OMNI-IO SKILLS: HARNESSING YOUR AGENT OMNI-NATIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose agents can plan, reason, and act over long horizons, yet their production capabilities remain fragmented across text, images, audio, video, documents, 3D assets, and code. Extending a foundation model to additional modalities ties capability growth to costly model updates, while assembling specialist models and tools leaves unresolved how procedures, dependencies, intermediate assets, and cross-turn revisions should be coordinated. We present **Omni-IO Skills**, a plug-and-play Agent Harness that makes existing agents omni-native through hierarchical Skills, a standardized multimodal execution interface, dependency-aware orchestration, and a persistent Asset Registry. Multi-asset workflows are represented as Declare Execution Graphs, which schedule independent operations concurrently and register successful outputs for downstream and cross-turn reuse across replaceable execution backends. Its 27 Skills cover 38 representative tasks spanning seven artifact modalities and four capability families: understanding, generation, reasoning, and retrieval. On UniM-90, the harness raises the input-support rates of GPT-5.6 Sol and Claude Sonnet 5 from 40.00% and 38.89% to 100%, while increasing relative Semantic–Quality Coupled Score from 26.99 to 74.94 and from 27.82 to 77.78, respectively; Strict Structure Score reaches 100.00 and 99.78. These results establish harness-level capability composition as a practical route to broad, evolvable Omni systems without changing the host agent's reasoning core. Our code is available at `https://anonymous.4open.science/r/Omni-IO-Skills-ICLR27/`.



Figure 1: Overview of Omni-IO Skills. Hierarchical Skills orchestrate multimodal understanding and generation, while generated assets are registered for downstream and cross-turn reuse.

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.

1 INTRODUCTION

Real-world tasks rarely remain within a single modality. Producing an online course, for example, may begin with lecture recordings and reference documents, continue through content analysis and visual design, and end with slides, illustrations, narration, and an explainer video. The resulting artifacts are not independent outputs: they share facts, style, timing, and production constraints. An effective Omni system must therefore receive and produce interleaved combinations of text, images, audio, video, documents, 3D assets, and code while preserving semantic and asset continuity across the entire workflow (Li et al., 2026c). Recent Omni foundation models have expanded native understanding and generation through unified autoregressive modeling (Lu et al., 2024; Zhan et al., 2024; Chameleon Team, 2024) and hybrid discrete–continuous designs (Wu et al., 2024; Zhou et al., 2025). This progress also exposes a persistent scaling pressure. As more modalities enter a shared model, their representations, objectives, and fidelity requirements must be reconciled, often with new data, codecs, decoders, and alignment stages (Wu et al., 2025). At the same time, specialized models and media engines continue to improve outside the shared backbone. The resulting ecosystem motivates a complementary route to Omni capability: a system layer that can harness heterogeneous and continuously evolving capabilities into coherent, application-level workflows.

General-purpose agents such as Codex (OpenAI, 2026a) and Claude Code (Anthropic, 2025) already provide strong instruction following, long-horizon planning, code execution, workspace operation, and iterative refinement. They supply much of the reasoning core needed to pursue complex goals, yet their end-to-end production surface remains centered on software and knowledge work. Tasks involving audio, video, 3D, professional documents, and coordinated media packages still depend on external capabilities and explicit orchestration. Model-coordination systems show that an agent can delegate to modality specialists without retraining its underlying model (Lin et al., 2025), while recent Agent Skills package reusable procedural knowledge for inference-time use (Li et al., 2026a; Chen et al., 2026; Zhang et al., 2026). These two ingredients do not by themselves provide an Omni workflow. A production task must also select the right capabilities, express control and data dependencies, move intermediate artifacts across tools, isolate local failures, and recover prior outputs for subsequent turns. The missing component is an *Omni agent harness*: a layer between agent reasoning and heterogeneous execution backends that turns scattered models, tools, and procedures into selectable, composable, executable, and traceable capabilities. This leads to our central question: *can a plug-and-play harness make an existing general-purpose agent omni-native while preserving its reasoning and planning core?*

We introduce **Omni-IO Skills**, a plug-and-play Omni-modal agent harness whose capabilities are carried by loadable Skills. As illustrated in Figure 1, the harness leaves the host agent unchanged and provides the procedural knowledge, execution interface, and persistent asset substrate required to turn a user request into a complete multimodal workflow. Its Atomic, Expert, and Scenario Skills capture reusable capability primitives, production procedures for concrete deliverables, and application-level tasks with multiple related outputs. The current implementation comprises 19 Atomic, 2 Expert, and 6 Scenario Skills. Together, these 27 Skills cover 38 representative tasks across understanding, generation, reasoning, and retrieval, spanning seven artifact modalities and application domains from education and research to marketing, creative production, and software engineering; Section 3 presents this user-facing scope. Underneath the Skills, a layered architecture separates task knowledge, tool access, provider binding, and artifact state. Dependency-aware execution and a persistent Asset Registry then coordinate task composition, artifact transfer, and later revision without binding a workflow to a particular service or file path; Sections 4 and 5 detail these mechanisms. Coupled with this runtime and asset layer, the Skills form an Agent Harness that can evolve with its models, tools, and application requirements.

We evaluate the harness on UniM-90, a controlled 90-instance subset of UniM (Li et al., 2026c), using GPT-5.6 Sol and Claude Sonnet 5 as two distinct host agents (OpenAI, 2026b; Anthropic, 2026). Omni-IO Skills raises their input-support rates from 40.00% and 38.89% to 100%. Across the full test set, relative Semantic–Quality Coupled Score (SQCS) increases from 26.99 to 74.94 for GPT-5.6 Sol and from 27.82 to 77.78 for Claude Sonnet 5, gains of 47.95 and 49.96 percentage points. Strict Structure Score reaches 100.00 and 99.78, respectively, while both hosts attain a Lenient Structure Score of 100.00. The absolute SQCS values after loading the harness are 74.94 and 77.78 over all instances, exceeding the Base Agents' 67.49 and 71.53 measured only on their narrower supported subsets. These results show that a shared harness can close substantial modality and workflow gaps across different hosts while delivering strong response quality on all 90 instances.

Our contributions are fourfold:

- We formulate an Omni-modal Agent Harness that extends a general-purpose agent into an omni-native system without retraining the host model.
- We realize this formulation as Omni-IO Skills, integrating multi-granularity procedural knowledge, heterogeneous execution backends, dependency-aware orchestration, and persistent artifact state within one extensible architecture.
- We build an application-facing system that covers seven artifact modalities, 27 Skills, and 38 representative tasks across understanding, generation, reasoning, and retrieval.
- We demonstrate consistent gains in modality coverage, semantic quality, interleaved coherence, and structural completeness across two different host agents.

2 Related Work

Omni Foundation Models. Multimodal research initially developed understanding models for mapping images, audio, and video into language-compatible representations (Radford et al., 2021; Zhang et al., 2023), and generative models for synthesizing visual, acoustic, and temporal content (Ramesh et al., 2022; Borsos et al., 2023; Ho et al., 2022). Omni foundation models increasingly unify these capabilities. Unified autoregressive models tokenize heterogeneous modalities into a common sequence (Lu et al., 2024; Zhan et al., 2024; Chameleon Team, 2024), whereas hybrid systems combine autoregressive reasoning with continuous representations, diffusion or flow objectives, and modality-specific decoders (Wu et al., 2024; Zhou et al., 2025; Ma et al., 2025; Xu et al., 2025a). Recent systems further explore streaming, full-duplex interaction, and modality-specific experts (Défossez et al., 2024; Xu et al., 2025b; Inclusion AI et al., 2025). These approaches unify modalities within the model, tying supported inputs and outputs to its training and architecture. Omni-IO Skills is complementary: it coordinates task execution and artifact flow across replaceable foundation models, specialist models, and media engines.

Agent Harnesses and Skills. Agent harnesses complement a model's reasoning policy with tool access, state management, execution control, verification, and recovery (Tang et al., 2026). ReAct provides an interleaved reasoning–action loop (Yao et al., 2023), while MCP standardizes tool and resource interfaces (Model Context Protocol, 2025). Multimodal agents extend this control loop by delegating to modality specialists (Yang et al., 2023; Lin et al., 2025; Li et al., 2026b), but production workflows additionally require explicit dependencies, intermediate-asset transfer, provenance, failure containment, and cross-turn revision. Agent Skills encode reusable procedural knowledge beyond atomic tool calls. Existing work evaluates curated expert procedures (Li et al., 2026a), parameterized computer-use graphs (Chen et al., 2026), and textual procedures paired with visual state representations (Zhang et al., 2026). Omni-IO Skills connects these lines by using hierarchical Skills to organize any-to-any multimodal procedures and a dependency-aware harness with persistent asset state to execute them as composable, traceable workflows.

3 System Capabilities and Task Coverage

Omni-IO Skills provides a task interface for requests whose source material, intermediate assets, and deliverables span multiple media types. A request may begin with a report, a recorded interview, product images, or a 3D model, then produce an analysis, a media artifact, or a coordinated package of outputs. The implemented Skills accept, produce, and connect seven artifact types:

Text **Image** **Video** **Audio** **Document** **3D** **Code**

We group the tasks supported by Omni-IO Skills into the following four categories. Cross-Modal Understanding extracts and organizes information from heterogeneous inputs; Cross-Modal Generation creates individual artifacts or coordinated deliverables from user requests and reusable assets; Cross-Modal Reasoning combines multimodal evidence with task constraints to produce plans, decisions, or revisions; and Cross-Modal Retrieval obtains grounded information from local documents or the web for direct answers or downstream use. Figure 2 shows representative tasks and their input–output modality flows. Appendix B provides detailed corresponding task-to-Skill mappings.

