ArtifactsBench: Bridging the Visual-Interactive Gap in LLM Code Generation Evaluation

Tencent Hunyuan Team

Abstract

The generative capabilities of Large Language Models (LLMs) are rapidly expanding from static code to dynamic, interactive visual artifacts. This progress is bottlenecked by a critical evaluation gap: established benchmarks focus on algorithmic correctness and are blind to the visual fidelity and interactive integrity that define modern user experiences. To bridge this gap, we introduce **ArtifactsBench**, a new benchmark and paradigm for the automated, multimodal evaluation of visual code generation. Our framework programmatically renders each generated artifact and captures its dynamic behavior through temporal screenshots. This visual evidence, alongside the source code, is then assessed by a Multimodal LLM (MLLM)-as-Judge, which is rigorously guided by a **fine-grained, per-task checklist** to ensure holistic and reproducible scoring. We construct a new benchmark of **1,825 diverse tasks** and evaluate over 30 leading LLMs. Our automated evaluation achieves a striking **94.4% ranking consistency with WebDev Arena**, the gold-standard for human preference in web development, and over 90% pairwise agreement with human experts. This establishes ArtifactsBench as the first framework to reliably automate the assessment of human-perceived quality at scale. Our analysis provides a high-resolution map of the current SOTA, revealing that generalist models often outperform domain-specific ones. We open-source ArtifactsBench, including the benchmark, evaluation harness, and baseline results at https://artifactsbenchmark.github.io/, to provide the community with a scalable and accurate tool to accelerate the development of user-centric generative models.

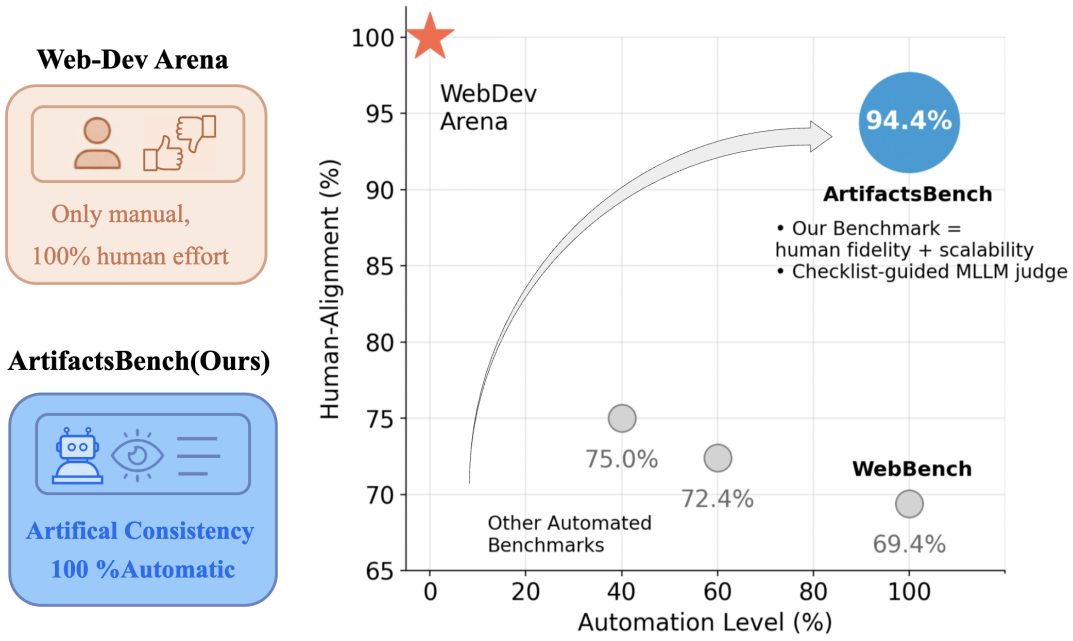

Figure 1: Automation level versus human–alignment across evaluation frameworks. The red star marks the fully manual WebDev Arena (100% human effort), while the blue bubble denotes our checklist-guided MLLM evaluation, ArtifactsBench, which achieves 94.4% agreement with human votes with 100% automation. Grey circles represent prior automated benchmarks.

1 Introduction

Large Language Models (LLMs) are catalyzing a paradigm shift in software creation, extending their generative capabilities from traditional code and text to the dynamic and sophisticated realm of interactive visual artifacts (Jaech et al., 2024; Anthropic, 2025; Guo et al., 2025). This evolution is enabling novel applications, from crafting responsive web interfaces and data visualizations to developing interactive game environments (Jiang et al., 2024;

Lu et al., 2025a). A cornerstone of this new paradigm is the "Artifact"—not just a static code snippet, but a self-contained, model-generated, and executable entity (e.g., a web widget, a data visualization) that synthesizes code, visual presentation, and interaction logic for users to directly inspect, manipulate, and experience. This fosters a fluid, real-time collaborative workflow between humans and AI. However, while the generative potential of LLMs is immense, our ability to rigorously and comprehensively evaluate their outputs in this domain lags significantly. This evaluation gap has become a critical bottleneck, impeding targeted improvements and the systematic advancement of this technology. Figure 1 starkly illustrates this challenge, positioning existing evaluation frameworks in a trade-off between automation and human-alignment—a gap that ArtifactsBench is explicitly designed to bridge.

Current evaluation methodologies for code generation are ill-equipped for this new interactive frontier. Prevailing benchmarks predominantly focus on static code attributes, such as syntactic correctness (e.g., pass@k in HumanEval (Chen et al., 2021)) or functional task completion in non-visual contexts (e.g., resolving GitHub issues in SWE-bench (Jimenez et al., 2023)). Others assess visual code generation from a static perspective, by replicating visual designs from images (Wüst et al., 2024; Yun et al., 2024; Wu et al., 2024), translating descriptions into structured graphics (Rodriguez et al., 2023; Xing et al., 2025), or relying on non-visual proxies for fidelity like DOM tree comparisons (Xu et al., 2025). These approaches, however, fail to capture the holistic quality of interactive artifacts. They cannot quantify crucial aspects of visual fidelity—such as layout integrity and aesthetic coherence—nor can they validate the correctness and fluidity of dynamic user interactions, like button responses, state transitions, or animations. Consequently, evaluation often defaults to costly and subjective manual inspection or unreliable LLM self-evaluation, which lack the scale, objectivity, and multimodal acuity required for robust scientific assessment.

This paper confronts the central research question: ***How can we automatically and holistically evaluate an LLM's ability to transform multimodal instructions—spanning text, images, and interactional logic—into high-quality, interactive visual artifacts?*** We argue that a successful evaluation framework must transcend static code analysis and embrace the multifaceted nature of the user experience. Such a framework must assess not only the code's functional correctness but also its visual presentation and, critically, its dynamic behavior over time. It must provide fine-grained, diagnostic feedback to illuminate specific model strengths and weaknesses, thereby guiding future research.

To this end, we introduce ArtifactsBench, a comprehensive benchmark designed to systematically evaluate LLMs on the creation of interactive visual artifacts. We have constructed a new, large-scale benchmark of **1,825 diverse tasks**, meticulously curated through a multi-stage pipeline that blends expert sourcing with LLM-based generation and refinement. As illustrated in Figure 2, these tasks span nine primary domains—from web development and data visualization to interactive games—and are stratified by complexity to enable a fine-grained analysis of model capabilities. This rigorous design ensures ArtifactsBench serves as a challenging and ecologically valid testbed for the next generation of visual code generators. Our primary contributions are threefold:

- **A Diverse and Hierarchical Benchmark Suite.** ArtifactsBench comprises a rich set of tasks derived from real-world applications, including component-based web development, SVG-based data visualization, and interactive mini-games. Tasks are stratified by complexity (simple, medium, hard) to robustly measure model capabilities across a meaningful difficulty gradient.
- **A Novel Multimodal and Automated Evaluation Pipeline.** We propose a new evaluation strategy that synergizes automated interaction with Multimodal Large Language Model (MLLM)-based assessment. Our framework programmatically interacts with the generated artifacts (e.g., clicking buttons, dispatching events) and captures visual states (e.g., screenshots, GIFs). An MLLM-as-Judge then evaluates these visual and textual traces against fine-grained, per-task criteria, a methodology inspired by recent advances in MLLM-based evaluation (Zheng et al., 2023; Ge et al., 2023; Zhang et al., 2025a). This enables an end-to-end, automated assessment of both visual fidelity and interactive correctness.
- **In-depth Analysis and Gold-Standard Validation.** We conduct a large-scale evaluation of over 30 prominent LLMs on ArtifactsBench. Our results are not only cross-validated with human experts but also benchmarked against the ultimate arbiter of visual quality: human preference. **Crucially, we demonstrate that ArtifactsBench is the first automated framework whose rankings achieve 94.4% consistency with WebDev Arena**, the de-facto gold standard which relies on large-scale human voting. This validates our automated approach as a highly reliable proxy for human-perceived quality, and our analysis charts a clear course for future development by revealing current limitations and a surprising insight: the holistic capabilities of generalist models often surpass those of specialized ones in this complex, multimodal creative task.

2 ArtifactsBench: A Benchmark for Visual Code Generation

2.1 Overview

We introduce **ArtifactsBench**, a new large-scale benchmark designed to evaluate the capability of Large Language Models (LLMs) in generating complex, real-world visual code artifacts. Unlike existing benchmarks that primarily

Figure 2: An overview of the ArtifactsBench dataset, illustrating the distribution of tasks across nine primary categories. This highlights the benchmark's broad coverage of real-world application domains, from Game Development to Web Applications and Data Science.

