GENCLAW: CODE-DRIVEN AGENTIC IMAGE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Image generation models have evolved from text-conditioned pixel synthesis toward multimodal agents endowed with visual comprehension and tool invocation capabilities. However, prompt-centered generation workflows often control visual synthesis through textual instructions to an underlying image model, leaving detailed spatial construction implicit. The programming capabilities of LLMs suggest a complementary interface: executable visual code that makes intermediate structural decisions explicit, editable, and available for inspection before final image synthesis. In this paper, we propose GENCLAW, a code-driven agentic image generation paradigm that empowers the agent to create like a human artist: first conceptualizing, then sketching, and finally coloring. Specifically, the agent first constructs the conceptual knowledge and context through search and reasoning. It then utilizes code (e.g., SVG, HTML, Three.js) to render executable visual sketches. Finally, it employs an image generation model to supplement textures, materials, and photorealism. In this workflow, code serves as a controllable intermediate canvas bridging linguistic reasoning and pixel synthesis. By exposing conceptualization, structure, and rendering as distinct stages, GENCLAW supports controllable and interpretable image generation.

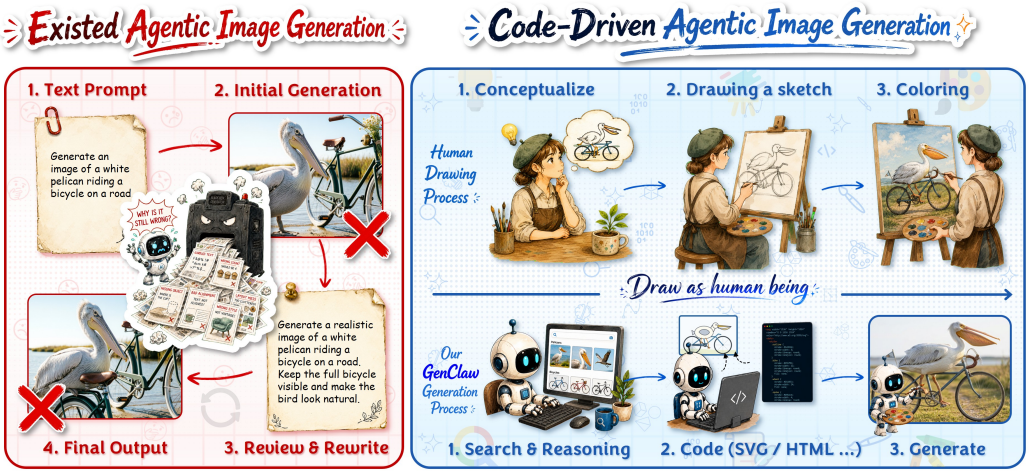


Figure 1: **Code-driven agentic image generation.** GenClaw decouples comprehension from generation through a Conceptualize → Sketch → Color workflow: search and reasoning establish context, code constructs the visual layout, and an image model supplies appearance.

1 INTRODUCTION

Image generation is evolving from text-conditioned synthesis (Rombach et al., 2022; Ramesh et al., 2022; Zhang et al., 2023) toward unified models that combine visual understanding and genera-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tion (Chen et al., 2025b; Deng et al., 2025b; Wu et al., 2025a). This progress provides a foundation for agents that integrate image synthesis with reasoning and tool use.

Recent work moves image synthesis toward an agentic paradigm through planning, tool invocation, and feedback. Commercial systems reflect this shift: Nano-Banana Pro (Raisinghani, 2025) and Seedream 5.0 Lite (ByteDance Seed, 2026a) support search-grounded generation, while Seedream 5.0 Pro (ByteDance Seed, 2026b) emphasizes reasoning, layout planning, and interactive editing. Think-Then-Generate (Kou et al., 2026) and GenAgent (Jiang et al., 2026) decouple high-level understanding from concrete generation. Mind-Brush (He et al., 2026) introduces search and reasoning tools to address real-time knowledge gaps and complex intent. JarvisEvo (Lin et al., 2025b) and RefineEdit-Agent (Liang et al., 2025) develop closed-loop editing through multimodal reasoning and editor–evaluator coordination. Along a complementary direction, CoCo (Li et al., 2026) generates structured sketches through executable code before image refinement. These advances motivate a broader question: how can an agent participate directly in constructing visual structure, in addition to acquiring context and reviewing outputs?

For agents that communicate with an image model primarily through natural-language prompts, structural control remains indirect. As illustrated in Figure 1, repeated prompt rewriting can improve a request, but leaves the realization of exact counts, coordinates, typographic layouts, and occlusion relations to the image model. Human artists offer a useful analogy: they develop an idea, organize the composition, sketch its structure, and then add color and detail. We seek a similarly staged interface in which the agent can inspect and revise a visual plan before synthesis. Executable visual code provides a "digital brush" suited to the programming and debugging capabilities of code agents.

Visual code generation and layered representations offer structured, editable representations for this purpose (Yang et al., 2026; Wang et al., 2025a; Lin et al., 2025a; Yin et al., 2025). SVG and HTML express objects, coordinates, text, and rendering order explicitly; executing the code makes the planned structure visible and available for revision. Frontier LLMs can already draw with code and render front ends, constructing an image skeleton much like a painter sketching line art. However, compact code sketches emphasize logic and structure, while photorealistic lighting, hair, and natural textures are difficult to specify with simple primitives. Image generation models complement this representation by supplying appearance and detail. Our aim is therefore to use code as an intermediate canvas within an agent workflow, with task-dependent tools for composition, typography, physical references, and layered editing.

Based on this complementarity, we propose code-driven agentic image generation and instantiate it in GenClaw (Figure 1, right). The agent first gathers entity knowledge and context through search and reasoning (*Conceptualize*); it then writes code as its "digital brush" to express visual intent on a canvas (*Sketch*), planning object positions, sizes, text layout, layer occlusion (z-order), and physical or geometric references. The image generation model acts as a "colorist," using the rendered sketch as a structural condition while supplying textures, materials, and realism (*Color*). Figure 2 illustrates the capabilities of this decoupled workflow:

- **Controllable composition.** Coordinate and quantity references in visual code improve control of object counts, spatial relations, and attribute assignments.
- **Reliable text layout.** SVG or HTML renders text explicitly and controls font size, alignment, and hierarchy, reducing spelling errors from pixel synthesis.
- **Physical reasoning.** Python, Canvas, or Three.js constructs geometric and physical references, such as lighting, perspective, and mirror reflections.
- **Structured visual-condition editing.** The agent modifies visual code to provide the generation model with explicit structural conditions for the requested edit.
- **Layered image editing.** Segmented layers and structured JSONL records isolate edit targets and help preserve unmodified regions.

The staged workflow also makes intermediate decisions inspectable. Users can examine retrieved context, edit the code sketch, and compare the planned structure with the final image to investigate failures in retrieval, construction, or rendering. GenClaw thus explores how code agents can serve visual creation through executable, revisable intermediate representations. Our quantitative results and qualitative cases evaluate the resulting agent as an integrated system across composition, text rendering, editing, and knowledge-intensive generation.

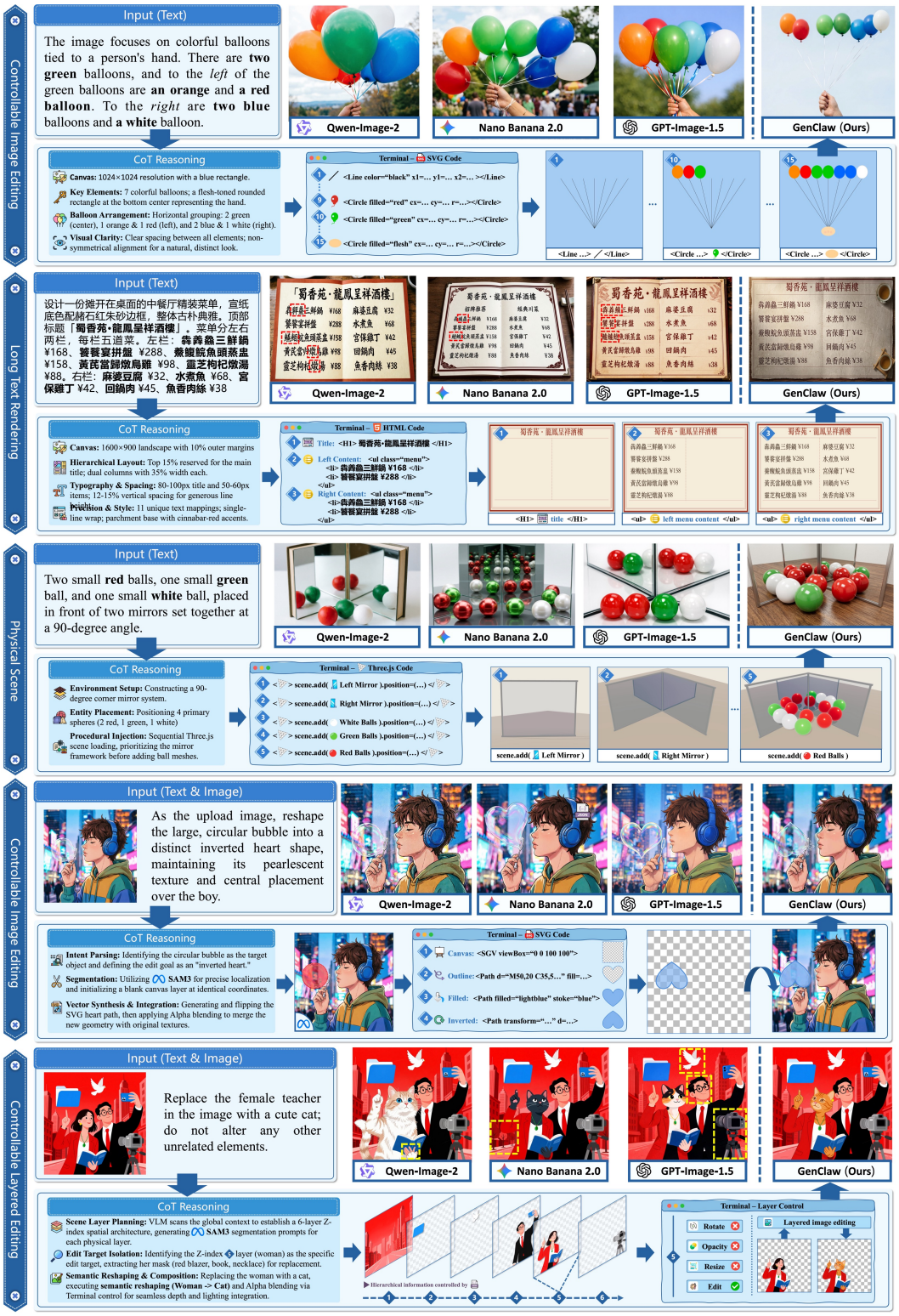


Figure 2: **Capability overview.** GenClaw uses executable intermediate canvases for scene composition, text rendering, physical reasoning, visual-condition editing, and layered image editing. Each case connects the request and comparison images with the agent's intermediate construction.

2 Related Work

Image generation and agents. Image generation has evolved from text-conditioned pixel synthesis toward unified large multimodal models that support visual understanding and generation (Jiang et al., 2025b;a). Models such as BAGEL (Deng et al., 2025a), Qwen-Image (Wu et al., 2025a), and Nano-Banana (Fortin et al., 2025) provide the visual-decoding substrate for our framework. This progress has also motivated agentic approaches (Feng et al., 2026; Chen et al., 2026; Ren et al., 2026). Think-Then-Generate (Kou et al., 2026) and GenAgent (Jiang et al., 2026) explicitly decouple high-level understanding from concrete generation. Mind-Brush (He et al., 2026) incorporates search and reasoning tools into open-domain creation to bridge real-time knowledge gaps. JarvisEvo (Lin et al., 2025b) combines interleaved multimodal reasoning, editing tools, and editor–evaluator coordination; RefineEdit-Agent (Liang et al., 2025) likewise uses visual feedback for iterative editing. These works establish context gathering and feedback as important parts of visual creation. GenClaw builds on this agentic setting by exposing the planned visual structure through an executable canvas that the agent can revise directly.

