models, and key failure modes in hardware grounding, pin grounding and executable code generation.

Taken together, these contributions position EmbedCopilot-Bench as a practical testbed and stepping stone for developing the next generation of LVLMs that can reliably assist real-world embedded development workflows.

2 Related Work

2.1 Multi-image Multimodal Models

Early large vision–language models (LVLMs) such as Flamingo [2], BLIP-2 [27], and LLaVA [32] demonstrated strong multimodal intelligence by combining pretrained vision encoders with large language models, with most early studies focusing on single-image or short-context settings. These systems achieve impressive performance on generic vision–language tasks such as VQA, captioning, and instruction-following. Building on these advances, recent work has pushed toward *multi-image* LVLMs that can jointly reason over multiple views or frames. Models such as LLaVA variants with multi-image input [25], Qwen2-VL [45]/Qwen2.5-VL [8], and InternVL3 [55] extend the context window to sequences of images and text, enabling cross-image reasoning and richer contextual integration. These capabilities are particularly relevant to our setting, where embedded development involves multiple complementary views that should be interpreted jointly.

2.2 Code LLMs and Program Synthesis

Code generation has emerged as one of the most prominent applications of large language models [19, 22, 24, 30]. A wide range of code LLMs have been proposed and evaluated on general-purpose programming benchmarks [6,23,39,50], achieving strong performance in algorithmic problem solving and software synthesis. Notable models and agents include Codex [10], Claude Code [3], and systems such as GitHub Copilot [15] and Cursor [5], which already support real-world software engineering workflows. Recent work has begun to explore LLM-centered embedded and IoT development systems. IoTPilot [16], EmbedGenius [49], and AutoIOT [42], for example, use LLMs to generate embedded or AIoT applications from natural-language requirements, optionally supported by structured component knowledge. EmbedBench [47] evaluates LLMs on embedded programming, circuit-design, and cross-platform-migration tasks. Englhardt *et al.* [12] further study LLM-assisted embedded development through hardware-in-the-loop evaluation and user interaction. These efforts are complementary, but primarily assess LLM-based program generation or artifact-oriented reasoning rather than visually grounded LVLM behavior from captured hardware states and development-environment screenshots. In contrast, EmbedCopilot-Bench evaluates LVLMs on end-to-end embedded-development tasks that require joint reasoning over textual instructions, hardware images, and IDE screenshots. It covers hardware operation, software configuration, and code generation,

and evaluates not only semantic answer quality but also functional deployment through emulator- and hardware-based execution.

2.3 Existing Vision–Language Benchmarks

As discussed in Sec. 1, many vision–language (VL) benchmarks focus on static image understanding and generic multimodal reasoning, targeting tasks such as visual recognition [18, 20], OCR [14, 34], multidisciplinary figure-based reasoning [51, 52], mathematical problem solving [36, 44, 56], and code-related multimodal reasoning [28, 46]. In particular, MMCode [28] evaluates visually rich programming problems, whereas Plot2Code [46] evaluates code generation from scientific plots. Recent domain-specific benchmarks further move toward adjacent electrical and EDA settings, *e.g.*, diagram-centric Electrical and Electronics Engineering problem solving and PCB/EDA artifact understanding [26, 29]. However, these benchmarks primarily evaluate reasoning over programming figures, scientific plots, circuit diagrams, or PCB artifacts. They do not target the downstream embedded-development loop in which a model must jointly interpret real hardware wiring states, toolchain or IDE screenshots, and textual task requirements to produce actionable operations and executable firmware.

EmbedCopilot-Bench is specifically targeted at evaluating LVLMs for visually grounded embedded development from real hardware images and development-environment screenshots. Rather than isolated perception, artifact understanding, or abstract code reasoning, EmbedCopilot-Bench examines end-to-end workflows that couple hardware wiring, software configuration, and executable firmware generation. This bridges the gap between multimodal real-world understanding and software system development, and provides a testbed for practical AI agents operating in embedded development. Consistent with prior manually curated VL benchmarks, EmbedCopilot-Bench is modest in size (216 triplets), comparable to MM-Vet [51] (218), Plot2Code [46] (368), and LLaVA-Bench [32] (150). Rather than exhaustively enumerating all isolated embedded-development tasks, EmbedCopilot-Bench prioritizes carefully curated, realistic, and reproducible diagnostic coverage of representative hardware–software–code workflows.

2.4 LLM-as-a-Judge Evaluation

EmbedCopilot-Bench benchmarks LVLMs based on an open-ended evaluator, specifically leveraging an independent LLM for evaluation on questions and answers matching without necessitating discrete answer such as multiple choice or short questions. LLM-as-a-Judge has been widely adopted for model evaluation in many complex settings [33, 35, 51], enabling free-form assessment where traditional metrics on open-ended answers fall short. Inspired by this line of work, our extended evaluation metric is tailored to multimodal tasks and adapts well to the diverse outputs in our benchmark.

Fig. 2: Hardware in EmbedCopilot-Bench. (a) Supported development boards. (b) Representative peripheral modules (sensors, UI components, and actuators).

3 EmbedCopilot-Bench

3.1 Benchmark Construction Pipeline

Our goal is to develop a multimodal benchmark to assess whether an LVLM can function as an embedded programming copilot. Therefore, we need to construct comprehensive Q&A samples, as exemplified in Fig. 3(b), where the question part contains a task query and its hardware and software context in visual form, while the answer provides the specific steps required to complete the task. Our core intuition is to leverage rich and high-quality tutorial videos on mainstream video platforms (*e.g.*, YouTube) as a reliable source for constructing such a benchmark. We adopt a three-stage pipeline to develop EmbedCopilot-Bench.