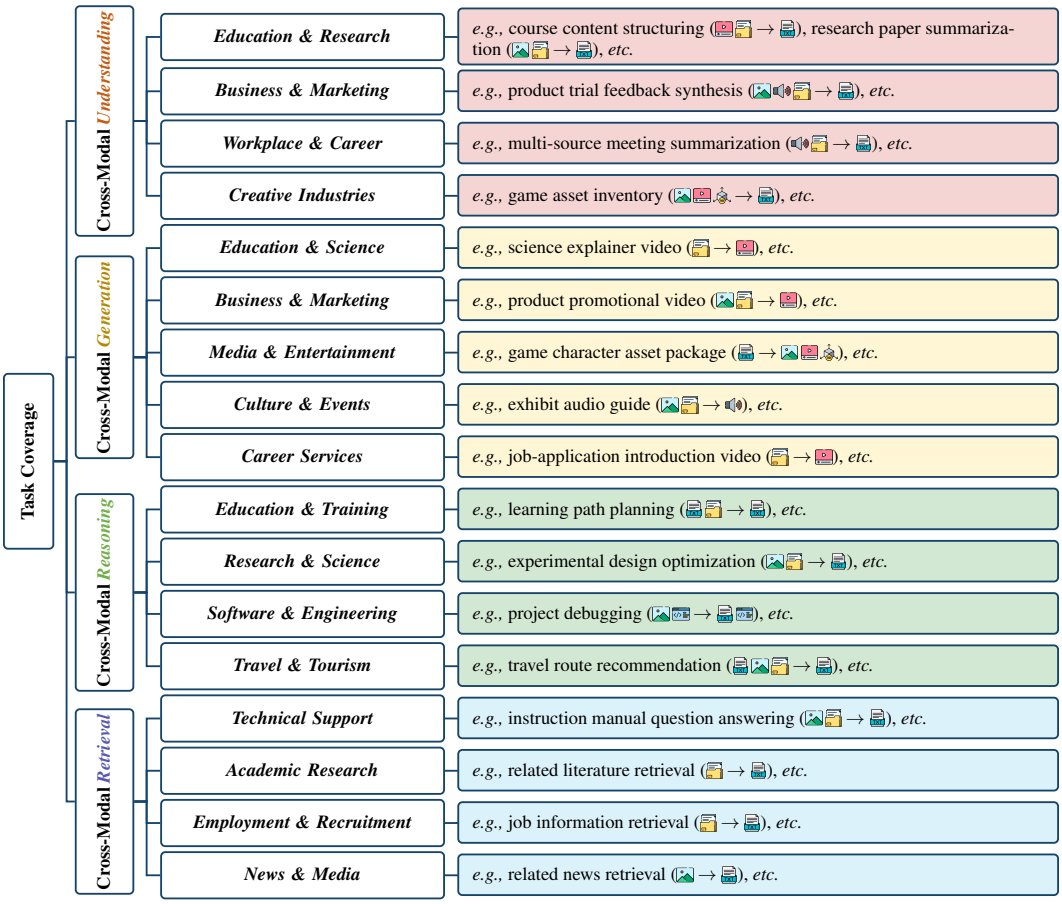

Figure 2: Real-world task coverage supported by Omni-IO Skills. Parenthesized icons show representative modality flows: icons to the left and right of an arrow denote inputs and outputs, respectively, and multiple icons indicate tasks that consume or produce multiple artifact types.

4 Layered Architecture of Omni-IO Skills

4.1 Architecture Overview

As illustrated in Figure 3, Omni-IO Skills sits between the host agent and external multimodal tools. Its four-layer architecture comprises the **Skill Entry**, **MCP Tool Service**, **Provider and Configuration**, and **Asset Registry** layers. Together, these layers separate task knowledge, tool interfaces, service implementations, and persistent outputs, allowing the host agent to plan multimodal workflows without coupling them to specific providers or workspaces. The Skill Entry layer represents a workflow as a ***Declare Execution Graph (DEG)***, while the remaining layers provide the abstractions needed to execute its nodes and manage their outputs.

4.2 Skill Entry Layer

As shown in Figure 4, the Skill Entry layer organizes reusable Skills by task granularity and compositional scope rather than by modality, using three levels: **Atomic Skills**, **Expert Skills**, and **Scenario Skills**. For example, the Event Material Scenario Skill can invoke Poster Design and Complex Video Production, which expand into Atomic generation Skills.

4.2.1 Declarative Skill Representation and Hierarchical Expansion

Skills at all three levels share the declarative representation $s = \langle c_s, I_s, P_s, O_s, H_s \rangle$, which specifies applicability, inputs, procedure, outputs, and cross-Skill relationships. The host agent selects appropriate Skills from the task context and recursively expands higher-level Skills until the terminal tasks become executable DEG nodes. Appendix D provide further details.

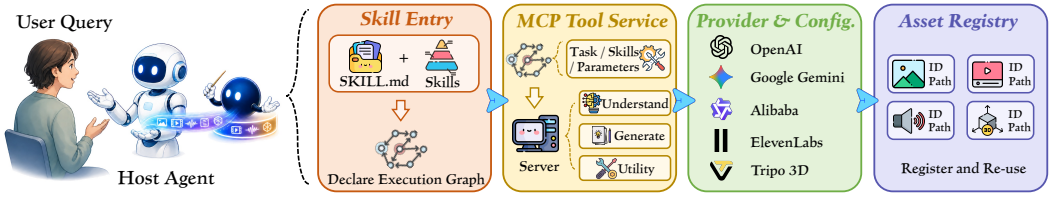

Figure 3: Architecture overview of Omni-IO Skills, which presents the four-layer architecture.

4.2.2 Hierarchical Omni-IO Skills

Atomic Skills. Atomic Skills are the ***smallest executable units***, each encapsulating one multimodal understanding, generation, or tool operation. The host agent invokes them directly for single-step requests. Operations requiring specialist services run through MCP tools, while code and Markdown generation run natively. These reusable primitives form the basis of Expert and Scenario Skills.

Expert Skills. Expert Skills produce one final deliverable through a ***production workflow*** of requirement analysis, planning, asset generation, assembly, inspection, and localized revision. For requests requiring decomposition and artifact-level inspection, they expand into existing Atomic Skills, then assemble and inspect the result. Defects trigger rework only of affected components, while validated assets are reused. Thus, Expert Skills expose professional production capabilities to higher-level Scenario Skills without duplicating the underlying Atomic Skills or execution logic.

Scenario Skills. Scenario Skills coordinate ***interrelated deliverables*** within an application context. They define trigger boundaries, typical inputs, deliverable-selection rules, and expected outputs without fixing the media mix or execution steps. Explicitly requested deliverables set the scope; underspecified requests follow scenario conventions, with clarification when the boundary remains ambiguous. For each deliverable, the Skill selects an Expert or Atomic Skill according to complexity and coordinates dependencies, shared constraints, and asset reuse across related deliverables.

4.3 MCP Tool Service Layer

The MCP Tool Service layer converts the semantic task specifications produced by the Skill Entry layer into standardized executable operations. It organizes external capabilities into three functional groups: **understanding** tools that consume multimodal inputs and return structured analyses; **generation** tools that produce image, video, audio, document, or 3D outputs; and **utility** tools that provide operations such as search and browsing. Each externally executed DEG node is submitted through a common contract containing its task type, self-contained prompt, optional parameters, and dependency-resolved asset inputs. The service uses the task type to select the appropriate tool interface without exposing provider-specific API details to the Skill definition.

The service also defines the boundary between external tool execution and host-native execution. Operations that depend on specialist multimodal models are invoked through MCP tools, whereas supported operations such as code and Markdown generation can be completed directly by the host agent. Both routes follow the same DEG dependency semantics and return outputs that can be normalized as system assets. For external calls, the tool service converts the provider response into a common result containing the output type, local path, description, and effective parameters before forwarding it to the Asset Registry. Consequently, upper-level Skills can compose heterogeneous capabilities through a stable tool contract even when their underlying APIs and output formats differ.

4.4 Provider and Configuration Layer

The Provider and Configuration layer separates a tool capability from the service implementation currently used to execute it. For each externally executable task type, it maintains a binding to the corresponding tool, provider, model, credential reference, default parameters, and fallback policy. A Skill therefore refers to a semantic capability such as image generation or speech synthesis rather than directly naming a provider-specific endpoint. The MCP Tool Service consumes this binding when it prepares the concrete invocation. This separation supports implementation-level substitution without rewriting the procedural knowledge encoded by Skills. A provider or model can be added, replaced, or reconfigured within the lower layer while the Skill description and DEG structure remain unchanged. Default parameters provide a consistent baseline for each tool, whereas node-level parameters allow a task to specialize the invocation without altering the shared provider configuration.



Figure 4: Hierarchical organization of Omni-IO Skills. Higher-level Skills organize and invoke lower-level Skills. **Scenario Skills** coordinate application-level objectives, **Expert Skills** encapsulate reusable production workflows, and **Atomic Skills** provide executable capability primitives.

4.5 Asset Registry Layer

The Asset Registry provides a shared data abstraction for user-provided, intermediate, and final artifacts. A registered asset is logically represented as

$$a = \langle \texttt{asset_id}, \texttt{type}, \texttt{subtype}, \texttt{path}, \texttt{description}, \texttt{params}, \texttt{turn_id}, \texttt{source_asset_id} \rangle. \quad (1)$$

The globally unique `asset_id` gives upper layers a stable reference that is independent of the physical file path. The `type` identifies the broad asset category, while `subtype` distinguishes finer classes within that category. The description and effective parameters preserve generation context, and `turn_id` associates the artifact with its interaction turn. When an artifact is revised or derived from an earlier one, `source_asset_id` records the provenance relationship.

The registry applies this representation uniformly to artifacts produced through MCP tools and by the host agent. Records are persisted in an append-only JSON registry rather than updated in place, allowing multiple versions of a deliverable to remain traceable. Registry writes are atomic and serialized with a file lock so that parallel nodes or sessions cannot overwrite one another. Through its lookup and registration interfaces, the layer decouples asset consumers from concrete workspace paths and provides the persistent references required for downstream and cross-turn reuse.