Visual code and layered representations. OmniSVG (Yang et al., 2026) and InternSVG (Wang et al., 2025a) study visual content represented as editable vector programs. VCode (Lin et al., 2025a) explores SVG as a symbolic visual representation; Vec2Pix (Guo et al., 2026) uses hierarchical vector graphics to guide pixel generation; and Qwen-Image-Layered (Yin et al., 2025) studies inherent editability through layer decomposition. Closest to our work, CoCo (Li et al., 2026) trains a unified model to generate executable sketch code and refine the rendered draft. GenClaw instead organizes a decoupled agent workflow around search and reasoning, task-dependent executable canvases, and an off-the-shelf image generator. Code serves as an intermediate representation for composition, typography, and layered construction, while the generator supplies textures and photorealistic appearance. Extended background is provided in Appendix F.

3 Method

3.1 Overall Framework

GenClaw organizes image creation into three conceptual layers (Figure 3). The **Cognitive Structuring Layer** uses a VLM/LLM with search and reasoning tools to understand intent, reference images, world knowledge, and constraints. The **Executable Canvas Layer** converts these records into code that explicitly constructs objects, coordinates, text, and layers. The **Visual Generation and Review Layer** invokes an image model to enrich the canvas, followed by VLM or user review. The agent autonomously selects and executes tools according to the request and intermediate results. Appendix A expands the method; Appendix D provides selected tool prompts and response formats.

3.2 Cognitive Structuring Layer

This layer decouples intent understanding and reasoning from pixel generation. The agent parses intent, completes missing knowledge, and converts reasoning conclusions into visual constraints. Ordinary requests may need only lightweight organization; long-tail entities, real-time events, locations, or specialized objects trigger search for facts and reference images. Mathematical, geographic, and physical tasks first require explicit reasoning about the underlying relations.

JSONL-style records specify objects, literal text, spatial relations, and the knowledge supporting each decision. They prioritize executability and traceability over descriptive richness, connecting intent to an editable canvas and allowing users to inspect or correct factual content before rendering.

3.3 Executable Canvas Layer

The agent selects a programming backend according to the task and compiles the records into executable code. The rendered sketch carries the core layout, text, and structural relations. Figure 2 illustrates the following uses of this task-dependent canvas.

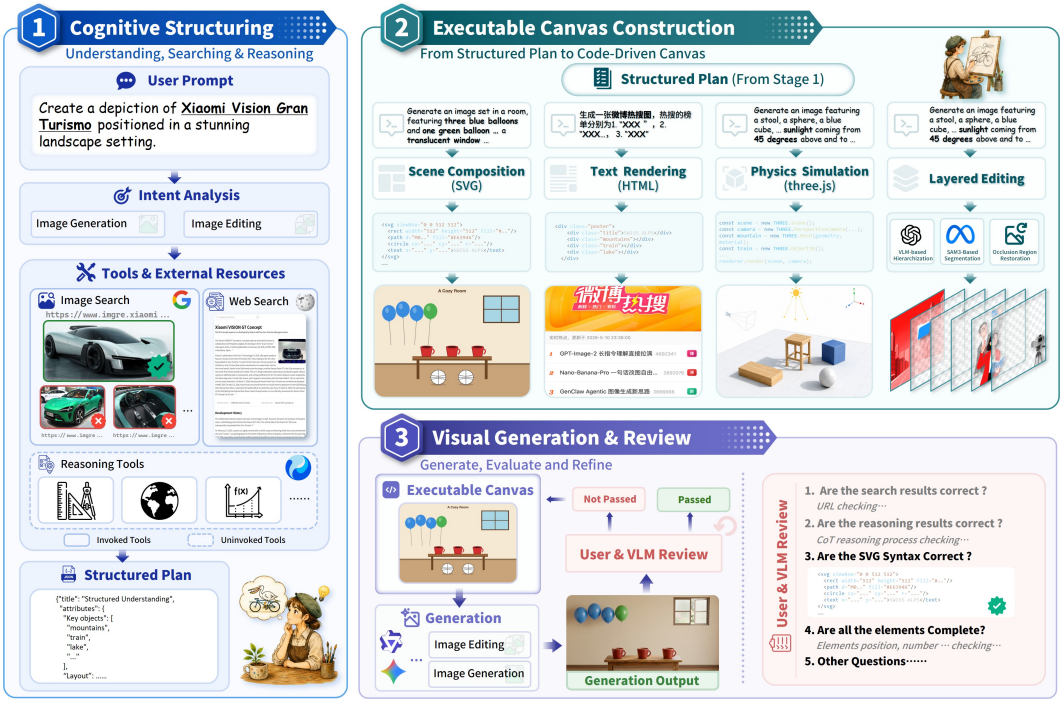


Figure 3: **Overall framework.** Cognitive structuring establishes intent; code constructs the canvas; generation and review produce and inspect the final image.

Composition and typography. SVG encodes object counts and spatial relations through nodes with explicit coordinates, sizes, and rendering order. The agent directly specifies counts and left–right or occlusion relations. For text-intensive tasks, HTML/CSS or SVG renders titles, paragraphs, prices, and section information with controlled font size, alignment, and hierarchy. Code-rendered text can also be combined with a model-generated background.

Physical and geometric references. Python, Canvas, or lightweight 3D code constructs explicit geometric or physical references before synthesis. For example, Three.js (Three.js Authors, 2024) places mirrors and objects and renders their reflections (Figure 2).

Layered editing. For layered edits, a VLM identifies objects, SAM 3 (Carion et al., 2025) segments them, and an image model completes occlusions when needed. The resulting layers are organized in JSONL with transparency and rendering order. To move an object, replace text, or modify a local region, the agent locates the corresponding layer or mask and changes its position, content, or attributes. The agent inspects execution results to revise counts, text, positions, and layer relations.

3.4 Visual Generation and Review Layer

The third layer supplies the rendered sketch and text conditions to an image generation or editing model, such as Qwen-Image or Nano Banana (Comanici et al., 2025). The model supplements texture, lighting, materials, and realism on the planned structure, combining code's explicit control with the generator's appearance modeling. For editing, a target layer or mask guides local changes and helps preserve surrounding content (Section 4.2).

The Review module compares the output with the request and structured records. Retrieved facts can be checked against their sources, while missing objects or text can be traced through the sketch and final image. These records help distinguish cognitive, structural, and visual failures and guide targeted correction. A correct sketch does not guarantee that constraints survive synthesis: counts, text, or geometry may still change, and the intermediate records make these deviations inspectable.

Table 1: Quantitative comparison on **GenEval++** (Ye et al., 2025b). Model references are provided in Table 7.

Method	Count	Color	Color/Count	Color/Pos	Pos/Count	Pos/Size	Multi-Count	Overall
Open-source models								
SD-3.5 M	0.600	0.400	0.250	0.250	0.075	0.400	0.300	0.325
Janus Pro 7B	0.300	0.450	0.125	0.300	0.075	0.350	0.125	0.246
T2I-R1	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
BLIP3-o	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
OmniGen2	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
Bagel	0.600	0.325	0.250	0.325	0.250	0.475	0.375	0.371
Qwen-Image	0.725	0.875	0.725	0.600	0.475	0.725	0.550	0.668
Closed-source models								
GPT-Image-1	0.675	0.900	0.725	0.625	0.600	0.800	0.850	0.739
Nano-Banana	0.700	0.800	0.680	0.530	0.680	0.780	0.800	0.710
GPT-Image-1.5	0.950	0.850	0.775	0.500	0.600	0.775	0.800	0.750
Nano-Banana Pro	0.900	0.800	0.750	0.700	0.725	0.725	0.850	0.761
Gemini-3.1 Flash-Image	0.850	0.850	0.800	0.750	0.675	0.725	0.775	0.775
Agentic generation models								
PromptEnhancer	0.625	0.500	0.225	0.375	0.125	0.450	0.375	0.382
GenAgent	0.775	0.775	0.650	0.800	0.600	0.725	0.750	0.725
Mind-Brush	0.700	0.775	0.775	0.750	0.850	0.775	0.850	0.782
GenClaw	0.950	0.875	0.825	0.850	0.925	0.825	0.900	0.879

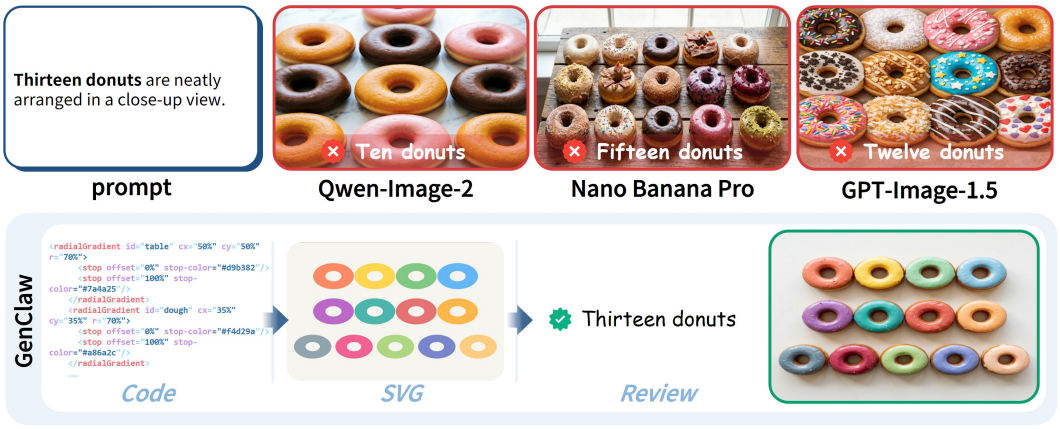


Figure 4: **Compositional control.** An SVG sketch specifies thirteen donuts before visual synthesis.

4 Experiments

4.1 Experimental Setup

We evaluate GenClaw on GenEval++ (Ye et al., 2025b) for scene instruction following, LongText-Bench (Geng et al., 2025) for text rendering, ImgEdit (Ye et al., 2025e) for editing, and Mind-Bench (He et al., 2026) for knowledge and reasoning. Baselines include GPT-Image (OpenAI, 2025a), Qwen-Image (Wu et al., 2025a), Nano-Banana (Fortin et al., 2025), BAGEL (Deng et al., 2025a), and generation agents GenAgent and Mind-Brush.

The agent backbone is Claude-Opus-4.6 (Anthropic, 2026), and the default generator is Gemini-3.1 Flash-Image (Raisinghani, 2026). The agent constructs structured records and executable canvases; the generator renders images conditioned on sketches, text layers, masks, or editing constraints. Rendering backends follow Section 3.3. Appendix B provides evaluation details and a direct-generation comparison; Appendix C provides additional visualizations.

Table 2: Quantitative comparison on **LongText-Bench**. We report the official metrics for both English and Chinese long-text rendering.

Model	LongText-Bench-EN ↑	LongText-Bench-ZH ↑
FLUX.1-dev Labs (2024)	0.607	0.005
OmniGen2 Wu et al. (2025b)	0.561	0.059
BAGEL Deng et al. (2025b)	0.373	0.310
CoCo Li et al. (2026)	0.755	0.753
Lumina-DiMOO Xin et al. (2025)	0.437	0.047
InternVL-U Tian et al. (2026)	0.738	0.860
X-Omni Geng et al. (2025)	0.900	0.814
LongCat-Next Meituan LongCat Team et al. (2026)	0.932	0.891
Z-Image Cai et al. (2025)	0.935	0.936
Qwen-Image Wu et al. (2025a)	0.943	0.946
Emu3.5 Cui et al. (2025)	0.976	0.928
GPT-Image-1 OpenAI (2025a)	0.956	0.619
Seedream 4.5 ByteDance Seed (n.d.)	0.989	0.987
Gemini-3.1 Flash-Image Raisinghani (2026)	0.981	0.949
GenClaw	0.989	0.988



Figure 5: **Long-text poster design: Preface to the Pavilion of Prince Teng.** The original request, model comparisons, retrieved text and visual references, and GenClaw output show how knowledge completion and explicit typography support poster construction.