(1) Video collection and subtitle-based task segmentation. We manually search channels on YouTube using curated keywords and identify a collection of videos that cover more than 5 mainstream embedded platforms and 27 functional modules, as illustrated in Fig. 2. These videos provide step-by-step instructions, where instructors demonstrate on-board hardware operations while sharing their computer screens and offering verbal explanations. We observed that subtitles can serve as an effective signal for segmenting individual tasks, thereby facilitating subsequent Q&A generation. Therefore, we crawl the videos and extract their subtitles using Whisper [41]. We then instruct GPT-5 to interpret each video's subtitles and decompose them into fine-grained, step-wise tasks, as shown in Fig. 3(a). The prompt for this process is detailed in supplementary materials.

(2) Q&A data generation. For each identified task, we construct a Q&A triplet with a textual task query, visual context, and answer, as shown in Fig. 3(b). The original subtitles are treated as the ground-truth answers, and we utilize GPT-5 to generate a task query based on the answer, which captures the overall objective of the task. In addition, we sample representative frames capturing the hardware status and development environment, and human annotators then correct Whisper transcription errors, refine ambiguous queries, and verify that the selected frames provide complete context for each Q&A triplet.

(3) Benchmark construction. We define three core aspects of embedded programming and label each Q&A triplet accordingly, enabling more fine-grained quantitative analysis of LVLM behavior. As shown in Fig. 3(c), we have:

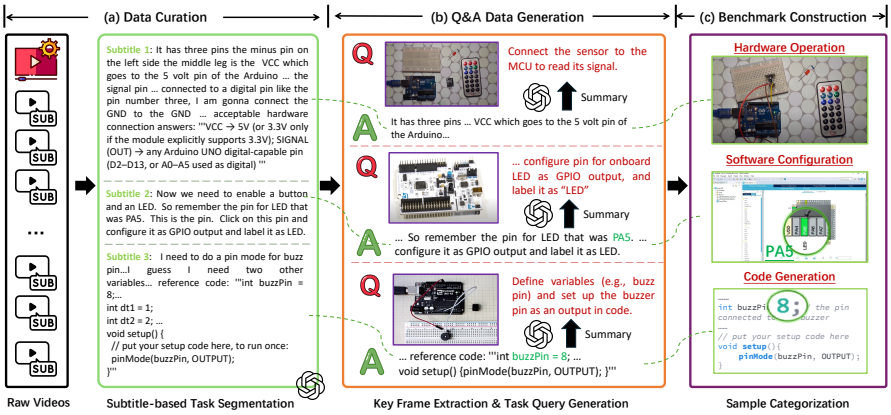

Fig. 3: Benchmark construction pipeline for EmbedCopilot-Bench. (a) **Data Curation:** we collect real-world embedded programming videos from YouTube and use Whisper transcripts to segment each video into fine-grained tasks. (b) **Q&A Data Generation:** for each task, GPT-5 summarizes the subtitles into a textual task query, while we extract key frames that capture the hardware setup and software environment, yielding multimodal Q&A triplets. (c) **Benchmark Construction:** each triplet is categorized into hardware operation, software configuration, or code generation, forming a structured benchmark for evaluating LVLMs as embedded programming copilots.

- **Hardware operation**: This refers to physical-world interactions with embedded devices, including practical wiring (*e.g.*, GPIO connections, breadboard wiring), SD card insertion, and USB connections between the embedded device and the computer. This capability requires the model to exhibit real-world understanding of wiring practices and underlying electronic principles, such as mapping module pins to the corresponding pins on the embedded device via a breadboard.
- **Software configuration**: This refers to operating-system-level and UI-level installation and setup, such as installing ESP32 drivers, flashing Ubuntu images for Raspberry Pi, selecting device ports, and other software-related operations required to prepare the hardware, all of which are completed solely on the computer. The models are evaluated on their ability to understand UI components of different applications, such as Raspberry Pi OS, Visual Studio, and the Device Manager in Windows.
- **Code generation**: This refers to scenarios where code must be produced to implement a specific function that satisfies the user's requirements. The models are evaluated on their ability to generate deployable code in multiple programming languages across different applications and IDEs.

Dataset statistics. Following the above pipeline, we obtain a comprehensive multimodal, multi-dimensional Q&A triplet dataset, EmbedCopilot-Bench, which contains 216 unique Q&A triplets and 235 capability labels. 63 triplets are tagged as *hardware operation*, 98 as *software configuration*, and 74 as *code*

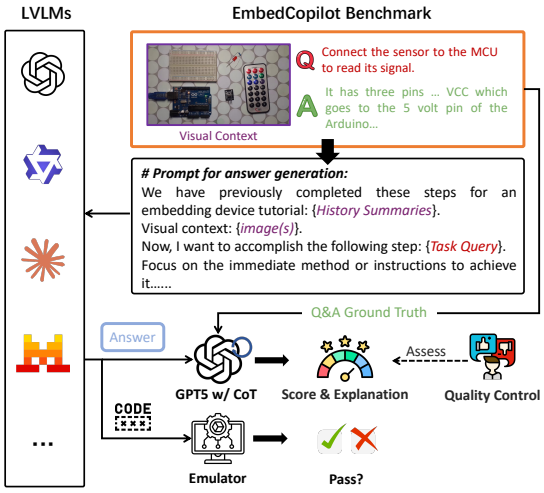

Fig. 4: Evaluation pipeline: Models predict next-step instructions from visual and textual context, with outputs scored by an LLM-as-a-Judge, verified by human annotators, and further validated via hardware (or an emulator) for code-related performance.

generation. Since a triplet may receive multiple labels (*e.g.*, both *hardware* and *code*), these per-dimension counts sum to more than the number of triplets.

