JarvisHub: An Open Harness for Canvas-Native Multimodal Creative Agents

JarvisX Team

Abstract

Creative AI is moving from single-step asset generation toward long-horizon multimodal production. Although recent generative models can synthesize high-quality images, videos, audio clips, UI elements, storyboards, slides, and other creative assets, real-world creative work requires more than isolated prompt-output interactions. It involves references, drafts, alternatives, edits, failed attempts, version relations, tool actions, evaluation signals, and human feedback, which together form an evolving project state. Existing prompt-based, chat-based, and node-based generation systems only partially support this state, as they often discard intermediate context, rely on linear conversations, or require manually specified workflows. Recent commercial systems indicate a shift toward agent-assisted creative production, but their closed architectures make it difficult to study how agents represent context, choose tools, revise artifacts, recover from failures, and maintain consistency over time. To address this gap, we introduce JarvisHub, a canvas-native creative agent harness for long-horizon multimodal creation. JarvisHub treats an editable canvas as the user workspace, the agent's external memory, action space, and shared project state, representing multimodal artifacts, dependencies, versions, and feedback as typed canvas nodes and links. Through a three-layer architecture of canvas state, protocol bridge, and agent runtime, JarvisHub enables agents to act within an inspectable and editable creative state. This design moves creative agents beyond isolated tool use toward sustained, human-steerable creative automation, where agents can progressively plan, generate, revise, and organize multimodal projects while users remain able to inspect, guide, and intervene throughout the process.

Project Page: https://www.jarvishub.site/

GitHub Repo: LYL1015/JarvisHub

1 Introduction

Recent advances in multimodal generation have made high-quality images, videos, audio clips, UI elements, and other creative assets substantially easier to produce [7, 8, 10, 12, 14, 17, 23, 25, 33, 39, 41, 42]. These models are increasingly used in visual communication, UI/UX design, storyboarding, video production, slide-deck creation, and marketing content generation. Practical creative work, however, rarely follows a single prompt-output interaction. Creators typically collect references, specify styles or characters, plan layouts or shots, generate multiple candidates, revise local details, compare alternatives, incorporate feedback, and assemble intermediate results into a final deliverable. These intermediate materials—including prompts, reference images, drafts, candidates, edits, failed attempts, versions, and feedback—are not incidental by-products of creation. They form the evolving state of a creative project and provide the context needed for subsequent planning, revision, and evaluation.

This project-state view creates a concrete challenge for creative agents. They must work with an evolving workspace rather than an isolated prompt. To make a useful next move, an agent needs to know which materials already exist, how they are related, which candidates were accepted or rejected, and which parts of the project should be updated next. This context is hard to store in a plain conversation because it includes

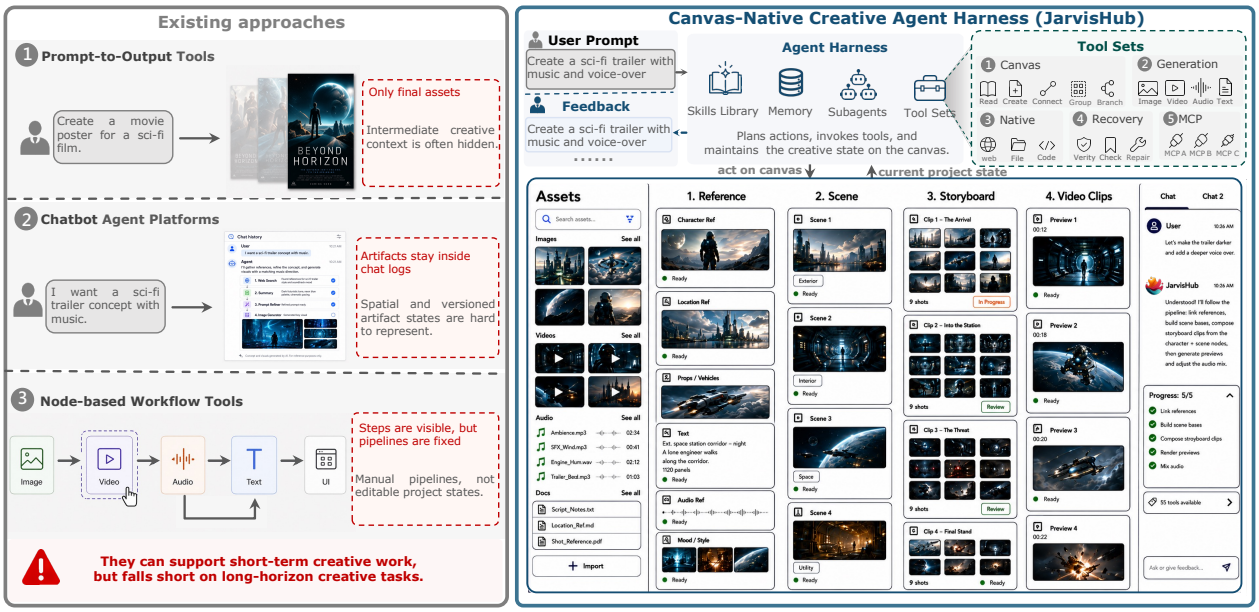

Figure 1 Comparison of existing creative systems with JarvisHub. Prompt-to-output tools, chatbot-based agents, and manual workflow systems each support parts of creative production, but they do not keep a unified project state for long-horizon work. JarvisHub uses the canvas as a shared workspace that agents can inspect and update, so multimodal assets, dependencies, edits, feedback, and revision traces remain accessible throughout the creative process.

spatial layouts, version branches, references, feedback, and unfinished artifacts. The key problem is therefore not simply how to call stronger models, but how to give agents a workspace they can read, update, and keep consistent across an extended workflow.

As shown in Figure 1, existing systems only partially address this requirement. Prompt-to-output tools [2, 3, 6, 9, 13, 19, 29, 34, 36–38, 43, 45, 46, 50–52] are effective for producing individual assets, but they typically hide or discard intermediate decisions, failed trials, alternative candidates, and revision history. Chat-based creative agents [4, 5, 11, 16, 27, 28, 33] can follow multi-step instructions and call tools, but their primary context remains a linear conversation, which makes it difficult to represent spatial layouts, asset relationships, version branches, and local editing targets. Node-based workflow tools [18, 20, 21, 31, 44, 47–49] make execution steps more visible, but they are usually organized around manually specified pipelines rather than continuously editable project states that an agent can inspect, revise, extend, and verify. Related visual and graph-structured creative systems further explore prompt chains, branching narrative graphs, and dynamic storyboards [20, 21, 24, 35, 44], but they remain tied to particular creative domains or manually edited graph structures rather than a general runtime harness for canvas-native agents. As a result, current systems may support useful creative actions, but they do not provide a complete runtime framework for long-horizon creative agents.

Recent commercial creative products show the same shift. Claude Design [1] supports conversational design generation and iterative refinement. Google Stitch [22] connects prompt-based UI generation with multi-page prototypes and front-end code. TapNow [40] and LibTV [26] organize scripts, characters, storyboards, images, videos, and audio for narrative content creation. MiniMax Hub [30] coordinates image, video, audio, and text generation tools in multimodal workflows. These systems suggest that creative AI is moving from isolated asset generation toward workflows where AI helps plan, revise, and organize projects across multiple stages.