4.2 Main Results

Compositional control with executable canvases. Table 1 reports results on GenEval++, which evaluates instruction following for complex scene layouts. GenClaw obtains an overall score of 0.879, compared with 0.775 for Gemini-3.1 Flash-Image and 0.782 for Mind-Brush. The results show advantages in object counting and spatial relations. GenClaw writes object nodes, positions, sizes, and layers into an SVG canvas before generation, providing an explicit structural condition for the visual decoder. Figure 4 shows a counting example with its intermediate SVG sketch and final image.

Text rendering and poster generation. On LongText-Bench, GenClaw obtains 0.989 on English and 0.988 on Chinese (Table 2). These scores match Seedream 4.5 on English and are slightly higher on Chinese (0.987). Text is deterministically rendered through HTML/SVG layers, while the image

Table 3: Quantitative comparison on **ImgEdit**. We utilize the VLM-Score to measure the efficacy of editing instruction execution, while employing PSNR and SSIM to evaluate the image consistency within unedited regions.

Method	VLM-Score ↑	PSNR ↑	SSIM ↑
FLUX.1 Kontext Labs (2025)	3.43	19.59	0.592
Qwen-Image-Edit Wu et al. (2025a)	3.70	17.64	0.526
Uniworld-V2 (Qwen-Image-Edit) Li et al. (2025)	3.72	16.13	0.482
CoCoEdit Wu et al. (2026)	3.79	19.13	0.555
FLUX.2 Klein 9B Black Forest Labs (2026c)	4.33	22.50	0.695
Gemini-3.1 Flash-Image Raisinghani (2026)	4.51	20.19	0.583
GPT-Image-1.5 OpenAI (2025b)	4.69	16.36	0.433
Ours	4.29	27.87	0.718

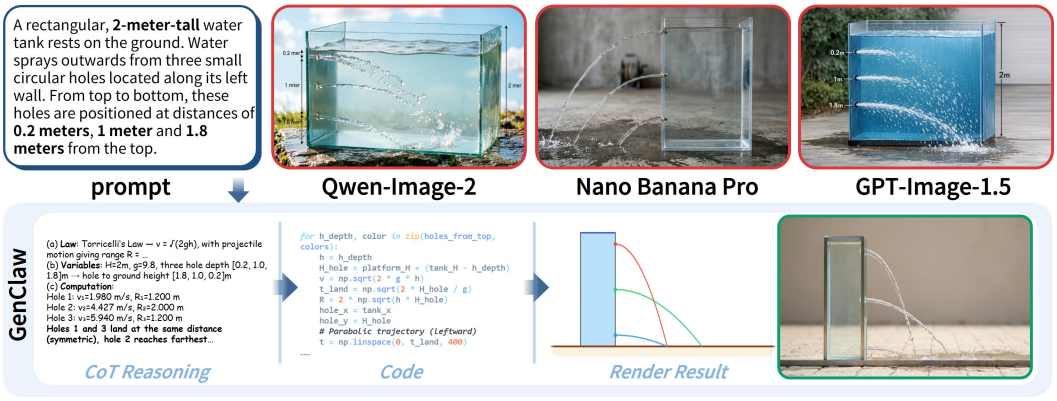


Figure 6: **Physical reasoning through executable sketches.** Code converts the inferred water-jet relations into explicit trajectories before the image model renders the scene.

model supplies backgrounds, style, and visual detail. This division is useful for text-intensive HTML pages, slides, and posters. Figure 5 shows the *Preface to the Pavilion of Prince Teng* case: search supplies the classical text and cultural references, and the executable canvas organizes the poster before visual rendering.

Image editing. We evaluate ImgEdit using VLM-based holistic scoring and, following Co-CoEdit (Wu et al., 2026), PSNR and SSIM computed exclusively on unedited areas using mask annotations. Table 3 shows that GenClaw achieves higher preservation scores (27.87 PSNR and 0.718 SSIM), consistent with the protection afforded by layer-wise editing. Its VLM-Score of 4.29 is lower than the 4.51 of Gemini-3.1 Flash-Image and 4.69 of GPT-Image-1.5, indicating a trade-off between editing effectiveness and preservation of non-target regions. Layer decomposition remains rudimentary, but provides a useful interface for localized edits.

Physical simulation. Our qualitative exploration uses Python, Canvas, or Three.js to construct simplified physical or geometric models before visual rendering. These executable drafts encode relations such as spring deformation, buoyancy, reflection, and water-jet range. Figure 6 shows the water-jet case: the intermediate calculation predicts that the middle hole reaches farthest, while the upper and lower holes have equal range. The code renders these trajectories before the image model supplies appearance. These cases illustrate the potential of code as an intermediate physical model rather than constituting a quantitative physics benchmark.

Knowledge grounding. Mind-Bench (He et al., 2026) tests whether a system understands implicit intent and external facts before generation. GenClaw obtains 0.57 overall, compared with 0.38 for Nano-Banana Pro and 0.31 for Mind-Brush; the per-category results appear in Table 4. GenClaw uses multi-round search and selects reference images from multiple retrieved candidates. Figure 7 illustrates map-grounded street views and factual content retrieval, with intermediate evidence available for inspection.

Table 4: Knowledge and reasoning results on **Mind-Bench**. Full model references appear in Table 8. "–" denotes an inapplicable I2I task.

Model	Knowledge-Driven					Reasoning-Driven					Overall
	SE	Weather	MC	IP	WK	SL	Poem	Life	GU	Math	
FLUX 1 dev	0.04	0.00	0.00	0.00	0.02	0.02	0.04	-	-	-	0.02
FLUX 1 Kontext	0.02	0.00	0.00	0.00	0.02	0.00	0.00	-	-	-	0.01
BAGEL	0.02	0.00	0.00	0.00	0.00	0.02	0.02	0.02	0.00	0.08	0.02
Z-Image	0.02	0.00	0.08	0.02	0.00	0.00	0.00	-	-	-	0.02
Qwen-Image	0.08	0.00	0.04	0.00	0.00	0.04	0.00	0.04	0.00	0.00	0.02
GPT-Image-1	0.32	0.06	0.22	0.02	0.16	0.32	0.10	0.24	0.10	0.12	0.17
GPT-Image-1.5	0.36	0.18	0.22	0.04	0.30	0.34	0.08	0.34	0.10	0.02	0.21
FLUX 2 Max	0.26	0.34	0.02	0.00	0.34	0.32	0.52	0.20	0.18	0.10	0.23
Nano Banana	0.24	0.20	0.12	0.00	0.36	0.32	0.40	0.28	0.08	0.24	0.22
FLUX 2 Pro	0.28	0.32	0.02	0.00	0.20	0.36	0.58	0.20	0.16	0.12	0.22
Nano-Banana Pro	0.52	0.24	0.20	0.04	0.52	0.60	0.72	0.28	**0.44**	0.28	0.38
Mind-Brush	0.54	0.16	0.62	0.18	0.40	0.26	0.54	0.10	0.16	0.14	0.31
GENCLAW	**0.64**	**0.44**	**0.66**	**0.32**	**0.64**	**0.78**	**0.90**	**0.38**	0.32	**0.60**	**0.57**

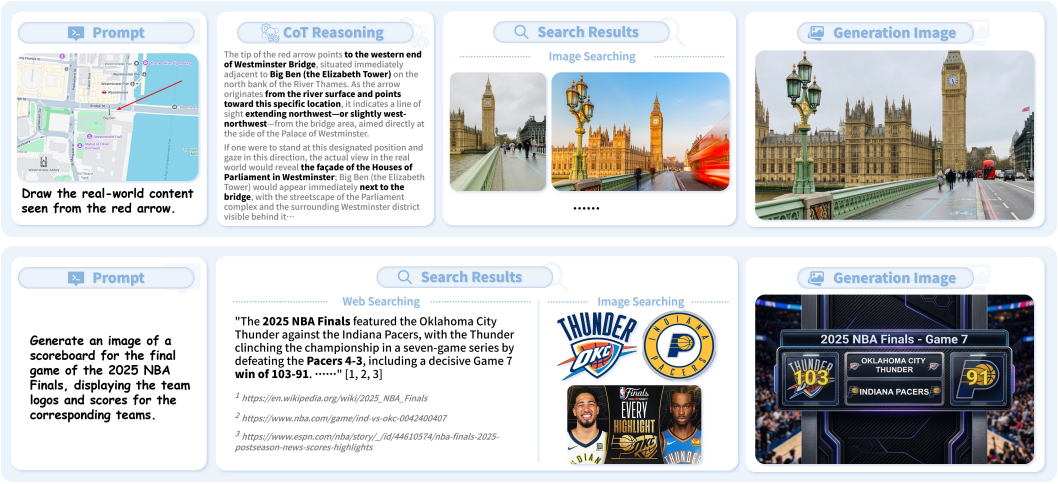


Figure 7: **GenClaw on Mind-Bench.** The agent invokes reasoning and search tools to obtain contextual information before visual rendering.

For street views, the agent retrieves images of the map-inferred location; for an NBA scoreboard, it retrieves game information. These references expose the factual basis for inspection.

5 CONCLUSION

We introduce the Code-Driven Agentic Image Generation paradigm and instantiate it in GENCLAW. Through the *Conceptualize* → *Sketch* → *Color* workflow, the LLM focuses on logic and structure, while the image generation model supplies pixels and texture. This decoupled design supports controllable composition, text rendering, physical simulation, and layered editing. Executable intermediate canvases also make the creation process inspectable: users can examine and revise the planned structure, and investigate failures in retrieval, code construction, or final rendering.

Looking ahead, just as code agents like Claude Code and Cursor are reshaping software engineering, we believe image generation is progressively merging into this broader agentic wave. While previous generative models predominantly operated in a passive, chat-based response mode, future visual generation systems may increasingly adopt a more proactive paradigm, using tools and editable intermediate representations throughout the creative process. GENCLAW represents our initial exploratory step in this direction. We hope this work provides a useful reference for visual creation systems with greater controllability, interpretability, and reasoning capability.

AI USE STATEMENT

Generative AI tools assisted the implementation of experimental code and the writing of this paper, including manuscript editing and LaTeX formatting. The GenClaw system itself invokes language and image generation models for planning, code construction, and visual synthesis, as described in Section 3.

REPRODUCIBILITY STATEMENT

Section 3 describes the method and Section 4.1 identifies the model backbones and rendering backends. Appendix A expands the method, Appendix B provides additional quantitative comparisons, and Appendix C preserves further qualitative examples. Appendix D provides selected tool prompts and response formats, with their provenance recorded alongside the excerpts.

REFERENCES

Anthropic. Claude. https://www.anthropic.com/claude, 2024. Accessed: 2026-05-07.

Anthropic. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026. February 5, 2026.

Black Forest Labs. Flux 2 max: Next generation image synthesis. https://bfl.ai/models/flux-2-max, 2026a. Accessed: 2026-01-26.

Black Forest Labs. Flux 2 pro: State-of-the-art quality at maximum speed. https://bfl.ai/models/flux-2, 2026b. Accessed: 2026-01-26.

Black Forest Labs. FLUX.2 [klein]: Towards Interactive Visual Intelligence. https://bfl.ai/blog/flux2-klein-towards-interactive-visual-intelligence, 2026c. Accessed: 2026-05-07.

ByteDance Seed. Deeper thinking, more accurate generation | introducing Seedream 5.0 Lite. https://seed.bytedance.com/en/blog/deeper-thinking-more-accurate-generation-introducing-seedream-5-0-lite, February 2026a. Published February 13, 2026.

