CODEOMNI: A CODE-DRIVEN HARNESS FOR EXTENSIBLE OMNI-MODAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

The code-as-action/tool paradigm has been widely explored in prior work. However, existing approaches either underutilize the expressiveness of code or remain tailored to a single scenario. To address these limitations, we propose **CodeOmni**, a code-driven harness that uses executable programs to orchestrate interactions between models and harness components, providing a unified execution substrate for image, video, audio, and audio-visual tasks. In CodeOmni, frozen models generate Python programs that jointly support tool orchestration, control flow, exact computation, and task-local state management. A variable-mediated workspace further stores intermediate artifacts outside the main conversational context and exposes them only when needed, reducing unnecessary context accumulation. New tasks can be incorporated through lightweight runtime profiles and task-grounded skills, which guide tool selection and program structure. Extensive experiments involving eight models and seven multimodal benchmarks show that unconstrained code generation alone is insufficient and can lead to suboptimal performance. In contrast, equipping the harness with task-grounded skills consistently improves performance over direct model inference in most evaluation settings. As a preliminary exploration of code-driven harnesses, CodeOmni demonstrates the potential of executable code as a shared and extensible substrate for multimodal model interaction, and provides insights toward the development of more capable and general harnesses.

1 INTRODUCTION

Omni-modal foundation models provide a unified interface for text, images, audio, and video (Xu et al., 2025). However, unified perception alone does not determine how a model should search a long video, isolate relevant audio, invoke exact computation, preserve intermediate results, or verify evidence. Such operations often require specialized tools, deterministic algorithms, and reusable task-local state. The effectiveness of an omni-modal system therefore depends not only on the underlying model, but also on the harness that coordinates perception, reasoning, tools, and state.

Code offers a natural representation for this harness. Tool-using agents commonly express actions through textual commands or function calls (Yao et al., 2023; Schick et al., 2023), while program-aided methods externalize computation into executable programs (Chen et al., 2022). CodeAct (Wang et al., 2024) further treats Python as a unified action space for tool composition, control flow, state preservation, and runtime correction. Multimodal systems such as VisProg (Gupta & Kembhavi, 2023), ViperGPT (Suris et al., 2023), CodeDance (Song et al., 2026), and MUSE (Lu et al., 2026) similarly demonstrate that executable procedures can combine learned perception with symbolic operations and verification. However, existing approaches are often specialized to a particular modality, depend on post-training, or rely on task-specific interaction protocols.

In this work, we introduce CodeOmni, a training-free, code-driven harness for extensible omni-modal reasoning. A frozen model generates Python programs over registered perception tools, media-processing functions, deterministic algorithms, and modality-specialized sub-agents. Code serves as a unified interface for three purposes: composing actions, performing exact computation, and preserving intermediate state across execution steps. In addition, CodeOmni uses a variable-mediated workspace to separate computational state from the language context. Python variables retain in-process values, while dedicated stores maintain structured values and large artifacts such

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.

as frames, clips, and audio files. The planner receives compact execution feedback and retrieves detailed artifacts only when needed, therefore largely reducing unnecessary context growth.

The framework further separates *runtime policy* from *task guidance*. Runtime profiles define allowed tools, evidence requirements, execution budgets, and output constraints. Frozen task skills provide benchmark-level strategies or executable wrappers that narrow the program-synthesis space without modifying the shared planner–executor loop. New modalities are supported by registering tools and sub-agents, while new tasks are integrated through profiles, skills, adapters, and scorers.

Extensive experiments reveal an important boundary of code-driven reasoning. Across four image benchmarks and eight models, the profile-only harness does not consistently outperform direct inference because the enlarged action space can lead to ineffective decompositions and unnecessary execution errors. In contrast, pairing the shared executable core with task-grounded skills produces substantial and broadly consistent improvements. Similar improvements are observed on audio and audio-visual tasks, whereas gains on Video-MME (Fu et al., 2025) remain modest because strong native video models already solve many instances without additional orchestration. These results show that free-form code enlarges the planning space and introduces failures in decomposition, tool selection, syntax, and execution, whereas fixed task pipelines limit extensibility. A practical harness must therefore retain the flexibility of code while constraining execution through reusable task knowledge and explicit runtime policies.