However, these products remain largely closed. Researchers can observe the user-facing features, but they cannot inspect how project state is represented, how actions are checked, how tools are scheduled, how feedback is used, or how failures are repaired. This makes it difficult to study how creative agents maintain context over long workflows. The missing piece is therefore not another creative generation product, but an open harness for studying how agents operate over evolving multimodal projects.

Table 1 Core capabilities supported by the JarvisHub harness.

Capability	What it enables
Persistent project state	Keeps prompts, references, drafts, versions, outputs, and feedback in an editable canvas rather than in transient chat history.
Controlled canvas action	Lets the agent read and modify the canvas only through checked operations allowed in the current turn.
Tool and media execution	Connects the canvas to generation, native execution, and external tools that return inspectable artifacts.
Feedback-guided revision	Uses human or model feedback to continue from accepted results, repair failed nodes locally, ask for clarification, or stop.
Traceability and recovery	Records requests, actions, observations, feedback, and checkpoints so the creative process can be inspected and restored.

To address this gap, we introduce JarvisHub, a canvas-native creative agent harness for long-horizon multimodal creation. The core idea is that the canvas is not only a visual interface for users, but also the shared project state that agents can read and update. In JarvisHub, prompts, references, images, videos, audio clips, UI components, storyboards, candidate results, version relations, edits, and user feedback are represented as editable canvas nodes and links. The agent observes the current canvas, interprets the available materials and their relationships, calls appropriate tools, creates or modifies nodes, and writes the results back to the same canvas. In this way, the creative process becomes visible and traceable rather than being hidden inside a sequence of prompts and tool calls.

JarvisHub has three key merits. First, the canvas state layer stores project materials such as nodes, attributes, layouts, versions, and dependency links. Second, the protocol bridge checks how agents read from and write to the canvas, including permissions, operation formats, validation, and logging. Third, the agent runtime chooses granted actions, calls tools, synchronizes state, and records trajectories. It exposes tool families for canvas editing, media generation, native execution, recovery, and Model Context Protocol (MCP)-backed extension. Skills, memory, and sub-agents then help organize longer workflows. Together, these layers let agents work on a shared project while preserving what they did, what evidence they used, and why the canvas changed.

Our contributions can be summarized as follows:

- We formalize long-horizon multimodal creation as an agent process over an editable project graph.
- We propose and implement a canvas-native creative agent harness that unifies project state, controlled runtime execution, and trajectory recording.
- We demonstrate JarvisHub on high-value long-horizon cases spanning narrative media generation, interactive web development, and presentation deck generation.

2 Method

2.1 Overview

JarvisHub is a canvas-native agent harness for long-horizon multimodal creation. It treats the canvas, not the chat history, as the main project memory. The canvas stores artifacts, dependencies, revisions, feedback, and intermediate results so that both users and agents can return to them later. As shown in Figure 2, each turn follows a simple loop: the runtime observes the canvas, interprets the user input, selects a permitted action through the protocol bridge, invokes the required capability, and writes the returned observation back to the canvas. Table 1 summarizes the core capabilities supported by JarvisHub. The following subsections then detail how these capabilities are realized by the canvas state, the protocol bridge, the agent runtime, and the trajectory record.

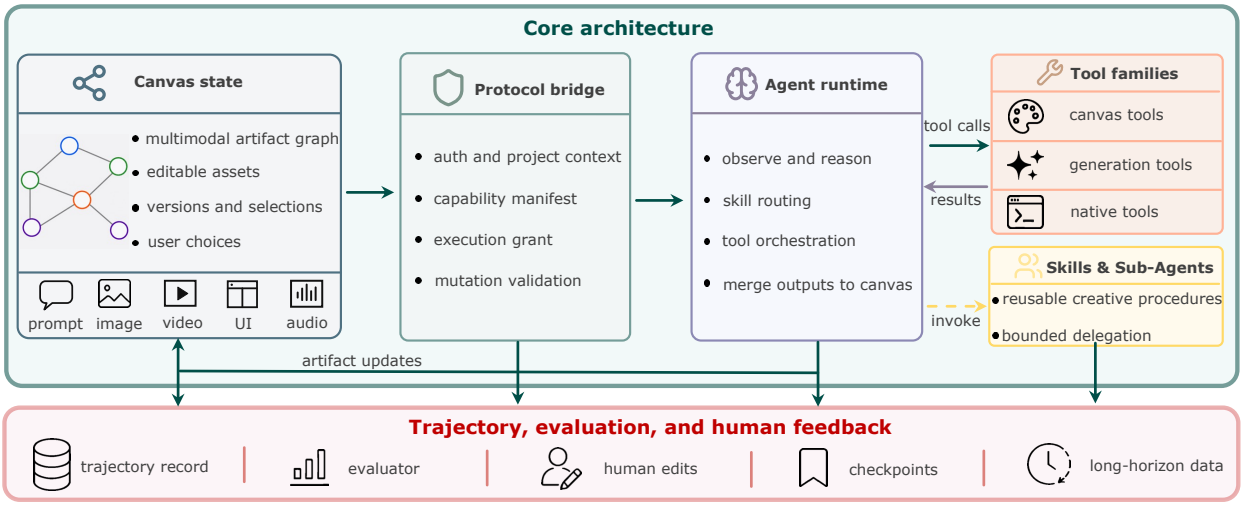

Figure 2 Overview of the JarvisHub architecture. JarvisHub organizes long-horizon creative work as a canvas-centered execution loop built on three core components: (1) a canvas state layer that stores multimodal artifacts, dependencies, versions, and user choices; (2) a protocol bridge that validates granted actions and controls canvas reads and writes; and (3) an agent runtime that plans, invokes tools and skills, and updates canvas artifacts. The harness further records trajectories, evaluator signals, human edits, and checkpoints as evidence for feedback, recovery, and future analysis.

2.2 Canvas-Native Project State

We first define the canvas state. It is the shared project object that users edit and agents read or update. At turn t, we represent a creative project as:

$$\mathcal{C}_t = (\mathcal{G}_t, \mathbf{X}_t, \mathbf{M}_t, \mathbf{U}_t, \mathbf{L}_t), \quad \mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t). \tag{1}$$

Here, $\mathcal{G}_t$ is a typed artifact graph. $\mathcal{V}_t$ denotes the set of canvas nodes, and $\mathcal{E}_t \subseteq \mathcal{V}_t \times \mathcal{R} \times \mathcal{V}_t$ denotes directed typed relations. Each edge can be written as (v_i, r, v_j), where $r \in \mathcal{R}$ indicates a relation such as reference use, version lineage, generation dependency, grouping, or workflow continuation. The other terms store node contents and interaction records: $\mathbf{X}_t$ stores editable contents and artifact payloads, $\mathbf{M}_t$ stores provenance, execution metadata, and runtime status, $\mathbf{U}_t$ records user selections, edits, and feedback, and $\mathbf{L}_t$ stores spatial positions and group layouts. This separation lets the agent see what an artifact contains, how it was produced, how users interacted with it, and where it appears on the canvas.