ByteDance Seed. Beyond generation, it understands design | introducing Seedream 5.0 Pro. https://seed.bytedance.com/en/blog/beyond-generation-it-understands-design-introducing-seedream-5-0-pro, July 2026b. Published July 8, 2026.

ByteDance Seed. Seedream4.5. https://seed.bytedance.com/en/seedream4_5, n.d. Official model page; accessed September 26, 2026.

Huanqia Cai, Sihan Cao, Ruoyi Du, Peng Gao, Steven Hoi, Zhaohui Hou, Shijie Huang, Dengyang Jiang, Xin Jin, Liangchen Li, et al. Z-image: An efficient image generation foundation model with single-stream diffusion transformer. *arXiv preprint arXiv:2511.22699*, 2025.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Dídac Surís, Chaitanya K. Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, J. Greer, Meng Wang, Peize Sun, Roman Radle, Triantafyllos Afouras, E. Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, S. Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, H. Cheng, Piotr Doll'ar, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. Sam 3: Segment anything with concepts. *ArXiv*, abs/2511.16719, 2025. doi: 10.48550/arXiv.2511.16719. URL https://arxiv.org/abs/2511.16719.

Jiuhai Chen, Zhiyang Xu, Xichen Pan, Yushi Hu, Can Qin, Tom Goldstein, Lifu Huang, Tianyi Zhou, Saining Xie, Silvio Savarese, et al. Blip3-o: A family of fully open unified multimodal models-architecture, training and dataset. *arXiv preprint arXiv:2505.09568*, 2025a.

Shuang Chen, Quanxin Shou, Hangting Chen, Yucheng Zhou, Kaituo Feng, Wenbo Hu, Yi-Fan Zhang, Yunlong Lin, Wenxuan Huang, Mingyang Song, et al. Unify-agent: A unified multimodal agent for world-grounded image synthesis. *arXiv preprint arXiv:2603.29620*, 2026.

Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025b.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Yufeng Cui, Honghao Chen, Haoge Deng, Xu Huang, Xinghang Li, Jirong Liu, Yang Liu, Zhuoyan Luo, Jinsheng Wang, Wenxuan Wang, Yueze Wang, Chengyuan Wang, Fan Zhang, Yingli Zhao, Ting Pan, Xianduo Li, Zecheng Hao, Wenxuan Ma, Zhuo Chen, Yulong Ao, Tiejun Huang, Zhongyuan Wang, and Xinlong Wang. Emu3.5: Native multimodal models are world learners, 2025. URL https://arxiv.org/abs/2510.26583.

DeepSeek-AI. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence. Technical Report, 2026.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025a.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025b.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Kaituo Feng, Manyuan Zhang, Shuang Chen, Yunlong Lin, Kaixuan Fan, Yilei Jiang, Hongyu Li, Dian Zheng, Chenyang Wang, and Xiangyu Yue. Gen-searcher: Reinforcing agentic search for image generation. *arXiv preprint arXiv:2603.28767*, 2026.

Alisa Fortin, Guillaume Vernade, Kat Kampf, and Ammaar Reshi. Introducing Gemini 2.5 Flash Image, our state-of-the-art image model. https://developers.googleblog.com/introducing-gemini-2-5-flash-image/, August 2025. Google Developers Blog, August 26, 2025.

Zigang Geng, Yibing Wang, Yeyao Ma, Chen Li, Yongming Rao, Shuyang Gu, Zhao Zhong, Qinglin Lu, Han Hu, Xiaosong Zhang, et al. X-omni: Reinforcement learning makes discrete autoregressive image generative models great again. *arXiv preprint arXiv:2507.22058*, 2025.

Lanqing Guo, Xi Liu, Yufei Wang, Zhihao Li, and Siyu Huang. Controlling your image via simplified vector graphics. *arXiv preprint arXiv:2602.14443*, 2026.

Jun He, Junyan Ye, Zilong Huang, Dongzhi Jiang, Chenjue Zhang, Leqi Zhu, Renrui Zhang, Xiang Zhang, and Weijia Li. Mind-Brush: Integrating agentic cognitive search and reasoning into image generation, 2026. URL https://arxiv.org/abs/2602.01756. Accepted to SIGGRAPH Asia 2026.

Dongzhi Jiang, Ziyu Guo, Renrui Zhang, Zhuofan Zong, Hao Li, Le Zhuo, Shilin Yan, Pheng-Ann Heng, and Hongsheng Li. T2i-r1: Reinforcing image generation with collaborative semantic-level and token-level cot. *arXiv preprint arXiv:2505.00703*, 2025a.

Dongzhi Jiang, Renrui Zhang, Haodong Li, Zhuofan Zong, Ziyu Guo, Jun He, Claire Guo, Junyan Ye, Rongyao Fang, Weijia Li, et al. Draco: Draft as cot for text-to-image preview and rare concept generation. *arXiv preprint arXiv:2512.05112*, 2025b.

Kaixun Jiang, Yuzheng Wang, Junjie Zhou, Pandeng Li, Zhihang Liu, Chen-Wei Xie, Zhaoyu Chen, Yun Zheng, and Wenqiang Zhang. Genagent: Scaling text-to-image generation via agentic multimodal reasoning, 2026. URL https://arxiv.org/abs/2601.18543.

Siqi Kou, Jiachun Jin, Zetong Zhou, Ye Ma, Yugang Wang, Quan Chen, Peng Jiang, Xiao Yang, Jun Zhu, Kai Yu, et al. Think-then-generate: Reasoning-aware text-to-image diffusion with llm encoders. *arXiv preprint arXiv:2601.10332*, 2026.

Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024.

Black Forest Labs. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space, 2025.

Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space, 2025. URL https://arxiv.org/abs/2506.15742.

Haodong Li, Chunmei Qing, Huanyu Zhang, Dongzhi Jiang, Yihang Zou, Hongbo Peng, Dingming Li, Yuhong Dai, ZePeng Lin, Juanxi Tian, Yi Zhou, Siqi Dai, Jingwei Wu, and Pheng-Ann Heng. CoCo: Code as CoT for text-to-image preview and rare concept generation, 2026. URL https://arxiv.org/abs/2603.08652. Accepted to ECCV 2026.

Weijia Li, Jun He, Junyan Ye, Huaping Zhong, Zhimeng Zheng, Zilong Huang, Dahua Lin, and Conghui He. Crossviewdiff: A cross-view diffusion model for satellite-to-street view synthesis. *arXiv preprint arXiv:2408.14765*, 2024.

Zongjian Li, Zheyuan Liu, Qihui Zhang, Bin Lin, Feize Wu, Shenghai Yuan, Zhiyuan Yan, Yang Ye, Wangbo Yu, Yuwei Niu, Shaodong Wang, Xinhua Cheng, and Li Yuan. Uniworld-v2: Reinforce image editing with diffusion negative-aware finetuning and mllm implicit feedback, 2025. URL https://arxiv.org/abs/2510.16888.

Zihan Liang, Jiahao Sun, and Haoran Ma. An llm-lvlm driven agent for iterative and fine-grained image editing. *arXiv preprint arXiv:2508.17435*, 2025.

Kevin Qinghong Lin, Yuhao Zheng, Hangyu Ran, Dantong Zhu, Dongxing Mao, Linjie Li, Philip Torr, and Alex Jinpeng Wang. Vcode: a multimodal coding benchmark with svg as symbolic visual representation. *arXiv preprint arXiv:2511.02778*, 2025a.

Yunlong Lin, Linqing Wang, Kunjie Lin, Zixu Lin, Kaixiong Gong, Wenbo Li, Bin Lin, Zhenxi Li, Shiyi Zhang, Yuyang Peng, et al. Jarvisevo: Towards a self-evolving photo editing agent with synergistic editor-evaluator optimization. *arXiv preprint arXiv:2511.23002*, 2025b.

Meituan LongCat Team, Bin Xiao, Chao Wang, Chengjiang Li, et al. LongCat-Next: Lexicalizing modalities as discrete tokens, 2026. URL https://arxiv.org/abs/2603.27538.

OpenAI. Gpt-4o. https://openai.com/index/introducing-4o-image-generation, 2025.

OpenAI. Introducing our latest image generation model in the API. https://openai.com/index/image-generation-api/, April 2025a. Introducing GPT Image 1; April 23, 2025.

OpenAI. The new ChatGPT Images is here. https://openai.com/index/new-chatgpt-images-is-here/, December 2025b. Introducing GPT Image 1.5; December 16, 2025.

Naina Raisinghani. Introducing Nano Banana Pro. https://blog.google/innovation-and-ai/products/nano-banana-pro/, November 2025. Google, November 20, 2025.

Naina Raisinghani. Nano Banana 2: Combining Pro capabilities with lightning-fast speed. https://blog.google/innovation-and-ai/technology/ai/nano-banana-2/, February 2026. Google, February 26, 2026.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Tianfei Ren, Zhipeng Yan, Yiming Zhao, Zhen Fang, Yu Zeng, Guohui Zhang, Hang Xu, Xiaoxiao Ma, Shiting Huang, Ke Xu, et al. Scope: Structured decomposition and conditional skill orchestration for complex image generation. *arXiv preprint arXiv:2605.08043*, 2026.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Three.js Authors. Three.js: Javascript 3d library. https://threejs.org, 2024. Accessed: 2026-05-07.

Changyao Tian, Danni Yang, Guanzhou Chen, Erfei Cui, Zhaokai Wang, Yuchen Duan, Penghao Yin, Sitao Chen, Ganlin Yang, Mingxin Liu, Zirun Zhu, Ziqian Fan, Leyao Gu, Haomin Wang, Qi Wei, Jinhui Yin, Xue Yang, Zhihang Zhong, Qi Qin, Yi Xin, Bin Fu, Yihao Liu, Jiaye Ge, Qipeng Guo, Gen Luo, Hongsheng Li, Yu Qiao, Kai Chen, and Hongjie Zhang. InternVL-U: Democratizing unified multimodal models for understanding, reasoning, generation and editing, 2026. URL https://arxiv.org/abs/2603.09877.

Haomin Wang, Jinhui Yin, Qi Wei, Wenguang Zeng, Lixin Gu, Shenglong Ye, Zhangwei Gao, Yaohui Wang, Yanting Zhang, Yuanqi Li, et al. Internsvg: Towards unified svg tasks with multimodal large language models. *arXiv preprint arXiv:2510.11341*, 2025a.

Linqing Wang, Ximing Xing, Yiji Cheng, Zhiyuan Zhao, Donghao Li, Tiankai Hang, Jiale Tao, Qixun Wang, Ruihuang Li, Comi Chen, et al. Promptenhancer: A simple approach to enhance text-to-image models via chain-of-thought prompt rewriting. *arXiv preprint arXiv:2509.04545*, 2025b.

Siwei Wen, Peilin Feng, Hengrui Kang, Zichen Wen, Yize Chen, Jiang Wu, Conghui He, Weijia Li, et al. Spot the fake: Large multimodal model-based synthetic image detection with artifact explanation. *Advances in Neural Information Processing Systems*, 38:58972–59005, 2026.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025a.

Chenyuan Wu, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, et al. Omnigen2: Exploration to advanced multimodal generation. *arXiv preprint arXiv:2506.18871*, 2025b.

Yuhui Wu, Chenxi Xie, Ruibin Li, Liyi Chen, Qiaosi Yi, and Lei Zhang. Cocoedit: Content-consistent image editing via region regularized reinforcement learning. *ArXiv*, abs/2602.14068, 2026. doi: 10.48550/arXiv.2602.14068. URL https://arxiv.org/abs/2602.14068.