3.2 CoT LLM-as-a-Judge for Open-ended Outputs

The questions and answers in EmbedCopilot-Bench are inherently open-ended: LVLM outputs could include natural-language explanations, multi-step procedures, and executable code. This makes it difficult to design simple token-level or exact-match metrics that faithfully capture quality differences between a generated answer and the ground-truth answer. Inspired by recent benchmark studies that use LLMs as automatic judges for free-form responses [35, 51], we adopt GPT-5 as our primary evaluator.

Evaluation pipeline. As shown in Fig. 4, for each Q&A triplet from EmbedCopilot-Bench, we feed the task query and visual context to the LVLM under evaluation. Additionally, we provide *history summaries* that explicitly describe what has already been done to reach the current state. The answer produced by the evaluated LVLM is then compared against the ground truth by the judge model. Inspired by prior work [9,33], we instruct the judge model to follow a rubric covering correctness, grounding, and actionability (see supplementary materials) and to assign a score on a 5-point Likert scale ($1 = \textit{very poor}$, $5 = \textit{excellent}$), along with a detailed explanation.

CoT LLM-as-a-Judge. To improve robustness and reduce sensitivity to local noise or single-pass hallucinations, we adopt an *iterative* CoT judging procedure, Scoring-Refine, inspired by self-reflection methods [33,37] and detailed

Algorithm 1 SCORING-REFINE (Iterative Judge with Final Reconciliation).

Require: q (question), g (ground truth), $\hat{y}$ (prediction), judge model $\mathcal{M}$; reflection rounds R; prompts $\mathcal{P} = \{p_{\text{init}}, p_{\text{ref}}, p_{\text{fin}}\}$

$(s_1, e_1) \leftarrow \mathcal{M}(p_{\text{init}} \,\|\, q \,\|\, g \,\|\, \hat{y})$ ▷ Initial score & explanation
$\mathcal{H} \leftarrow [(s_1, e_1)]$ ▷ Initialize scoring history
for $t = 2$ **to** R **do** ▷ Stop condition
  $(s_t, e_t) \leftarrow \mathcal{M}(p_{\text{ref}} \,\|\, q \,\|\, g \,\|\, \hat{y} \,\|\, (s_{t-1}, e_{t-1}))$ ▷ Self-reflect
  $\mathcal{H}$.append$((s_t, e_t))$ ▷ Store history
end for
$(s_{\text{final}}, e_{\text{final}}) \leftarrow \mathcal{M}(p_{\text{fin}} \,\|\, q \,\|\, g \,\|\, \hat{y} \,\|\, \mathcal{H})$ ▷ Reconcile all rounds
return $(s_{\text{final}}, e_{\text{final}})$

in Algorithm 1. Here, $\mathcal{P}$ denotes three instruction prompts for the judge model $\mathcal{M}$: an initial scoring prompt p_{init}, a self-reflection prompt p_{ref} that asks the model to reconsider its previous judgment, and a final reconciliation prompt p_{fin} that aggregates the scoring history into a single decision. In this procedure, $\mathcal{M}$ first generates an initial score and explanation (s_1, e_1) for $(q, g, \hat{y})$, then performs rounds of self-reflection by revisiting its previous scores and explanations, and finally outputs a reconciled decision $(s_{\text{final}}, e_{\text{final}})$, which we use as the evaluation output for that sample. In all experiments, we set the number of reflection rounds to $R = 3$ (one initial judgment and two self-reflection passes), which we found to provide a good trade-off between robustness and inference cost. In a nutshell, the judge model produces a raw Likert score $r_i \in \{1, \ldots, 5\}$ for each sample, conditioned on the question, ground-truth answer, and model output. We normalize r_i to $[0, 1]$ by $s_i = (r_i - 1)/4$. The overall score is $S = \frac{1}{N} \sum_{i=1}^{N} s_i \times 100\%$, and the capability-wise score is $S_C = \frac{1}{N_C} \sum_{i \in C} s_i \times 100\%$, where N is the total sample number, C denotes the subset of samples for a capability label, and $N_C = |C|$.

3.3 Execution-based Judge for Code Outputs

To complement LLM-as-a-Judge, for code generation, we further apply a stricter and more precise evaluation by executing the generated code in a hardware emulator environment to verify whether it achieves the target behavior.

Execution-based judge. We mainly employ Wokwi [1] as the test environment, a web-based platform that supports a wide range of development platforms, functional modules and peripheral configurations (*e.g.*, most Arduino- and ESP32-based setups), enabling us to emulate I/O connections, firmware execution, and observable behaviors such as pin toggling and serial output. We treat multiple consecutive tasks whose outputs contribute code toward a common goal (*e.g.*, enabling an LED blinking pattern or producing a correct sensor readout) as a single meta-task. For each meta-task, we set up a Wokwi project that mirrors the ground-truth hardware wiring diagram and inject the model-generated code to verify whether the target behavior is achieved correctly. In addition, we adopt hardware-in-the-loop execution on real devices to quantify code quality in cases not supported by Wokwi, including STM32 projects that

rely on STM32CubeIDE and Raspberry Pi workflows. For each meta-task, we apply a rigorous human audit process as follows. First, a trained rater (i) assembles and compiles the full model-generated program, (ii) flashes the resulting binary to the target MCU, (iii) runs the physical rig (*e.g.*, Arduino + motor), and (iv) performs *blinded human adjudication*, verifying whether all pre-registered acceptance criteria are satisfied (*e.g.*, speed, direction, latency, serial output). Second, we ask a second rater to review the outcome, and any disagreements are resolved by consensus. With dataset release, we also provide our Wokwi project files, together with a detailed description of the physical hardware setups, and complete per-meta-task evaluation metadata, including the target behavior, pre-registered acceptance criteria, and scripts for reproducing the ESR evaluation.