We instantiate the graph through typed and addressable nodes. Let $\mathcal{V}_t = \{v_i\}_{i=1}^{N_t}$ contain N_t canvas nodes. Each node is represented as:

$$v_i = (\mathrm{id}_i, k_i, \mathbf{p}_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{m}_i, s_i), \quad k_i \in \mathcal{K}_{\mathrm{node}}, \tag{2}$$

where id_i is a stable identifier, k_i is the node kind, $\mathbf{p}_i$ stores position and local layout, $\mathbf{x}_i$ stores editable inputs such as prompts or configuration fields, $\mathbf{y}_i$ stores generated outputs or artifact handles, $\mathbf{m}_i$ stores provenance and execution diagnostics, and s_i denotes runtime status. User interaction records in $\mathbf{U}_t$ point back to the nodes, edges, or canvas regions they affect. In JarvisHub, the node-kind space $\mathcal{K}_{\mathrm{node}}$ covers textual, visual, audiovisual, and workflow-boundary artifacts specified by the canvas schema and capability manifest.

This representation provides three key properties for long-horizon creation. First, artifacts are addressable: the agent can refer to a specific candidate image, video clip, webpage render, or slide rather than relying on an ambiguous conversational phrase. Second, artifacts are reusable: a reference, draft, rejected candidate, or intermediate result can later serve as input to another generation or editing step. Third, dependencies are inspectable: users and agents can trace which materials influenced a result, which version was selected, and which downstream artifacts depend on it. In addition, because runtime status is part of the canvas state, later steps can distinguish completed or selected artifacts from planned, running, failed, or otherwise unverified results.

Table 2 Core tool families used by the agent runtime.

Tool family	Runtime role	Representative operations
Canvas tools	Update project state.	Read, create, update, connect, group, select, and branch nodes; maintain artifact handles and runtime status.
Generation tools	Produce canvas artifacts.	Image, video, audio, and composed-media generation.
Native tools	Use external execution.	Browser, file, code, search, document, and presentation operations that return inspectable artifacts.
Recovery tools	Inspect and repair.	Structured feedback, verification, checkpointing, and local repair.
MCP tools	Extend external services.	MCP-provided capabilities under the same manifest-and-grant contract.

2.3 Protocol-Constrained Canvas Interaction

Long-horizon creation only works if canvas updates are explicit, valid, and recoverable. JarvisHub enforces this through a protocol bridge between the agent and the canvas. At turn t, the agent receives a user query q_t and observes the current canvas state $\mathcal{C}_t$. The bridge provides a capability manifest Γ_t, which lists the node types, mutation operations, tools, and artifact handles available in the current project. It then derives an execution grant Ω_t that limits what the agent may do for the current request. Conditioned on $(q_t, \mathcal{C}_t, \Gamma_t, \Omega_t)$, the agent proposes an action a_t. The harness executes this action only if it can be encoded as a checked tool call, canvas mutation, evaluation request, clarification request, or user-facing response.

Executing a valid action produces a tool or environment observation o_t. The turn may also receive a feedback signal f_t from a human user, a model-based critic, an auxiliary agent, or a structured evaluator. This feedback may induce a repair or follow-up decision r_t. The canvas then evolves as:

$$\mathcal{C}_{t+1} = \mathcal{F}(\mathcal{C}_t, a_t, o_t, f_t, r_t), \tag{3}$$

where $\mathcal{F}$ denotes the state-transition operator implemented by the bridge. It writes the accepted action and returned evidence to the current canvas, for example by creating nodes, updating fields, connecting dependencies, attaching artifacts, recording failures, creating checkpoints, or requesting human correction. Thus, canvas interaction becomes auditable rather than hidden inside language generation: if the agent modifies the canvas, the trajectory records the mutation; if it uses a generated artifact, the trajectory records both the artifact handle and the observation that produced it.

2.4 Agent Runtime for Creative Orchestration

As illustrated in Figure 3, the agent runtime decides what the agent can do in the current turn and turns that decision into a valid canvas update. Given $(q_t, \mathcal{C}_t, \Gamma_t, \Omega_t)$, it first interprets the user request against the current canvas. It then filters the available operations by the execution grant, invokes the required capability, and returns the resulting evidence through the protocol bridge. This gives each turn a granted action space:

$$\mathcal{A}_t = \mathcal{A}(\Omega_t, \Gamma_t, \mathcal{C}_t, q_t), \quad a_t \in \mathcal{A}_t. \tag{4}$$

Here, $\mathcal{A}_t$ denotes the actions that are both relevant to the request and permitted in the current turn. The formula is not a list of implementation tools. Instead, it states the runtime contract: an accepted action must be grounded in the observed canvas, exposed by the capability manifest, allowed by Ω_t, and committed through the protocol bridge. This keeps the shared canvas as the project state, rather than letting the runtime maintain a separate hidden state.

To implement this contract, the runtime groups available capabilities into a small set of tool families. Each family can be granted for a turn, invoked by the agent, and committed back to the canvas through the same bridge. Table 2 summarizes these families.

The tool families define what can be invoked. The runtime also uses three higher-level supports to decide how those capabilities should be sequenced across long creative tasks:

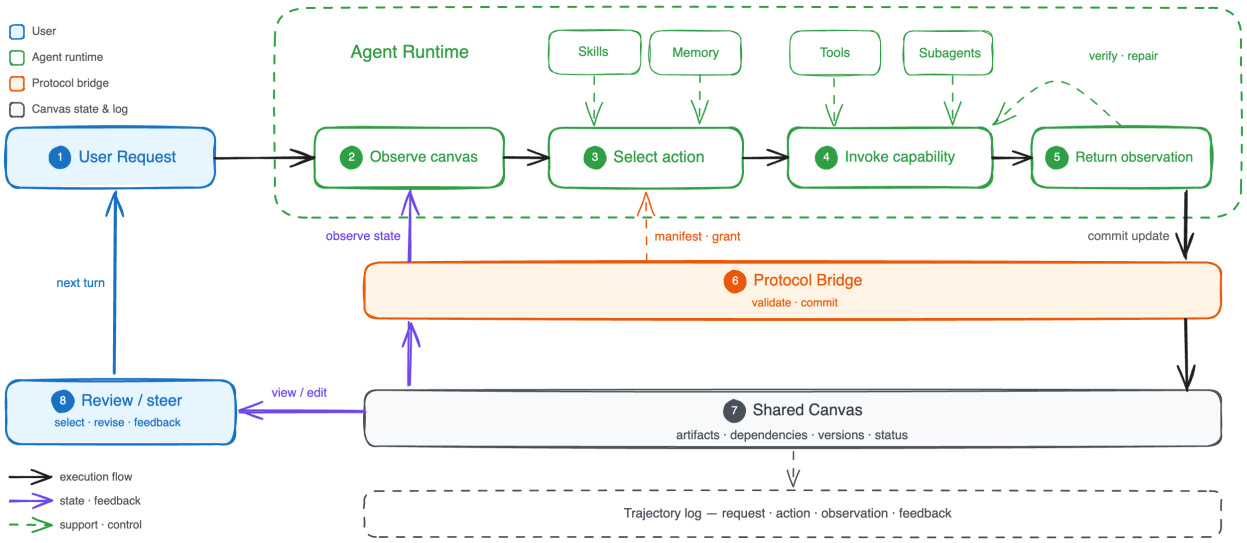

Figure 3 Agent runtime loop. The loop converts a user request into a protocol-checked canvas update. The runtime observes the shared canvas, selects an action under the current manifest and execution grant, invokes the corresponding capability with support from skills, memory, tools, and subagents, and returns the resulting observation to the protocol bridge. The bridge validates and commits the update to the shared canvas, where the user can inspect the result, provide feedback, and steer the next turn.