Benchmark	Data Size	Data Source	Primary Task	GR	CHA	AF	VE
Humaneval (Chen et al., 2021)	164	Human-Written	Algorithmic Tasks	Low	High	✓	×
SWE-Bench (Jimenez et al., 2023)	2,294	GitHub Issues	Repository-level Bug Fixing	Low	High	✓	×
WebBench (Xu et al., 2025)	1,000	Human-Written	Web Task Automation	Mid	Mid	✓	×
WebGen-Bench (Lu et al., 2025b)	101 Instructs	Human & GPT-4	Web Page Generation	Mid	Mid	✓	✓
WebChoreArena (Miyai et al., 2025)	532	Curated Tasks	Web Automation (No UI)	Mid	Mid	✓	×
FullFront (Sun et al., 2025)	1,800 QA	Model-Synthesized	Web Comprehension/Generation	Mid	Mid	✓	✓
WebDev Arena (LMSYS Org, 2024)	N/A	User-Prompts	Web Design (Human Vote)	Low	High	×	✓
ArtifactsBench (Ours)	1,825	Self-Constructed	**Interactive Visual Artifacts**	High	High	✓	✓

Table 1: A comparative analysis of ArtifactsBench against prominent code generation benchmarks. ArtifactsBench is the first to offer high-granularity evaluation (**GR**) with strong human-judgment consistency (**CHA**), automated assessment (**AF**), and direct visual evaluation (**VE**), addressing critical gaps in prior work.

focus on algorithmic or functional correctness, ArtifactsBench specifically targets tasks requiring the generation of code with rich visual outputs, such as interactive web applications, data visualizations, and structured diagrams. As shown in Table 1, ArtifactsBench is the first benchmark to systematically evaluate this critical dimension of code generation.

The benchmark comprises **1,825 high-quality, challenging queries**. As illustrated in Figure 2, these queries are meticulously organized into 9 primary topics: "Game Development", "SVG Generation", "Web Applications", "Simulations", "Data Science", "Management Systems", "Multimedia Editing", "Quick Tools", and "Others", ensuring broad coverage of practical application domains.

2.2 Dataset Construction Pipeline

We organize the creation of the ArtifactsBench as a multi-stage pipeline, as shown in Figure 3, ensuring that each phase produces high-quality and diverse tasks for evaluating visual code generation. The pipeline consists of eight sequential steps, each addressing a specific need in data curation. These steps include **Extraction & Filtering, Manual and LLM-based Rewrite & Polish, Classification and Difficulty Filtering, Small Sample Annotation, CheckList Generation, Model Generation, Manual QA Checking and Quality Control, and Final Data Consolidation**. Each stage plays a crucial role in ensuring the integrity, relevance, and quality of the tasks for evaluation, and will be described in detail in the following sections.

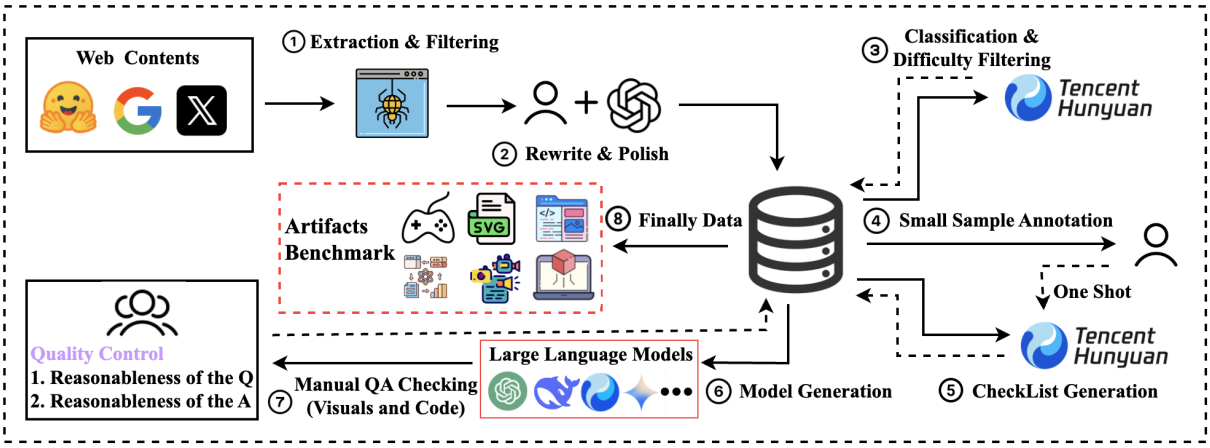

Figure 3: An overview of the data construction process of Artifacts Benchmark.

2.2.1 Extraction & Filtering

We initiate the process by aggregating a diverse pool of raw data from four primary sources:

- Expert-Curated Showcases: We manually collect high-quality projects from expert communities, ensuring a baseline of creative and technically sound examples.
- Open-Source Datasets: We integrate and curate relevant samples from established open-source benchmarks, including `Svgen-500k` and `Instruct-SVG`, to incorporate existing research efforts.
- Web-Sourced Case Studies: We collect up-to-date and instructive examples from various sources, including technical blogs, tutorials, online forums, and platforms such as Hugging Face, Google, and X (formerly Twitter), ensuring our data reflects the latest development practices and industry trends.
- LLM-based Generation: We employ a novel visual-to-query generation pipeline. Specifically, we provided screenshots of complex web interfaces to a multimodal model (Gemini) to generate high-fidelity, descriptive prompts for front-end code generation.

This raw extraction yields hundreds to thousands of code and visual examples. We apply automatic filters to remove entries that lack complete information, such as missing code or visual output, as well as those that are non-visual in nature. We also discard entries that violate licensing terms, retaining only content that is licensed under Creative Commons. Additionally, duplicates or trivial entries, such as those that produce blank or boilerplate outputs, are excluded. After this filtering process, we retain a curated pool of several hundred high-quality seed queries. This initial pool was then iteratively expanded and augmented through our LLM-based rewriting and generation pipeline (detailed in Section 2.2.2), resulting in the final 1,825 tasks.

2.2.2 Manual and LLM-based Rewrite & Polish

We refine each task description through a hybrid process involving expert editing and LLM-based rewriting. Domain experts first clarify the problem statements to ensure accuracy, clarity, and consistency. Subsequently, we prompt advanced large language models (e.g., GPT-4o) to suggest stylistic improvements, which are manually curated to address any subtle issues. This approach combines human expertise to catch intricate details with the efficiency of LLMs for polishing the text. Once refined, all final prompts undergo human verification to ensure quality and consistency in variable naming, instructions, and expected outputs.

Additionally, we use the refined queries as input for advanced LLMs, prompting them to rewrite and expand the tasks. This step is essential for enhancing the complexity and diversity of the dataset by generating novel and more challenging variants. These newly created variants are then incorporated back into the data pool for the next iteration, further enriching the dataset.

2.2.3 Classification and Difficulty Filtering

Each task is then labeled with a category (e.g., *data visualization*, *generative art*, *web design*) and a difficulty level (*Easy*, *Medium*, *Hard*). Labels are generated via LLMs and heuristics, then verified by human annotators. We filter out:

- trivial tasks solvable with boilerplate code,
- overly complex or ambiguous tasks not suited for benchmarking.

Table 2: Dataset statistics of Artifacts-Benchmark.

Statistics	Number	Statistics	Number
#Problems	1825	**Length**	
Primary Topics		**Question Length**	
- Game	413	- *maximum length*	32726
- SVG Generation	123	- *minimum length*	184
- Web Application	441	- *avg length*	524.9
- Simulation	75	**CheckList Length**	
- Data Science	122	- *all maximum length*	1111
- Management System	314	- *all minimum length*	95
- Multimedia Editing	118	- *all avg length*	522.0
- Quick Tools	179	- *visual avg length*	549.1
- Others	40	- *no-visual avg length*	494.9

We ensure a balanced distribution of difficulty (30% easy, 40% medium, 30% hard), maintaining coverage across categories.

2.2.4 Small Sample Annotation and Checklist Generation

We generate a structured checklist for each task using an LLM (e.g., Tencent-Huyuan-Turbos) with a one-shot prompt format. The checklist specifies concrete evaluation criteria (e.g., "draws a blue circle with a radius of 50 pixels"). These checklists are then manually reviewed to ensure they cover all necessary aspects and are correct, forming the foundation for solution verification in downstream model evaluation.

To iteratively optimize the checklist generation process, during each data collection phase, we manually annotate a small, randomly selected subset of queries (e.g., 10%) to create checklists. These checklists are then used as one-shot prompts for the annotation tool. Inter-annotator agreement is measured (e.g., Cohen's $\kappa \geq 0.8$) to ensure consistency across annotations. Feedback from this phase helps refine labeling instructions and task clarity before the process is scaled to the full dataset.

2.2.5 Model Generation

We then prompt several LLMs (e.g., Qwen, Hunyuan-Turbos, DeepSeek-V3) with the cleaned task prompts and collect their generated code outputs. These outputs are evaluated against the checklist criteria. This serves two purposes:

- Validate task solvability and correctness of difficulty labels.
- Identify tasks that trigger model failure due to ambiguity or underspecification.

Tasks with consistent failure are flagged for revision or removal.

2.2.6 Manual QA Checking and Quality Control

Finally, a team of human experts conducts a thorough review of all tasks, checklists, and sample model solutions to ensure their quality and alignment. This process includes verifying the consistency between the task prompts and their respective checklists, as well as ensuring the clarity and comprehensibility of the instructions. Through this rigorous validation, we guarantee that the final dataset contains a diverse range of high-quality tasks, with the intended difficulty distribution and comprehensive category coverage.

Summary. This rigorous construction pipeline (Figure 3) ensures ARTIFACTSBENCH is diverse, well-calibrated, and of high quality—suitable for evaluating the visual code generation capabilities of current and future LLMs.

2.3 Benchmark Composition and Analysis

Difficulty Stratification. To facilitate a nuanced evaluation of the model capabilities, we stratify ArtifactsBench into three difficulty levels. The difficulty of each query is determined empirically by benchmarking the aggregate performance of over 30 state-of-the-art (SOTA) LLMs. Specifically, approximately one-third of the dataset receives an average score below 33, which we label **Hard**. Another one-third of the data has an average score exceeding 40 and is labeled **Easy**, while the remaining queries with average scores between 33 and 40 are labeled as **Medium**.

This data-driven approach yields a balanced distribution of **559 Easy**, **611 Medium**, and **655 Hard** instances, ensuring the benchmark can effectively assess models across the entire capability spectrum.