5 Dependency-Aware Orchestration and Asset Lifecycle

5.1 Runtime Workflow

Upon receiving a user request that involves multimodal understanding or generation, the host agent invokes Omni-IO Skills. The Skill Entry layer first selects the relevant Skills and expands them into executable tasks. These tasks are instantiated as nodes in a DEG that expresses their control and data dependencies. Before execution, the DEG undergoes structural validation and is then scheduled in successive Waves. Each externally executed node is mapped to a concrete tool and provider through the MCP Tool Service and Provider and Configuration layers, while natively executed nodes follow the same scheduling protocol. The resulting assets are registered in the Asset Registry.

The runtime jointly organizes ***control flow*** and ***asset flow***. Control flow starts from Skill selection and DEG construction, proceeds through validation and dependency-aware scheduling, and ends with the final state of every node. Asset flow starts from user-provided, historical, and newly generated artifacts. Each artifact is normalized into a registered asset whose local path can be resolved and passed to downstream nodes. Figure 5 summarizes the interaction between these two flows.

5.2 DEG Representation and Construction

Let $G = (V, E)$ denote a DEG, where each node $v \in V$ is an executable task or an existing-asset source and each directed edge $(u, v) \in E$ indicates that v depends on u. An executable node can be represented as

$$v = \langle \texttt{id}, \texttt{type}, \texttt{prompt}, \texttt{params}, \texttt{depends_on} \rangle. \quad (2)$$

The `id` uniquely identifies the node, `type` specifies the required capability, `prompt` provides a self-contained operation description, `params` specializes the execution, and `depends_on` records

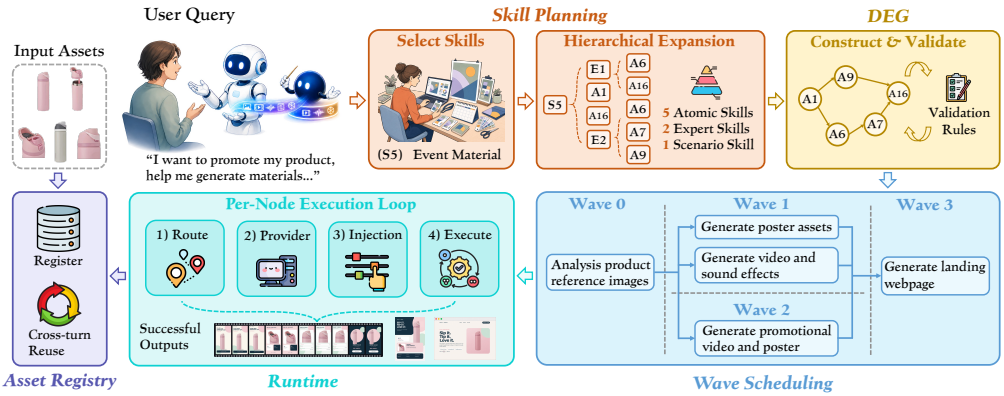

Figure 5: Runtime workflow of Omni-IO Skills, illustrated with a product promotion request. The selected S5 Event Material Scenario Skill expands through Expert Skills into five Atomic Skills, whose dependencies form a validated DEG and determine successive execution Waves. Each ready node follows the same execution loop of tool routing, provider resolution, asset injection, and execution, while successful outputs are registered for downstream use and cross-turn reuse.

upstream node or asset references. An edge acts as a control dependency because the downstream node cannot start before its predecessor completes, and as a data dependency when the predecessor's output is consumed by the downstream operation. The host agent constructs G from the tasks produced by Skill expansion. Shared intermediate results are represented once and referenced by every consumer. A historical asset is represented by an `asset_source` node containing its `asset_ref`; it is considered completed after the reference is resolved and issues no generation call.

5.3 Graph Validation, Wave Scheduling, and Failure Handling

Before execution, the DEG is validated to ensure that every dependency reference resolves to a node or registered asset and that the induced graph is acyclic. A graph that fails either check is treated as a planning error and is reconstructed rather than submitted in an invalid form. For a valid DEG, pending nodes whose predecessors have all completed form the next execution Wave. Nodes in the same Wave are mutually independent and are dispatched concurrently. Nodes with no unresolved predecessors enter the earliest Wave, and every remaining node enters the first later Wave in which all of its immediate predecessors have completed. The schedule is determined by declared dependencies rather than modality: image and audio generation can run concurrently when neither consumes the other, whereas an image-conditioned video node waits for its reference image. When a node fails, its pending descendants are cancelled because their required inputs are unavailable, while nodes on independent branches continue to execute. The system records the final state of every node and does not roll back outputs produced by completed nodes.

5.4 Runtime Tool Routing and Parameter Injection

For each executable node, the MCP Tool Service selects a tool according to the node's task type. The Provider and Configuration layer then resolves the corresponding provider, model, credentials, default parameters, and fallback policy. At invocation time, the effective parameters combine provider defaults, node-level overrides, and inputs injected from predecessor outputs.

Once an upstream node completes, the system resolves the local paths of its outputs and passes them to the downstream tool call according to the task relationship. An image-to-video dependency supplies the image as the initial visual condition, whereas an image-to-3D dependency supplies it as the reference image. Because these inputs are injected from the graph, downstream prompts do not need to embed paths. Code, Markdown, and tasks executed natively by the host agent bypass MCP generation services but retain the same input-resolution, status, and output-registration contract.

5.5 Asset Propagation and Cross-Turn Reuse

Once a node completes successfully, its output is written to the workspace and registered in the Asset Registry. MCP-tool outputs are registered automatically when their results are returned, whereas files produced natively by the host agent are registered explicitly after they are written. Within one

Table 1: Main results on UniM-90. We report the input-support rate τ together with the absolute and relative variants of Semantic–Quality Coupled Score (SQCS), Interleaved Coherence Score (ICS), Strict Structure Score (StS), and Lenient Structure Score (LeS). Higher is better for all metrics.

Base Agent	τ	Absolute				Relative			
		SQCS	ICS	StS	LeS	SQCS	ICS	StS	LeS
GPT-5.6 Sol	40%	67.49	86.53	47.84	72.22	26.99	34.61	19.14	28.89
+ **Omni-IO Skills**	100%	74.94	93.98	100.00	100.00	74.94	93.98	100.00	100.00
Claude Sonnet 5	38.89%	71.53	82.29	52.21	68.57	27.82	32.00	20.30	26.67
+ **Omni-IO Skills**	100%	77.78	83.28	99.78	100.00	77.78	83.28	99.78	100.00

turn, downstream nodes resolve upstream outputs through their asset identifiers and consume the corresponding local files. Across turns, a user can refer to a historical asset through `asset_ref`; the referenced record is materialized as an already-completed source node in the new DEG and becomes immediately available to its descendants. A revision creates a new record that may retain a reference to its source instead of updating the existing record in place. See Appendix G for further details.

6 Experiments

6.1 Experimental Setup

Baselines. We evaluate two general-purpose agents, GPT-5.6 Sol (OpenAI, 2026b) and Claude Sonnet 5 (Anthropic, 2026), under two configurations for each agent: *Base Agent* and *Agent + Omni-IO Skills*. The host agent runs in its default environment, retaining the built-in Skills and tools provided by the system while excluding any additionally installed third-party extensions or task-specific customizations. *Agent + Omni-IO Skills* further loads Omni-IO Skills and its associated MCP tool services into the same environment, while keeping all other built-in Skills, tool configurations, prompts, and execution budgets unchanged. Because the default capabilities of the two agents may differ, we focus on the within-agent gains introduced by Omni-IO Skills and do not interpret cross-agent score differences as a ranking of the models' capabilities.

Benchmark and evaluation metrics. We construct UniM-90 by selecting a fixed subset of 90 instances from UniM (Li et al., 2026c), covering text, image, audio, video, document, code, and 3D modalities, together with their representative interleaved combinations. The subset is selected independently of the native modality capabilities of any evaluated agent; the detailed selection protocol is provided in Appendix I.1. For evaluation, we adopt the UniM Evaluation Suite and report the input-support rate τ, together with the absolute and relative variants of Semantic–Quality Coupled Score (SQCS), Interleaved Coherence Score (ICS), Strict Structure Score (StS), and Lenient Structure Score (LeS). Here, τ denotes the proportion of instances for which an agent can fully receive and process all input modalities; the absolute score $\mathcal{X}^{\mathrm{abs}}$ measures performance over the subset of supported instances, whereas the relative score $\mathcal{X}^{\mathrm{rel}} = \tau\mathcal{X}^{\mathrm{abs}}$ further reflects performance over the complete test set. SQCS jointly evaluates semantic correctness and generation quality; StS and LeS measure strict output-structure consistency and modality-level coverage, respectively; and ICS assesses the overall coherence of interleaved multimodal responses. For instances whose inputs can be processed but whose required output modalities cannot be generated or whose target outputs cannot be completed, the resulting failures remain included in metric computation.