- **Skills.** Skills encode reusable creative procedures, such as storyboarding, reference-guided generation, design-to-web reconstruction, video prompting, or deck construction. They give the runtime a task structure without hard-coding every step into the bridge.
- **Memory.** Memory preserves user preferences, prior decisions, and procedural knowledge across turns. It helps the runtime choose actions that remain consistent with earlier project context.
- **Subagents.** Subagents handle independent subtasks when a workflow can branch. For example, separate video shots can be explored in parallel, while the parent agent still selects useful results and integrates them back into the shared canvas graph.

Together, the tool families provide executable capabilities, while skills, memory, and subagents help choose and sequence them. The resulting workflow can explore alternatives, but its state remains anchored in the canvas, execution grants, and protocol-checked updates.

2.5 Feedback and Traceability

Long-horizon creative work requires feedback before the final artifact is complete. A feedback signal f_t may come from a user, an evaluator, or a subagent critic. Users can select candidates, reject versions, revise prompts, adjust styles, or identify local defects, while JarvisHub can automatically inspect node outputs for consistency, missing evidence, visual quality, and task-specific constraints. The purpose of feedback is not merely to judge an intermediate result, but to determine the next state transition: whether the agent should continue from an accepted node, repair a failed node locally, ask the user for clarification, or stop when the available evidence is insufficient.

To make feedback actionable and traceable, the harness records each feedback signal together with the state and execution context in which it occurs. We formally define the trajectory as:

$$\tau = \{(q_t, \mathcal{C}_t, \Gamma_t, \Omega_t, a_t, o_t, f_t, r_t, \mathcal{C}_{t+1})\}_{t=1}^{T}, \tag{5}$$

where, q_t is the user request, $\mathcal{C}_t$ is the canvas state before execution, Γ_t specifies the available capabilities, Ω_t constrains the actions permitted in the current turn, a_t is the selected agent action, o_t is the tool observation

Table 3 High-value long-horizon creative tasks used in our experiments.

Task	Definition and challenge	Representative outputs
Narrative media generation	Convert a story, script, or scene brief into a temporally coherent visual or audiovisual sequence, testing narrative planning, identity preservation, style consistency, and cross-shot continuity.	Character references, scene designs, storyboard panels, shot plans, generated image sequences, video clips, and animatics.
Interactive web development	Convert a design goal, information structure, or interaction requirement into a rendered web artifact, testing coordination among layout design, interaction logic, frontend code, preview inspection, and iterative revision.	Static pages, dynamic websites, landing pages, web interfaces, interactive prototypes, rendered previews, and frontend implementati ons.
Presentation deck generation	Convert a topic, source document, report, or communication goal into a coherent multi-slide presentation, testing content selection, narrative organization, slide layout, visual synthesis, and cross-slide consistency.	Presentation decks, academic talks, project reports, pitch decks, interview presentations, visual summaries, and explanatory diagrams.

or generated evidence, f_t is the feedback signal, r_t is the repair or follow-up decision induced by the feedback, and $\mathcal{C}_{t+1}$ is the updated canvas state.

3 Experiments

We evaluate JarvisHub on representative long-horizon creative tasks to demonstrate its capability in realistic creative workflows. Specifically, the experiments examine whether a canvas-native harness can support high-value tasks that require persistent project context, iterative artifact generation, and feedback-driven refinement.

3.1 Experimental Setup

All tasks are run in the same JarvisHub environment. We use GPT-5.5 [32] as the main agent backend, GPT Image 2 [33] as the image-generation backend, Seedance 2.0 [41] as the video-generation backend, and Gemini 3.1 Pro [15] as the multimodal evaluation backend. More details about the model-backend configuration are provided in our open-source GitHub repository: `https://github.com/LYL1015/JarvisHub`.

3.2 Long-Horizon Creative Tasks

As shown in Table 3, we evaluate JarvisHub on three representative long-horizon creative tasks: narrative media generation, interactive web development, and presentation deck generation. These tasks cover common creative workflows where an open-ended goal must be developed into a coherent deliverable. They require planning, reference organization, intermediate artifact generation, feedback, and local revision.

3.3 Qualitative Results

For each task, we report two complementary views: a workspace trace of the canvas-centered production process and a generated artifact of the final deliverable. These paired views show how plans, references, assets, dependencies, and agent progress remain inspectable throughout long-horizon work, and how accumulated canvas state supports coherent outputs.

Narrative media generation. As shown in Figures 4 and 5, the narrative media case turns a short-drama prompt into a coherent visual sequence. The workspace trace makes story planning, visual references, dependencies, and generation progress inspectable. The output illustrates how the canvas preserves narrative and visual continuity across shots.

Task: Generate a short drama about a “cowboy robot zombie scavenger.”

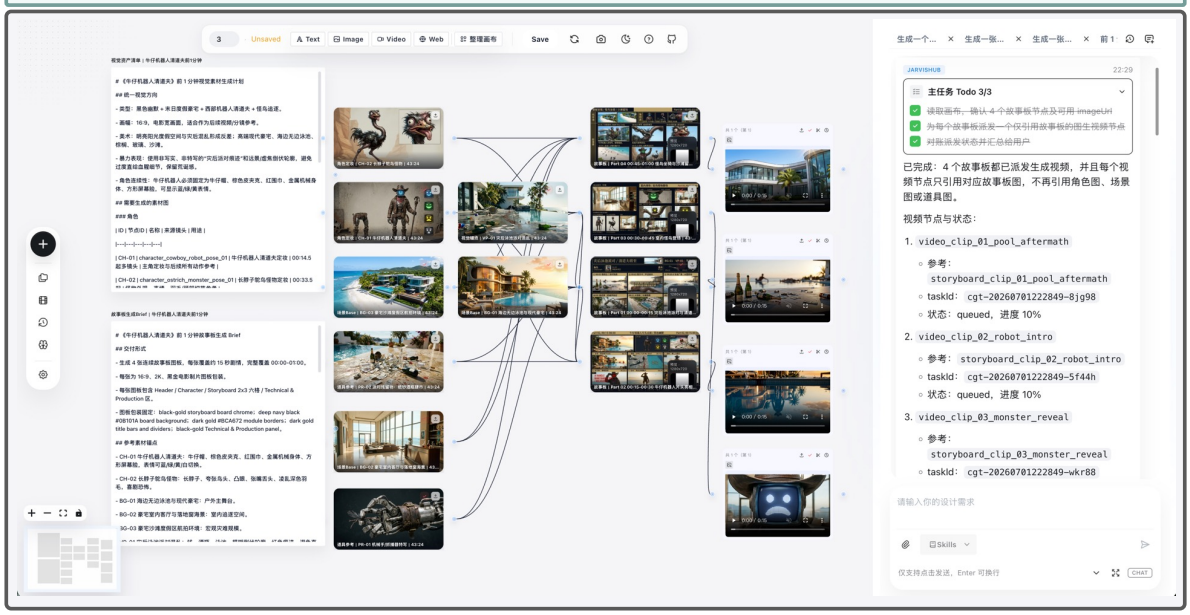

Figure 4 Workspace trace for narrative media generation. The JarvisHub canvas externalizes the task brief, planning notes, visual references, shot candidates, dependency links, and agent progress for a short-drama case.