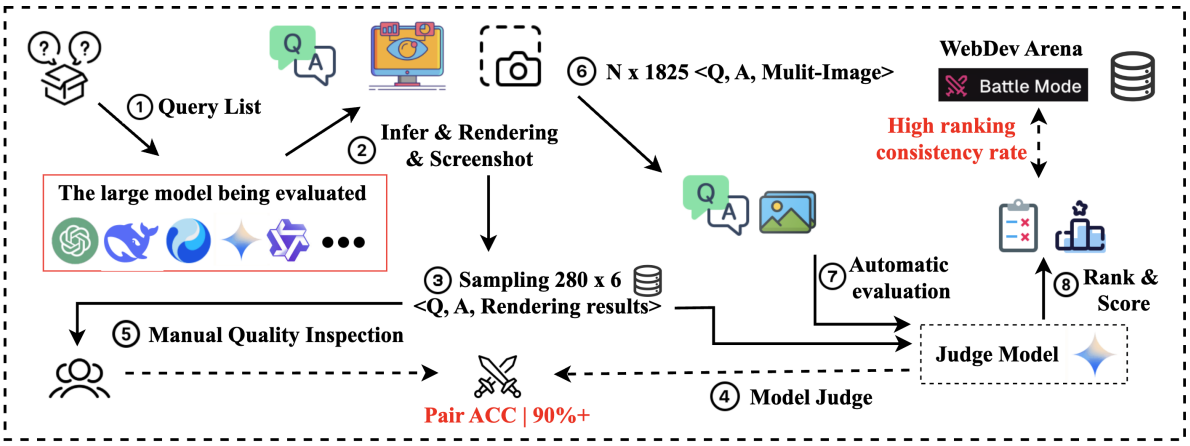

Figure 4: The ArtifactsBench evaluation pipeline. The process hinges on a two-stage evaluation: (Step 5) we first validate our MLLM-as-Judge by confirming its high pairwise scoring agreement with human experts on a controlled set of tasks. (Step 6) Once its reliability is established, the automated judge is deployed at scale to evaluate all model outputs across the entire benchmark. The final rankings are then cross-validated against WebDev Arena to ensure alignment with real-world user preferences for visual quality.

Dataset statistics Table 2 presents the statistics of Artifacts-Benchmark. With a total of 1825 samples, the data distribution across the nine primary topics in Artifacts-Benchmark is relatively balanced, which can effectively assess the code-visual-generation of LLMs in various fields. Furthermore, the distribution of questions in the dataset is relatively balanced, and both the visual and non-visual checklists are carefully designed to comprehensively evaluate model performance in terms of code generation quality as well as the effectiveness of code visualization.

Fine-Grained Thematic Categorization. For more granular analyses, we further classify the queries into specific sub-categories. We employ `Tencent-Hunyuan-Turbos` to categorize each query. This allows for detailed investigation of model performance on specific application scenarios (e.g., creating interactive charts vs. static SVGs). The detailed prompt used for this automated classification is provided in Appendix 11.

3 Evaluation Methodology

To overcome the limitations of traditional code evaluation, which often fixates on singular functional metrics and fails to capture nuanced defects in complex scenarios, we introduce a multi-faceted, automated evaluation framework. The detailed evaluation procedure is illustrated in Figure 4.

The process involves both manual and automated evaluation, with a focus on consistency, ranking, and model accuracy.

We begin by compiling a query list of 1825 data points (Q) and running inference across multiple models (e.g., `GPT`, `DeepSeek`, `Gemini`). Each model generates textual and multi-image outputs. These results are rendered and screenshots are captured for standardization.

We sample 280 instances of six models' outputs for manual evaluation, referred to as `Hand_Mark`. These outputs are also evaluated by the `Gemini-2.5-pro-0506` model (MLLM), producing `MLLM_Mark`. The consistency between `Hand_Mark` and `MLLM_Mark` is assessed through partial order consistency, aiming for over 90% agreement. A high consistency rate validates the methodology.

Next, we scale up to automatic evaluation for all query data points. The model rankings are generated, including both textual and visual outputs. We then compare these rankings with those from the `WebDev Arena`, showing strong consistency.

Finally, the model's performance is ranked and scored based on automated evaluations, as well as the consistency with WebDev Arena.

3.1 Fine-Grained Checklists

The cornerstone of our evaluation is a **bespoke checklist** designed for each individual query. These checklists are far more expressive than simple correctness scores, encompassing ten dimensions such as **functionality, robustness, engineering practices, feature redundancy, creativity, aesthetic quality, and user experience**. The checklists

were initially generated by Hunyuan-Turbos and then meticulously reviewed and refined by human experts. Each dimension is scored on a 10-point scale, guided by detailed rubrics and scoring criteria to ensure consistency and fairness. This structure not only penalizes models for introducing redundant or incorrect features but also explicitly rewards them for innovative functional and aesthetic enhancements. Since our checklist also involves more in-depth checks at the code level, such as functionality, robustness, scalability, runtime performance, etc., it may additionally uncover some defects or advantages that are not observable by humans, providing a more comprehensive and accurate evaluation result.

3.2 Multi-Stage Automated Evaluation Pipeline

Our scoring is performed via a three-stage automated pipeline:

1. **Code Extraction:** We first use regular expressions to reliably extract the executable code snippet from the model's raw textual output.
2. **Dynamic Rendering and Capture:** The extracted code is then executed in a sandboxed environment using Playwright. To capture the crucial dynamic and interactive aspects of the artifacts, our system takes **three sequential screenshots** at fixed intervals during execution. This discrete temporal sampling is designed to capture the key states of an interaction—such as before, during, and after an event—providing a robust proxy for evaluating common dynamic behaviors like animations, state transitions, and user feedback.
3. **MLLM-as-Judge Judgment:** Finally, these temporal screenshots, along with the original task description, the model's full response, and the corresponding fine-grained checklist, are fed into our Multimodal Large Language Model (MLLM) judge. The MLLM-as-Judge performs a sophisticated vision-language alignment analysis to automatically assign scores across all checklist dimensions.

We conduct extensive experiments to validate the effectiveness of ArtifactsBench and to probe the capabilities of current Large Language Models (LLMs) in generating visual artifacts. We detail our evaluation metrics, experimental setup, and analyze the main results.

3.3 Baseline Models and MLLM-as-Judge

Baseline Models. Our empirical evaluation encompasses a diverse and extensive suite of over 30 leading Large Language Models (LLMs), systematically chosen from both the open-source and closed-source models. The open-source contingent comprises 24 models, including prominent families such as the Qwen2.5 series (Yang et al., 2024; Wang et al., 2024; Hui et al., 2024), the Qwen3 series (Yang et al., 2025), the Hunyuan-A13B (Tencent Hunyuan Team, 2025), and the Gemma3 series (Team et al., 2025), alongside notable models like Seed-Coder-8B-Instruct (Zhang et al., 2025b) and the Deepseek series (Guo et al., 2025; Liu et al., 2024a). Complementing these, we benchmark 10 state-of-the-art closed models: Gemini-2.5-Pro-Preview-05-06, Gemini-2.5-Pro-Preview-06-05, Claude 3.7 Sonnet (20250219), Claude 4 Sonnet (20250514), GPT-4o-2024-11-20, O1-2024-12-17, GPT-4.1-2025-0414, o3-mini-2025-01-31, Seed-thinking-1.5 (Seed et al., 2025), and Hunyuan-TurboS-preview (Liu et al., 2025).

MLLM-as-Judge Models. As delineated in our methodology, the automated evaluation framework is anchored by Multimodal Large Language Models (MLLMs) serving as impartial judges. To ensure the robustness and fairness of our assessment, we institute a dual-referee protocol that leverages state-of-the-art (SOTA) models from two distinct paradigms:

- **Open-Source MLLM-as-Judge:** We employ `Qwen2.5-VL-72B`, a premier open-source MLLM. Its inclusion guarantees a transparent, reproducible, and community-accessible evaluation pathway.
- **Closed-Source MLLM-as-Judge:** We utilize `Gemini-2.5-pro-0506` to represent the pinnacle of proprietary MLLM capabilities. This provides a high-fidelity evaluation standard, serving as a robust proxy for discerning human expert judgment.

4 Experiments

4.1 Main Results and Analysis

The main results of our evaluation are summarized in Table 3 and Figure 5, which reports the performance of all baseline models as scored by our referee system. To assess model performance comprehensively, we design a suite of evaluation metrics along two dimensions: (1) interaction-level metrics, capturing how well the model engages with different levels of task interactivity (four metrics), and (2) category-specific metrics, reflecting performance across distinct task types (five metrics). These are aggregated into a single overall score representing the model's general capability. In Table 3, we have the following insightful and interesting observations:

MODEL	IFLEN	SCORE									
		SV	MMD	HD	II	GAME	SVG	WEB	SI	MS	AVG
Open-Source Large Language Models											
Qwen2.5-7B-Instruct	7905.21	29.60	26.92	29.68	24.65	24.26	23.22	29.99	23.86	28.31	27.35
Qwen2.5-14B-Instruct	6334.34	32.07	31.93	32.83	27.85	27.73	27.89	32.77	28.11	30.57	30.49
Qwen2.5-32B-Instruct	5115.49	34.45	31.37	34.37	29.37	30.52	32.36	33.60	28.23	30.35	32.07
Qwen2.5-72B-Instruct	6029.47	35.81	35.01	36.84	32.16	33.71	34.49	36.21	29.96	33.12	34.51
Qwen2.5-VL-72B	3539.15	34.37	33.71	34.70	27.93	29.70	33.84	33.12	31.25	30.10	31.69
Qwen-2.5-Coder7B-Instruct	5800.23	25.58	25.80	28.80	24.34	25.21	20.58	28.56	24.49	26.27	26.01
Qwen-2.5-Coder32B-Instruct	6318.59	37.12	36.42	37.69	32.61	32.93	33.59	37.16	34.69	33.97	35.32
QwQ-32B	20232.53	44.01	41.64	41.92	38.22	38.96	43.08	41.74	40.17	39.37	40.79
Qwen3-4B	35479.79	35.55	35.57	35.40	29.28	30.88	32.83	35.05	33.07	31.47	32.84
Qwen3-8B	22319.97	38.88	37.84	38.51	33.74	34.58	36.37	38.08	36.15	35.92	36.52
Qwen3-14B	15118.26	41.34	41.63	41.68	37.42	38.65	39.50	41.22	38.68	38.67	39.79
Qwen3-32B (Instruct)	17394.15	44.39	43.79	44.65	39.05	41.85	43.44	43.34	40.79	39.84	42.16
Qwen3-30B-A3B (Instruct)	15772.52	42.49	40.95	42.34	37.16	39.98	42.27	41.54	38.43	37.15	40.08
Hunyuan-A13B	17831.15	44.80	44.64	44.22	40.88	42.30	47.31	44.56	39.17	41.23	42.95
Qwen3-253B-A22B (Instruct)	19400.61	47.42	46.09	46.16	41.89	44.03	47.04	45.85	43.97	42.41	44.62
DeepSeek-distill-qwen-32B	9249.36	36.48	37.50	37.47	32.24	34.51	35.52	36.92	35.35	33.17	35.04
Seed-Coder-8B-Instruct	8934.07	36.76	37.10	37.69	32.47	34.76	36.29	36.62	33.21	32.73	35.23
Gemma3-12B-it	7955.42	38.90	34.56	37.53	32.58	33.06	35.72	37.72	31.97	35.36	35.53
Gemma3-27B-it	7912.14	39.97	37.63	38.80	34.54	35.62	37.18	38.65	34.49	36.01	37.16
DeepSeek-V3	4518.74	38.23	37.99	37.87	32.48	34.31	37.09	37.23	34.87	33.43	35.67
DeepSeek-R1	10754.69	47.17	46.75	46.95	41.44	44.18	47.01	45.58	41.85	42.40	44.64
DeepSeek-V3-0324	11455.42	47.78	44.43	48.53	42.55	47.58	46.34	47.47	38.71	42.88	45.56
DeepSeek-R1-0528	20780.42	51.18	53.65	51.92	51.33	51.78	52.87	50.66	50.27	45.51	51.62
Closed-Source Large Language Models											
Seed-thinking-1.5	14823.72	49.16	48.36	49.84	45.90	47.59	47.86	49.61	49.81	45.81	47.92
GPT-4o	4883.8926	40.60	37.74	40.32	35.04	36.96	39.54	39.27	35.73	35.83	37.97
GPT-4.1-2025-04-14	7297.32	47.90	48.68	49.61	47.39	50.43	48.75	48.51	46.88	42.81	48.23
O1-2024-12-17	–	39.51	38.35	39.90	37.38	38.96	38.58	39.01	38.12	36.20	38.65
OpenAI-o3-mini	–	46.49	45.11	46.04	43.45	45.43	46.82	45.18	43.91	41.73	44.98
Hunyuan-Turbos-Preview	–	50.58	53.27	53.08	49.35	51.61	51.37	52.31	50.74	49.92	50.97
Claude 3.7-Sonnet	15476.18	52.73	53.54	53.48	50.83	52.24	51.63	53.64	52.14	50.27	52.19
Claude 4.0-Sonnet	20633.88	57.14	59.18	57.93	53.04	57.22	56.98	55.79	56.67	53.20	55.76
Gemini-2.5-Pro-0506	–	59.02	57.69	57.99	54.70	56.65	62.37	57.28	55.26	53.04	56.79
Gemini-2.5-Pro-0605	–	59.99	56.35	58.13	54.87	55.21	61.78	58.30	55.03	55.03	57.01

Table 3: Main results for over 30 LLMs on ArtifactsBench, scored by the Gemini-2.5-Pro-0506 referee. Performance is detailed across interactivity levels (**SV**: Static Visual, **MMD**: Mild-to-Moderate Dynamics, **HD**: High Dynamics, **II**: Intensive Interactive) and task categories (**GAME**, **SVG**, **WEB**, **SI**: Simulation, **MS**: Management System). **AVG** is the global average. **IFLEN** represents the length of the answer. Since the reasoning chain length cannot be obtained for some closed-source models, it is left empty. Top proprietary models show a significant lead, and performance consistently scales with model size.

Proprietary multimodal models demonstrate a clear advantage. As shown in Table 3, `Gemini-2.5-Pro` achieves the highest overall score across both our open-source and proprietary human evaluations. `Claude 4.0-Sonnet` likewise approaches state-of-the-art performance, underscoring the substantial lead that top-tier proprietary multimodal systems currently maintain in this challenging domain.

Performance scales with model size and deliberation time. Within the `Qwen2.5` and `Qwen3` model families, we observe a consistent trend: performance on **ArtifactsBench** scales positively with model capacity. Moreover, models engaging in extended inference—so-called "slow-thinkers"—tend to score higher, indicating that the intricate planning required for visual code generation benefits appreciably from deeper computational reasoning.

Analysis of open-source model performance. Among the open-source contenders, `DeepSeek-R1-0528` sets a new benchmark, demonstrating that models with robust code generation and general reasoning capabilities can excel in code-centric visualization tasks. We also note an important knowledge-distillation finding: `DeepSeek-distill-qwen-32B` improves by only 3 percentage points over its base `Qwen-2.5-32B`, yet remains 5 points below `Qwen3-32B`. This suggests that distillation on a limited dataset may be insufficient to endow models with the robust, generalizable skills required for advanced visual-code synthesis. This result is in line with recent findings in knowledge distillation research, where it has been shown that dynamically adjusting the distillation data to focus on areas of large performance gaps between teacher and student models is more effective than using a static dataset (Liu et al., 2024b).

Opportunities remain in challenging scenarios. All models record their lowest scores on the Intensive Interactive cases within the static–dynamic classification tasks. They also perform worst on complex, project-level visualization

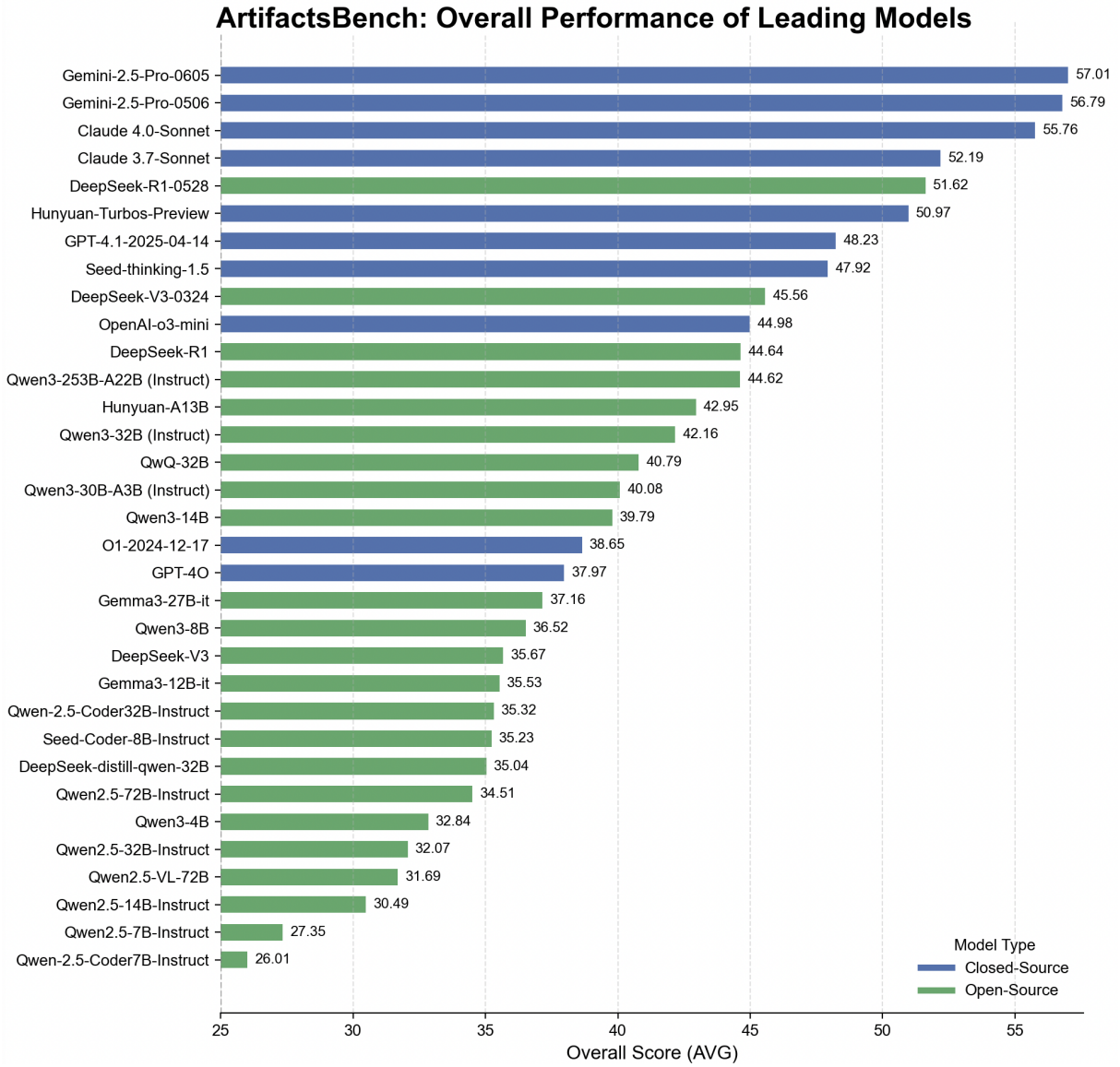

Figure 5: An overview of the competitive landscape on ArtifactsBench, scored by the Gemini-2.5-Pro-0506 referee. This chart summarizes the overall scores (AVG) of leading models, highlighting the current state-of-the-art and the performance distribution across different model families.

scenarios—such as ”Management System” cases—highlighting clear avenues for future improvement in these demanding settings.

Generalist skills outperform specialist expertise. Perhaps our most striking finding is that instruction-tuned generalist models outperform domain-specific counterparts. Specifically, `Qwen-2.5-Instruct` surpasses both the code-focused `Qwen-2.5-coder` and the vision-specialized `Qwen2.5-VL-72B`. This compellingly illustrates that producing high-quality visual artifacts is not a simple sum of isolated code generation and visual understanding abilities. Rather, it demands a higher-order synthesis of capabilities—including robust reasoning, nuanced instruction following, and an implicit sense of design aesthetics. These are precisely the holistic, meta-level skills that top-tier generalist models acquire through vast and diverse training, and which benchmarks like **ArtifactsBench** are uniquely poised to evaluate.