Our contributions are summarized as follows:

- **A unified code-driven harness for omni-modal reasoning.** We introduce CodeOmni, a training-free framework that uses model-generated Python as a shared execution substrate for frozen models across image, video, audio, and audio-visual tasks, without redesigning the central agent loop for each modality or benchmark.
- **A state-aware and extensible execution architecture.** CodeOmni unifies tool orchestration, deterministic computation, and task-local state within executable programs. Its variable-mediated workspace, registered tools, and modality-specialized sub-agents preserve reusable intermediate artifacts while limiting unnecessary growth of the main conversational context.
- **Profile- and skill-guided orchestration with systematic empirical validation.** We separate hard runtime policies from reusable task-grounded skills to constrain the open program-synthesis space while retaining the flexibility of code. Experiments across seven image, video, audio, and audio-visual benchmark settings and eight models show that unguided code execution can be unreliable, whereas skill-guided execution substantially improves accuracy and efficiency on tool-amenable tasks.

2 Related Work

2.1 Code-Driven Agents

CodeAct (Wang et al., 2024) first uses executable Python as a unified action representation that can combine multiple tools, preserve variables, and revise behavior from runtime feedback. Recent code-agent systems explore planning granularity, training, recursion, and runtime state. L0 (Zhang et al., 2025b) combines reinforcement learning with a concurrent sandboxed REPL agent scaffold. ReCode (Yu et al., 2025) represents high-level plans as abstract functions that are recursively decomposed into executable actions. Recursive Language Models (Zhang et al., 2025a) programmatically inspect long externalized inputs and invoke sub-model calls over selected portions. CaveAgent (Ran et al., 2026)introduces a dual-stream architecture in which a persistent Python runtime stores complex objects across turns, reducing dependence on the textual context. These works are closely related to CodeOmni's use of code and external state, but they primarily target general text-centric agent tasks, long-context processing, or learned agent policies. CodeOmni instead studies a frozen-model harness in which media artifacts, modality tools, deterministic solvers, and sub-agents share a common executable workspace.

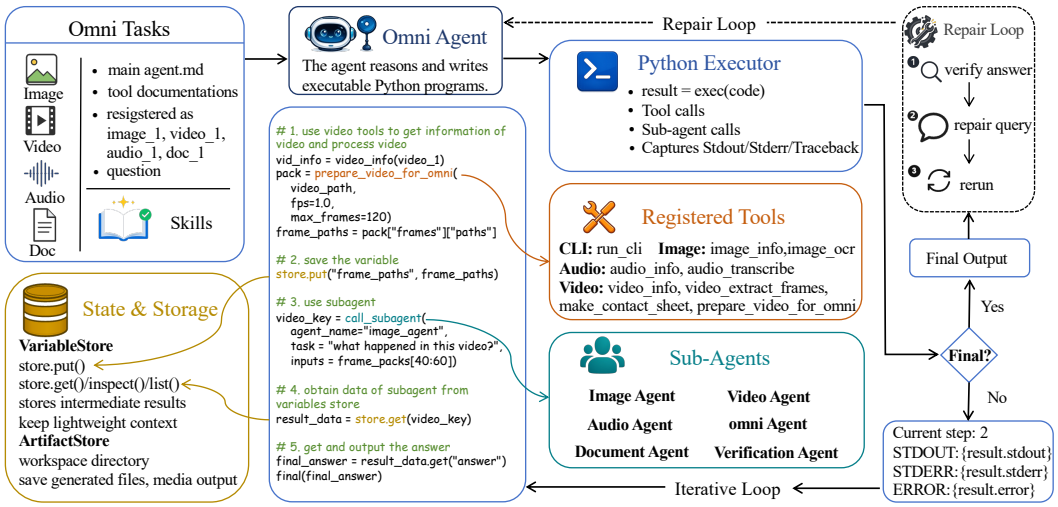


Figure 1: Overview of the proposed CodeOmni harness. The Omni Agent receives task instructions, multimodal inputs, and task-specific skills, then generates executable Python programs. Python executors handle code and call registered tools and specialized sub-agents. Intermediate results and generated artifacts are maintained through the state and storage modules, enabling lightweight context for main agent. The framework iteratively inspects execution outputs, including standard output, standard error, and tracebacks, until a final answer is produced. A repair loop further checks the answer, constructs a repair query, and reruns the workflow to improve the final output.