Execution success rate (ESR). For the execution-based judge, we use the ESR as the evaluation metric to quantify whether model-generated code satisfies the desired end-to-end device behavior. Following Pass@1-style code-generation evaluation [10, 42], we define two versions of the execution success criterion. **Strict:** a code sample is counted as correct only if (i) it drives all component pins according to the ground-truth hardware connections, and (ii) it successfully completes the target behavior (either in emulator or on physical hardware). **Permissive:** we define a *permissive*, pin-error-tolerant criterion, where the code is allowed to make mistakes only in pin-number recognition (*i.e.*, using a different pin index than the one wired in hardware), but is counted as correct if it can achieve the desired functional behavior after pin correction. We intentionally restrict permissiveness to pin-index errors to isolate the visual grounding bottleneck and avoid platform-dependent subjectivity from broader "small logic fixes", making the Strict–Permissive gap a clean proxy for visual–physical alignment deficits with minimal evaluation noise.

3.4 Quality Control with Human Annotators

We adopt two complementary procedures to ensure the quality of both the dataset and the evaluation judges.

(i) Benchmark construction. To ensure the quality of the Q&A triplets, human annotators manually review and correct transcription errors produced by Whisper. For each task query summarized by GPT-5, we also manually verify its completeness. In addition, we mask information that is already visible in the captured images so that the actual image-understanding capability of LVLMs can be properly evaluated. For selecting frames to serve as visual context, we adopt a hybrid pipeline in which we first apply heuristic rules to automatically propose candidate frames, and then manually screen them to select those that meet our quality requirements.

(ii) Human verification of the LLM judge. To verify the reliability of LLM-as-a-Judge, we adopt two complementary human-based checks. First, following prior LLM-as-a-Judge validation practices [35, 51], we randomly sample 172 judged responses to construct a manually scored gold subset. Three PhD-level researchers with expertise in embedded systems and programming independently assign 1–5 Likert scores using the same rubric as GPT-5, enabling a direct

assessment of human–LLM score agreement. Second, we require the judge to output both a score and a concise explanation for each open-ended QA comparison. 12 PhD-level researchers then assess the *consistency* between the score and its explanation via a dedicated web UI that supports efficient browsing and labeling. Together, these checks evaluate both the agreement between GPT-5 and human scores and whether the judge's rationales support its assigned scores.

4 Evaluation Results

4.1 Experiment Settings

We evaluate a suite of representative LVLMs on the EmbedCopilot-Bench. We consider two categories:❶ **Closed-source LVLMs**: Claude-Sonnet-4.5 [4], GPT-5 [40], GPT-4o [21], and Gemini-2.5-Pro [17]; and ❷ **Open-source LVLMs**: Qwen2.5-VL-7B [8], Qwen3-VL-2B/4B/8B [7], InternVL3-8B [55], Mistral-Small-3-24B [38], LLaVA-OneVision-7B [25], and Gemma-3-4B [43].

Each model is evaluated under two input configurations: *text-only* (T), where the model receives only the text query, and *with images* (+I), where we additionally provide visual context, *i.e.*, hardware status and computer screen snapshots. For all evaluated models, we apply a unified prompt as illustrated in Fig. 4. Model outputs are scored on a 1–5 scale by GPT-5 acting as an LLM-as-a-Judge, following the protocol described in Sec. 3.2. Since LLM-based judging can exhibit slight stochasticity, we repeat the judging process 5 times for each model and input configuration, and report mean ± standard deviation for the overall and capability-wise scores. In addition to these scores, we also evaluate the Execution Success Rate (ESR) described in Sec. 3.3.

4.2 LLM-based Results

Table 1 summarizes the performance of both closed- and open-source LVLMs on EmbedCopilot-Bench under our LLM-as-a-Judge evaluation (averaged over 5 independent GPT-5 judging runs). We highlight three key observations.

First, **closed-source LVLMs consistently lead in overall judge scores, but are not yet saturated on hardware grounding**. Under the with-images setting, Claude-Sonnet-4.5 and GPT-5 achieve the best overall scores (89.6 and 89.2), followed by GPT-4o (87.0) and Gemini-2.5-Pro (86.6). Across these models, software configuration remains relatively strong ($\text{LLM}_{\text{sw}} \approx 93$–$97$), while hardware operation and code generation still show headroom ($\text{LLM}_{\text{hw}} \approx 83$–$88$; $\text{LLM}_{\text{code}} \approx 82$–$84$), suggesting that fine-grained physical grounding and executable code synthesis remain challenging even for state-of-the-art LVLMs.

Second, **visual context generally improves judge scores, especially for LLM_{hw}, but gains are model-dependent**. For example, GPT-4o's LLM_{hw} score increases from 65.6 (T) to 84.7 (+I), accompanied by an overall gain from 80.3 to 87.0. Similarly, Qwen2.5-VL-7B improves on LLM_{hw} (43.2→59.0) and Code (56.1→66.1) when images are provided. However, the benefit is not universal: for Gemma-3-4B and LLaVA-OneVision-7B, with-images and text-only

Table 1: Performance comparison of LVLMs with ESR. All open-source models are evaluated in their instruction-tuned variants. Mod.: input modality (T: text-only; +I: with images). LLM_{hw}: hardware-operation; LLM_{sw}: software-configuration; LLM_{code}: code-generation. We additionally report execution success rates (ESR) on executable tasks per setting: E_{str} requires exact pin correctness and task completion; E_{per} also counts runs that complete the task with minor pin errors.