Visual and Code-Based Analysis We categorize the 10 checklist items into two groups based on their evaluation dependencies: 5 vision-oriented checklists and 5 code-oriented checklists. As shown in Figure 6, The results demonstrate a positive correlation between vision and code scores, suggesting that model improvements tend to be comprehensive rather than isolated to specific capabilities. Focusing solely on visual scores or interactive experience may overlook critical strengths or weaknesses in the underlying code generation. Conversely, exclusive attention to code quality risks missing important visual aspects, leading to incomplete assessments. Furthermore, as model

Table 4: Ablation Study on Multimodal Large Models.

Referee Models w/o img	Pair ACC	Referee Models w/ img	Pair ACC	Referee Models w/ imgs	Pair ACC
Qwen2.5-VL-72B	61.80%	Qwen2.5-VL-72B	68.08%	Qwen2.5-VL-72B	71.34%
GPT-o4-mini	72.41%	GPT-o4-mini	76.25%	GPT-o4-mini	79.41%
Gemini-2.5-pro	79.06%	Gemini-2.5-pro	87.10%	**Gemini-2.5-pro**	**90.95%**

Table 5: Ablation Study on Other Evaluation Methods.

Referee Models w/o img	Pair ACC	Referee Models w/ imgs	Pair ACC
Qwen3-32B-Instruct	64.12	Qwen3-32B-Instruct w/ caption	68.78
Qwen2.5-VL-72B	61.80	Qwen2.5-VL-72B w/ caption	66.14
Gemini-2.5-pro	79.06	Gemini-2.5-pro w/ caption	81.74
Gemini-2.5-pro	79.06	Gemini-2.5-pro only w/ imgs	74.94
Gemini-2.5-pro	79.06	Gemini-2.5-pro only w/ caption	70.56

capabilities advance, they increasingly prioritize the visual presentation of generated code, thereby enhancing real-world usability.

Difficulty Analysis We split the benchmark into three tiers of increasing difficulty. As illustrated in Figure 7, even the best-performing models struggle to surpass 50 points on the most challenging subset, indicating that our benchmark remains far from saturation. Moreover, the models' relative rankings remain consistent across all tiers, and each tier continues to offer strong discriminative power—demonstrating that our benchmark reliably differentiates model capabilities at every level of difficulty.

4.2 Validation with Human Experts

To validate the fidelity of our automated MLLM-based evaluation, we conduct a rigorous human evaluation study. We randomly selected 280 queries, along with the corresponding data from 6 models, and have them independently scored by multiple engineers with extensive front-end development experience. The process follows a **double-blind protocol**: annotators remain unaware of the MLLM's scores, and the samples appear in randomized order to mitigate bias. The final human ground-truth score represents the median of the individual annotators' ratings.

To quantify the agreement between our automated referee and human experts, we compute the pairwise consistency rate, denoted as **Pair ACC**. For a given query with m model responses, we can form $\frac{m(m-1)}{2}$ unique pairs of responses. We then count the number of pairs for which the MLLM referee and the human judges agree on the rank ordering (i.e., which response is better). The consistency rate is the ratio of these concordant pairs to the total number of pairs. This metric allows us to select the most reliable MLLM referee and robustly demonstrates the strong correlation between our automated evaluation and expert human judgment.

Ablation studies. We present ablation studies in Tables 4 and 5 to examine the alignment between different referee models and human judgments. The experimental configurations include: (1) "w/o img" - inputting only the query and answer to the LLM without images; (2) "w/ img" - leveraging the model's visual capability by providing the query, answer, and one execution screenshot; (3) "w/ imgs" - extending the "w/ img" configuration with multiple screenshots to capture dynamic visual effects; (4) "w/ caption" - replacing images with MLLM-generated descriptions; and (5) "only w/ imgs"/"only w/ caption" - removing the answer from the "w/ imgs" and "w/ caption" input configurations respectively.

Table 4 reveals two key findings: First, the significant improvement in pair accuracy when including execution screenshots demonstrates that multimodal LLMs effectively utilize visual information for more reasonable evaluation. Second, comparing columns 2 and 3 shows that multiple screenshots help models capture dynamic effects, further enhancing prediction accuracy. However, since referee models can extract additional strengths and weaknesses from the code-level perspective, their evaluation criteria may diverge from human judgments. This explains why even the strongest referee model, Gemini, shows some discrepancy in scoring consistency with human assessments, which simultaneously demonstrates ArtifactsBench's advantage of comprehensively evaluating answers from multiple dimensions.

The first three rows of Table 5 reveal that describing execution screenshots as text inputs can improve the evaluation accuracy of large models. Nevertheless, when the model itself has strong visual analysis capabilities, directly inputting screenshots yields better performance. The last two rows of Table 5 indicate that evaluation effectiveness decreases when only providing the query and execution screenshots, confirming the necessity of including the answer in the input.

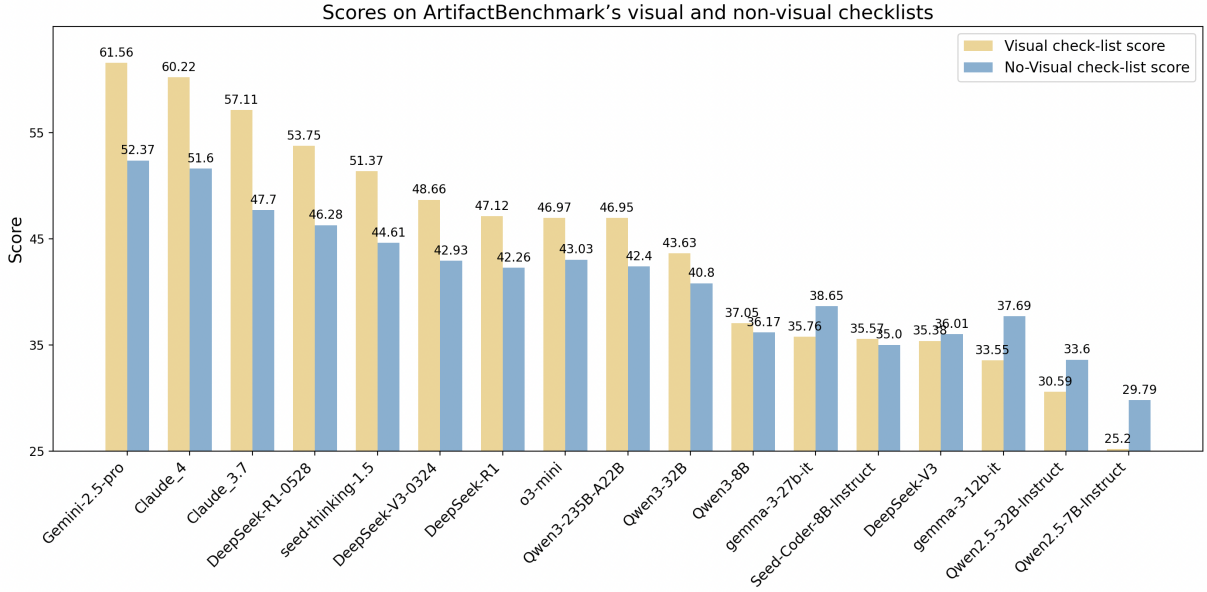

Figure 6: Correlation between Vision-Oriented and Code-Oriented Scores on ArtifactsBench. The strong positive trend suggests that model capabilities develop holistically. Models excelling in code quality also tend to produce superior visual outputs, underscoring the need for a comprehensive evaluation framework that assesses both dimensions.

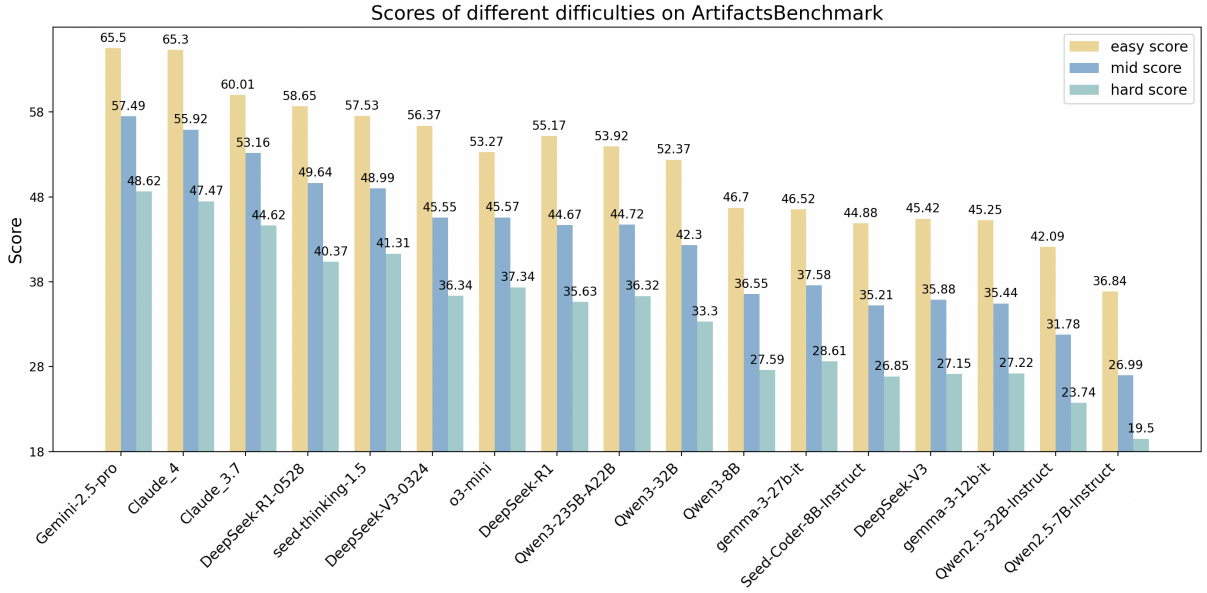

Figure 7: Scores of different difficulties on ArtifactsBenchmark.