2.2 Executable Multimodal Reasoning and Agentic Harnesses

Early multimodal tool systems connect language models to specialized perception modules. MM-ReAct (Yang et al., 2023) prompts a language model to coordinate a pool of vision experts through textual descriptions, coordinates, and file references. VisProg (Gupta & Kembhavi, 2023) generates modular visual programs whose intermediate steps call vision models, image-processing routines, and Python functions. ViperGPT (Suris et al., 2023) similarly uses a code-generation model to compose vision-and-language modules through executable Python. CodeDance (Song et al., 2026) extends executable visual reasoning through a dynamically tool-integrated MLLM trained to balance tool use, compose operations, and produce self-checkable visual artifacts. OmniAgent (Xing et al., 2026) formulates omni-modal understanding as an iterative observation–thought–action process and accumulates selectively acquired audio-visual evidence in persistent textual memory. CodeOmni shares the motivation of using code as a flexible multimodal reasoning medium, but differs in that it constructs a comprehensive, code-centric harness and adapts it to a diverse range of omni tasks. MUSE (Lu et al., 2026) wraps frozen MLLMs with unified task representation, visual processing, structured parsing, deterministic verification, and verifier-guided repair. It organizes composable modules around heterogeneous visual tasks, whereas CodeOmni makes generated Python the common control, computation, and state interface across visual, video, audio, and audio-visual settings.

3 The CodeOmni Harness

3.1 Design Principle and Overview

CodeOmni follows *core-preserving extensibility*: adding a modality or benchmark should not require rewriting the planner–executor loop. It separates a shared executable core from task-dependent policy and guidance: Shared Core + Runtime Profile + Registered Tools + Task Skill. A frozen backbone generates Python programs that execute in a task-local environment and return compact observations for the next step. Registered tools expose media processing, deterministic computation, and model-backed evidence analysis. A runtime profile constrains available tools and evidence requirements, while a task skill narrows the open program-synthesis space toward a reliable procedure. As shown in Figure 1, for each task, a runtime profile P selects the allowed tool set $\mathcal{T}_P$,

allowed sub-agent set $\mathcal{A}_P$, step and repair budgets, evidence requirements, and output constraints. At execution step t, the planner produces a Python program p_t. The program is executed in the registered environment and the current task-local workspace W_t:

$$(o_t, W_{t+1}) = \mathrm{Exec}(p_t; \mathcal{T}_P, \mathcal{A}_P, W_t).$$

The observation o_t contains compact execution feedback, including standard output, standard error, and an exception traceback when execution fails. The updated workspace remains available to later programs in the same attempt. Execution terminates when the program calls `final(...)` or writes a valid final answer.

3.2 Code as Action, Computation, and State

The planner writes executable Python, allowing code to simultaneously serve as action, computation, and state. First, code provides a concise and compositional representation for flexible workflow orchestration: a program can call multiple tools, iterate over artifacts, branch on intermediate results, invoke sub-agents, and integrate evidence within a unified execution process. Second, code connects learned model capabilities with deterministic algorithms and predefined functions, including graph search, string matching, arithmetic operations, and timestamp-based media extraction, making these operations explicit, reliable, and reproducible. Third, code naturally supports persistent state management, as variables, imports, intermediate artifacts, and helper functions can be retained and reused across execution steps without repeatedly translating them into natural language. For example, when processing a long video, the program can first construct a coarse temporal index and subsequently inspect only the intervals most relevant to the task.

3.3 Variable-Mediated Workspace

CodeOmni maintains three related state layers. First, the Python executor uses a shared global namespace within an attempt, so variables and imports created by earlier programs remain available to later programs. Second, a `VariableStore` keeps real Python values together with lightweight metadata such as type, summary, preview, and creation time. Third, an `ArtifactStore` persists generated frames, clips, audio files, contact sheets, spectrograms, and other large outputs on disk. Inputs are registered as Python variables such as `image_1` and as store entries. New values are not automatically serialized into the dialogue; the planner sees a compact inventory and explicitly calls `store.list()`, `store.inspect()`, or `store.get()` when needed. Large objects remain reusable without repeated context insertion.