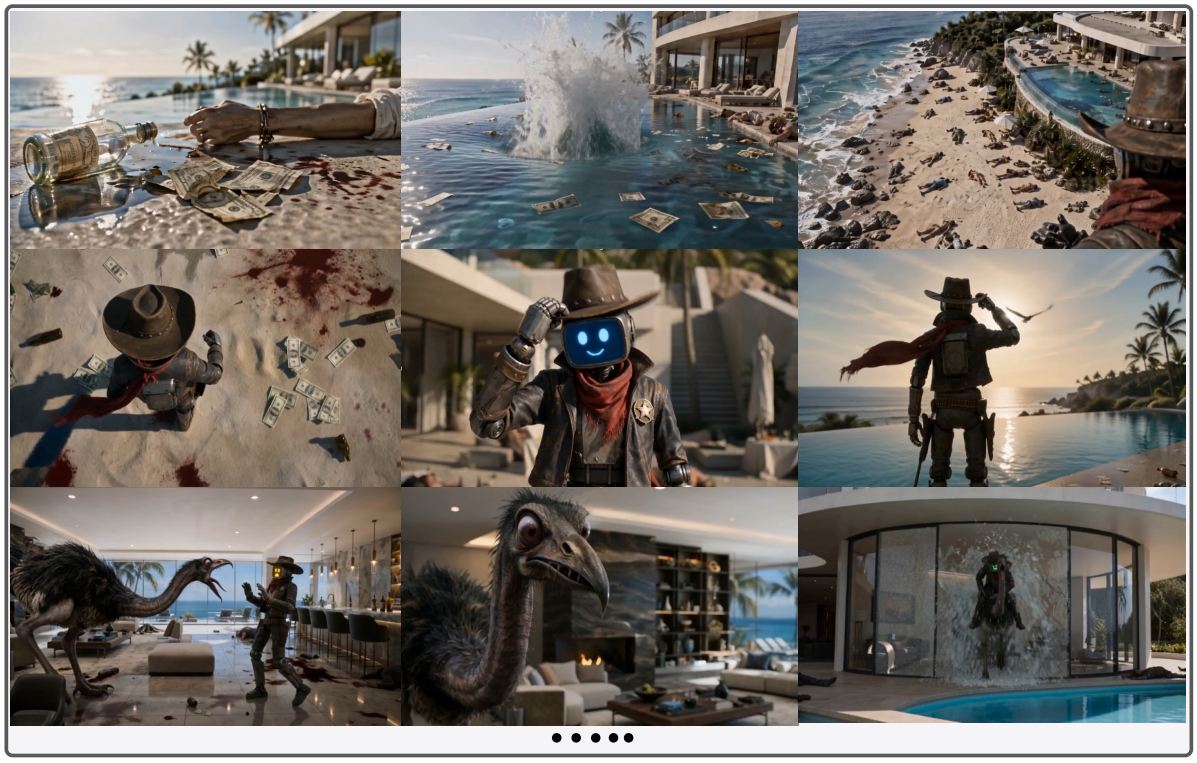

Figure 5 Generated artifact for narrative media generation. The final output presents key frames produced through the canvas-managed workflow, with recurring characters, setting cues, and action continuity across shots. This case is inspired by **Mx-Shell**'s original work *Zombie Sweeper*.

Interactive web development. As shown in Figures 6 and 7, the web development case turns an aesthetic and interaction brief into a rendered photography website. The workspace trace makes design references,

implementation progress, previews, and revision state inspectable. The output illustrates how the canvas preserves visual direction and interface consistency during iterative web construction.

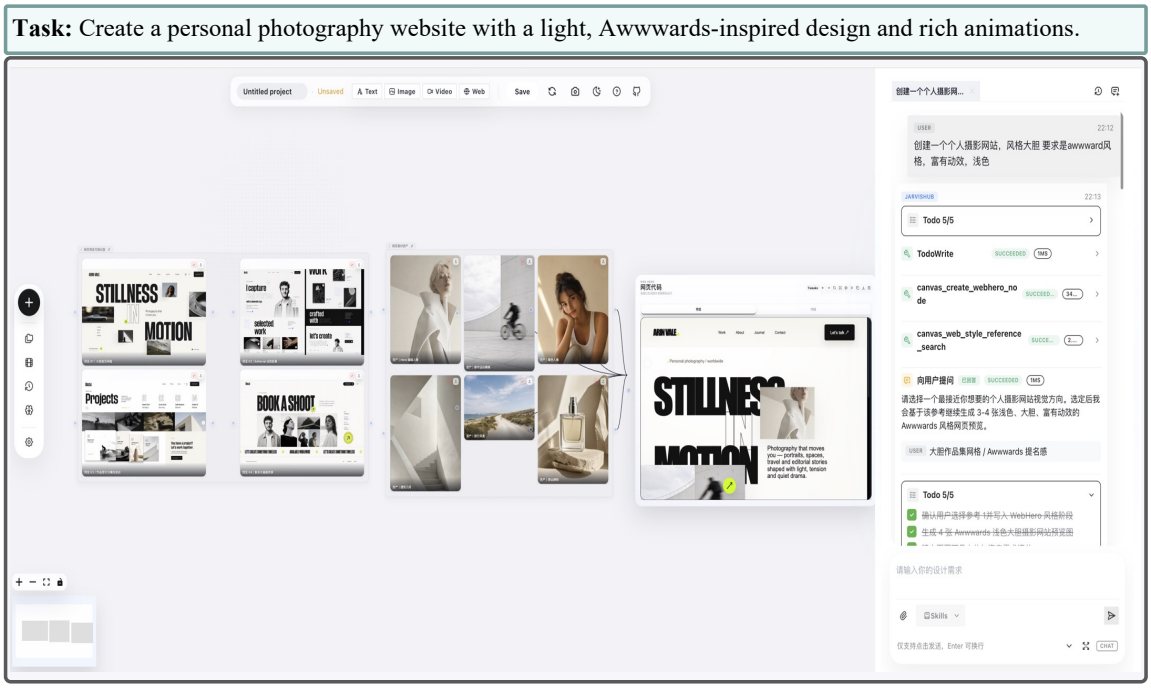

Figure 6 Workspace trace for interactive web development. The JarvisHub canvas tracks the web brief, visual references, layout drafts, implementation artifacts, previews, and revision state during website construction.