Model	Mod.	LLM_{hw}	LLM_{sw}	LLM_{code}	LLM_{all}	E_{str}	E_{per}
Closed-source LVLMs							
Claude-Sonnet-4.5	T	79.7±3.1	96.1±1.8	84.5±2.8	87.4±2.4	48.3%	82.8%
	+I	86.5±3.2	95.3±2.2	83.6±2.5	**89.6±2.3**	44.8%	86.2%
GPT-5	T	74.5±3.3	95.5±2.1	77.8±3.0	83.6±2.5	17.2%	27.6%
	+I	88.2±3.4	96.9±1.4	81.6±3.0	**89.2±2.3**	31.0%	58.6%
GPT-4o	T	65.6±3.9	93.8±2.2	76.9±2.1	80.3±2.4	37.9%	62.1%
	+I	84.7±3.9	93.3±2.4	81.7±2.7	**87.0±2.6**	48.3%	79.3%
Gemini-2.5-Pro	T	71.7±2.9	92.7±2.2	79.1±2.8	82.4±2.4	31.0%	44.8%
	+I	82.7±2.9	93.3±1.6	82.2±2.7	**86.6±2.3**	41.4%	55.2%
Open-source LVLMs							
Mistral-Small-3-24B	T	66.7±1.9	88.8±2.5	77.4±3.9	79.0±2.4	31.0%	58.6%
	+I	72.2±3.4	88.2±2.4	78.5±1.3	**80.8±2.2**	37.9%	82.8%
InternVL3-8B	T	56.0±3.6	80.6±3.9	68.9±3.7	70.2±3.8	13.8%	24.1%
	+I	65.0±4.0	83.5±3.3	70.5±2.8	**74.4±3.0**	13.8%	31.0%
Qwen3-VL-8B	T	57.4±3.5	81.2±4.7	63.0±4.3	68.3±4.1	10.3%	13.8%
	+I	71.9±3.6	77.9±3.5	69.1±3.3	**74.0±3.3**	6.9%	24.1%
Qwen2.5-VL-7B	T	43.2±2.5	56.3±3.8	56.1±3.0	52.1±3.3	13.8%	17.2%
	+I	59.0±3.3	73.8±4.3	66.1±4.2	**67.3±3.6**	20.7%	34.5%
Qwen3-VL-4B	T	54.4±2.7	75.6±3.2	61.2±2.4	64.9±2.8	10.3%	17.2%
	+I	60.8±2.7	70.1±2.2	62.8±2.5	**65.6±2.3**	13.8%	24.1%
Gemma-3-4B	T	53.3±2.0	71.9±2.5	59.5±1.8	**62.6±1.8**	6.9%	17.2%
	+I	52.0±3.4	71.6±3.6	58.9±2.0	62.2±2.6	6.9%	17.2%
LLaVA-OneVision-7B	T	50.2±3.3	73.5±2.9	56.3±2.4	**61.5±2.7**	3.4%	10.3%
	+I	52.6±3.0	71.0±4.2	55.3±2.6	60.6±3.3	3.4%	10.3%
Qwen3-VL-2B	T	39.7±1.0	53.5±3.0	48.7±1.7	47.9±1.9	3.4%	10.3%
	+I	47.1±1.6	61.9±1.3	53.9±1.0	**54.9±0.9**	0.0%	0.0%

scores are broadly comparable, suggesting that for smaller-capacity models, visual perception and cross-modal grounding can be unreliable in this setting. As a result, image inputs may introduce additional noise or misinterpretations that offset any potential gains from visual context.

Third, **within a controlled single-family comparison (Qwen3-VL-2B/4B/8B), effective use of images appears to require sufficient model capacity**. With images, the score increases monotonically from 54.9 (2B) to 65.6 (4B) and 74.0 (8B). Notably, the hardware dimension benefits the most at larger scale: Qwen3-VL-8B improves its LLM_{hw} from 57.4 (T) to 71.9 (+I), whereas smaller variants show smaller gains. This suggests that visual input becomes reliably actionable beyond a certain capacity, especially for hardware grounding.

4.3 Execution Success Rate (ESR) Results

Table 1 also reports ESR on executable tasks, which provides a stricter end-to-end validation beyond judge scores. We highlight three key insights.

First, **execution remains substantially harder than achieving high judge scores, even for closed-source models**. Although closed-source models attain the strongest $\mathrm{LLM}_{\mathrm{code}}$ scores (often > 80), their strict execution success rates are much lower (*e.g.*, $\mathrm{E}_{\mathrm{str}}$ peaks at 48.3%). This gap shows that plausible code generation alone is insufficient for end-to-end success, which additionally requires robust visual grounding and execution-faithful reasoning.

Second, **pin-related near-misses account for a large fraction of failures, and permissive reveals this clearly**. Across most models, $\mathrm{E}_{\mathrm{per}}$ is markedly higher than $\mathrm{E}_{\mathrm{str}}$, suggesting that many "failures" are due to minor pin mis-specification rather than incorrect task logic. For example, Mistral-Small-3-24B rises from 37.9% to 82.8% and Claude-Sonnet-4.5 from 44.8% to 86.2% (+I), indicating pin-level grounding as a key bottleneck.

Third, **stronger models are bottlenecked by pin-level grounding, while weaker models are dominated by real failures**. Top models achieve high permissive success, indicating that many remaining errors are near-misses primarily caused by pin mismatches. In contrast, smaller open-source models remain low even under the permissive criterion (*e.g.*, LLaVA-OneVision-7B stays at 10.3%), suggesting more fundamental reasoning or code-generation failures beyond pin selection. Overall, this decomposition implies that improving pin-level grounding is the main lever for strong models, whereas weaker models require broader gains in planning and execution-faithful code synthesis.