3.4 Runtime Profiles and Task Skills

Free-form code creates a large and error-prone action space. CodeOmni controls it through two mechanisms. The first is runtime profiles. A profile specifies hard capability and evidence policy: tool and sub-agent allow-lists, required evidence groups, step and repair budgets, answer format, and repair hints. The implementation contains 7 distinct profiles across image, video, audio, and audio-visual tasks. The second mechanism is task skills. A skill supplies task-grounded strategy, output constraints, or an executable wrapper. A *prompt skill* leaves program synthesis to the model. An *exact executable skill* contains a reserved marker and frozen Python block that is run at the first step; later steps return to model-generated code if needed. Deterministic helpers implement low-variance components such as BFS, seam comparison, pixel-grid search, or media preparation. The profile enforces what a run may and must do, whereas the skill suggests or instantiates how. The resulting interaction can be summarized as: Profile enforces policy + Skill narrows execution + Code preserves composition. This organization directly reflects the empirical finding that access to code and tools alone is not reliably beneficial: the executable substrate becomes effective when the search space is constrained by reusable task-grounded procedures.

3.5 Registered Tools and Evidence Sub-Agents

The registry exposes 32 Python tools covering command execution, image/OCR and spreadsheet processing, media probing and conversion, frame/clip extraction, synchronized evidence packs, temporal indexing, ASR, acoustic visualization, *etc.* CodeOmni also exposes six sub-agents:

`image_agent`, `audio_agent`, `video_agent`, `omni_agent`, `document_agent`, and `verification_agent`. A program calls one sub-agent with selected inputs; store keys are materialized, local media are collected, and the result is written back to the same store. The first four use modality-oriented prompts, while document and verification use a generic evidence prompt. Sub-agents reuse the planner model by default but may use heterogeneous backends. These tools and sub-agents are registered as callable variables, allowing the model to invoke them conveniently through function calls. By offloading specialized processing from the main agent, they keep its context concise while extending its capability boundaries.

3.6 Iterative Execution, Verification, and Repair

Within an attempt, CodeOmni follows a generate–execute–observe loop. The planner receives the public task, tool documentation, profile/skill instructions, and a compact variable inventory, then generates one Python block. Execution feedback—stdout, stderr, and tracebacks—becomes the next observation, enabling code repair and evidence refinement. The current attempt terminates when either the maximum number of iterative steps is reached or a call to the `final(...)` function is detected. After an attempt, a deterministic verifier checks execution, parses and normalizes the answer, compares it with a reference when available, and verifies required evidence groups from the trace. Benchmark runners may replace exact matching with task scorers such as valid-path or seam-based acceptance. Failures are categorized as format, task-logic, evidence-mismatch, or execution errors. Following Lu et al. (2026), we incorporate a repair loop into CodeOmni, it converts verifier feedback into a fresh attempt. The query contains the unchanged public task, compressed failure pattern, recent answers, and profile hints, but not the correct answer. A new agent and workspace are created and the same skill is reinjected.

4 Experiments

4.1 Experimental Setup

4.1.1 Benchmarks.

We evaluate CodeOmni on four image benchmarks including VSP-Grid (Wu et al., 2025), CoMT (Cheng et al., 2025), BLINK-Jigsaw (Fu et al., 2024), TIR-Bench Word Search (Li et al., 2025), a video benchmark Video-MME (Fu et al., 2025), an audio benchmark MMAU Test-mini (Sakshi et al., 2025) and an audio-visual benchmark AVUT (Yang et al., 2025). VSP-Grid presents a rendered grid environment containing a player, a goal, and hazardous cells, and requires the model to generate a sequence of discrete moves that reaches the goal without entering a hazard. For CoMT, we use the creation split, which contains multiple-choice multimodal reasoning problems that predominantly involve plane-geometry inference from visual and textual inputs. The Jigsaw task from BLINK presents an incomplete image and two candidate patches, and asks the model to select the patch that best completes the missing region. The Word Search task from TIR-Bench presents a dense grid populated with visually similar glyphs, including Chinese characters, Latin letters, or digits. Given a designated target glyph, the model must either count its occurrences or report its row and column position. Video-MME evaluates question answering over videos spanning diverse domains and a broad range of durations. MMAU Test-mini evaluates audio comprehension, information extraction, and reasoning through multiple-choice questions paired with audio clips spanning speech, sound, and music. For AVUT, we evaluate three multiple-choice audio-visual alignment tasks: Audio-Visual Character Matching (AVCM), Audio-Visual Object Matching (AVOM), and Audio-Visual Text Matching (AVTM).