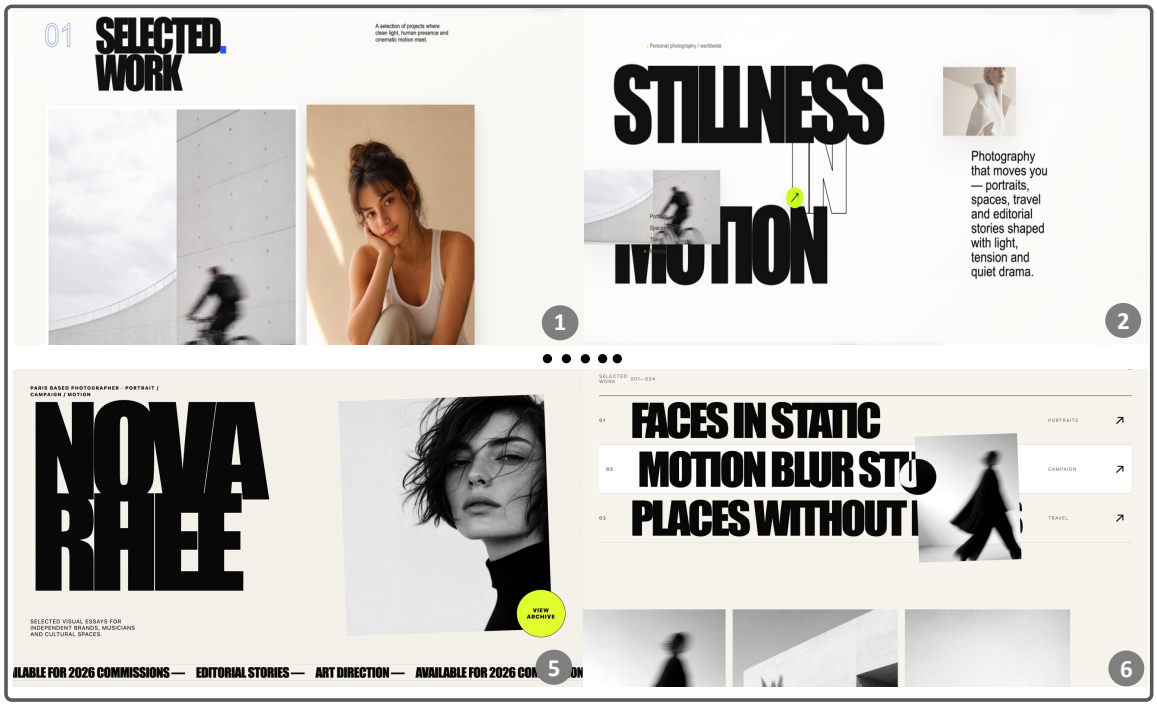

Figure 7 Generated artifact for interactive web development. The final output shows website screens with consistent typography, image placement, page structure, and visual direction.

Presentation deck generation. As shown in Figures 8 and 9, the presentation case turns a machine-learning lecture topic into a structured slide deck. The workspace trace makes content planning, visual assembly, previews, and task progress inspectable. The output illustrates how the canvas preserves topic structure and visual style across a multi-slide deliverable.

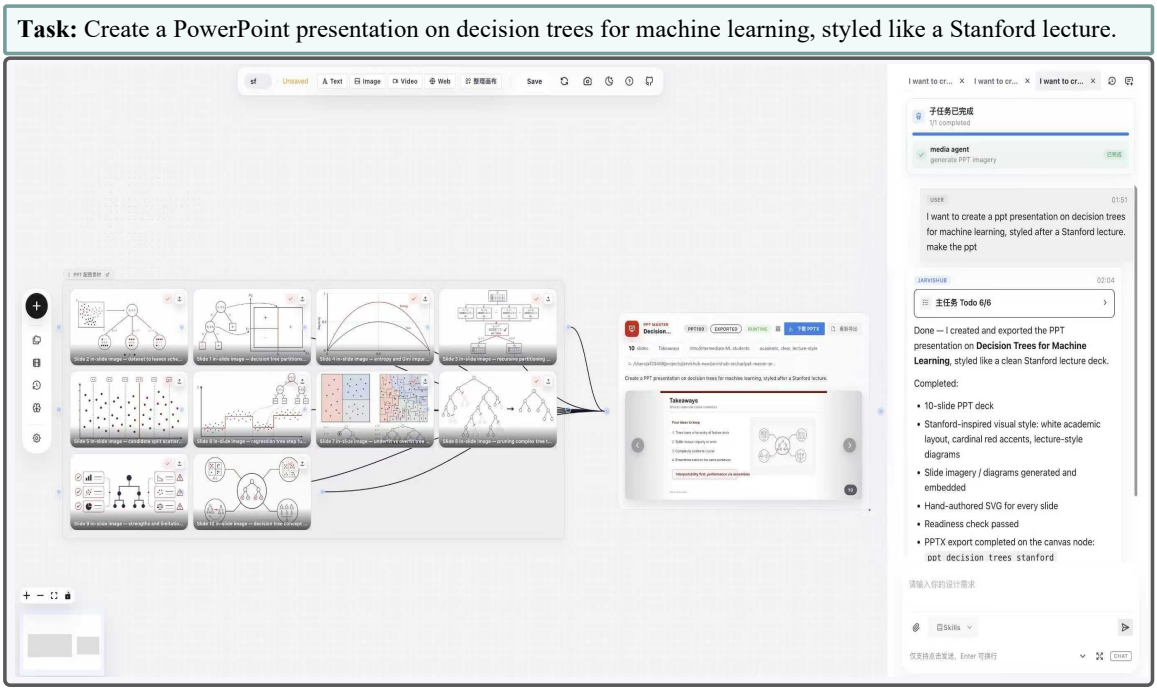

Figure 8 Workspace trace for presentation deck generation. The JarvisHub canvas records lecture content, generated diagrams, slide drafts, dependency links, PowerPoint previews, and revision state.

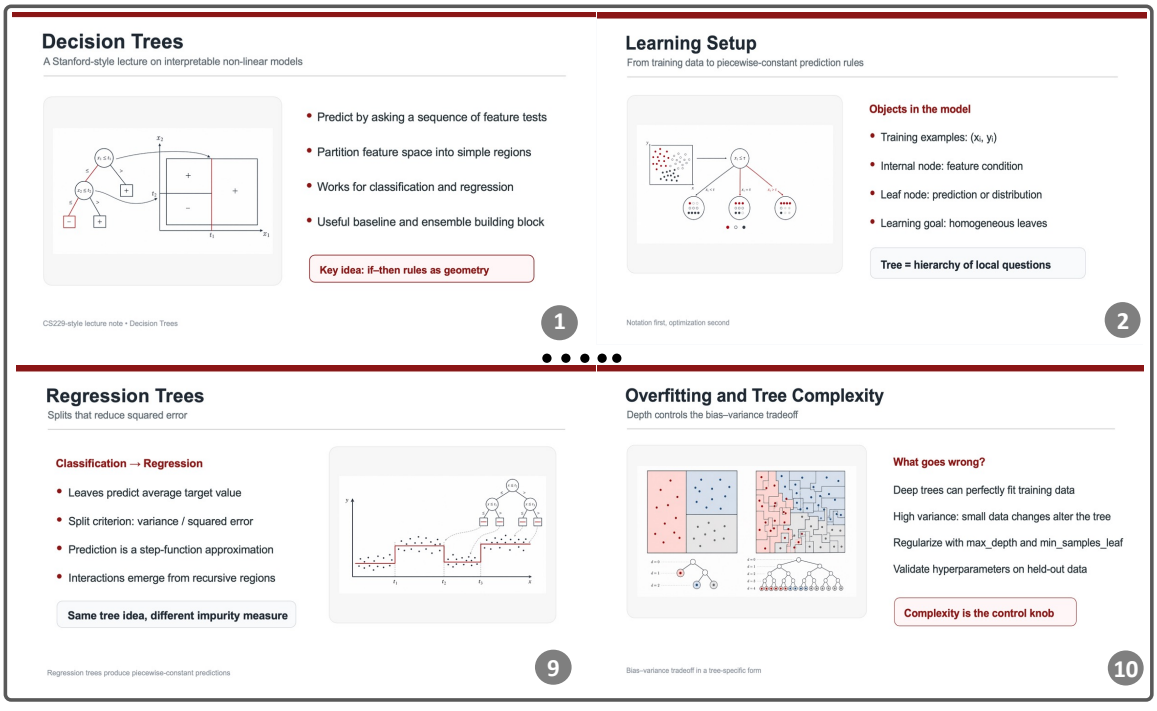

Figure 9 Generated artifact for presentation deck generation. The final output shows lecture slides with consistent layout, diagram style, emphasis color, and slide-level organization.