6.2 Main Results

As shown in Table 1, Omni-IO Skills yields substantial and consistent improvements for both base agents. First, the input-support rates of GPT-5.6 Sol and Claude Sonnet 5 increase from 40.00% and 38.89%, respectively, to 100% in both cases. This result shows that Omni-IO Skills effectively addresses the limitations of base agents in handling heterogeneous multimodal inputs and outputs, including audio, video, documents, and 3D content, thereby enabling them to cover all tasks in UniM-90. In terms of the relative metrics, SQCS increases from 26.99 and 27.82 to 74.94 and 77.78, while ICS increases from 34.61 and 32.00 to 93.98 and 83.28, respectively, demonstrating substantial improvements in semantic quality and interleaved multimodal coherence over the full test set. For the

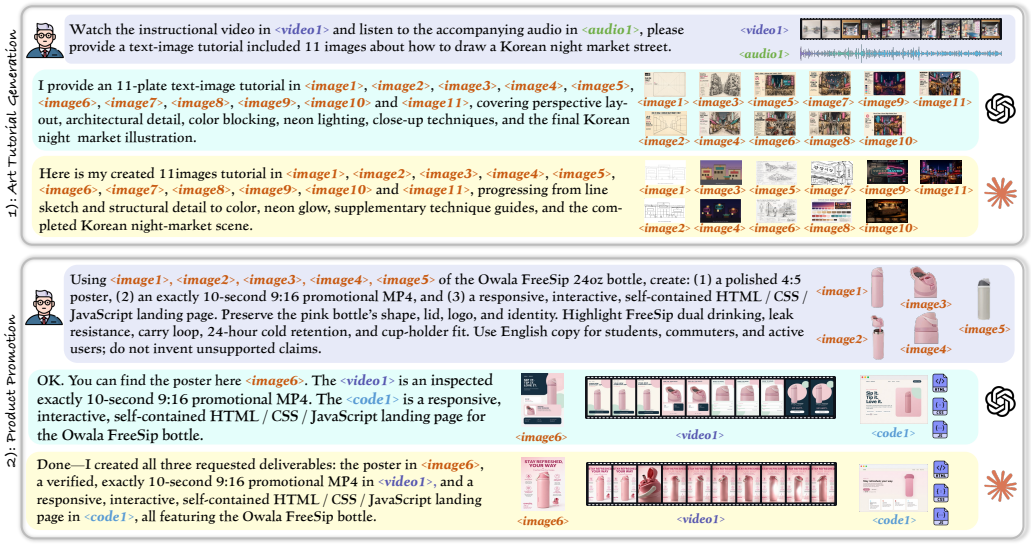

Figure 6: Qualitative case studies of Omni-IO Skills with GPT-5.6 Sol and Claude Sonnet 5.

absolute metrics, SQCS increases from 67.49 and 71.53 to 74.94 and 77.78, respectively, indicating that Omni-IO Skills not only broadens modality support but also improves the semantic correctness and generation quality of task outputs. Absolute ICS also rises from 86.53 to 93.98 and from 82.29 to 83.28 for the two agents, respectively. For each agent, the gains are achieved with its original prompts, showing that Omni-IO Skills improves multimodal performance without prompt retuning. The gains are particularly pronounced for the structure metrics: GPT-5.6 Sol achieves 100% on both StS and LeS, while Claude Sonnet 5 reaches 99.78% and 100%, respectively, validating the effectiveness of Omni-IO Skills in multimodal tool invocation, asset organization, and output-structure control.

6.3 Case Studies

As shown in Figure 6, we qualitatively analyze the end-to-end performance of Omni-IO Skills with GPT-5.6 Sol and Claude Sonnet 5 through two cases: art tutorial generation and product promotion. In the art tutorial generation case, Omni-IO Skills invokes the Education Sharing Skill to plan the tutorial and coordinate the workflow, the Video Understanding Skill to extract the drawing steps from the video, the Audio Understanding Skill to recover the accompanying explanations and procedural order, and the Image Generation Skill to produce the 11-image tutorial. In the product promotion case, Omni-IO Skills invokes the Image Understanding Skill to extract product attributes from the images, the Poster Design Skill to design the poster, the Complex Video Production Skill to produce the promotional video, and the Code Generation Skill to build the landing page, thereby generating the requested deliverables. Both GPT-5.6 Sol and Claude Sonnet 5 produce all requested outputs while preserving consistency across tutorial steps, product identity, and deliverables, demonstrating the stable performance of Omni-IO Skills in multimodal understanding, generation, and the orchestration of interdependent outputs from source interpretation through final artifact assembly in both cases.

7 Conclusion

Omni-IO Skills presents a plug-and-play Agent Harness that makes general-purpose agents omni-native without altering their reasoning core. The system organizes multimodal capabilities as hierarchical Skills, maps complex requests into dependency-aware DEGs, routes tasks across replaceable specialist backends, and preserves generated artifacts through a persistent Asset Registry. On UniM-90, the harness raises the input-support rates of GPT-5.6 Sol and Claude Sonnet 5 to 100%, increases relative Semantic–Quality Coupled Score by 47.95 and 49.96 points, and reaches Strict Structure Scores of 100.00 and 99.78. These results show that harness-level composition provides a practical route to Omni capability while keeping procedures, providers, and assets independently extensible. Omni-IO Skills therefore offers an application-oriented foundation for agents that coordinate heterogeneous media and reusable outputs across multi-turn production workflows.

AI USE STATEMENT

Generative AI tools were used to assist with LaTeX formatting, manuscript polishing, code development, and the generation of Figure 1. All AI-assisted changes were reviewed by the authors, who take responsibility for the final content of this work, including its text, claims, and artifacts.

REFERENCES

Anthropic. Claude Code: Best practices for agentic coding. Technical report, Anthropic, 2025. URL https://www.anthropic.com/engineering/claude-code-best-practices.

Anthropic. Claude Sonnet 5 System Card. Technical report, Anthropic, 2026. URL https://www.anthropic.com/claude-sonnet-5-system-card.

Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. AudioLM: A language modeling approach to audio generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024.

Tianyi Chen, Yinheng Li, Michael Solodko, et al. CUA-Skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*, 2026.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, et al. Moshi: A speech-text foundation model for real-time dialogue. *arXiv preprint arXiv:2410.00037*, 2024.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Proceedings of the NeurIPS*, 2022.

Inclusion AI, Biao Gong, Cheng Zou, et al. Ming-Omni: A unified multimodal model for perception and generation. *arXiv preprint arXiv:2506.09344*, 2025.

Xiangyi Li, Yimin Liu, Wenbo Chen, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026a.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, et al. OmniGAIA: Towards native omni-modal AI agents. *arXiv preprint arXiv:2602.22897*, 2026b.

Yanlin Li, Minghui Guo, Kaiwen Zhang, Shize Zhang, Yiran Zhao, Haodong Li, Congyue Zhou, Weijie Zheng, Yushen Yan, Shengqiong Wu, Wei Ji, Lei Cui, Furu Wei, Hao Fei, Mong-Li Lee, and Wynne Hsu. Unim: A unified any-to-any interleaved multimodal benchmark. In *Proceedings of the CVPR*, 2026c.

Huawei Lin, Yunzhi Shi, Tong Geng, Weijie Zhao, Wei Wang, and Ravender Pal Singh. Agent-Omni: Test-time multimodal reasoning via model coordination for understanding anything. *arXiv preprint arXiv:2511.02834*, 2025.

Jiasen Lu, Christopher Clark, Sangho Lee, Zichen Zhang, Savya Khosla, Ryan Marten, Derek Hoiem, and Aniruddha Kembhavi. Unified-IO 2: Scaling autoregressive multimodal models with vision, language, audio, and action. In *Proceedings of the CVPR*, 2024.

Yiyang Ma, Xingchao Liu, Xiaokang Chen, et al. JanusFlow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation. In *Proceedings of the CVPR*, pp. 7739–7751, 2025. doi: 10.1109/CVPR52734.2025.00725.

Model Context Protocol. Model context protocol specification. Technical report, Model Context Protocol, 2025. URL https://modelcontextprotocol.io/specification/2025-06-18.

OpenAI. Introducing the Codex app. Technical report, OpenAI, 2026a. URL https://openai.com/index/introducing-the-codex-app/.

OpenAI. GPT-5.6 System Card. Technical report, OpenAI, 2026b. URL https://deployment safety.openai.com/gpt-5-6.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the ICML*, 2021.

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.

Xinyu Tang, Han Peng, Guoxin Chen, Yuze Shi, Zitao Su, Peiyu Liu, Wayne Xin Zhao, Yawen Li, and Zhe Xue. Agent systems with harness engineering. OpenReview, 2026. URL https://openreview.net/forum?id=nM5tDHrQsx.

Chengyue Wu, Xiaokang Chen, Zhiyu Wu, Yiyang Ma, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, Chong Ruan, and Ping Luo. Janus: Decoupling visual encoding for unified multimodal understanding and generation. In *Proceedings of the CVPR*, pp. 12966–12977, 2025. doi: 10.1109/CVPR52734.2025.01210.

Shengqiong Wu, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. NExT-GPT: Any-to-any multimodal LLM. In *Proceedings of the ICML*, pp. 53366–53397, 2024.

Jin Xu, Zhifang Guo, Jinzheng He, et al. Qwen2.5-Omni technical report. *arXiv preprint arXiv:2503.20215*, 2025a.

Jin Xu, Zhifang Guo, Hangrui Hu, et al. Qwen3-Omni technical report. *arXiv preprint arXiv:2509.17765*, 2025b.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. MM-REACT: Prompting ChatGPT for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the ICLR*, 2023.

Jun Zhan, Junqi Dai, Jiasheng Ye, et al. AnyGPT: Unified multimodal LLM with discrete sequence modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9637–9662, 2024. doi: 10.18653/v1/2024.acl-long.521.

Hang Zhang, Xin Li, and Lidong Bing. Video-LLaMA: An instruction-tuned audio-visual language model for video understanding. In *Proceedings of the EMNLP*, 2023.

Kangning Zhang, Shuai Shao, Qingyao Li, et al. MMSkills: Towards multimodal skills for general visual agents. *arXiv preprint arXiv:2605.13527*, 2026.

Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. In *Proceedings of the ICLR*, 2025.

A Complete Catalog of Omni-IO Skills

Table 2: Complete catalog of Omni-IO Skills across the Atomic, Expert, and Scenario Skills.