Yi Xin, Qi Qin, Siqi Luo, Kaiwen Zhu, Juncheng Yan, Yan Tai, Jiayi Lei, Yuewen Cao, Keqi Wang, Yibin Wang, Jinbin Bai, Qian Yu, Dengyang Jiang, Yuandong Pu, Haoxing Chen, Le Zhuo, Junjun He, Gen Luo, Tianbin Li, Ming Hu, Jin Ye, Shenglong Ye, Bo Zhang, Chang Xu, Wenhai Wang, Hongsheng Li, Guangtao Zhai, Tianfan Xue, Bin Fu, Xiaohong Liu, Yu Qiao, and Yihao Liu. Lumina-DiMOO: An omni diffusion large language model for multi-modal generation and understanding, 2025. URL https://arxiv.org/abs/2510.06308.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *Advances in Neural Information Processing Systems*, 38:113670–113696, 2026.

Junyan Ye, Jun He, Weijia Li, Zhutao Lv, Yi Lin, Jinhua Yu, Haote Yang, and Conghui He. Leveraging bev paradigm for ground-to-aerial image synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 28451–28461, 2025a.

Junyan Ye, Dongzhi Jiang, Zihao Wang, Leqi Zhu, Zhenghao Hu, Zilong Huang, Jun He, Zhiyuan Yan, Jinghua Yu, Hongsheng Li, et al. Echo-4o: Harnessing the power of gpt-4o synthetic images for improved image generation. *arXiv preprint arXiv:2508.09987*, 2025b.

Junyan Ye, Baichuan Zhou, Zilong Huang, Junan Zhang, Tianyi Bai, Hengrui Kang, Jun He, Honglin Lin, Zihao Wang, Tong Wu, et al. Loki: A comprehensive synthetic data detection benchmark using large multimodal models. In *International Conference on Learning Representations*, volume 2025, pp. 70440–70522, 2025c.

Junyan Ye, Leiqi Zhu, Yuncheng Guo, Dongzhi Jiang, Zilong Huang, Yifan Zhang, Zhiyuan Yan, Haohuan Fu, Conghui He, and Weijia Li. Realgen: Photorealistic text-to-image generation via detector-guided rewards. *arXiv preprint arXiv:2512.00473*, 2025d.

Yang Ye, Xianyi He, Zongjian Li, Bin Lin, Shenghai Yuan, Zhiyuan Yan, Bohan Hou, and Li Yuan. Imgedit: A unified image editing dataset and benchmark. *arXiv preprint arXiv:2505.20275*, 2025e.

Shengming Yin, Zekai Zhang, Zecheng Tang, Kaiyuan Gao, Xiao Xu, Kun Yan, Jiahao Li, Yilei Chen, Yuxiang Chen, Heung-Yeung Shum, et al. Qwen-image-layered: Towards inherent editability via layer decomposition. *arXiv preprint arXiv:2512.15603*, 2025.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3836–3847, 2023.

SUPPLEMENTARY MATERIAL

The supplement expands the method, reports additional quantitative and qualitative results, and documents prompt excerpts from the public implementation. The final sections discuss limitations and extended related work.

CONTENTS

A Extended Method Details 15
B Evaluation Details and Quantitative Results 17
C Additional Qualitative Results 20
Text Rendering and Poster Design 20
Physical Simulation 21
D Agent and Tool Prompts 22
E Limitations and Future Work 27
F Extended Related Work 27

A EXTENDED METHOD DETAILS

This section expands the three layers summarized in Section 3, including backend selection, editable records, and error tracing.

A.1 COGNITIVE STRUCTURING LAYER

The goal of this layer is to decouple the understanding and reasoning tasks before image generation from the image generation model, and to establish a cognitive trajectory between the user's intent and the executable canvas. This corresponds to the thinking and conception stage in human drawing. Existing generation models are good at mapping conditions to visual content, but they are not always suitable for complex intent parsing, world-knowledge retrieval, or symbolic reasoning. Different from directly expanding a prompt, this layer does not try to directly generate the final image description. Instead, it explicitly parses the user's intent, completes missing knowledge, or performs necessary reasoning analysis, and organizes the results as structured records.

Specifically, the agent first performs intent understanding. For ordinary image generation tasks, the agent may only conduct lightweight prompt organization. However, when the task involves specific or dynamic concepts, such as long-tail entities, real-time events, geographic locations, cultural symbols, or professional objects, the model's internal knowledge is often insufficient to support accurate generation. At this time, the agent calls search tools to complete the relevant facts, thereby filling the cognitive gap. For requests involving mathematics, geography, physics, and other tasks that require complex understanding and reasoning, the agent first explicitly obtains intermediate conclusions based on the VLM's reasoning ability, and then converts implicit relations into visual constraints. For instance, in geometry tasks, the agent computes the numerical answer prior to rendering it visually. This process makes the cognitive work before image generation explicit, instead of leaving all understanding pressure to the final image model.

After completing intent understanding, knowledge completion, and reasoning analysis, the agent organizes the results into JSONL-style structured records (implementation-specific interfaces are documented in Appendix D). Unlike natural-language prompts, these records do not pursue descriptive richness, but emphasize executability and traceability: they need to specify which objects should appear, which text should be rendered, which relations should be preserved, and which knowledge facts support these visual decisions. Such structured records are also useful for interaction between human users and the agent. For example, in drawing a science-popularization poster, the user can explicitly query whether the knowledge content is wrong.

A.2 Executable Canvas Layer

After completing the conception of the drawing content, the agent constructs sketch content in a way similar to human drawing. The agent selects an appropriate programming backend according to the task type, and compiles the objects, text, spatial relations, and constraints into executable code. After execution, the code generates a sketch-like intermediate image, which carries the core layout, text content, and structural relations of the image.

For complex compositional scene generation, such as tasks that emphasize quantity and spatial relations, the agent can use SVG as the canvas backend. SVG can explicitly represent each object as a node and control layout through coordinates and size. For example, in scenes that require generating a fixed number of objects, strict left-right relations, or occlusion relations, the agent can directly create the corresponding number of object nodes in SVG, rather than relying on the image model to understand natural-language descriptions such as "how many", or "located at the center".

For text-intensive tasks, the agent can use HTML/CSS to build the canvas. Menus, course schedules, webpage cards, instruction pages, and infographics often contain large amounts of Chinese, English, prices, titles, and section information. At this time, the agent deterministically draws the text content through a renderer, and then hands it to the subsequent image model for visual enhancement, avoiding asking the image model to directly "guess-write" text as pixel texture. In addition, in some tasks where the text does not need to be re-rendered and regenerated, we can also adopt a strategy where the text is directly rendered by code and the background image is generated by the image model.

For tasks involving physical laws, the agent can call Python plotting, Canvas, or lightweight 2D/3D code to build geometric references. The key to this type of task is not visual style, but correct relations. For example, based on Three.js (Three.js Authors, 2024) code, the system can explicitly place a mirror and render the reflection result of a small ball, as shown in Figure 2. Here, code provides a simplified geometric model of mirror reflection before final image generation.

For editing tasks, the agent tends to first build a layered representation of the image. The agent first understands the image content based on a VLM, then divides the objects in each layer, and then uses SAM 3 (Carion et al., 2025) tools for object segmentation. It also uses an image model to complete occluded regions. After layered representation, the agent can treat the contents as editable layers and control transparency and rendering order through a JSONL format. For example, when the user asks to move an object, replace a piece of text, or modify only a certain region in the image, the agent can first locate the corresponding object layer or mask region, and modify its position, content, or attributes.

Prompt-centered generation workflows influence the image model mainly through textual conditions, while code-based canvases additionally expose an editable spatial representation. In contrast, code provides the agent with a more natural operation interface, and it is highly matched with the agent's capability form. The agent is better at reading and writing structured text, calling tools, modifying local code, and checking execution results, rather than drawing stroke by stroke through a mouse in a GUI like a human. Therefore, we do not ask the agent to simulate human GUI operations, but let it directly construct the image structure through CLI/code. Object count, text content, spatial position, and layer relations can all be clearly expressed in code.

A.3 Visual Generation and Review Layer

The third layer is where the agent calls existing image generation or editing models to complete the final visual realization, and uses a VLM or multimodal evaluation ability to review the result. Based on the executable code and its rendered sketch obtained from the second layer, the agent uses it as visual-condition input, and calls Qwen-Image, Nano Banana (Comanici et al., 2025), or other models with image generation and editing capabilities to complete the final generation. At this point, the image model no longer needs to plan the scene structure from scratch, but supplements texture, lighting, material, and realism on the basis of the existing sketch and structural constraints.

In generation scenarios, the image model more often plays the role of visual realizer: it performs naturalized rendering based on the code sketch and text condition, concentrating model capability on texture, lighting, material, details, and overall realism, rather than simultaneously undertaking tasks such as complex planning, counting, text layout, or physical reasoning. Compared with relying only on an LLM to generate a lightweight code sketch, the image model adds photorealistic detail

beyond that sketch. Code sketches are good at expressing structure, layout, and text, but are difficult to use for complex natural textures, realistic lighting, and open-scene details; the image generation model can preserve the sketch structure while extending visual content from simple UI or schematic diagrams to more natural and realistic scene generation.

In editing scenarios, the agent performs modifications based on editable objects, layered representations, or local masks provided by the second layer. For example, in tasks such as "move a cup", "replace the title in a poster", or "change the color of an object", the agent can first determine the object layer or region that needs to be modified, and then call an image editing model to complete the local visual update. Providing an explicit target layer or mask narrows the editing scope and can help preserve non-target regions.

After generation, the agent's Review module compares the final output with the user's objectives and the initial structured records. In conventional end-to-end image generation, although a VLM or user can ultimately detect synthesis errors, this black-box observation lacks fine-grained interpretability, making it difficult to pinpoint the root cause. GenClaw's intermediate records provide additional evidence for this inspection. A VLM or user can investigate potential sources of error by examining the representations recorded throughout the pipeline. For instance, if an error stems from flawed cognitive comprehension or inaccurate external knowledge, the agent can trace back to the first layer to verify the accuracy of the URL contents retrieved by the search tool. For issues such as incorrect object counts or missing text, the system can cross-reference the intermediate code outputs with the final rendered image to help localize the failure. These checks help distinguish errors introduced during knowledge gathering, canvas construction, or visual rendering, and guide targeted revisions.

A correct code sketch does not guarantee that all constraints survive image synthesis: the visual generator can still change counts, text, or geometry. Intermediate records make these deviations inspectable. Appendix D provides the agent and tool prompts for planning, canvas construction, and review, together with structured response schemas.

B Evaluation Details and Quantitative Results

B.1 Evaluation Protocol

Most baseline scores are taken from the corresponding published papers; a smaller subset is reproduced using the provided default settings and evaluation protocols. For GenEval++, we use the updated Echo-4o reference results. We retain the metrics defined by each benchmark: compositional instruction following on GenEval++, English and Chinese text rendering on LongText-Bench, editing quality and non-target-region preservation on ImgEdit, and knowledge and reasoning on Mind-Bench. For ImgEdit, the benchmark masks define the unedited regions used to compute PSNR and SSIM; the VLM score evaluates the edited image holistically. Reported baseline scores and reproduced scores are therefore interpreted under their respective evaluation protocols, rather than as measurements from a single matched-budget run.

GenClaw uses a shared agent policy across tasks. The agent selects and composes tools from the request and intermediate observations, executes them, and uses their outputs to determine subsequent actions. Search, code rendering, image generation or editing, and review are task-dependent choices within this loop; the three conceptual layers in Section 3 describe their roles rather than separate fixed benchmark pipelines.

B.2 Comparison with a Trained Code-as-CoT Model

CoCo (Li et al., 2026) fine-tunes a unified BAGEL model with code-generation and draft-refinement supervision. GenClaw instead uses a separate code agent to orchestrate task-dependent tools and off-the-shelf image models, without task-specific joint training. Table 2 includes CoCo's published LongText-Bench scores from its Table 2: 0.755 on English and 0.753 on Chinese, compared with 0.989 and 0.988 for GenClaw. The comparison illustrates the performance of these different system designs under their reported settings; the different backbones and rendering procedures do not isolate the effect of training strategy alone.