4 Discussion

Why should the canvas be treated as an agent workspace? The canvas is useful not only because it gives users a visual interface, but because it can serve as a shared workspace for both humans and agents. In JarvisHub, prompts, references, candidates, edits, versions, dependencies, and feedback are represented as typed and addressable canvas nodes and links. This turns the canvas into both an external memory and an action space for the agent. Users can inspect and guide the same state that the agent reads and modifies. As a result, the agent can reuse prior artifacts, perform local updates, maintain dependencies, and continue unfinished work without hiding the process in private tool calls or transient chat history.

What can an open harness enable for future creative-agent research? An open creative-agent harness can support research artifacts beyond a single system demonstration. First, it enables project-state benchmarks, where each task specifies an initial canvas, reference materials, available tools, constraints, feedback events, and expected checkpoints rather than only a prompt and a target answer. Second, it supports evaluation protocols that combine final artifact quality with process-level measures, such as context preservation, tool-use appropriateness, dependency correctness, feedback adherence, and repair success. Third, it creates a data flywheel for improving future models: each run produces structured examples of canvas states, agent actions, tool observations, feedback signals, repair decisions, and final outcomes. With proper consent, anonymization, copyright filtering, and quality control, these trajectories can be used to train future creative agents for planning, tool selection, multimodal state tracking, local repair, and feedback-guided revision.

Why should trajectories be the object of analysis? For creative tasks, the final artifact alone does not fully characterize agent behavior. Multiple outputs may be acceptable, and a visually strong result may still be produced by a brittle or incorrect process. Conversely, an imperfect final result may still contain useful intermediate artifacts and recoverable decisions. The trajectory records how the agent moves from one canvas state to the next: what context it uses, which action it selects, what tool evidence is returned, what feedback is received, and how the canvas is repaired or updated. This makes process-level failures visible, such as ignoring references, losing accepted choices, overwriting useful variants, or regenerating globally when a local repair is needed.

5 Conclusion and Limitations

We presented JarvisHub, a canvas-native agent harness for long-horizon multimodal creation. JarvisHub represents a creative project as an editable canvas graph. Through a protocol bridge, agents can inspect project context, call tools, update canvas artifacts, incorporate feedback, and record state changes. The runtime integrates canvas operations, media generation, native tools, repair, checkpointing, and trajectory capture in one shared workspace. Experiments on representative creative tasks show that JarvisHub can support more inspectable and reusable agentic creative workflows. The recorded trajectories also provide useful data for building benchmarks, evaluating long-horizon behavior, and training future creative agents.

Several limitations remain. First, our experiments are qualitative demonstrations rather than a completed benchmark or leaderboard. Second, JarvisHub focuses on orchestration and project-state management, so final artifact quality still depends on the external models and tools used by the runtime. Third, the protocol bridge makes canvas actions explicit and recoverable, but it does not guarantee that the agent's creative decisions are semantically correct. Finally, trajectory records are valuable for analysis and training, but raw trajectories require quality filtering, consent, anonymization, and copyright review before they can be used as research data.

6 Contributions

The contributors' names are listed in alphabetical order (A to Z) by first name.

Core Contributors

Yunlong Lin, Zixu Lin, Zhaohu Xing

Contributors

Biqiang Li, Chenxin Li, Haonan Wang, Haitao Wu, Hengyu Liu, Jianghai Chen, Kaituo Feng, Kaixin Li, Shawn Chen, Shijue Huang, Sixiang Chen, Tsung-Yi Ho, Wenxuan Huang, Xiangyan Liu, Xiaomeng Hu, Xuanhua He, Yan Sun, Yunqing Zhao, Zhiqin Yang, Zehan Wang, Zhengyang Tang

Academic Advisors

Tianyu Pang, Xiangyu Yue

References

[1] Anthropic. Claude design. https://www.anthropic.com/news/claude-design-anthropic-labs, 2025. Accessed 2026-05-10.

[2] Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, et al. FLUX.1 Kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025.

[3] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 18392–18402, 2023.

[4] ByteDance Seed Team. Seedream 5.0 Lite. Model documentation, https://seed.bytedance.com/seedream, 2026. Accessed: 2026-06-07.

[5] Haoyu Chen, Keda Tao, Yizao Wang, Xinlei Wang, Lei Zhu, and Jinjin Gu. Photoartagent: Intelligent photo retouching with language model-based artist agents. *arXiv preprint arXiv:2505.23130*, 2025.

[6] 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.

[7] SiXiang Chen, Jianyu Lai, Jialin Gao, Tian Ye, Haoyu Chen, Hengyu Shi, Shitong Shao, Yunlong Lin, Song Fei, Zhaohu Xing, et al. Postercraft: Rethinking high-quality aesthetic poster generation in a unified framework. *arXiv preprint arXiv:2506.10741*, 2025.

[8] Sixiang Chen, Zhaohu Xing, Tian Ye, Xinyu Geng, Yunlong Lin, Jianyu Lai, Xuanhua He, Fuxiang Zhai, Jialin Gao, and Lei Zhu. Genevolve: Self-evolving image generation agents via tool-orchestrated visual experience distillation. *arXiv preprint arXiv:2605.21605*, 2026.

[9] 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*, 2025.

[10] Ding Ding, Zeqian Ju, Yichong Leng, Songxiang Liu, Tong Liu, Zeyu Shang, Kai Shen, Wei Song, Xu Tan, Heyi Tang, et al. Kimi-audio technical report. *arXiv preprint arXiv:2504.18425*, 2025.

[11] Niladri Shekhar Dutt, Duygu Ceylan, and Niloy J Mitra. MonetGPT: Solving puzzles enhances MLLMs' image retouching skills. *ACM Transactions on Graphics*, 44(4):1–12, 2025.

[12] 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.

[13] Tsu-Jui Fu, Wenze Hu, Xianzhi Du, William Yang Wang, Yinfei Yang, and Zhe Gan. Guiding instruction-based image editing via multimodal large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[14] Tong Ge, Yashu Liu, Jieping Ye, Tianyi Li, and Chao Wang. Advancing vision-language models in front-end development via data synthesis. *arXiv preprint arXiv:2503.01619*, 2025.

[15] Google DeepMind. Gemini 3.1 Pro Model Card. Model card, https://deepmind.google/models/model-cards/gemini-3-1-pro/, February 2026. Accessed: 2026-05-31.