ID	Skill	Description
		Atomic Skills
A1	Image Understanding	Analyzes image content, text, objects, visual style, composition, and atmosphere.
A2	Video Understanding	Summarizes video content and analyzes timelines, speech, key frames, and audiovisual style.
A3	Audio Understanding	Transcribes speech and analyzes music, sound effects, speakers, timing, and acoustic scenes.
A4	Document Understanding	Extracts and summarizes text, structure, tables, and slide content from PDF, Word, PowerPoint, and Excel files.
A5	3D Understanding	Inspects 3D geometry, dimensions, materials, topology, semantic category, and rendered appearance.
A6	Image Generation	Generates images from text prompts with configurable aspect ratios and output formats.
A7	Video Generation	Generates text- or image-conditioned videos with configurable duration, resolution, and aspect ratio.
A8	Music Generation	Generates instrumental or vocal music from descriptions of style, mood, instrumentation, and tempo.
A9	Sound-Effect Generation	Generates environmental sounds and effects with controllable duration, source, intensity, and spatial character.
A10	Speech Generation	Synthesizes multilingual speech with configurable voice, language, and speaking speed.
A11	3D Generation	Generates GLB models from text prompts or reference images for downstream visualization and production.
A12	PPT Generation	Produces structured PowerPoint presentations from agent-planned slide titles and key points.
A13	Word Generation	Produces structured Word documents from agent-planned headings and paragraphs.
A14	PDF Generation	Produces structured PDF documents from agent-planned sections and body text.
A15	Excel Generation	Produces styled, multi-sheet Excel workbooks from structured headers and data rows.
A16	Code Generation	Creates and registers code or web artifacts, supports versioned iteration, and integrates multiple upstream assets.
A17	Markdown Generation	Creates and registers structured Markdown documents, supports versioned edits, and integrates multiple upstream assets.
A18	Web Search	Retrieves up-to-date web information and returns structured results with source links.
A19	Web Browsing	Navigates webpages, interacts with page elements, and extracts content through a local browser.
		Expert Skills
E1	Poster Design	Creates and reviews polished posters, invitations, promotional graphics, and social covers by combining generated or supplied visuals with exact typography and deterministic layout.
E2	Complex Video Production	Produces and reviews complete multi-scene videos by coordinating scripts, storyboards, visual clips, narration, music, subtitles, branding, and final assembly.
		Scenario Skills
S1	Social-Media Post	Orchestrates platform-aware copy with suitable image, video, or audio assets for social publishing.
S2	Office Documents	Converts text, recordings, and source documents into structured presentations, reports, minutes, or spreadsheets.
S3	Job Application	Creates resumes, cover letters, self-introduction presentations, speech, or video for a target role.
S4	Education Sharing	Produces teaching and explanatory materials such as slides, documents, illustrations, narration, and videos.
S5	Event Material	Creates invitations, posters, atmosphere music, and promotional or retrospective videos for events.
S6	Game Asset	Orchestrates concept art, 3D models, showcase videos, and optional web presentations for game assets.

B REPRESENTATIVE REAL-WORLD TASK COVERAGE

Table 3: Representative real-world task coverage of Omni-IO Skills. Both Skill IDs and modality combinations denote representative configurations rather than exhaustive implementations. Icons denote text, image, audio, video, document, code, and 3D.

Task	Skill ID(s)	Input Modality(s)	Output Modality(s)
Cross-Modal Understanding			
Multi-Source Meeting Summarization	A3, A4		
Interview Topic Analysis	A3, A4		
Course Content Structuring	A2, A4		
Research Paper Summarization	A1, A4		
Academic Presentation Question Answering	A2, A4		
Product Demonstration Analysis	A2, A4		
Podcast Content Analysis	A3, A4		
Product Trial Feedback Synthesis	A1, A3, A4		
Brand Material Analysis	A1, A2, A4		
Job Application Materials Analysis	A2, A4		
3D Design Analysis	A1, A5		
Game Asset Inventory	A1, A2, A5		
Exhibit Content Organization	A1, A3, A4		
Sales Material Summarization	A1, A3, A4		
Event Record Review	A1, A2, A3		
Experiment Record Organization	A1, A2, A4		
Cross-Modal Generation			
Academic Poster Generation	E1, A1, A4		
Video Thumbnail Design	E1, A2		
Podcast Promotional Post	S1, E1, A3		
Product Promotional Poster	E1, A1, A4		
Event Invitation Design	S5, E1, A1		
Product Social-Media Post	S1, A1, A4, A6		
Short-Video Social Post	S1, E1, A2		
Course Video Handout	S4, A2, A13		
Exhibit Audio Guide	A1, A4, A10		
Product Promotional Video	S1, E2, A1, A4		
Science Explainer Video	S4, E2, A4		
Event Teaser Video	S5, E2, A1		
Job-Application Introduction Video	S3, E2, A4		
Game Character Asset Package	S6, A6, A7, A11		
Cross-Modal Reasoning			
Learning Path Planning	A4, A18, A19		
Experimental Design Optimization	A1, A4		
Project Debugging	A1, A16		
Travel Route Recommendation	A1, A4, A18, A19		
Cross-Modal Retrieval			
Instruction Manual Question Answering	A1, A4		
Related Literature Retrieval	A4, A18, A19		
Job Information Retrieval	A4, A18, A19		
Related News Retrieval	A1, A18, A19		

Table 4: Implemented Omni-IO Skills and their hierarchical composition.

Atomic Skills					
A1	Image Understanding	A2	Video Understanding	A3	Audio Understanding
A4	Document Understanding	A5	3D Understanding	A6	Image Generation
A7	Video Generation	A8	Music Generation	A9	Sound-Effect Generation
A10	Speech Generation	A11	3D Generation	A12	PPT Generation
A13	Word Generation	A14	PDF Generation	A15	Excel Generation
A16	Code Generation	A17	Markdown Generation	A18	Web Search
A19	Web Browsing				
Expert Skills					
E1	Poster Design	→	A1 A6 A16		
E2	Complex Video Production	→	A2 A3 A6 A7 A8 A9 A10		
Scenario Skills					
S1	Social-Media Post	→	E1 E2 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10		
S2	Office Documents	→	A3 A4 A12 A13 A14 A15		
S3	Job Application	→	E1 E2 A4 A10 A12 A13		
S4	Education Sharing	→	E1 E2 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14		
S5	Event Material	→	E1 E2 A6 A7 A8 A9 A10		
S6	Game Asset	→	A6 A7 A11 A16		

C MCP Tools and Provider Bindings

C.1 Tool Inventory

Table 5 lists the tools of the MCP Tool Service layer, grouped into understanding, generation, utility, and registry management. Capabilities that require an external service are exposed as MCP tools. Image understanding uses the host agent's native vision, while code and Markdown generation use the host agent's native file writing. Generation tools register their outputs before returning; the natively executed `code` and `markdown` task types require an explicit `register_asset` call. Document generation tools receive a planned `structure` and content from the host agent and render them into the requested document files.

C.2 Provider Bindings and Fallback

Table 6 lists the provider, model, and API key bound to each capability in the configuration used throughout this work. A binding is a single field in the configuration file; changing a provider requires editing that field and restarting the tool service, and never touches a Skill specification, the DEG format, or the Asset Registry.

C.3 Default Parameters

Each generation capability carries a set of defaults in the configuration layer. A node in the DEG may override any of them through its `params` field, and the effective values are recorded in the resulting asset record, so that an asset can always be reproduced from what was registered. Table 7 lists the defaults used in this work.

C.4 Availability under Partial Configuration

Omni-IO Skills does not require a complete credential set. Image understanding; PPT, Word, PDF, and Excel generation; code and Markdown generation; speech synthesis; and web browsing run natively in the host agent or locally, so they remain available without credentials. On the first multimodal request, the host calls `check_config`, which reports ready and missing capabilities, free alternatives when available, and capabilities that are always available.

Table 5: Tools exposed by the MCP Tool Service layer. *Registers* indicates whether the tool writes its output to the Asset Registry before returning. Image understanding, code generation, and Markdown generation are executed natively by the host agent.

Tool	Modality	Key parameters and returned content	Registers
		Understanding	
`understand_video`	video	`video_url`; returns a summary description and a segmented timeline	✗
`understand_audio`	audio	`audio_url`, `subtype`; returns a transcript for speech and a style analysis for music and sound effects	✗
`understand_document`	document	`document_url`; returns a summary and the extracted body text in Markdown	✗
`understand_3d`	3D	`file_url`; returns mesh metadata, rendered projection paths, and the renderer actually used	✗
		Generation	
`generate_image`	image	`prompt`, `aspect_ratio`, `quality`	✓
`generate_video`	video	`prompt`, `duration_seconds`, `aspect_ratio`, `first_frame_url`	✓
`generate_music`	audio	`prompt`, `duration_seconds`, `instrumental`	✓
`generate_sfx`	audio	`prompt`, `duration_seconds`	✓
`generate_speech`	audio	`text`, `voice`, `language`, `speed`	✓
`generate_3d`	3D	`prompt`, `reference_image_url`; always emits GLB	✓
`generate_ppt`	document	`structure` (title and per-slide key points), `style`	✓
`generate_word`	document	`structure` (title, headings, paragraphs), `style`	✓
`generate_pdf`	document	`structure` (title, headings, paragraphs), `style`	✓
`generate_excel`	document	`structure` (per-sheet headers and rows), `style`	✓
		Utility	
`search`	text	`query`, `count`; returns titles, links, and snippets	✗
`browse`	text	`url`, `max_chars`; returns extracted page text with a truncation flag	✗
`check_config`	—	returns ready, missing, and always-free capabilities together with the active providers	✗
		Registry management	
`set_turn`	—	`turn_id`; stamps every asset registered in the current interaction turn	✗
`register_asset`	any	`asset_type`, `subtype`, `path`, `description`, `params`, `source_asset_id`; used for natively produced files	✓
`get_asset`	any	`asset_id`; returns the full record, including the local path	✗
`list_assets`	any	returns all registered assets in reverse completion order	✗

D Skill Specification Format and Extensibility

D.1 Shared Specification Schema

Every Skill specification is a self-contained document loaded on demand by the Skill Entry layer. Table 8 lists the fields common to all three levels and the fields specific to each. Levels differ in what they declare, not in how they are declared: an Atomic Skill declares the tool it invokes, an Expert Skill declares the workflow it expands into existing atomic tasks, and a Scenario Skill declares the rules by which it selects deliverables. None of them declares a service endpoint, a model name, or an API key. Figure 7 shows a specification in full. Specifications are written in the working language of their author; the specification in the figure is rendered in English for presentation.