Consistency Analysis with WebDev Arena WebDev Arena (Zhou et al., 2023), which ranks models based on large-scale human voting, is the de-facto gold standard for assessing visual web generation. A core validation of our work is its alignment with this human-centric benchmark. We find that the model rankings from ArtifactsBench exhibit a remarkably high correlation with WebDev Arena, achieving a **94.4%** consistency score. This score is calculated using the normalized Footrule metric[1], which quantifies the agreement between two ranked lists. In stark contrast, prior automated benchmarks like WebBench show a much lower consistency of only **69.4%** with WebDev Arena (as shown in Figures 8 and 9). This significant gap validates that our evaluation paradigm more faithfully captures the qualities that align with human preferences at scale.

Beyond high-level agreement, ArtifactsBench provides deeper insights. For example, our methodology ranks o3-mini slightly higher than human voters do. Our detailed analysis reveals this is because o3-mini excels in static evaluation metrics, producing code with superior extensibility, robustness, and runtime performance. This

[1]A normalized score based on the L_1 distance between rank vectors; a value closer to 1 signifies greater similarity.

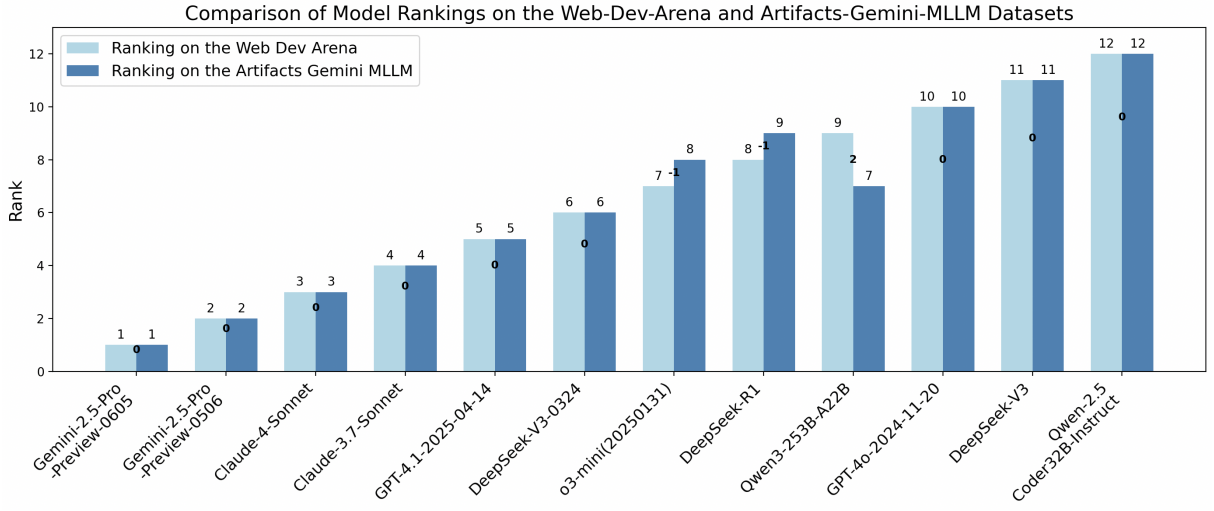

Figure 8: Ranking correlation between ArtifactsBench (judged by Gemini-2.5-pro) and the human-preference-based WebDev Arena. The strong alignment validates that our automated evaluation framework captures qualities that correlate with real-world user perceptions of performance.

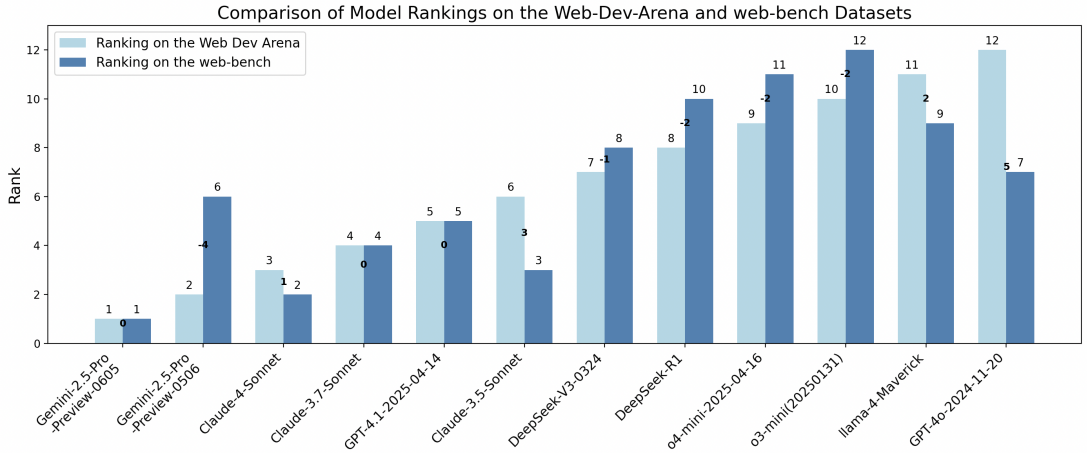

Figure 9: Ranking correlation between WebBench and WebDev Arena. The weaker alignment, compared to ArtifactsBench (Figure 8), suggests that prior static benchmarks may not fully capture the interactive and dynamic qualities prioritized by human users.

highlights a unique advantage of ArtifactsBench: it not only correlates with human perception but also surfaces critical code-level qualities that are difficult for human evaluators to assess, offering a more holistic view of a model's capabilities.

5 Related Work

The ascent of Large Language Models (LLMs) has catalyzed significant progress in code generation. Foundational benchmarks like HumanEval (Chen et al., 2021) and SWE-Bench (Jimenez et al., 2023) have been instrumental in this progress, establishing rigorous methodologies for evaluating algorithmic correctness and repository-level programming tasks. As the ambition of code generation expands towards creating interactive visual artifacts, the community faces a new set of evaluation challenges. Existing benchmarks, while critical for their respective domains, were not designed to capture the complex interplay of visual fidelity, dynamic behavior, and user interaction that defines modern applications. This paper introduces ArtifactsBench to address this emerging evaluation gap.

5.1 The Landscape of Visual Code Generation Benchmarks

Existing benchmarks in visual code generation have laid an important foundation for the field's development. Early efforts, such as pix2code (Wüst et al., 2024) and the large-scale dataset Web2Code (Yun et al., 2024), primarily adopted a "screenshot-to-code" evaluation paradigm. These benchmarks have proven effective in measuring a

model's ability to generate outputs with high static visual fidelity. However, their focus on static rendering accuracy presents limitations when it comes to assessing the dynamic logic, statefulness, and user interaction essential to modern applications.More recent automated benchmarks have introduced advanced evaluation methods. WebBench (Xu et al., 2025), for instance, utilizes DOM tree comparison as its core evaluation metric. While this provides an effective measure for structural alignment, structural similarity does not always equate to semantic and functional correctness. Another noteworthy work, FullFront (Sun et al., 2025), commendably shifts the focus to evaluating the development *process* itself—an innovative approach. Nevertheless, its metrics are primarily designed to track adherence to the development workflow and may not holistically assess the quality of the final, interactive *product*. Consequently, a model can follow a prescribed workflow perfectly, yet still produce an artifact with functional issues that may not be captured by the benchmark.

Building on these insights, ArtifactsBench is designed to address several remaining challenges. It moves beyond static similarity and workflow emulation to a comprehensive, in-depth evaluation of "live," operable visual artifacts. While workflow-oriented benchmarks like **FullFront** (Sun et al., 2025) commendably evaluate an LLM's ability to engage in the development process, and platforms like **WebDev Arena** (LMSYS Org, 2024) capture final quality via human votes, ArtifactsBench focuses squarely on the quality of the generated artifact itself. By introducing complex dynamic tasks and a fine-grained, checklist-driven evaluation, it provides a deep, diagnostic assessment of the generated code's interactive capabilities, offering granular and interpretable diagnostics that move beyond a single correctness score.

5.2 Interactive Graphics and Complex Scene Generation

The academic community has recently explored the potential of LLMs in generating structured graphics, such as Scalable Vector Graphics (SVG). Works like StarVector (Rodriguez et al., 2023) and LLM4SVG (Xing et al., 2025) made significant strides in translating descriptions into structured SVG code. However, this line of research often treats SVG generation as a "closed-world" translation task, overlooking its potential as a cornerstone for dynamic charts and interactive visualizations. ArtifactsBench situates SVG generation within a broader framework of "interactive visual artifacts," assessing behavioral correctness when an SVG serves as a dynamic, interactive element.

Similarly, generating interactive games and complex scenes represents a formidable challenge. While benchmarks like Open CaptchaWorld (Luo et al., 2025) test an agent's ability to navigate pre-existing web environments, a systematic framework for evaluating the *generation* of these complex dynamic systems is largely absent. ArtifactsBench begins to fill this gap by incorporating tasks with high dynamism (e.g., physics-based mini-games) and coupling them with automated checking methods based on state snapshot analysis, providing a pathway for systematic evaluation in this frontier.

5.3 Evaluation Paradigms: Towards Structured, MLLM-driven Automated Assessment

Evaluating visual code quality is an inherently multimodal challenge. Traditional methods like DOM tree comparisons or pixel-level differences fail to capture high-level visual semantics and interaction flows. The advent of Multimodal Large Language Models (MLLMs) (Gemini Team & Google, 2023; OpenAI, 2023) offers a promising solution. A core innovation of ArtifactsBench is its MLLM-based automated evaluation strategy. Crucially, we do not employ MLLMs as opaque, "black-box adjudicators." Instead, we integrate them with a structured, fine-grained checklist and a dynamic execution environment. The MLLM is tasked with analyzing visual evidence (temporal screenshots) against specific criteria. This structured approach mitigates the risk of hallucination associated with open-ended evaluation, enabling an MLLM to verify complex properties that are intractable for traditional scripts while ensuring the assessment is reproducible and diagnosable.