4.1.2 Models.

We evaluate 8 models, comprising 5 open-source Qwen models and three proprietary models from OpenAI and Google. The open-source group includes Qwen3.5-27B (Qwen Team, 2026a), Qwen3.5-35B (Qwen Team, 2026a), Qwen3.6-27B (Qwen Team, 2026b), Qwen3.6-35B (Qwen Team, 2026c), and Qwen3-Omni-30B (Xu et al., 2025), covering multiple Qwen model series and capacity levels, with Qwen3-Omni-30B providing an additional omni-modal variant. The proprietary group consists of GPT-5.4-mini (OpenAI, 2026a), GPT-5.6-luna (OpenAI, 2026b), and Gem-

Table 1: Performance on the VSP-Grid benchmark. V: valid-path rate; O: optimal-length rate; Avg.: average across Levels 3–6.

Model	Method	Level 3		Level 4		Level 5		Level 6		Avg.	
		V	O	V	O	V	O	V	O	V	O
Qwen3.5-27B	DIRECT	0.91	0.90	0.82	0.80	0.69	0.67	0.44	0.44	0.72	0.70
	HARNESS	0.89	0.88	0.68	0.65	0.61	0.58	0.50	0.48	0.67	0.65
	HARNESS+SKILL	**1.00**	**0.98**	**0.93**	**0.89**	**0.90**	**0.89**	**0.77**	**0.74**	**0.90**	**0.88**
Qwen3.5-35B	DIRECT	0.21	0.17	0.10	0.04	0.06	0.01	0.04	0.03	0.10	0.06
	HARNESS	0.37	0.34	0.31	0.27	0.36	0.32	0.26	0.21	0.33	0.29
	HARNESS+SKILL	**0.99**	**0.98**	**0.98**	**0.98**	**0.94**	**0.94**	**1.00**	**1.00**	**0.98**	**0.98**
Qwen3.6-27B	DIRECT	0.56	0.46	0.43	0.39	0.24	0.20	0.12	0.08	0.34	0.28
	HARNESS	0.33	0.32	0.45	0.45	0.45	0.44	0.46	0.46	0.42	0.42
	HARNESS+SKILL	**1.00**	**0.98**	**0.94**	**0.92**	**0.92**	**0.90**	**0.88**	**0.88**	**0.94**	**0.92**
Qwen3.6-35B	DIRECT	0.69	0.69	0.51	0.49	0.45	0.43	0.24	0.24	0.47	0.46
	HARNESS	0.30	0.29	0.34	0.31	0.41	0.39	0.44	0.43	0.37	0.36
	HARNESS+SKILL	**0.90**	**0.86**	**0.89**	**0.78**	**0.85**	**0.82**	**0.86**	**0.85**	**0.88**	**0.83**
Qwen3-Omni-30B	DIRECT	0.13	0.02	0.02	0.00	0.04	0.02	0.02	0.00	0.05	0.01
	HARNESS	0.19	0.13	0.17	0.14	0.19	0.14	0.06	0.03	0.15	0.11
	HARNESS+SKILL	**0.70**	**0.63**	**0.60**	**0.57**	**0.56**	**0.49**	**0.31**	**0.28**	**0.54**	**0.49**
GPT-5.4-mini	DIRECT	0.44	0.35	0.32	0.23	0.19	0.12	0.17	0.14	0.28	0.21
	HARNESS	**0.64**	**0.58**	**0.58**	**0.53**	0.46	0.43	0.39	0.36	**0.52**	0.47
	HARNESS+SKILL	0.58	0.55	0.45	0.41	**0.49**	**0.49**	**0.52**	**0.52**	0.51	**0.49**
GPT-5.6-luna	DIRECT	0.98	0.93	0.96	0.93	0.96	0.94	0.93	0.88	0.96	0.92
	HARNESS	0.96	0.95	0.97	0.96	0.97	0.96	0.98	0.94	0.97	0.95
	HARNESS+SKILL	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.97**	**1.00**	**0.99**	**0.99**	**0.98**
Gemini 3.1 Pro	DIRECT	0.88	0.88	0.78	0.74	0.67	0.64	0.38	0.36	0.68	0.66
	HARNESS	0.80	0.79	0.82	0.81	0.88	0.88	0.80	0.80	0.82	0.82
	HARNESS+SKILL	**0.99**	**0.99**	**1.00**	**1.00**	**0.98**	**0.98**	**0.98**	**0.98**	**0.99**	**0.99**