D.2 Routing Precedence and Invocation Direction

Skill invocation is strictly downward. A Scenario Skill may invoke Expert or Atomic Skills, an Expert Skill may invoke Atomic Skills, and an Atomic Skill invokes no other Skill. Prohibiting upward and lateral calls keeps workflow expansion finite and the resulting DEG acyclic.

Table 6: Provider bindings in the Provider and Configuration layer. Alternatives are selectable by editing the corresponding configuration fields.

Capability	Default provider (model/backend)	Alternative (model/backend)	API Key
	Understanding		
Image	Host agent	—	—
Video	Google (`gemini-2.5-flash`)	—	`GEMINI_API_KEY`
Speech	Google (`gemini-2.5-flash`)	OpenAI (`whisper-1`)	`GEMINI_API_KEY` / `OPENAI_API_KEY`
Music and sound effects	Google (`gemini-2.5-flash`)	Alibaba (`Qwen2-Audio`)	`GEMINI_API_KEY` / `QWEN_API_KEY`
Document	Mistral AI (`mistral-ocr-latest`)	Host agent	`MISTRAL_API_KEY`
3D	Blender (`headless renderer`)	Local (`multi-view renderer`)	—
	Generation		
Image	OpenAI (`gpt-image-2`)	fal.ai (`fal-ai/flux/schnell`)	`OPENAI_API_KEY` / `FAL_API_KEY`
Video	Alibaba (`wan2.7`)	fal.ai (`Kling 1.6`)	`WAN_API_KEY`
Music	ElevenLabs (`music_v2`)	Replicate (`musicgen`)	`ELEVENLABS_API_KEY` / `REPLICATE_API_TOKEN`
Sound effects	ElevenLabs (`text_to_sound_v2`)	—	`ELEVENLABS_API_KEY`
Speech	OpenAI (`tts-1`)	Microsoft (`Edge TTS`)	`OPENAI_API_KEY`
3D	Tripo AI (`v2.5`)	Meshy (`API v1/v2`)	`TRIPO_API_KEY` / `MESHY_API_KEY`
PPT / Word / PDF / Excel	Local (`document renderers`)	—	—
Code / Markdown	Host agent	—	—
	Utility		
Web search	Brave (`Search API`)	—	`SEARCH_API_KEY`
Web browsing	Local (`headless browser`)	—	—

Table 7: Default generation parameters. Node-level `params` override these values, and the effective values are persisted in the asset record as `params`.

Capability	Parameter	Default
Image	`aspect_ratio` / `quality`	`1:1` / `standard`
Video	`duration_seconds` / `aspect_ratio`	`10` / `16:9`
Sound effects	`duration_seconds`	`15`
Music	`duration_seconds` / `instrumental`	`30` / `true`
Speech	`language` / `speed`	`zh-CN` / `1.0`

D.3 Skills Extension

Skill extension in Omni-IO Skills follows its hierarchical architecture. Atomic Skills add foundational executable capabilities, Expert Skills organize these capabilities into production workflows for concrete deliverables, and Scenario Skills compose lower-level Skills to support new application contexts. At all three levels, the extension is recorded in a specification that follows the format of Table 8 and is registered in the Skill Entry index. The extension procedure for each level is described below.

Atomic Skills extension. To add an Atomic Skill, the developer first encapsulates a capability that the host agent cannot execute natively as a new MCP tool and configures its provider binding. The developer then defines the operation's trigger conditions, inputs, execution procedure, and outputs in the Skill specification and binds the specification to the new tool.

Expert Skills extension. To add an Expert Skill, the developer specifies its target deliverable, decomposes the production workflow into existing Atomic Skills, and declares their dependencies. The workflow also defines final artifact assembly, inspection, and localized revision.

image_generation_skill.md

```
# Skill: Image Generation

## Trigger
A node of type "image" is present in the DEG.

## Tool binding
Configuration key : generate.image
MCP tool          : generate_image(prompt, aspect_ratio, quality)

## Inputs
prompt        <- node.prompt    complete, self-contained description
     of the image
aspect_ratio  <- node.params or configuration default
quality       <- node.params or configuration default

## Execution steps
1. Resolve parameters, filling omitted fields from the configuration
     defaults.
2. Invoke the bound provider through the MCP tool.
3. Write the returned image to the workspace output directory.
4. Register the asset and return the record.

## Outputs
An asset object of type "image" with an asset_id, a local path, a
    natural
language description, and the parameters that took effect.

## Interactions
downstream video : the local path is injected as the first video
    frame
downstream 3d    : the local path is injected as the reference image

## Error handling
missing credential : report the required key, mark the node failed
policy rejection   : report the rejected prompt, mark the node
    failed
network timeout    : retry once, then mark the node failed
```

Figure 7: A complete Atomic Skill specification of image generation. The specification names a configuration key rather than a provider, so rebinding the provider in Table 6 leaves the specification unchanged.

Scenario Skills extension. To add a Scenario Skill, the developer first defines its application context and trigger boundaries, then specifies the rules for selecting deliverables from a user request. For each deliverable, the Scenario Skill selects an appropriate Expert or Atomic Skill and declares cross-deliverable dependencies, parallelism, and asset reuse. A Scenario Skill coordinates existing lower-level Skills rather than introducing new executable operations.

E Expert Skill Production and Review Loop

E.1 Input and Expansion Contracts

An Expert Skill receives the deliverable goal and its use context; the available inputs and reusable assets; content constraints whose accuracy must be preserved; the output specifications and production requirements; and the stylistic direction together with any exclusions. It asks only for missing information that would materially affect the result and supplies defaults for adjustable details.

An Expert Skill distinguishes generative tasks from production operations that require precise control. Content suited to generation is delegated to the corresponding Atomic Skills, whereas elements whose

Table 8: Fields of a Skill specification. The first block is shared by all three levels; the remaining blocks are level-specific. A specification is loaded only when the entry layer selects it.

Field	Description
Shared by all levels	
Trigger condition	The request pattern or graph node type under which this Skill applies.
Trigger boundary	The neighbouring cases that must *not* route here, with the Skill that should handle them instead.
Inputs	The fields consumed, their source, and whether each is required.
Execution steps	The ordered procedure the host agent follows once the Skill is selected.
Outputs	The asset object produced.
Interactions	Which other Skills consume this Skill's output and what is injected downstream.
Error handling	Per-error-class recovery, covering missing credentials, policy rejections, and timeouts.
Atomic Skills only	
Tool binding	The configuration key and the MCP tool invoked, or the statement that execution is native.
Parameter defaults	The configuration defaults applied when the node omits a parameter.
Expert Skills only	
Expansion contract	The existing atomic task types the workflow expands into, and their dependency pattern.
Assembly step	The node that combines the generated assets into the final artifact.
Inspection criteria	What is checked on the assembled artifact and with which understanding capability.
Scenario Skills only	
Deliverable rules	How the set of deliverables is inferred from the request, and the convention applied when it is underspecified.
Level selection	For each deliverable, whether an Expert Skill or an Atomic Skill is invoked.
Cross-deliverable reuse	Which deliverables share assets, and which may proceed in parallel.

Table 9: Implemented Expert Skills. Each expands into existing atomic task types, assembles the final artifact through a `code` node, and inspects the result with the understanding capability appropriate to the artifact's modality.

Expert Skill	Workflow stages	Inspection
Poster Design	Lock the brief and exact copy → fix the visual direction → generate only the necessary visual assets → typeset copy and assets deterministically → inspect.	Native vision on the assembled image, checking copy fidelity, hierarchy, and legibility.
Complex Video Production	Script and storyboard → generate shots, narration, and music → assemble with subtitles and branding → inspect.	Video understanding on the assembled cut, with audio understanding where narration or music must be verified.

accuracy, structure, or timing must be guaranteed are handled through controllable content-processing or assembly steps. This division depends on the deliverable rather than on a fixed production pipeline.

An Expert Skill is not itself an execution unit. It decomposes the complete production workflow into existing executable tasks, declares their dependencies, and emits a flat task fragment. The fragment is merged with the rest of the graph and validated and scheduled by the common execution procedure. The two implemented Expert Skills instantiate this contract through deliverable-specific production stages and modality-appropriate inspection, as summarized in Table 9. Figure 8 illustrates this expansion for poster production.

E.2 Assembly

Assembly uses a controllable programmatic workflow to combine generated content and existing assets into the final artifact. A poster is typeset programmatically over its generated background so

Table 10: Defect attribution and the corresponding minimal rework. Validated assets are reused across rework rounds, so a localized defect never triggers regeneration of the whole artifact.

Responsible layer	Symptom	Minimal rework
Atomic asset	A generated visual or shot is off-brief, malformed, or carries spurious text.	Re-run that atomic node only, then re-assemble.
Written content	Copy, narration, or lyrics misstate a locked fact.	Re-run the affected content node, then re-assemble.
Assembly	Layout, cut, subtitle timing, transition, or mix is wrong while the assets are correct.	Modify the `code` node only; no generation call is issued.
Planning	The deliverable itself was misconceived or the brief was incomplete.	Return to the user with the specific decision required.

that copy is rendered with exact glyphs rather than sampled from a generative model; a video is cut, mixed, and subtitled by a local media pipeline.

E.3 Inspection and Localized Rework

An Expert Skill does not deliver on the completion of its atomic nodes. The assembled artifact is read back with the understanding capability of its own modality and checked against the locked brief. When a defect is found, it is localized to the smallest responsible part of the workflow, and only that part is reworked. Table 10 summarizes the corresponding actions.