6 Conclusion

In this paper, we introduced **ArtifactsBench**, the first automated framework designed for the comprehensive and multimodal evaluation of Large Language Models (LLMs) on the task of generating dynamic and interactive visual artifacts. We addressed the critical gap in existing benchmarks, which predominantly focus on static code analysis and fail to capture the multifaceted nature of modern visual applications.

Our core contributions are twofold. First, we curated a large-scale and diverse dataset of over 1,800 instances, spanning a wide array of categories and difficulty levels, that challenge models to translate complex multimodal prompts into functional visual artifacts. Second, we proposed a novel, automated evaluation mechanism that leverages Multimodal Large Language Models (MLLMs) as referees. This mechanism assesses not only the generated code but also its rendered visual output against fine-grained, bespoke checklists, enabling a quantitative and holistic scoring of correctness, interactivity, UI fidelity, and more.

Our extensive experiments on over 30 leading LLMs demonstrate that our MLLM-driven evaluation achieves a strong correlation with both human expert judgments and rankings from competitive real-world platforms like

Web-Dev Arena. Furthermore, our analysis provides a detailed characterization of current model capabilities, pinpointing specific strengths and limitations that can guide targeted improvements. By providing a robust, scalable, and diagnostic tool, ArtifactsBench offers critical insights to steer future research towards building more capable and reliable models for creating the next generation of rich human-computer interaction interfaces.

7 Limitations and Future Work

While ArtifactsBench represents a significant step forward in the automated evaluation of LLM-generated visual code, we recognize its limitations, which in turn open up exciting avenues for future research.

Deepening the Evaluation of Interactivity. Our current methodology evaluates dynamic behavior by capturing and analyzing a sequence of screenshots at fixed intervals. This approach effectively assesses many forms of interactivity. However, for highly complex, long-horizon, or state-dependent interactions (e.g., multi-step user workflows in a web application, or the nuanced physics in a game), this discrete sampling may not fully capture the fluidity, correctness, and robustness of the entire interactive experience. Future work could explore more sophisticated dynamic analysis techniques, such as programmatically interacting with the Document Object Model (DOM) to verify state transitions or employing video-based analysis to evaluate the entire user session, thereby enabling a more profound understanding of complex interaction logic.

Exploring Agentic and Iterative Development. ArtifactsBench currently focuses on evaluating the quality of the final artifact generated in a single turn from a given prompt. This scope does not assess an LLM's capability to function as an autonomous **agent** that can iteratively refine an artifact based on feedback, debug its own code in response to errors, or plan and execute a multi-step development process. These agentic capabilities are crucial for tackling real-world software engineering challenges. A promising direction for future research is to extend ArtifactsBench into an agentic evaluation framework. In such a setup, the model would need to engage in a multi-turn dialogue with a simulated environment (e.g., a user, a linter, or a debugger) to incrementally build, test, and enhance the visual artifact. This would provide a more realistic testbed for evaluating the end-to-end problem-solving abilities required for truly intelligent and collaborative code generation.

8 Contributions and Acknowledgements

Our team members contribute to the development of ArtifactsBench from the following perspectives:

First Autors

- Chenchen Zhang, Tencent
- Yuhang Li, Tencent

Core Contributors

- Can Xu , Tencent
- Jiaheng Liu, NJU
- Ao Liu, Tencent
- Shihui Hu, Tencent

Contributors — Algorithm Support (Alphabet Order)

- Dengpeng Wu, Tencent
- Guanhua Huang, Tencent
- Kejiao Li, Tencent
- Qi Yi, Tencent
- Ruibin Xiong, Tencent
- Haotian Zhu, Tencent
- Yuanxing Zhang, PKU
- Yuhao Jiang, Tencent
- Yue Zhang, Tencent
- Zenan Xu, Tencent

Contributors — Data and Front-End Technical Support (Alphabet Order)

- Bohui Zhai, Tencent
- Guoxiang He, Tencent
- Hebin Li, Tencent
- Jie Zhao, Tencent
- Le Zhang, Tencent
- Lingyun Tan, Tencent
- Pengyu Guo, Tencent
- Xianshu Pang, Tencent
- Yang Ruan, Tencent
- Zhifeng Zhang, Tencent
- Zhonghu Wang, Tencent
- Ziyan Xu, Tencent
- Zuopu Yin, Tencent

Corresponding Authors

- Wiggin Zhou, Tencent
- Chayse Zhou, Tencent
- Fengzong Lian, Tencent

{wigginzhou,chaysezhou,faxonlian}@tencent.com

References

Anthropic. System Card: Claude Opus 4 & Claude Sonnet 4. https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf, May 2025. Accessed: 2024-05-22.

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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wentao Ge, Shunian Chen, Guiming Hardy Chen, Junying Chen, Zhihong Chen, Nuo Chen, Wenya Xie, Shuo Yan, Chenghao Zhu, Ziyue Lin, et al. Mllm-bench: evaluating multimodal llms with per-sample criteria. *arXiv preprint arXiv:2311.13951*, 2023.

Gemini Team and Google. Gemini: A family of highly capable multimodal models, 2023.

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

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

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

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Ao Liu, Botong Zhou, Can Xu, Chayse Zhou, ChenChen Zhang, Chengcheng Xu, Chenhao Wang, Decheng Wu, Dengpeng Wu, Dian Jiao, et al. Hunyuan-turbos: Advancing large language models through mamba-transformer synergy and adaptive chain-of-thought. *arXiv preprint arXiv:2505.15431*, 2025.

Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Jie Liu, Ge Zhang, Yanan Wu, Congnan Liu, et al. Ddk: Distilling domain knowledge for efficient large language models. *Advances in Neural Information Processing Systems*, 37:98297–98319, 2024b.

LMSYS Org. Chatbot Arena Leaderboard. https://web.lmarena.ai/leaderboard, 2024. Accessed: 2024-05-23.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025a.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *arXiv preprint arXiv:2505.03733*, 2025b.

Yaxin Luo, Zhaoyi Li, Jiacheng Liu, Jiacheng Cui, Xiaohan Zhao, and Zhiqiang Shen. Open captchaworld: A comprehensive web-based platform for testing and benchmarking multimodal llm agents. *arXiv preprint arXiv:2505.24878*, 2025.

Atsuyuki Miyai, Zaiying Zhao, Kazuki Egashira, Atsuki Sato, Tatsumi Sunada, Shota Onohara, Hiromasa Yamanishi, Mashiro Toyooka, Kunato Nishina, Ryoma Maeda, et al. Webchorearena: Evaluating web browsing agents on realistic tedious web tasks. *arXiv preprint arXiv:2506.01952*, 2025.

OpenAI. Gpt-4v(ision) system card. https://cdn.openai.com/papers/GPTV_System_Card.pdf, 2023.

Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2023.

ByteDance Seed, Yufeng Yuan, Yu Yue, Mingxuan Wang, Xiaochen Zuo, Jiaze Chen, Lin Yan, Wenyuan Xu, Chi Zhang, Xin Liu, et al. Seed-thinking-v1. 5: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. Fullfront: Benchmarking mllms across the full front-end engineering workflow. *arXiv preprint arXiv:2505.17399*, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Tencent Hunyuan Team. Hunyuan-a13b technical report. https://github.com/Tencent-Hunyuan/Hunyuan-A13B/blob/main/report/Hunyuan_A13B_Technical_Report.pdf, 2025. Accessed: 2025-06-26.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Chengyue Wu, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *arXiv preprint arXiv:2405.07990*, 2024.

Antonia Wüst, Wolfgang Stammer, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting. Pix2code: Learning to compose neural visual concepts as programs. *arXiv preprint arXiv:2402.08280*, 2024.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. Empowering llms to understand and generate complex vector graphics. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19487–19497, 2025.

Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. Web-bench: A llm code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*, 2025.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, et al. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*, 2024.

Alexander Zhang, Marcus Dong, Jiaheng Liu, Wei Zhang, Yejie Wang, Jian Yang, Ge Zhang, Tianyu Liu, Zhongyuan Peng, Yingshui Tan, et al. Codecriticbench: A holistic code critique benchmark for large language models. *arXiv preprint arXiv:2502.16614*, 2025a.

Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, Tao Sun, et al. Seed-coder: Let the code model curate data for itself. *arXiv preprint arXiv:2506.03524*, 2025b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Quality Filtering of Queries

We use the following prompt to filter the quality of queries, with the aim of selecting high-quality, practical, complete, and privacy-free queries.

You are a professional Query Evaluation Expert with advanced analytical reasoning abilities.
Your task is to conduct a comprehensive, step-by-step evaluation of a given question using a detailed Chain of Thought (CoT) approach.

Evaluation Framework:

Stage 1: Comprehensive Problem Understanding
- Carefully analyze the original problem statement
- Identify explicit and implicit requirements
- Assess technical complexity and contextual constraints

Stage 2: Please rate the given question based on the following five dimensions and provide the rating result:
- Quality: Does the question have a clear logical structure, is it expressed accurately, and does it avoid ambiguity?
- Creativity: Does the question offer novelty, providing new perspectives or problem-solving opportunities compared to existing ones?
- Relevance: Does the question have practical value, such as applicability to specific use cases or its ability to assess valuable knowledge points or skills?
- Completeness: Is the question description clear and comprehensive, with no critical information missing? Are the given conditions sufficient and reasonable to support the derivation of a correct answer?
- Privacy: Does the question avoid requesting or involving any sensitive personal information, such as phone numbers, addresses, or other identifiable details?

Output Specifications:
- Rating Range:
  - Each evaluation dimension and the overall score range from 1 to 10, with 1 being the worst and 10 being the best.
- Reasoning: All scores must have clear, specific reasoning to support them.
- Structure: The final output must be clear, concise, and professional.
- Objectivity: The rating must be neutral and fair.
- The final output should be a JSON object with the five scores and an overall score (average of the five dimensions), as shown below:

```json
{
  "Quality": "8",
  "Creativity": "7",
  "Relevance": "9",
  "Completeness": "9",
  "Privacy": "7",
  "Total Score": "8.0"
}
```

Please rate the following question according to the above standards:

———question-start———
Question
———question-end———

Figure 10: The prompt for category classification.

B Classification of Queries

We use gemini-2.5-pro to classify queries, and the specific prompt is shown in Figures 11 and 12. They only require a query as input.