ini 3.1 Pro (Google DeepMind, 2026). These models enable comparisons across model families, capacity levels, and modality capabilities over visual, video, and audio reasoning tasks.

4.1.3 INFERENCE.

We adopt four inference settings: DIRECT: one native multimodal model call with no external code execution or tool orchestration; HARNESS: one code-agent attempt with the shared runtime and selected profile, but without the additional frozen benchmark-level skill; HARNESS+SKILL: one code-agent attempt with the same runtime and profile plus the frozen benchmark-level skill; HARNESS+REPAIR: skill-guided execution followed, when necessary, by verifier-guided fresh attempts. Results of HARNESS+REPAIR are presented in the supplementary material.

4.1.4 METRICS

For the image benchmarks, we follow the metrics of Lu et al. (2026), for the VSP-Grid, we use the sample of levels 3–6, and report valid-path rate (**V**) and optimal-length rate (**O**). For the remaining three benchmarks, we measure the number of correctly answered instances, denoted as *Valid*, and the total number of model invocations, denoted as *Calls*. For Video-MME, we report accuracy for short-, medium-, and long-duration videos, together with their mean across durations (Avg.). For MMAU, we report accuracy for Sound, Music, and Speech and their arithmetic mean. For AVUT, we report accuracy for AVCM, AVOM, and AVTM, as well as the arithmetic mean across the three categories.

Table 2: Results on BLINK-Jigsaw, CoMT, and Word Search. "Valid" counts correct answers, "Calls" counts all model calls, and ΔValid is measured relative to the corresponding DIRECT result.

Model	Method	BLINK-Jigsaw			CoMT			Word Search		
		Valid	Calls	ΔValid	Valid	Calls	ΔValid	Valid	Calls	ΔValid
Qwen3.5-27B	DIRECT	113/150	150	+0	172/200	200	+0	20/100	100	+0
	HARNESS	118/150	363	+5	192/200	576	+20	3/100	422	−17
	HARNESS+SKILL	**150**/150	150	+37	**194**/200	506	+22	**91**/100	117	+71
Qwen3.5-35B	DIRECT	115/150	150	+0	169/200	200	+0	22/100	100	+0
	HARNESS	72/150	204	−43	**183**/200	464	+14	4/100	376	−18
	HARNESS+SKILL	**150**/150	150	+35	179/200	455	+10	**91**/100	117	+69
Qwen3.6-27B	DIRECT	115/150	150	+0	167/200	200	+0	21/100	100	+0
	HARNESS	83/150	815	−32	**197**/200	759	+30	7/100	399	−14
	HARNESS+SKILL	**150**/150	150	+35	**197**/200	684	+30	**91**/100	117	+70
Qwen3.6-35B	DIRECT	110/150	150	+0	161/200	200	+0	22/100	100	+0
	HARNESS	72/150	654	−38	**177**/200	559	+16	3/100	267	−19
	HARNESS+SKILL	**150**/150	150	+40	173/200	618	+12	**91**/100	117	+69
Qwen3-Omni-30B	DIRECT	82/150	150	+0	143/200	200	+0	8/100	100	+0
	HARNESS	18/150	1291	−64	174/200	550	+31	0/100	260	−8
	HARNESS+SKILL	**144**/150	150	+62	**186**/200	558	+43	**88**/100	117	+80
GPT-5.4-mini	DIRECT	83/150	150	+0	97/200	200	+0	**12**/100	100	+0
	HARNESS	**91**/150	360	+8	110/200	268	+13	8/100	169	−4
	HARNESS+SKILL	79/150	150	−4	**116**/200	314	+19	4/100	104	−8
GPT-5.6-luna	DIRECT	110/150	150	+0	**199**/200	200	+0	46/100	100	+0
	HARNESS	114/150	390	+4	196/200	356	−3	47/100	223	+1
	HARNESS+SKILL	**138**/150	238	+28	190/200	351	−9	**63**/100	111	+17
Gemini 3.1 Pro	DIRECT	117/150	150	+0	193/200	200	+0	31/100	100	+0
	HARNESS	98/150	356	−19	**195**/200	305	+2	19/100	313	−12
	HARNESS+SKILL	**150**/150	151	+33	**195**/200	248	+2	**97**/100	118	+66