4.4 Effectiveness of LLM-as-a-Judge Evaluation

Following the human-verification protocol described in Sec. 3.4, we assess the reliability of GPT-5 from two complementary perspectives: agreement with human scores and consistency between its assigned scores and accompanying rationales.

Human–LLM score agreement. On the manually scored gold subset of 172 sampled model responses, GPT-5 achieves averaged absolute differences of 0.058, 0.048, and 0.080 relative to the three independent expert annotators, respectively, with an overall mean of 0.062 (Table 2).

Human audit of judge rationales. We audit whether the judge's scores are well supported by its rubric-based explanations. Twelve PhD-level embedded-systems experts independently rated explanation–score consistency for 172 judged responses on a 5-point Likert scale. The mean consistency rating is 4.7/5, with a skewed distribution (75.4% rated 5; 96.1% rated ≥ 4). The **within-1 agreement** is the mean pairwise agreement over all $\binom{12}{2} = 66$ rater pairs: for each pair, we compute the fraction of audited items on which their ratings differ by at most 1 point, and then average this fraction across pairs; we obtain 93.7%. These audits suggest that the judge's rubric-based rationales are generally consistent with its assigned scores.

Table 2: GPT-5–human score agreement on the gold subset. All raters used the same 1–5 rubric. Scores are normalized to $[0, 1]$.

Metric	GPT-5–Human Agreement (3 Human Raters)
Averaged absolute difference	0.058 / 0.048 / 0.080
Overall mean	**0.062**

Table 3: Inter-judge robustness of LLM-as-a-Judge. We report the model-level Spearman correlation of average scores across 12 models, and the sample-level MAE between two judges' per-sample scores on the normalized 0–1 scale.

Alternative Judge	Open-weight?	Model-level ρ	Sample MAE (0–1)
Qwen3-235B	✓	0.909	0.145
GLM-4.7	✓	0.916	0.152
DeepSeek-V3.2	✓	0.832	0.206
Claude-Sonnet-4.5	×	0.909	0.195
Gemini-2.5-Pro	×	0.888	0.163
GPT-4o	×	0.916	0.108

Judge robustness and reproducibility. To ensure our evaluation is not tied to a single proprietary judge, we compare GPT-5 against diverse open-weight and closed-source judges, including Qwen3-235B [48], GLM-4.7 [53], DeepSeek-V3.2 [31], Claude-Sonnet-4.5 [4], Gemini-2.5-Pro [17], and GPT-4o [21] (Table 3).

Across 12 prediction models (5,184 judged responses), model-level rankings remain highly consistent across judges (Spearman $\rho = 0.832$–0.916), indicating that our key conclusions are largely judge-agnostic. At the sample level, judges show modest calibration differences, measured by mean absolute error (MAE) between two judges' per-sample scores on the normalized 0–1 scale (MAE = 0.108–0.206); importantly, these shifts primarily affect fine-grained score calibration rather than the overall ordering trends captured by model-level correlations. Finally, open-weight judges achieve similar model-level agreement with GPT-5, supporting reproducible evaluation independent of proprietary APIs.

4.5 Case Study

Our ESR experiments indicate that the permissive evaluation reveals a large fraction of near-miss failures, especially those related to pin localization. To better understand these errors, we present a representative case study. As shown in Fig. 5, we analyze the strongest coding model under our permissive metric and investigate why its strict success rate remains unsatisfactory.

In this task, the LVLM is first asked to identify the two yellow signal wires connecting a servo motor and a potentiometer to an Arduino Uno, and to output the corresponding pin numbers. Based on this inferred wiring, the model must then generate Arduino code that correctly initializes the signal pins for both components. Claude-Sonnet-4.5 correctly localizes the servo signal wire on digital pin 9 and propagates this assignment into the code. However, for the

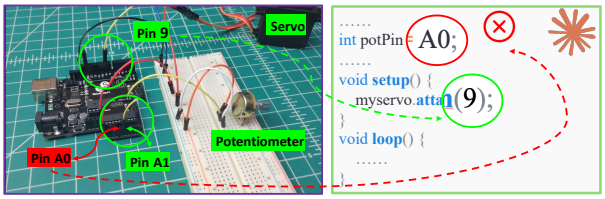

Fig. 5: Case study of pin-localization near miss. Claude-Sonnet-4.5 correctly identifies the servo signal wire on digital pin 9 and initializes it in code, but misidentifies the potentiometer signal wire as analog pin A0 instead of the correct pin A1, leading to an execution failure under strict evaluation.

potentiometer, the model predicts analog pin A0 instead of the correct pin A1, leading to an incorrect code initialization. This mistake likely stems from subtle occlusion of the silkscreen legends on the PCB and the slightly rotated camera viewpoint, which make the printed labels less salient. In contrast, a real-world user can reliably infer the correct pin by reasoning about the relative layout of the headers and counting pins from a known reference, even when the text label is partially occluded. This example illustrates both the promise and the limitations of current LVLMs: while they can often localize visible pins correctly, small viewpoint changes or occlusions can disrupt their predictions, revealing a deeper lack of structured hardware reasoning and highlighting room for improvement.

5 Conclusion

We presented EmbedCopilot-Bench, a multimodal benchmark for end-to-end embedded-device development, comprising annotated QA triplets across heterogeneous hardware platforms and peripheral modules, together with a hybrid evaluation protocol that combines rubric-driven LLM judging and execution-based ESR. Experiments show that visual context helps, yet models remain far from reliable under execution. Our ESR analysis further identifies pin grounding as a key bottleneck (large Strict–Permissive gap), motivating future work on more robust pin localization in LVLMs under viewpoint changes and occlusions to better align visual grounding with executable firmware.