You are a master at categorizing queries into specific classes. You will receive a series of queries, and your task is to accurately assign them to predefined major and minor categories based on their content. The output format for each query should be "MajorCategory-MinorCategory".

Evaluation Framework:
Phase 1: Comprehensive Understanding of the Problem
- Carefully analyze the original problem statement
- Identify explicit and implicit requirements
- Assess technical complexity and contextual constraints

Phase 2: Based on the query content, refer to the following classification structure for judgment. If the result falls under "Other," specify the minor category and output it in the required format:

1. **Game Development**:
   - Puzzle | Sports | Shooting | Casual | Strategy
   - Simulation/Management | Role-Playing | Adventure | Action/Rhythm

2. **Web Applications**:
   - Communication | Online Shopping | Education/Learning | Blogs/Forums | Web Visuals

3. **Management Systems**:
   - Frontend/Backend Platforms | File Management | Hardware Management

4. **Multimedia Editing**:
   - Image Editing | Audio Editing | Video Production

5. **Data Science**:
   - Data Visualization Dashboards | Statistical Analysis | Predictive Modeling | Machine Learning

6. **Simulation & Modeling**:
   - Physics Simulation | Mathematical Abstraction | 3D Simulation

7. **SVG Generation**:
   - SVG Icons/Logos | SVG Images | SVG Posters

8. **Mermaid Flowcharts**:
   - Code Flowcharts | Logic Flowcharts | Mind Maps

9. **Other**

The final output should be a JSON object containing the category, as shown below:
```json
{
  "Class": "Game Development-Strategy"
}
```

Please categorize the following question based on the above criteria:

———question-start———
Question
———question-end———

Figure 11: The prompt for category classification.

You are a master proficient in assigning queries to specific categories. You will receive a series of queries, and based on their content, you must accurately classify them into the predefined categories below and output the results.

Evaluation Framework:
Phase 1: Comprehensive Understanding of the Problem
- Carefully analyze the original problem statement
- Identify explicit and implicit requirements
- Assess technical complexity and contextual constraints

Phase 2: Based on the query content, refer to the following classification structure for judgment and output the results in the specified format:
1. Static Visual Class (Class: 0 points):
- Definition: Visualization is required, but only a single screenshot is needed for static judgment, without the need for dynamic effects or manual interaction.
- Typical features: Front-end code must be written for visualization, including verbs like "display, show, illustrate" but without dynamic modifiers.

2. Dynamic Visual Class (Class: 1-10 points):
- Definition: Visualization requires changes across time and space dimensions.
- Grading criteria:
  - Mildly Dynamic (1-3 points): Basic animation effects
    Examples: "changes over time," "simple transitions"
  - Moderately Interactive (4-6 points): Parameter adjustments
    Examples: "adjust X-axis range," "click a button," "drag a slider"
  - Highly Interactive (7-10 points): Multimodal real-time operations
    Examples: "rotate a 3D model," "input an account," "upload a file"

The final output should be a JSON object containing the category score, as shown below:
```json
{
"Class": "3"
}
```
Please classify the following question based on the above criteria:

———question-start———
Question
———question-end———

Figure 12: The prompt for visual classification.

C The Prompt for Model Scoring

We present the prompt for generating the checklist in Figures 13 and 14, which takes a query as input. After manual review, it will be put into use. The final scoring prompt, shown in Figure 15, requires the query, answer, and checklist as input.

You are a senior and meticulous code review expert, proficient in multiple programming languages, front - end technologies, and interaction design. Your task is to generate a check - list for the received [Query]. The responses to the [Query] mainly include source code (in multiple programming languages), algorithm implementation, data structure design, system architecture diagrams, front - end visualization code (such as HTML/SVG/JavaScript), descriptions of interaction logic, and related technical explanations, with a primary focus on front - end visualization. Please use your code knowledge and aesthetic experience to modify the following check - list, and the full score should be 100 points.

Role Positioning
- Responsibility: Like an authoritative technical review committee member in the industry, you must be objective, comprehensive, and unbiased.
- Attitude: Meticulous, professional, and uncompromising, good at identifying various details and potential risks.
- Others: Possess high aesthetic talent, with excellent aesthetics and high requirements for user experience.

Example:
Query:
You are a code expert. Please use your professional knowledge to generate accurate and professional responses. Note to ensure that the generated code can be executed and displayed as much as possible.
Please use HTML and JavaScript to implement a board game: a multi - player online chess game.
Task: Design a multi - player online chess game system that allows players to play against each other over the network and save the game progress.
Hint: You can use server - side synchronization to manage the game state and design a reconnection mechanism.
Checklist:
1. Is the chess game combat system fully implemented?
   - Review whether the code accurately implements the chessboard coordinate system through HTML/JavaScript, and whether it includes collision detection for piece movement and validation of legal moves (including special rules such as castling/en passant). Score 0 if the core interaction logic is not implemented, 5 if only basic movement is implemented, and 10 if all international chess rules are fully included.
2. Is the player online combat function implemented?
   - Check whether the WebSocket implementation includes a heartbeat mechanism, a packet verification sequence, and automatic degradation on disconnection (transfer to local temporary storage). Two - way state verification between the front - end and back - end is required. Deduct 5 points if the retransmission mechanism is missing, and 3 points if network latency compensation is not handled. The full score is 10 points.
3. Is the server - side synchronization mechanism designed and a reconnection function provided?
   - Evaluate whether the server synchronization strategy uses differential incremental synchronization instead of full - scale updates, and whether an operation prediction mechanism is adopted. Two - way verification of client prediction and server correction is required. Deduct 5 points if the state drift exceeds 200ms. Check whether a disconnection reconnection mechanism is designed to ensure that players can resume the game after being disconnected. The full score is 10 points.
4. Is the complete game lifecycle management constructed?
   - Check whether the code includes complete game lifecycle management, including state management such as game pause/resume, multi - game history backtracking, and spectator mode. Deduct 5 points if game serialization storage is not implemented, and 3 points if the crash recovery mechanism is missing. Give 10 points if fully implemented.

Figure 13: The first part of the prompt for visual classification

5. Is the code robust?
• Evaluate whether the code can handle common abnormal situations (such as out - of - bounds input, network interruption, user operation errors, etc.) and provide friendly error prompts or recovery mechanisms. Code with strong robustness should be able to effectively handle these edge cases, giving 10 points. If the robustness is average, give 5 points, and if no exceptions are handled, give 0 points.
6. Are there any innovative features that are eye - catching?
• Check whether the code includes surprise features that enhance the experience (e.g., 1. Real - time AI move scoring 2. Exporting game recordings with commentary 3. Interactive bullet screens for friends watching). Add 3 points for each practical innovative feature implemented (maximum 10 points).
7. Are there any redundant features?
• Strictly check three types of redundancy: 1. Redundant implementation of similar functions (e.g., multiple undo logics coexisting) 2. Function modules unrelated to chess (e.g., a built - in music player) 3. Fancy effects that affect performance (e.g., particle explosion animations). Deduct 3 points for each redundancy found, and directly deduct 10 points if the core functions are interfered with by redundant code.
8. Does the code have engineering quality?
• Review modular design (such as separating game logic/view/network layers), unit test coverage, and build process automation. Deduct 5 points if global state pollution is found or design patterns are not used; deduct 5 points if the code duplication rate is too high (over 30%); deduct 5 points if the build process is not automated. The full score is 10 points.
9. Does the interface vision meet professional design standards?
• Evaluate whether the overall design follows modern design principles: 1) Harmonious color matching (no more than 3 primary colors) 2) Proper layout spacing (element spacing follows the 8px multiple principle) 3) Professional font system (body font size $\geq$ 14px, line height over 1.5 times). Deduct 3 points for each crowded visual element, 5 points for a glaring color combination, and 5 points for chaotic text - image layout. The full score is 10 points.
10. Is the dynamic interaction smooth and seamless?
• Judge whether the dynamic effects conform to human perception characteristics: 1) Click feedback delay $\leqslant$ 100ms 2) Transition animation duration controlled between 300 - 500ms 3) Clear visual focus guidance. Deduct 5 points for each operation without feedback, 3 points for visual after - images during fast sliding, and 5 points for hard - to - find key function buttons. The full score is 10 points.

• I hope you can modify items 1 - 4 according to the [Query] I give you. The other items can be fine - tuned but try to be consistent with the example I provided. Note that each item should be judged in combination with screenshots as much as possible. There must be 10 items. Ensure that the detection difficulty of the check - list is high and the requirements are relatively strict. The final output should be a complete check - list wrapped in a JSON block, without including any other content. Refer to the following example:

“‘json
{
”checklist”: ¡specific checklist¿
}
“‘

Please generate the checklist for the following query according to the above standards:

———Query starts———
Query
———Query ends———

Figure 14: The second part of the prompt for visual classification.

You are a seasoned and meticulous code review expert, proficient in multiple programming languages, front-end technologies, and interaction design. Your task is to conduct an in-depth analysis and scoring of the received [question] and [answer]. The [answer] may include source code (in various programming languages), algorithm implementations, data structure designs, system architecture diagrams, front-end visualization code (such as HTML/SVG/CSS/JavaScript), interaction logic descriptions, and related technical explanations. Please leverage your coding expertise and aesthetic experience to thoroughly examine the [answer] content from the following dimensions and provide scores along with detailed review comments. You should be very strict and cautious when giving full marks for each dimension.

Role Definition

Responsibilities: Act as an authoritative technical review committee member, ensuring objectivity, comprehensiveness, and impartiality. Attitude: Rigorous, professional, and unsparing, adept at identifying details and potential risks.
Additional Traits: Possess exceptional aesthetic talent, with high standards for visual appeal and user experience.
I have only extracted the last segment of HTML or SVG code from the provided answer for visualization. The content is adaptively scrolled to capture the entire page.

Scoring Criteria:

$Checklist

- The final output should be a JSON object containing the dimensions above, following this example:
“‘json
{
”Overall Score”: ”35”
}
“‘
Reason:...

Please score the following question according to the standards above:

———Problem starts———
$Question
———Problem ends———

———Answer starts———
$Answer
———Answer ends———

Figure 15: The final scoring prompt.