B.3 Direct Generation versus the Full Agent

Tables 5 and 6 compare direct Gemini-3.1 Flash-Image generation with the complete GenClaw system, using the results reported in Tables 1 and 2. The agent uses this image model as its default generator and organizes planning, task-dependent tools, executable canvases, and review into a code-driven workflow. For text-intensive tasks, deterministic HTML/SVG rendering also contributes to the final result.

Table 5: **Direct generation versus GenClaw on GenEval++.** Both settings use Gemini-3.1 Flash-Image as the image generator. All category scores are reproduced from Table 1; higher is better. Gains are absolute differences.

Setting	Count	Color	Color/ Count	Color/ Pos	Pos/ Count	Pos/ Size	Multi- Count	Overall
Direct generation	0.850	0.850	0.800	0.750	0.675	0.725	0.775	0.775
GenClaw	**0.950**	**0.875**	**0.825**	**0.850**	**0.925**	**0.825**	**0.900**	**0.879**
Absolute gain	+0.100	+0.025	+0.025	+0.100	+0.250	+0.100	+0.125	+0.104

Table 6: **Direct generation versus GenClaw on LongText-Bench.** The direct baseline is Gemini-3.1 Flash-Image. Scores are reproduced from Table 2; higher is better. Gains are absolute differences.

Setting	English	Chinese
Direct generation	0.981	0.949
GenClaw	**0.989**	**0.988**
Absolute gain	+0.008	+0.039

The complete system improves all reported GenEval++ categories, with the largest absolute gain on Position/Count (0.250) and an overall gain of 0.104. LongText-Bench improves by 0.008 on English and 0.039 on Chinese. These results support the effectiveness of the integrated agentic workflow built around an executable code canvas, where planning, construction, and review operate together.

B.4 Complete Quantitative Comparisons

Tables 7 and 8 provide model references for the GenEval++ and Mind-Bench comparisons in the main paper. All scores are identical to those in Tables 1 and 4.

Table 7: Complete quantitative comparison on **GenEval++** Ye et al. (2025b).

Method	Count	Color	Color/Count	Color/Pos	Pos/Count	Pos/Size	Multi-Count	Overall
Open-source Models								
SD-3.5 M Esser et al. (2024)	0.600	0.400	0.250	0.250	0.075	0.400	0.300	0.325
Janus Pro 7B Chen et al. (2025b)	0.300	0.450	0.125	0.300	0.075	0.350	0.125	0.246
T2I-R1 Jiang et al. (2025a)	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
BLIP3-o Chen et al. (2025a)	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
OmniGen2 Wu et al. (2025b)	0.325	0.675	0.200	0.350	0.075	0.250	0.300	0.311
Bagel Deng et al. (2025b)	0.600	0.325	0.250	0.325	0.250	0.475	0.375	0.371
Qwen-Image Wu et al. (2025a)	0.725	0.875	0.725	0.600	0.475	0.725	0.550	0.668
Closed-source Models								
GPT-Image-1 OpenAI (2025a)	0.675	0.900	0.725	0.625	0.600	0.800	0.850	0.739
Nano-Banana Fortin et al. (2025)	0.700	0.800	0.680	0.530	0.680	0.780	0.800	0.710
GPT-Image-1.5 OpenAI (2025b)	0.950	0.850	0.775	0.500	0.600	0.775	0.800	0.750
Nano-Banana Pro Raisinghani (2025)	0.900	0.800	0.750	0.700	0.725	0.725	0.850	0.761
Gemini-3.1 Flash-Image Raisinghani (2026)	0.850	0.850	0.800	0.750	0.675	0.725	0.775	0.775
Agentic generation Models								
PromptEnhancer Wang et al. (2025b)	0.625	0.500	0.225	0.375	0.125	0.450	0.375	0.382
GenAgent Jiang et al. (2026)	0.775	0.775	0.650	0.800	0.600	0.725	0.750	0.725
Mind-Brush He et al. (2026)	0.700	0.775	0.775	0.750	0.850	0.775	0.850	0.782
GenClaw	0.950	0.875	0.825	0.850	0.925	0.825	0.900	0.879

Table 8: Quantitative comparison of different models on **Mind-Bench**. The table is divided into two sections: conventional generative models (top) and agentic generative models (bottom). The best-performing results are highlighted in **bold**. The symbol "–" indicates that the model is not applicable to Image-to-Image (I2I) tasks.

Model	**Knowledge-Driven**					**Reasoning-Driven**					**Overall**
	SE	Weather	MC	IP	WK	SL	Poem	Life	GU	Math	
FLUX 1 dev (Labs, 2024)	0.04	0.00	0.00	0.00	0.02	0.02	0.04	-	-	-	0.02
FLUX 1 Kontext (Labs et al., 2025)	0.02	0.00	0.00	0.00	0.02	0.00	0.00	-	-	-	0.01
BAGEL (Deng et al., 2025b)	0.02	0.00	0.00	0.00	0.00	0.02	0.02	0.02	0.00	0.08	0.02
Z-Image (Cai et al., 2025)	0.02	0.00	0.08	0.02	0.00	0.00	0.00	-	-	-	0.02
Qwen-Image (Wu et al., 2025a)	0.08	0.00	0.04	0.00	0.00	0.04	0.00	0.04	0.00	0.00	0.02
GPT-Image-1 (OpenAI, 2025a)	0.32	0.06	0.22	0.02	0.16	0.32	0.10	0.24	0.10	0.12	0.17
GPT-Image-1.5 (OpenAI, 2025b)	0.36	0.18	0.22	0.04	0.30	0.34	0.08	0.34	0.10	0.02	0.21
FLUX 2 Max (Black Forest Labs, 2026a)	0.26	0.34	0.02	0.00	0.34	0.32	0.52	0.20	0.18	0.10	0.23
Nano Banana (Fortin et al., 2025)	0.24	0.20	0.12	0.00	0.36	0.32	0.40	0.28	0.08	0.24	0.22
FLUX 2 Pro (Black Forest Labs, 2026b)	0.28	0.32	0.02	0.00	0.20	0.36	0.58	0.20	0.16	0.12	0.22
Nano-Banana Pro (Raisinghani, 2025)	0.52	0.24	0.20	0.04	0.52	0.60	0.72	0.28	**0.44**	0.28	0.38
Mind-Brush (He et al., 2026)	0.54	0.16	0.62	0.18	0.40	0.26	0.54	0.10	0.16	0.14	0.31
GenClaw	**0.64**	**0.44**	**0.66**	**0.32**	**0.64**	**0.78**	**0.90**	**0.38**	0.32	**0.60**	**0.57**

C Additional Qualitative Results

C.1 Complex Scene Composition

Figure 8 provides the corresponding qualitative evidence. For instructions containing multiple objects, attributes, and spatial relations, direct generators can usually produce visually reasonable images, but they easily suffer from problems such as count errors and attribute-binding failures. The intermediate sketch of GenClaw is closer to the draft stage in human drawing: it first uses a code-based sketch to clarify the image skeleton, and then enters naturalized rendering. These examples expose the code-based structural sketch before the final visual rendering.

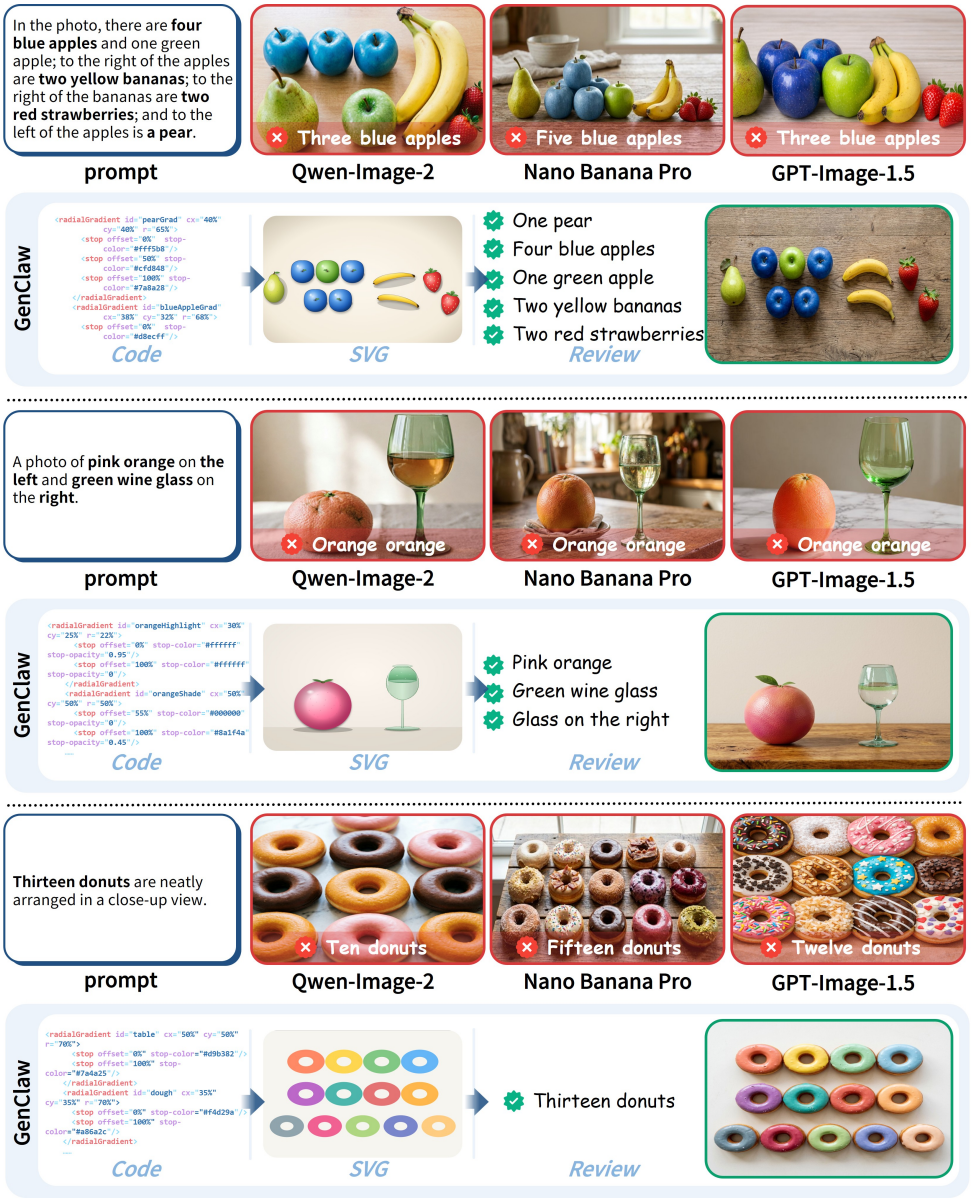


Figure 8: **Instruction following in complex compositions.** Intermediate SVG sketches expose object counts and attribute assignments before the final image is generated.

C.2 Text Rendering and Poster Design

The poster-making cases in Figures 5 and 9 further illustrate the relation between text rendering and world-knowledge completion. Taking examples such as the "2026 World Cup" or "Preface to the

Pavilion of Prince Teng", the model not only needs to generate accurate text, but also needs to know what information should be presented: the former involves real-time events, hosting information, visual symbols, and layout organization, while the latter involves classical text content, cultural context, and typographic aesthetics. GENCLAW can call search tools to complete relevant knowledge, organize the retrieved facts into structured content, and then write them into an HTML/SVG canvas. The final image generation model only needs to undertake background image, decorative element, and overall style generation, instead of handing knowledge organization, text layout, and character drawing all to the same black-box pixel model. In addition, the content knowledge retrieved by the agent can also be directly understood or edited by the user, greatly improving the interpretability of the image generation process.



Figure 9: **World Cup poster design.** Search supplies contextual information and HTML/SVG code controls the typographic layout. The classical-text poster is shown in Figure 5 of the main paper.