After rework, assembly must use the newly generated assets rather than continue referencing rejected versions. Each new artifact is inspected again. Autonomous rework is limited to two rounds by default; if the issue remains unresolved, the Expert Skill reports the outstanding problem, delivers the best available version, and asks the user for the smallest decision needed to resolve it.

poster_expert_fragment.json

```
{
  "tasks": [
    { "id": "poster_bg", "type": "image",
      "prompt": "Contemporary music-festival poster
    background, deep indigo and
                 fluorescent orange, abstract sound waves
    carrying the rhythm,
                 subject weighted to the lower half, large
     clean negative space
                 across the upper half, no letters, digits,
     logos, watermarks
                 or pseudo-text of any kind",
      "params": { "aspect_ratio": "3:4" }
    }, { "id": "poster_logo", "type": "image",
      "asset_ref": "img_7c1e04"
    }, { "id": "poster_final", "type": "code",
      "prompt": "Typeset the locked copy over the
    background, place the logo in
                 the lower right margin, export at 300 dpi
     as PNG and PDF",
      "depends_on": ["poster_bg", "poster_logo"] }
  ]
}
```

Figure 8: An Expert Skill expansion fragment for a poster. Only existing task types appear: the visual asset is an `image` node, a registered logo enters as an asset-source node, and assembly is a `code` node that depends on both. Task identifiers name the deliverable so that fragments from several Expert Skills can be merged without collision.

F Scenario Skill Deliverable Planning and Coordination

F.1 Input and Planning Contracts

A Scenario Skill receives the user's objective, source materials, reusable assets, explicitly requested deliverables, and constraints shared across outputs. It first records the requirements shared by all deliverables in a common scenario brief, then prepares a separate production brief for the specific requirements of each deliverable. Explicit user scope is preserved; when the output form is unspecified, the Skill selects the smallest conventional set that satisfies the objective. It asks for clarification only when ambiguity would materially affect scope, cost, or intended use. Table 11 lists the candidate outputs of the six implemented Scenario Skills.

Table 11: Deliverable-selection domains of the implemented Scenario Skills.

Scenario Skill	**Application boundary**	**Candidate deliverables**
Social-Media Post	Platform-facing publication or promotion around supplied or generated source material.	Post copy, supporting image or poster, short video, and optional audio.
Office Documents	Transformation of source material into structured workplace artifacts.	Presentation, report, PDF, minutes, and spreadsheet.
Job Application	A coordinated application package for a target role.	Resume, cover letter, self-introduction presentation, speech, and video.
Education Sharing	Teaching or explanation for a specified audience and learning objective.	Slides, handout, illustration, narration, and explainer video.
Event Material	Promotion, participation, or retrospective material for one event.	Invitation, poster, atmosphere music, teaser video, and retrospective video.
Game Asset	A coordinated presentation of one game concept or production asset.	Concept art, 3D model, showcase video, and optional web presentation.

For each deliverable, the Scenario Skill chooses the lowest Skill level capable of completing the required procedure. Tasks that require coordinated asset generation, deterministic assembly, artifact-level inspection, and localized rework are delegated to an Expert Skill; tasks that can be completed as one independent operation are delegated directly to an Atomic Skill. The expanded task fragments are merged into the DEG, and shared constraints are written into every relevant brief.

Consider a forest-elf game-asset request containing a character illustration, a short turntable clip, a 3D model, and a single-page showcase, with an existing background track reused. Figure 9 shows the resulting deliverable plan: the illustration serves as the shared identity anchor; the video and 3D model run in parallel after it completes; the background track is reused directly; and the showcase page aggregates all assets at the end.

F.2 Cross-Deliverable Coordination and Reuse

The common scenario brief is the consistency anchor for the package. Locked facts, names, dates, claims, brand requirements, and exclusions are propagated unchanged to every affected deliverable. Stylistic direction is specialized only where the output medium requires it, without changing the common identity of the package. For example, a social-media caption and a promotional video may use different lengths and pacing while preserving the same product facts and campaign direction.

Source analysis and generated assets are represented once whenever several deliverables consume the same result. If a downstream output must read or transform an upstream artifact, the dependency is declared explicitly. If two outputs share only the common scenario brief and consume no common artifact, their prompts are made self-contained and the branches may proceed in parallel. This distinction preserves both asset identity and the parallelism available to the Wave scheduler.

F.3 Completion and Revision

After execution, the Scenario Skill performs a package-level completion and check. It verifies that every selected deliverable has a terminal state, that completed outputs preserve the locked facts and declared relationships, and that intended reuse points reference the selected asset versions. If a shared dependency fails, dependent deliverables become unavailable while independent branches

game_asset_scenario_plan.json

```
{
  "scenario": "game_asset",
  "shared_brief": "Forest elf, emerald hair, silver light
    armour,
                    stylised realism, consistent identity
    across outputs",
  "deliverables": [
    { "id": "character_key_art", "invoke": "A6 Image
    Generation"
    }, { "id": "turntable_clip", "invoke": "A7 Video
    Generation",
      "reuse": ["character_key_art"]
    }, { "id": "character_model", "invoke": "A11 3D
    Generation",
      "reuse": ["character_key_art"]
    }, { "id": "background_track", "asset_ref": "
    mus_b4e8d2"
    }, { "id": "showcase_page", "invoke": "A16 Code
    Generation",
      "reuse": ["character_key_art", "turntable_clip",
                "character_model", "background_track"] }
  ]
}
```

Figure 9: A Scenario Skill deliverable plan for a forest-elf game-asset package. The plan records lower-level Skill invocations and asset-reuse relationships before being expanded into a DEG.

may still complete. The Scenario Skill reports the package as complete, partial, or failed and lists any unavailable deliverables. User revisions are applied according to their scope. Local changes affect only the target deliverable and its descendants, while changes to locked facts, shared assets, or global direction update all affected briefs and dependencies. Completed assets remain available, and revised branches register new versions.

G Declare Execution Graph Specification

G.1 Reference Orchestration Procedure

Algorithm 1 records the reference orchestration procedure underlying Figure 5. The procedure makes explicit how Omni-IO Skills expansion produces terminal tasks, how a DEG is validated and scheduled in parallel Waves, and how successful outputs and failure states are propagated through the Asset Registry.

The procedure maintains separate control and asset states. Node status determines which tasks become schedulable or cancelled, while the registry records only successfully materialized artifacts. This separation allows independent graph branches to continue after a local failure and makes completed assets available to later nodes and future turns.

G.2 Node Schema

Table 12 lists the fields of a DEG node. A node is either a generation node carrying a `prompt` or an asset-source node carrying an `asset_ref`. The two fields are mutually exclusive. Prompts are self-contained and describe everything a node needs without referring to another node's prompt. Semantic consistency across concurrent nodes is established by writing their prompts within the same planning context.

G.3 Task Types and Execution Sites

Table 13 lists the admissible node types. Most types correspond to one MCP tool call with a fixed parameter signature, and therefore cannot accept a variable number of upstream assets; `code` and

Algorithm 1 Dependency-aware orchestration in Omni-IO Skills.

Require: user request q, Skill index $\mathcal{S}$, Asset Registry $\mathcal{R}$
Ensure: outputs and final execution states of all nodes

$\mathcal{T} \leftarrow \textsc{SkillEntry}(q, \mathcal{S})$ ▷ selected Skills expanded into terminal tasks
$G = (V, E)$ with $V \leftarrow \mathcal{T}$ and E induced by `depends_on`
if $\textsc{Validate}(G, \mathcal{R})$ fails **then**
  return planning error ▷ unresolved reference or dependency cycle
end if
$\text{status}[v] \leftarrow \textsc{Pending}$ for all $v \in V$
for all $v \in V$ with `asset_ref` present **do**
  $o[v] \leftarrow \mathcal{R}.\textsc{Get}(v.\texttt{asset_ref})$; $\text{status}[v] \leftarrow \textsc{Completed}$
end for
while some $v \in V$ has $\text{status}[v] = \textsc{Pending}$ **do**
  $\mathcal{W} \leftarrow \{\, v \in V : \text{status}[v] = \textsc{Pending}$ and all predecessors completed $\}$
  for all $v \in \mathcal{W}$ **in parallel do**
    $\pi \leftarrow \textsc{ResolveProvider}(v.\texttt{type})$
    $\theta \leftarrow \pi.\text{defaults} \oplus v.\texttt{params} \oplus \textsc{Inject}\big(o[\text{pred}(v)]\big)$
    $o[v] \leftarrow \textsc{Execute}(\pi, v.\texttt{prompt}, \theta)$
    if execution succeeded **then**
      $\mathcal{R}.\textsc{Register}(o[v])$; $\text{status}[v] \leftarrow \textsc{Completed}$
    else
      $\text{status}[v] \leftarrow \textsc{Failed}$
      mark all pending descendants cancelled ▷ independent nodes continue
    end if
  end for
end while
return (o, status)

Table 12: Fields of a DEG node. An asterisk marks the two mutually exclusive fields. Exactly one of `prompt` and `asset_ref` is present. Upstream file paths are supplied by the scheduler at invocation time, so a prompt never contains a path or an asset identifier.

Field	Type	Req.	Meaning
`id`	string	✓	Unique identifier within the graph; the target of `depends_on`.
`type`	string	✓	Task type, drawn from Table 13.
`prompt`	string	*	Complete, self-contained description of what this node must produce.
`params`	object	✗	Parameter overrides applied on top of the configuration defaults.
`depends_on`	string | array	✗	Upstream node identifiers; an array is admissible only for `code` and `markdown`.
`asset_ref`	string	*	Identifier of a previously registered asset; makes this an asset-source node.

Table 13: Admissible node types and where they execute. Only the two natively executed types can consume multiple upstream assets, and only they require explicit registration.

Type	Execution site	Multi-dependency	Auto-registered
`image`, `video`, `3d`	MCP tool	✗	✓
`music`, `sfx`, `speech`	MCP tool	✗	✓
`ppt`, `word`, `pdf`, `excel`	MCP tool (local renderer)	✗	✓
`code`, `markdown`	Host agent native file writing	✓	✗
existing asset (`asset_ref` present)	None; resolved from the registry	✗	—

`markdown` are written natively by the host agent, which resolves each upstream identifier through `get_asset` before writing, and are consequently the only types for which a dependency array is meaningful.