Acknowledgments

We sincerely thank the anonymous AC and reviewers for their valuable comments. This work is partially supported by Guangdong General Higher Education Institution (Young Innovative Talents) (No. 2025KQNCX113), Guangzhou Municipal Science and Technology Project (No. SL2024A04J01392), Guangzhou-HKUST(GZ) Joint Funding Program (No. SL2024A03J01192), the Department of Education of Guangdong Province (Innovation Team, No. 2024KCXTD008), and Guangdong Provincial Key Lab of Integrated Communication, Sensing and Computation for Ubiquitous Internet of Things (No. 2023B1212010007).

References

1. Wokwi: Online simulator for embedded development. https://wokwi.com/, accessed: 2025-10-13
2. Alayrac, J.B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al.: Flamingo: a visual language model for few-shot learning. Advances in neural information processing systems **35**, 23716–23736 (2022)
3. Anthropic: Claude code. https://www.claude.com/product/claude-code (2025), accessed: 2026-06-24
4. Anthropic: Claude sonnet 4.5 system card. https://assets.anthropic.com/m/12f214efcc2f457a/original/Claude-Sonnet-4-5-System-Card.pdf (2025), accessed: 2025-11-12
5. Anysphere, Inc.: Cursor: An ai-powered coding assistant. https://cursor.com (2023), accessed: 2026-06-24
6. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021)
7. Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., et al.: Qwen3-vl technical report. arXiv preprint arXiv:2511.21631 (2025)
8. Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al.: Qwen2. 5-vl technical report. arXiv preprint arXiv:2502.13923 (2025)
9. Bai, Y., Ying, J., Cao, Y., Lv, X., He, Y., Wang, X., Yu, J., Zeng, K., Xiao, Y., Lyu, H., et al.: Benchmarking foundation models with language-model-as-an-examiner. Advances in Neural Information Processing Systems **36**, 78142–78167 (2023)
10. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
11. Chiang, C.H., Lee, H.y.: Can large language models be an alternative to human evaluations? In: Rogers, A., Boyd-Graber, J., Okazaki, N. (eds.) Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 15607–15631. Association for Computational Linguistics, Toronto, Canada (Jul 2023). https://doi.org/10.18653/v1/2023.acl-long.870, https://aclanthology.org/2023.acl-long.870/, accessed: 2026-06-24
12. Englhardt, Z., Li, R., Nissanka, D., Zhang, Z., Narayanswamy, G., Breda, J., Liu, X., Patel, S., Iyer, V.: Exploring and characterizing large language models for embedded system development and debugging. In: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems. CHI EA '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3613905.3650764, https://doi.org/10.1145/3613905.3650764, accessed: 2026-06-24
13. Fu, J., Ng, S.K., Jiang, Z., Liu, P.: Gptscore: Evaluate as you desire. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). pp. 6556–6576 (2024)
14. Fu, L., Kuang, Z., Song, J., Huang, M., Yang, B., Li, Y., Zhu, L., Luo, Q., Wang, X., Lu, H., Li, Z., Tang, G., Shan, B., Lin, C., Liu, Q., Wu, B., Feng,

H., Liu, H., Huang, C., Tang, J., Chen, W., Jin, L., Liu, Y., Bai, X.: OCR-Bench v2: An improved benchmark for evaluating large multimodal models on visual text localization and reasoning. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2025), https://openreview.net/forum?id=Vb6i3Dp24N, accessed: 2026-06-24
15. GitHub, Inc.: Github copilot. https://github.com/features/copilot (2021), accessed: 2025-11-12
16. Gong, K., Dong, W., Peng, Y., Wang, H., Gao, Y.: Poster: Enabling iot application programming in natural language with iotpilot. In: Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems. pp. 903–904 (2024)
17. Google DeepMind: Gemini 2.5 pro model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Pro-Model-Card.pdf (2025), accessed: 2025-11-12
18. Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., Parikh, D.: Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 6904–6913 (2017)
19. Guo, D., Xu, C., Duan, N., Yin, J., McAuley, J.: Longcoder: A long-range pre-trained language model for code completion. In: International Conference on Machine Learning. pp. 12098–12107. PMLR (2023)
20. Hudson, D.A., Manning, C.D.: Gqa: A new dataset for real-world visual reasoning and compositional question answering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 6700–6709 (2019)
21. Hurst, A., Lerer, A., Goucher, A.P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al.: Gpt-4o system card. arXiv preprint arXiv:2410.21276 (2024)
22. Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S.: A survey on large language models for code generation. arXiv preprint arXiv:2406.00515 (2024)
23. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: Swe-bench: Can language models resolve real-world github issues? arXiv preprint arXiv:2310.06770 (2023)
24. Lachaux, M.A., Roziere, B., Chanussot, L., Lample, G.: Unsupervised translation of programming languages. arXiv preprint arXiv:2006.03511 (2020)
25. Li, B., Zhang, Y., Guo, D., Zhang, R., Li, F., Zhang, H., Zhang, K., Zhang, P., Li, Y., Liu, Z., Li, C.: Llava-onevision: Easy visual task transfer. arXiv preprint arXiv:2408.03326 (2024)
26. Li, J., Chen, L., Yang, B., Zhu, J., Wang, Y., Ma, Y., Yang, M.: PCB-bench: Benchmarking LLMs for printed circuit board placement and routing. In: The Fourteenth International Conference on Learning Representations (2026), https://openreview.net/forum?id=Q5QLu7XTWx, accessed: 2026-06-24
27. Li, J., Li, D., Savarese, S., Hoi, S.: Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In: International conference on machine learning. pp. 19730–19742. PMLR (2023)
28. Li, K., Tian, Y., Hu, Q., Luo, Z., Huang, Z., Ma, J.: Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems. In: Findings of the Association for Computational Linguistics: EMNLP 2024. pp. 736–783 (2024)
29. Li, M., Zhong, J., Chen, T., Lai, Y., Psounis, K.: Eee-bench: A comprehensive multimodal electrical and electronics engineering benchmark. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 13337–13349 (2025)