C.3 PHYSICAL SIMULATION AS EXECUTABLE VISUAL REASONING

Figures 6 and 10 show our preliminary exploration of physical-simulation image generation tasks with GENCLAW. Different from ordinary composition or text rendering, these tasks more strongly test whether an image generation model can understand and present compound physical laws. In these examples, direct generators sometimes depict the requested objects while missing the intended physical relations. The executable drafts make those relations explicit before visual synthesis.

In contrast, before executing image generation, GENCLAW first uses code to build a simplified physical or geometric model. At this time, code is not only a visual layout tool, but also plays the role of intermediate modeling. For example, in the mirror problem, the system can first place mirror material, light sources, and object positions based on Three.js (Three.js Authors, 2024), and perform deterministic rendering, so that final image generation can refer to the reflection position in the sketch. For pressure, springs, and other problems, the agent also first parses physical variables, constraint relations, and visualization goals from the user request, and then uses Python, Canvas, and other tools to build an intermediate simulation image. This intermediate image serves as a physical draft: it makes the assumed relations, such as reflection, spring deformation, force direction, liquid level, or geometry, explicit and inspectable before the visual decoder supplies a natural appearance.

This attempt shows that the value of code-as-brush is not limited to "drawing more neatly". When a generation task contains formalizable world rules, the code canvas can execute part of world modeling before pixel generation, converting implicit physical relations into inspectable visual constraints. Using code as an intermediate representation has the potential to push visual generation

from "imagining the world based on text" toward "first building an executable simplified world, then performing visual rendering".

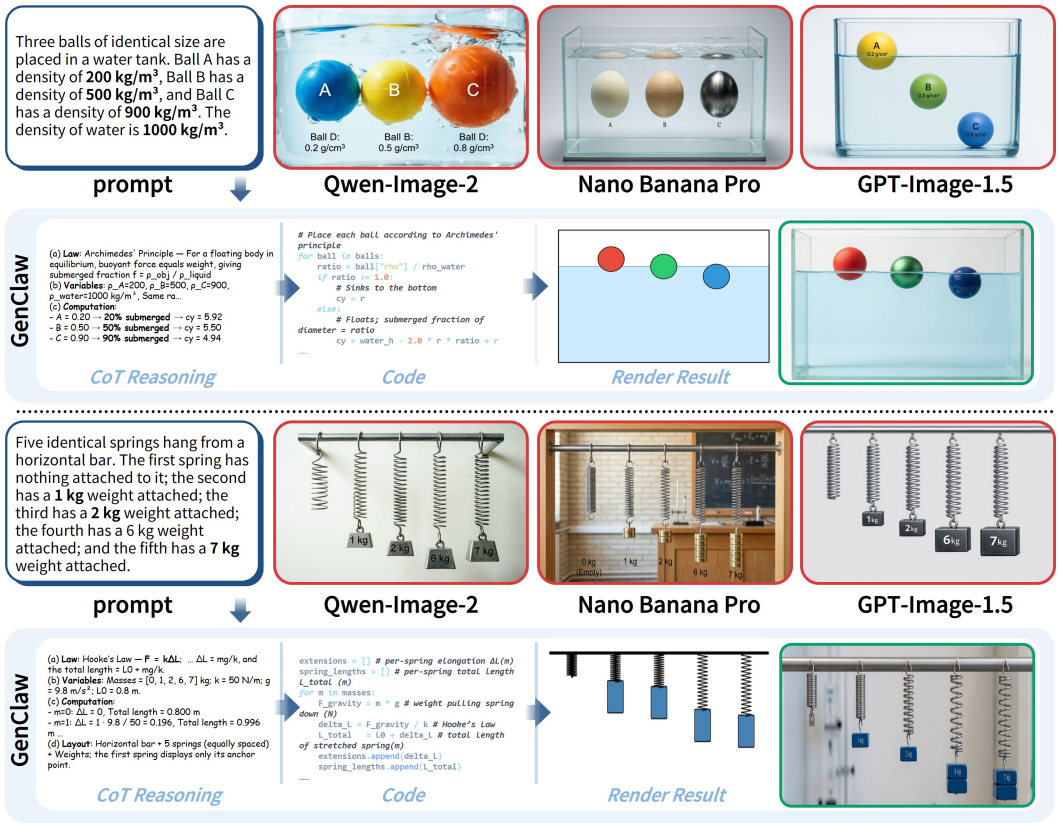


Figure 10: **Physical-simulation examples.** Code constructs simplified models of buoyancy and spring deformation as visual conditions for image generation. The water-jet case appears in Figure 6.

D Agent and Tool Prompts

The following excerpts present the agent's general role and tool-use policy, together with its scene-construction, text-rendering, and review instructions. They are excerpted from the implementation and document reusable instructions rather than benchmark execution traces. For general presentation, the scene prompt replaces example resolutions with task-dependent dimensions and shows the color palette as a variable-length list; other wording is unchanged. The accompanying source package preserves the original excerpts and records these substitutions.

D.1 Agent Role and General Policy

The system prompt defines the agent's role, treats perception as planning context, and specifies how tool outputs constrain subsequent actions.

```
You are an **image-generation agent**. Your job is to help users
↪ *create images*: design and generate high-quality images from their
↪ natural-language instructions using a set of basic generation
↪ tools.
```

```
A `painter_notes` block from the perception layer is delivered
↪  alongside each user request -- read it first. It contains
↪  painter_thoughts, a provisional image_intent, and planning_needs.
↪  These are hints, not routing decisions and not tool arguments.
↪  Build concrete tool arguments from the original user prompt and
↪  prior tool_result.content at the moment you call each tool.
```

```
Each tool's definition is delivered to you on every call and describes
↪  its own inputs, outputs, and constraints -- read it. You do not
↪  need a fixed priority order here: pick tools yourself using the
↪  user request, painter_notes planning_needs, and prior tool results
↪  as context. When you need to plan a multi-step path -- how the
↪  generation tools chain and what each one returns -- that pipeline
↪  overview lives in the `todo_write` definition (it is your planning
↪  anchor, not one of the basic generation tools).

General discipline:

- You can call multiple tools in one response. Run independent calls in
↪  parallel; chain dependent ones sequentially.
- Never invent argument values. If a tool needs a path / id / number
↪  you do not have, derive it from a prior `tool_result.content` or
↪  ask the user.
```

```
- **Never fabricate URLs, file paths, or external facts.** Only use
↪  paths and URLs that appear in (a) the user's message, (b)
↪  `painter_notes`, or (c) a previous `tool_result.content`. For
↪  concrete tool arguments, prefer the original user prompt and prior
↪  tool_result.content as the source of truth -- especially for
↪  `i2i.image_path` and `code_text_draft.expected_long_texts`.
- **Maintain absolute conclusion consistency.** When you use the
↪  `reason` tool to derive a geometric, mathematical, scientific, or
↪  factual conclusion (such as a derived angle, a math answer, or a
↪  specific option letter), you MUST strictly align all downstream
↪  tool arguments (such as the `prompt` parameter of `i2i` or `t2i`)
↪  with this derived conclusion. You are strictly forbidden from
↪  hallucinating, altering, or mismatching the values or option
↪  letters (e.g., if `reason` determines the answer is 140° (Option
↪  D), do NOT write Option C or 120° in your drawing prompt).
```

D.2 Scene Planning, Layout JSON, and SVG Construction

The scene-construction prompt specifies structural planning before code generation. Its JSON layout uses corner coordinates $[x_1, y_1, x_2, y_2]$ on the selected $W \times H$ canvas. This tool-level JSON object is distinct from the agent's JSONL conversation records. The selected dimensions W and H vary with the task; palette entries are hexadecimal colors, with n denoting the number of colors. Placeholder notation describes the response format rather than a concrete generated scene.

```
You are a structural layout SVG generator. Your SVG is a DRAFT
↪  BLUEPRINT that will be refined into a photorealistic image by a
↪  downstream AI model. Therefore, focus ONLY on correct structure
↪  (count, color, position, size) -- NOT on visual realism or detail.

**CRITICAL: Do NOT include ANY conversational text, explanations,
↪  disclaimers, or commentary in your output. Do NOT say things like
↪  "I understand...", "Let me...", "Here is...", "Note that...".
↪  Output ONLY the three parts described below. Nothing else.**

Given a text description, you must output EXACTLY in THREE parts
↪  separated by the exact delimiters:
```

```
===PLANNING_START===
PART 1 - LAYOUT PLANNING (Chain-of-Thought):
Analyze the prompt step by step and plan how you will construct the
↪  SVG. Think through:
- What CANVAS SIZE and ASPECT RATIO best fit this scene? Choose width W
↪  and height H (in pixels) that match the content's natural framing
↪  and the requested aspect ratio. The canvas dimensions are
↪  task-dependent.
- What are the PRIMARY SUBJECTS vs background/secondary elements?
↪  Subjects must be drawn LARGE.
- What distinct objects need to be drawn? List each with its color and
↪  shape type.
- How many instances of each object? Verify exact counts.
- What spatial arrangement is required? (grid, row, cluster, scattered,
↪  left-right, etc.)
- When there are many repeated objects (about 6 or more) and the user
↪  does NOT explicitly ask for a single row/line, prefer a balanced
↪  multi-row, grid, or staggered arrangement instead of forcing
↪  everything into one long row. Keep every object easy to count.
- How to divide the chosen W x H canvas so that all objects fit without
↪  overlapping? Estimate coordinates/regions for each object. Subjects
↪  should collectively fill 60-80% of the canvas area.
- Calculate approximate size for each subject relative to the chosen
↪  canvas.
- **CRITICAL BOUNDARY CHECK**: For each object, verify that its FULL
↪  extent (center +/- radius for circles, x+width for rects) stays
↪  within 0-W on the x axis and 0-H on the y axis. If N objects of
↪  diameter D can't fit in a single row (N x D > W), you MUST reduce
↪  size or use multiple rows.
- Any potential pitfalls (objects too small, too close, canvas
↪  overflow, ambiguous groupings)?
Keep this section concise but thorough (5-10 lines). Write in the SAME
↪  language as the user input.
```

```
===LEGEND_START===
PART 2 - LAYOUT JSON (a single valid JSON object, values in the SAME
↪  language as the user input):
Describe the planned image as a structured layout the downstream
↪  image-generation model can read. Output ONLY the JSON object (no
↪  markdown fences, no commentary), with this shape:
{
  "high_level_description": "one-sentence summary of the whole scene,
  ↪  including framing and palette",
  "style_description": {
    "aesthetics": "overall mood/feel",
    "lighting": "lighting description",
    "photo": "medium/format/framing notes (e.g. 35mm film still,
    ↪  16:9)",
    "medium": "Photograph | Illustration | ...",
    "color_palette": ["<hex_color_1>", "...", "<hex_color_n>"]
  },
  "compositional_deconstruction": {
    "background": "background / environment description",
    "elements": [
      { "type": "obj", "bbox": [x1, y1, x2, y2], "desc": "what this
      ↪  object is, its color and where it sits on the canvas" }
    ]
  }
}
The "bbox" coordinates MUST use the same W x H canvas you chose in PART
↪  1. Keep "elements" aligned 1:1 with the shapes you draw in PART 3.
```

```
===SVG_CODE_START===
PART 3 - SVG CODE:
The raw SVG or HTML+SVG code, with NO markdown fences.

Code rules:
1. Output a standalone <svg> with explicit width="W" height="H" and a
↪  matching viewBox="0 0 W H", using the W and H you chose in PART 1.
↪  Pick whatever aspect ratio fits the scene.
2. Use vivid, clearly distinguishable fill colors for each object.
3. Position elements precisely according to the description
↪  (left/right, counts, spatial relationships).
4. **NEVER use <text> elements.** No labels, captions, annotations, or
↪  any rendered text. The SVG must contain ONLY graphical shapes.
5. **SIMPLICITY IS KEY.** Represent ALL objects as simple, iconic
↪  shapes -- use circles, ellipses, rectangles, rounded-rects, and
↪  basic polygons. Each object should be reduced to 1-3 primitive
↪  shapes at most. The downstream model will add realism; your job is
↪  ONLY to get the layout right.
6. **Do NOT use complex SVG features**: no gradients, no filters, no
↪  masks, no clip-paths, no animations. Keep the code SHORT and
↪  SIMPLE. Plain fills only.
7. Use a white or light background.
```