[16] Google DeepMind. Gemini 3.1 Flash Image Preview. Model card, https://deepmind.google/models/gemini-image/flash/, 2026. Also known as Nano Banana 2. Accessed: 2026-06-07.

[17] Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification. *arXiv preprint arXiv:2603.26648*, 2026.

[18] Oucheng Huang, Yuhang Ma, Zeng Zhao, Mingrui Wu, Jiayi Ji, Rongsheng Zhang, Zhipeng Hu, Xiaoshuai Sun, and Rongrong Ji. ComfyGPT: A self-optimizing multi-agent system for comprehensive ComfyUI workflow generation. *arXiv preprint arXiv:2503.17671*, 2025.

[19] Mude Hui, Siwei Yang, Bingchen Zhao, Yichun Shi, Heng Wang, Peng Wang, Yuyin Zhou, and Cihang Xie. Hq-edit: A high-quality dataset for instruction-based image editing. *arXiv preprint arXiv:2404.09990*, 2024.

[20] Alexander Htet Kyaw and Lenin Ravindranath Sivalingam. Node-based editing for multimodal generation of text, audio, image, and video. *arXiv preprint arXiv:2511.03227*, 2025. Accepted to the NeurIPS 2025 Workshop on Generative and Protective AI for Content Creation.

[21] Alexander Htet Kyaw and Lenin Ravindranath Sivalingam. StoryNodes: Human-AI co-creation for multimodal media generation using an agentic node-based interface. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 4540–4548, 2026.

[22] Google Labs. Stitch: A new way to design user interfaces using ai. https://blog.google/innovation-and-ai/models-and-research/google-labs/stitch-ai-ui-design/, 2025. Accessed 2026-05-10.

[23] Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.

[24] Jorge Leandro, Sudha Rao, Michael Xu, Weijia Xu, Nebosja Jojic, Chris Brockett, and Bill Dolan. GENEVA: GENErating and visualizing branching narratives using LLMs. *arXiv preprint arXiv:2311.09213*, 2024. Accepted at IEEE Conference on Games 2024.

[25] Chenxin Li, Zhengyang Tang, Mingxin Huang, Yunlong Lin, Shijue Huang, Shengyuan Liu, Bowen Ye, Rang Li, Lei Li, Benyou Wang, et al. Claw-eval-live: A live agent benchmark for evolving real-world workflows. *arXiv preprint arXiv:2604.28139*, 2026.

[26] LibTV Labs. Libtv skills. https://github.com/libtv-labs/libtv-skills, 2025. Accessed 2026-05-10.

[27] Yunlong Lin, Zixu Lin, Kunjie Lin, Jinbin Bai, Panwang Pan, Chenxin Li, Haoyu Chen, Zhongdao Wang, Xinghao Ding, Wenbo Li, et al. Jarvisart: Liberating human artistic creativity via an intelligent photo retouching agent. *arXiv preprint arXiv:2506.17612*, 2025.

[28] 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*, 2025.

[29] Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, et al. Step1x-edit: A practical framework for general image editing. *arXiv preprint arXiv:2504.17761*, 2025.

[30] MiniMax. Minimax hub. https://hub.minimax.io/, 2026. Accessed 2026-05-25.

[31] Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, et al. Magentic-UI: Towards human-in-the-loop agentic systems. *arXiv preprint arXiv:2507.22358*, 2025.

[32] OpenAI. GPT-5.5 System Card. https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf, 2026. Published April 23, 2026. Accessed: 2026-05-31.

[33] OpenAI. GPT Image 2. OpenAI API documentation, https://developers.openai.com/api/docs/models/gpt-image-2, 2026. Accessed: 2026-05-31.

[34] 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.

[35] Anyi Rao, Xuekun Jiang, Yuwei Guo, Linning Xu, Lei Yang, Libiao Jin, Dahua Lin, and Bo Dai. Dynamic storyboard generation in an engine-based virtual environment for video production. *arXiv preprint arXiv:2301.12688*, 2023.

[36] 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*, pages 10684–10695, 2022.

[37] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[38] Seedream Team, Yunpeng Chen, Yu Gao, Lixue Gong, Meng Guo, Qiushan Guo, Zhiyao Guo, Xiaoxia Hou, Weilin Huang, Yixuan Huang, et al. Seedream 4.0: Toward next-generation multimodal image generation. *arXiv preprint arXiv:2509.20427*, 2025.

[39] Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3956–3974, 2025.

[40] TapNow. Tapnow ai creative platform documentation. https://docs.tapnow.ai/en/docs, 2025. Accessed 2026-05-10.

[41] Team Seedance, De Chen, Liyang Chen, Xin Chen, Ying Chen, Zhuo Chen, Zhuowei Chen, et al. Seedance 2.0: Advancing video generation for world complexity. *arXiv preprint arXiv:2604.14148*, 2026. https://arxiv.org/abs/2604.14148.

[42] Zeyue Tian, Binxin Yang, Zhaoyang Liu, Jiexuan Zhang, Ruibin Yuan, Hubery Yin, Qifeng Chen, Chen Li, Jing Lyu, Wei Xue, et al. Audio-omni: Extending multi-modal understanding to versatile audio generation and editing. *arXiv preprint arXiv:2604.10708*, 2026.

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

[44] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. PromptChainer: Chaining large language model prompts through visual programming. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–10. Association for Computing Machinery, 2022. doi: 10.1145/3491101.3519729.

[45] Shitao Xiao, Yueze Wang, Junjie Zhou, Huaying Yuan, Xingrun Xing, Ruiran Yan, Shuting Wang, Tiejun Huang, and Zheng Liu. Omnigen: Unified image generation. *arXiv preprint arXiv:2409.11340*, 2024.

[46] Jinheng Xie, Zhenheng Yang, and Mike Zheng Shou. Show-o2: Improved native unified multimodal models. *arXiv preprint arXiv:2506.15564*, 2025.

[47] Zhenran Xu, Yiyu Wang, Xue Yang, Longyue Wang, Weihua Luo, Kaifu Zhang, Baotian Hu, and Min Zhang. ComfyUI-R1: Exploring reasoning models for workflow generation. *arXiv preprint arXiv:2506.09790*, 2025.

[48] Zhenran Xu, Xue Yang, Yiyu Wang, Qingli Hu, Zijiao Wu, Baotian Hu, Longyue Wang, Weihua Luo, and Kaifu Zhang. ComfyUI-Copilot: An intelligent assistant for automated workflow development. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 632–643, 2025.

[49] Xiangyuan Xue, Zeyu Lu, Di Huang, Zidong Wang, Wanli Ouyang, and Lei Bai. ComfyBench: Benchmarking LLM-based agents in ComfyUI for autonomously designing collaborative AI systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24614–24624, 2025.

[50] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gaurav Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022.

[51] Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. Magicbrush: A manually annotated dataset for instruction-guided image editing. *Advances in Neural Information Processing Systems*, 36:31428–31449, 2023.

[52] Haozhe Zhao, Xiaojian Shawn Ma, Liang Chen, Shuzheng Si, Rujie Wu, Kaikai An, Peiyu Yu, Minjia Zhang, Qing Li, and Baobao Chang. Ultraedit: Instruction-based fine-grained image editing at scale. *Advances in Neural Information Processing Systems*, 37:3058–3093, 2024.