G.4 Validation Rules

Before any tool is invoked, the graph is checked against the rules in Table 14. Each rule imposes a structural requirement on the declaration that is decidable without executing anything. A graph that

Table 14: Validation rules applied to the DEG before execution. Every rule is decidable from the declaration alone. Failure returns the graph to planning rather than to execution.

Rule	What it demands	What it prevents
Identifier uniqueness	Every `id` occurs once in the graph.	A dependency resolving to the wrong node.
Type admissibility	Every `type` appears in Table 13.	An Expert Skill inventing a node type that no execution site can run.
Node completeness	Exactly one of `prompt` and `asset_ref` is present, and a `prompt` is self-contained.	A generation call issued with an empty or context-dependent description.
Dependency resolution	Every identifier in `depends_on` names a node in the same graph.	A node waiting forever on a task that was never declared.
Asset resolution	Every `asset_ref` names a record in the registry.	An asset-source node resolving to a path that does not exist.
Acyclicity	The graph induced by `depends_on` contains no cycle.	A Wave partition that never terminates.
Dependency arity	An array `depends_on` occurs only on `code` and `markdown` nodes.	A fixed-signature tool call being handed a variable number of upstream assets.
Conservative declaration	A dependency is declared whenever a node consumes an upstream output, references its content, or must remain semantically consistent with it.	A downstream node starting before the asset it depends on exists.

violates any rule is a planning error and is reconstructed rather than submitted for execution. When it is unclear whether a dependency is required, the system declares it.

G.5 Failure and Interruption Semantics

Node failure and user interruption both cancel affected downstream work without rebuilding the graph. A failed node is marked failed, its pending descendants are cancelled, and independent branches continue. Failed and cancelled nodes are reported but not registered.

User instructions modify the graph in flight. For each pending or in-flight node, the host agent determines whether its output is affected. Affected in-flight nodes are cancelled and re-issued, affected pending nodes are edited before dispatch, and unaffected nodes continue. Completed assets remain registered. Table 15 summarizes the handling of five interruption classes.

Table 15: Interruption classes and their handling. Completed assets remain registered; only pending and in-flight nodes are affected.

Class	Recognized by	Handling
Local revision	One deliverable changes.	Update the target; pending descendants wait for its new output, while completed descendants remain linked to the prior version.
Global revision	A shared scene, style, or tone changes.	Cancel and re-issue affected in-flight nodes; regenerate completed assets only on request.
Extension	A deliverable is added.	Insert new nodes; dispatch independent nodes immediately and schedule dependent nodes in later Waves.
Cancellation	A deliverable is withdrawn.	Cancel the node and its pending descendants; register no output for them.
Topic switch	The instruction is unrelated to the graph.	Confirm, cancel remaining work, and plan a new graph; keep completed assets.

G.6 Worked Example

Figure 10 declares a graph for a request that combines a reused asset, two independent generations, and an aggregation, and Table 16 gives the resulting Wave partition. Two properties are visible. The asset-source node occupies Wave 0 as an already completed node and issues no generation call, so reusing a previous turn's output costs nothing. The two nodes of Wave 1 are mutually independent and are dispatched in a single concurrent round, and their failure modes are independent: were the 3D node to fail, the video node would still complete, and only the aggregation node would be cancelled.

Table 16: Wave partition of Figure 10. Nodes in the same Wave are mutually independent and are dispatched in one concurrent round. The asset-source node is complete on entry and issues no call.

Wave	Nodes	Calls issued
0	`img1`, `mus1`	one image generation; `mus1` resolves from the registry
1	`vid1`, `3d1`	one video and one 3D generation, concurrently
2	`code1`	none; written natively after resolving four asset paths

game_asset_deg.json

```
{
  "tasks": [
    { "id": "img1", "type": "image",
      "prompt": "Full-body character illustration of a forest
    elf, emerald hair,
                  silver light armour, front view, plain
    background, game concept
                  art style",
      "params": { "aspect_ratio": "3:4" } },
    { "id": "mus1", "type": "music",
      "asset_ref": "mus_b4e8d2" },
    { "id": "vid1", "type": "video",
      "prompt": "Slow orbit around the standing character,
    soft shifting forest
                  light, no camera cuts",
      "depends_on": "img1", "params": { "duration_seconds": 10
     } },
    { "id": "3d1", "type": "3d",
      "prompt": "Forest elf character, emerald hair, silver
    light armour, T-pose,
                  stylised realism",
      "depends_on": "img1" },
    { "id": "code1", "type": "code",
      "prompt": "Single-page asset showcase embedding the
    illustration, the turntable
                  clip, the model viewer and the background
    track",
      "depends_on": ["img1", "vid1", "3d1", "mus1"] }
  ]
}
```

Figure 10: A DEG combining cross-turn reuse, parallel generation, and aggregation. The video and 3D nodes both consume the character illustration, and the page node aggregates all four assets.

The product-promotion workflow in Figure 5 provides a complementary four-Wave example:

Table 17: Illustrative four-Wave DEG execution for the product-promotion case in Figure 5.

Stage	Node(s)	Dependency and output
Wave 0	Analyze product reference images	Extracts product appearance and visual constraints.
Wave 1	Generate poster assets	Generates poster-ready visuals from the product analysis.
	Generate video and sound effects	Generates video and sound-effect assets in parallel.
Wave 2	Generate promotional video and poster	Combines assets into the promotional video and poster.
Wave 3	Generate landing web page	Assembles a landing page from the promotional outputs.

H Asset Registry Schema and Cross-Turn Reuse

H.1 Record Schema

Table 18 lists the fields of a registry record. The `description` field is written at generation time and supports semantic retrieval in subsequent turns. The `turn_id`, set once per user message, disambiguates between assets of the same type produced in different turns.

Table 18: Fields of an Asset Registry record. Records are keyed by `asset_id` and persisted as a single JSON file in the workspace.

Field	Meaning
`asset_id`	Globally unique identifier, formed from a type prefix and a random suffix.
`type`	`image`, `video`, `audio`, `3d`, `document`, or `code`.
`subtype`	Finer classification: `music`, `sfx`, `speech`; `ppt`, `word`, `pdf`, `excel`, `markdown`; `poster`, `produced_video` for assembled artifacts.
`path`	Absolute local path of the file in the workspace output directory.
`description`	Natural language description written at generation time, used for semantic retrieval in later turns.
`params`	The parameters that actually took effect, after defaults and node-level overrides were merged.
`turn_id`	Identifier of the interaction turn, set once per user message.
`source_asset_id`	Optional lineage pointer to the record this asset was derived from.

H.2 Registration Paths

Assets reach the registry along two paths. Outputs of MCP generation tools are registered by the tool service before the call returns, so the host agent performs no bookkeeping. Files written natively by the host agent, namely `code` and `markdown` nodes and the artifacts assembled by Expert Skills, are registered by an explicit call after the file is written.

H.3 Concurrency and Versioning

The registry is a JSON file in the workspace, shared across sessions. Writes are atomic, performed to a temporary file and renamed into place, and serialized with a file lock on platforms that provide one, so that concurrent sessions registering into the same workspace do not overwrite one another.

Records are appended, never updated in place. Regenerating a deliverable produces a new record with a new `asset_id` that may point back at the record it was derived from, so successive versions of the same artifact remain distinguishable and individually retrievable.

H.4 Cross-Turn Reuse

Figure 11 shows the mechanism end to end. In the first turn a node generates an image and its record enters the registry. In a later turn, possibly in a different session, the user refers to that image; the host agent locates the record by semantic match over the stored descriptions, and materializes it in the new graph as an asset-source node. That node is complete on entry, issues no generation call.

I Experiment Details

I.1 Construction of UniM-90

UniM-90 is a fixed subset of 90 instances drawn from UniM (Li et al., 2026c). We sample one easy, one medium, and one hard case from each of UniM's 30 domains. If a difficulty level is unavailable, it is replaced by an additional case from another available level within the same domain, yielding 90 cases in total.

I.2 Configuration

The *Base Agent* configurations retain only the Skills natively built into Codex and Claude Code, respectively, with no additional Skills installed, whereas the *Agent + Omni-IO Skills* configurations

cross_turn_reuse.json


```
{
  "turns": [
    { "turn_id": "turn_20260618_1437",
      "graph": { "tasks": [
          { "id": "img1", "type": "image",
            "prompt": "Tokyo street at 2 a.m., neon on wet asphalt,
    low angle, no figures" }
        ] },
      "record": { "asset_id": "img_a3f2c1", "type": "image",
        "path": "<workspace>/img_a3f2c1.png",
        "description": "Night Tokyo street, low angle, cold blue
    palette, wet neon asphalt, no figures.",
        "params": { "aspect_ratio": "16:9" },
        "turn_id": "turn_20260618_1437"
      }
    }, { "turn_id": "turn_20260619_0902",
      "user": "Turn that night-street image from yesterday into a
    short clip.",
      "graph": { "tasks": [
          { "id": "img1", "type": "image", "asset_ref": "img_a3f2c1"
     }, {
            "id": "vid1", "type": "video",
            "prompt": "Slow dolly along the street, no figures, rain
     continuing",
            "depends_on": "img1", "params": { "duration_seconds": 10
     } }
        ] },
      "waves": {
        "wave_0": { "node": "img1", "action": "resolve_from_registry
    " },
        "wave_1": { "node": "vid1", "tool": "generate_video",
          "first_frame_url": "<workspace>/img_a3f2c1.png" }
      }
    }
  ]
}
```


Figure 11: Cross-turn reuse represented as a JSON object. The second turn regenerates nothing: the referenced asset enters the new graph as a completed node, and only the video node issues a call.

add Omni-IO Skills and its associated MCP tool services while keeping all other settings, including tool configurations, system prompts, and execution budgets, unchanged. The provider bindings in Table 6 are held fixed across all instances and both agents, ensuring that the observed differences are not attributable to different underlying generators. This controlled setup therefore isolates the contribution of Omni-IO Skills.