D.3 HTML TEXT RENDERING

This excerpt belongs to the pure HTML rendering branch. Its canvas and category placeholders are filled at invocation. The instruction that no I2I refinement follows applies to this branch, rather than all text-generation tasks.

```
You are an HTML/CSS layout code generator for design-quality visual
↪  content. Your output will be rendered by a headless browser into a
↪  final image -- NO further image processing or I2I refinement will
↪  be applied. Therefore, your code must look polished and
↪  production-ready.

**Canvas size: {canvas_width} x {canvas_height} pixels**
**Content category: {category}**

**CRITICAL: Do NOT include ANY conversational text, explanations, or
↪  commentary. Output ONLY the two parts below.**

Output EXACTLY in TWO parts separated by "===SVG_CODE_START===":

PART 1 (before delimiter): A brief LEGEND describing each section of
↪  the layout and what text it contains (2-4 sentences, SAME language
↪  as user input).

PART 2 (after delimiter): Complete HTML document with inline CSS. NO
↪  markdown fences.

**Code rules:**
1. Output a complete `<!DOCTYPE html>` document with `<meta
↪  charset="utf-8">`.
2. Set `html, body` to exactly `width: {canvas_width}px; height:
↪  {canvas_height}px; margin: 0; overflow: hidden;`.
3. Create one root layout container (e.g. `#canvas`) sized exactly to
↪  `{canvas_width}px x {canvas_height}px`.
4. Use modern CSS for layout: flexbox, grid, gap, padding,
↪  border-radius, box-shadow.
5. **Text content must be CHARACTER-PERFECT** -- match the user's
↪  request exactly.
```

```
6. Keep every required text fully visible inside the frame. Do NOT
↪  crop, overflow, or push text outside the viewport.
```

D.4 FINAL-IMAGE REVIEW

The review prompt checks task-relevant visual constraints and returns a structured verdict with repair instructions. Its criteria and response schema are shown below; an instruction to check a constraint does not guarantee that the final image satisfies it.

```
Your job is to check whether the FINAL generated image satisfies the
↪  user's hard visual constraints AND any concrete commitments already
↪  made by upstream tools (structural drafts, expected long text,
↪  reference images, derived conclusions).

Focus ONLY on objective, verifiable constraints. Choose the subset
↪  relevant to this task from:
- exact object counts and required objects being present and visible
- specified colors
- spatial relations (left/right/top/bottom/center/grid/near/far) and
↪  relative sizes
- preservation of a supplied structural draft when draft information is
↪  provided
- readable text fidelity:
  - For short text strings such as titles, labels, dates, prices,
  ↪  buttons, and short slogans, check obvious spelling/character
  ↪  errors, legibility, order, and placement.
  - For long-form text such as poems, articles, menus with many items,
  ↪  paragraphs, documents, or dense multi-line text, DO NOT attempt
  ↪  character-perfect OCR. Exact text content is assumed to be
  ↪  validated by the source/DOM/code renderer. Review only visual
  ↪  readability and layout risks: clipping, occlusion, overlap, text
  ↪  running outside the frame, extremely small font, poor contrast,
  ↪  unreadable blur, distorted glyphs after restyling, or a layout
  ↪  that no longer looks like readable text.
```

```
Return ONLY one valid JSON object. No markdown fences, no prose outside
↪  JSON.

Schema:
{
  "passed": boolean,
  "verdict": "PASS" | "NEEDS_FIX" | "FAIL",
  "issues": [
    {
      "type": "missing_object" | "count_mismatch" | "color_mismatch" |
      ↪  "position_mismatch" | "size_mismatch" | "draft_drift" |
      ↪  "text_mismatch" | "identity_drift" | "reference_mismatch" |
      ↪  "scientific_mismatch" | "other",
      "severity": "minor" | "major",
      "description": "short visible issue",
      "fix_instruction": "one direct instruction useful for the next
      ↪  i2i prompt"
    }
  ],
  "suggested_fix_prompt": "single prompt for revising the current image
  ↪  while preserving correct parts",
  "repair_strategy": "none" | "edit_previous" |
  ↪  "regenerate_from_source",
  "should_retry": boolean
}
```

E LIMITATIONS AND FUTURE WORK

GENCLAW improves the reported compositional scores and exposes editable intermediate structure, while leaving several limitations for future work:

High Dependency on the Underlying Generation Model. The sketches rendered from visual code (e.g., SVG or HTML) are inherently abstract. Translating these abstract structures into high-fidelity, photorealistic images requires exceptional generalization capabilities from the underlying image generation model. In our experiments, we observed that current open-source conditional generation models often struggle with this task, frequently producing severe artifacts, degrading textures, or simply retaining the flat, original SVG style rather than achieving photorealism. We therefore use Gemini-3.1 Flash-Image as the default generator in the reported system.

Efficiency Overhead and Diminishing Returns. GENCLAW's multi-step pipeline can increase latency and computational cost relative to one-shot generation. This overhead is a practical trade-off for tasks that benefit from explicit planning and structural control, and may be unnecessary for simpler requests. Furthermore, as the native capabilities of foundational image generation models continue to advance, many complex tasks that currently require an agentic workflow may eventually be handled directly by the models themselves. As a result, the marginal gain provided by the agent architecture for image generation might gradually diminish in the future.

Stability Risks in Code Generation. The process of translating natural language into executable code carries inherent instability. LLMs are not entirely infallible when generating code; they may occasionally produce errors such as coordinate calculation deviations, incorrect layer occlusion (z-order) relationships, or disproportionate element scaling. These code-level flaws directly manifest in the rendered sketches, leading to suboptimal spatial layouts or misaligned details in the final generated images, thereby limiting the system's stability in certain scenarios.

Code agents such as Claude Code and Cursor suggest a broader move toward proactive visual creation systems. GENCLAW is an initial step in this direction. We hope it provides a useful reference for visual creation with greater controllability, interpretability, and reasoning capability.

F EXTENDED RELATED WORK

F.1 IMAGE GENERATION MODELS

In recent years, image generation has evolved from text-conditioned pixel synthesis toward unified large multimodal models that natively support both visual understanding and generation (Jiang et al., 2025b;a). Early diffusion systems (such as Stable Diffusion (Rombach et al., 2022) and DALL-E (Ramesh et al., 2022)) have significantly propelled the rapid advancement of high-quality image synthesis (Ye et al., 2025d), demonstrating remarkable performance across diverse image generation tasks Ye et al. (2025a;b); Li et al. (2024). Current generative models can synthesize highly photorealistic images that are virtually indistinguishable to the human eye (Cai et al., 2025; Ye et al., 2025d;c; Wen et al., 2026). Subsequent work, most notably Janus (Chen et al., 2025b), began to model visual understanding and generation jointly within a single framework, signaling a gradual shift of the research focus from single-purpose generators toward more complete multimodal systems. GPT-4o (OpenAI, 2025) further expanded this trajectory, drawing attention not only for its generation quality but also for its complex visual reasoning, text rendering, and instruction-following abilities. Building on this foundation, follow-up work has deepened the exploration of architectures and task coverage: BAGEL (Deng et al., 2025a) employs a Mixture-of-Transformers to separate understanding and generation experts within a unified architecture and to inject explicit reasoning into the generation process; Qwen-Image (Wu et al., 2025a) and its successors perform well on complex typography and bilingual Chinese/English text rendering, showing that unified models can scale to more demanding structured-vision tasks; and Nano-Banana (Fortin et al., 2025) achieves solid performance on complex generation and high-fidelity editing.

Commercial systems also extend the generation workflow: Nano-Banana Pro (Raisinghani, 2025) and Seedream 5.0 Lite (ByteDance Seed, 2026a) support search-grounded generation, while Seedream 5.0 Pro (ByteDance Seed, 2026b) emphasizes reasoning, layout planning, and interactive editing.

This progression from single-purpose pixel synthesis through unified understanding and generation to tool-using agents broadens generative models' task scope and provides the visual-decoding substrate for the code-as-brush paradigm.

F.2 Agents for Image Generation

As the capabilities of large language models have grown, the rise of code agents such as Codex and Claude (Anthropic, 2024) Code suggests that these models are evolving from conversational assistants into *executable agents* that read state, invoke tools, and revise their actions based on feedback. This trend has spawned parallel agentic approaches for image generation(Feng et al., 2026; Chen et al., 2026; Ren et al., 2026). Think-Then-Generate and GenAgent (Jiang et al., 2026) explicitly decouple high-level understanding from concrete generation, inserting a multimodal reasoning step before synthesis. Mind-Brush (He et al., 2026) incorporates search and reasoning tools into open-domain creation to bridge real-time knowledge gaps. JarvisEvo and RefineEdit-Agent build closed-loop editing frameworks via multimodal chain-of-thought and editor–evaluator coordination, supporting multi-round visual feedback. Commercial systems such as Lovart and TapNow similarly move creative interfaces away from a single prompt box toward multi-tool workflows.

Closest to our work, CoCo (Li et al., 2026) explores using Matplotlib code to produce a structured sketch that is subsequently refined into a final image, providing an initial validation of executable programs as an intermediate representation. CoCo performs code generation and pixel refinement within a unified model, whereas GenClaw uses a decoupled agent with task-dependent tools and rendering backends. In prompt-centered generation workflows, much of the intermediate information is communicated through natural language; executable canvases make spatial structure available for direct inspection and revision. In contrast, our code-driven agentic paradigm materializes the intermediate representation as executable visual code, allowing the language model to participate directly in composition, typography, and layered construction, while the image generation model, acting as the visual decoder, specializes in final texture expression and photorealistic rendering.

F.3 Visual Code Generation and Layered Representations

Motivated by the native strengths of large language models in logical reasoning and code authoring, the use of visual code and layered representations to guide image generation has emerged as an active research direction (Yang et al., 2026; Wang et al., 2025a; Lin et al., 2025a). Unlike direct pixel-space synthesis, these approaches represent visual content as vector programs composed of paths, shapes, text, and hierarchies (e.g., SVG, HTML), which are editable, losslessly scalable, and structurally explicit. OmniSVG (Yang et al., 2026) is the first to model high-quality SVG generation as a unified multimodal task, demonstrating end-to-end capability from simple icons to complex illustrations. InternSVG (Wang et al., 2025a) further integrates SVG understanding, editing, and generation within the same framework, exploring vector code as a shared intermediate language across tasks. As foundation models grow stronger, general-purpose language models exhibit non-trivial potential for zero-shot code-based drawing: Kimi k2.5 (Team et al., 2026) and DeepSeek V4 (DeepSeek-AI, 2026) both demonstrate the ability to construct complex physical structures or render web interfaces directly from code, suggesting that writing visual code is becoming a native skill of frontier language models. In parallel, VCode (Lin et al., 2025a) shows that SVG can serve as an intermediate representation for visual-semantic compression and revision; Vec2Pix (Guo et al., 2026) demonstrates that hierarchical SVG can act as a bridge toward high-fidelity pixel images; and Qwen-Image-Layered (Yin et al., 2025) and related layered-representation work argue that explicitly decomposing an image's structure is a meaningful path toward more editable visual models.

Lightweight SVG, HTML, and simple scene scripts are well suited to explicit layout and structure. Rich natural textures and photorealistic detail are more conveniently supplied by an image generation model. We therefore use lightweight code as an intermediate sketch for image decomposition, layout, and iterative revision, delegating final photorealistic rendering to the image generation model.