30. Liang, J.T., Yang, C., Myers, B.A.: A large-scale survey on the usability of ai programming assistants: Successes and challenges. In: Proceedings of the 46th IEEE/ACM international conference on software engineering. pp. 1–13 (2024)
31. Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., et al.: Deepseek-v3. 2: Pushing the frontier of open large language models. arXiv preprint arXiv:2512.02556 (2025)
32. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. Advances in neural information processing systems **36**, 34892–34916 (2023)
33. Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-eval: Nlg evaluation using gpt-4 with better human alignment. arXiv preprint arXiv:2303.16634 (2023)
34. Liu, Y., Li, Z., Huang, M., Yang, B., Yu, W., Li, C., Yin, X.C., Liu, C.L., Jin, L., Bai, X.: Ocrbench: on the hidden mystery of ocr in large multimodal models. Science China Information Sciences **67**(12) (Dec 2024). https://doi.org/10.1007/s11432-024-4235-6, http://dx.doi.org/10.1007/s11432-024-4235-6, accessed: 2026-06-24
35. Liu, Z., Chu, T., Zang, Y., Wei, X., Dong, X., Zhang, P., Liang, Z., Xiong, Y., Qiao, Y., Lin, D., et al.: Mmdu: A multi-turn multi-image dialog understanding benchmark and instruction-tuning dataset for lvlms. Advances in Neural Information Processing Systems **37**, 8698–8733 (2024)
36. Lu, P., Bansal, H., Xia, T., Liu, J., Li, C., Hajishirzi, H., Cheng, H., Chang, K.W., Galley, M., Gao, J.: Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. arXiv preprint arXiv:2310.02255 (2023)
37. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al.: Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems **36**, 46534–46594 (2023)
38. Mistral AI: Mistral small 3.1. https://mistral.ai/news/mistral-small-3-1 (2025), accessed: 2025-11-12
39. Muennighoff, N., Liu, Q., Zebaze, A., Zheng, Q., Hui, B., Zhuo, T.Y., Singh, S., Tang, X., Von Werra, L., Longpre, S.: Octopack: Instruction tuning code large language models. In: NeurIPS 2023 workshop on instruction tuning and instruction following (2023)
40. OpenAI: Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf (2025), accessed: 2025-11-12
41. Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust speech recognition via large-scale weak supervision. In: International conference on machine learning. pp. 28492–28518. PMLR (2023)
42. Shen, L., Yang, Q., Zheng, Y., Li, M.: Autoiot: Llm-driven automated natural language programming for aiot applications. In: Proceedings of the 31st Annual International Conference on Mobile Computing and Networking. pp. 468–482 (2025)
43. Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al.: Gemma 3 technical report. arXiv preprint arXiv:2503.19786 (2025)
44. Wang, K., Pan, J., Shi, W., Lu, Z., Ren, H., Zhou, A., Zhan, M., Li, H.: Measuring multimodal mathematical reasoning with math-vision dataset. Advances in Neural Information Processing Systems **37**, 95095–95169 (2024)
45. Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., et al.: Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. arXiv preprint arXiv:2409.12191 (2024)

46. Wu, C., Liang, Z., Ge, Y., Guo, Q., Lu, Z., Wang, J., Shan, Y., Luo, P.: Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In: Findings of the Association for Computational Linguistics: NAACL 2025. pp. 3006–3028 (2025)
47. Xu, R., Cao, J., Wu, M., Zhong, W., Lu, Y., He, B., Han, X., Cheung, S.C., Sun, L.: Embedagent: Benchmarking large language models in embedded system development. arXiv preprint arXiv:2506.11003 (2025)
48. Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al.: Qwen3 technical report. arXiv preprint arXiv:2505.09388 (2025)
49. Yang, H., Li, M., Han, M., Li, Z., Xu, W.: Embedgenius: Towards automated software development for generic embedded iot systems. arXiv preprint arXiv:2412.09058 (2024)
50. Yang, J., Prabhakar, A., Narasimhan, K., Yao, S.: Intercode: Standardizing and benchmarking interactive coding with execution feedback. Advances in Neural Information Processing Systems **36**, 23826–23854 (2023)
51. Yu, W., Yang, Z., Li, L., Wang, J., Lin, K., Liu, Z., Wang, X., Wang, L.: Mm-vet: Evaluating large multimodal models for integrated capabilities. arXiv preprint arXiv:2308.02490 (2023)
52. Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., et al.: Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9556–9567 (2024)
53. Zeng, A., Lv, X., Zheng, Q., Hou, Z., Chen, B., Xie, C., Wang, C., Yin, D., Zeng, H., Zhang, J., et al.: Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471 (2025)
54. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al.: Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in neural information processing systems **36**, 46595–46623 (2023)
55. Zhu, J., Wang, W., Chen, Z., Liu, Z., Ye, S., Gu, L., Tian, H., Duan, Y., Su, W., Shao, J., et al.: Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. arXiv preprint arXiv:2504.10479 (2025)
56. Zou, C., Guo, X., Yang, R., Zhang, J., Hu, B., Zhang, H.: Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. arXiv preprint arXiv:2411.00836 (2024)