4.2 PERFORMANCE ACROSS IMAGE BENCHMARKS

Table 1 and Table 2 compare direct model inference with the base harness and the skill-enhanced harness across four visual reasoning benchmarks. On VSP-Grid, the skill-enhanced harness achieves the strongest performance for almost every model and difficulty level. And the advantage generally becomes more pronounced at higher difficulty levels, where direct inference often degrades sharply. For example, Qwen3.5-35B improves from average valid-path and optimal-length rates of 0.10 and 0.06 under direct inference to 0.98 and 0.98 with the skill-enhanced harness. These results suggest that the skill-enhanced harness can compensate for deficiencies in a model's native visual planning ability by converting an implicit reasoning problem into a more structured and executable procedure. Strong models also benefit, although their improvements are naturally smaller because of ceiling effects. GPT-5.6-luna already achieves 0.96/0.92 with direct inference, but the skill-enhanced harness further improves its performance to 0.99/0.98. In contrast, the base harness is considerably less reliable. Although it improves some models, such as Qwen3.5-35B and Gemini 3.1 Pro, it reduces performance for others, including Qwen3.5-27B and Qwen3.6-35B. This observation indicates that providing access to code execution is insufficient by itself. Without task-specific guidance, the model may generate an inappropriate decomposition, select ineffective operations, or introduce additional execution errors.

The same overall pattern is observed on the remaining three benchmarks. The largest improvement occurs on Word Search, where the average valid rate increases by more than 54 percentage points. This benchmark requires systematic localization and verification over many candidate positions, making it particularly suitable for code-based search rather than unconstrained visual inspection. On BLINK-Jigsaw, the skill-enhanced harness obtains perfect scores for four Qwen models and Gemini 3.1 Pro. CoMT presents a different pattern. The base harness is already effective on this benchmark, increasing the average valid rate from 81.3% to 89.0%. The skill-enhanced harness provides a smaller additional improvement to 89.4%. This suggests that CoMT naturally benefits from iterative tool use and intermediate computation.

The skill-enhanced harness is also substantially more efficient than the base harness. On BLINK-Jigsaw, it uses approximately 161 calls on average, compared with 554 for the base harness, while achieving much higher accuracy. On Word Search, the corresponding averages are 115 and 304 calls. The most striking case is Qwen3-Omni-30B on BLINK-Jigsaw, where the base harness solves

Table 3: Performance (%) on Video-MME across different video durations. Avg. is the mean over all durations.

Model	Method	Short	Med.	Long	Avg.
Qwen3.6-35B	DIRECT	**81.44**	70.00	60.89	70.78
	HARNESS	55.56	54.89	49.78	53.41
	HARNESS+SKILL	81.22	**70.67**	**61.78**	**71.22**
Qwen3.6-27B	DIRECT	81.11	69.56	61.78	70.81
	HARNESS	73.56	61.11	55.33	63.33
	HARNESS+SKILL	**81.44**	**70.56**	**63.22**	**71.74**

only 18 examples using 1,291 calls, whereas the skill-enhanced harness solves 144 examples using 150 calls. Although a few exceptions remain, such as GPT-5.4-mini on BLINK-Jigsaw, the overall results demonstrate that task-grounded skills transform open-ended code generation into a more reliable, accurate, and efficient reasoning process.

Verified visual planning. Appendix B and Figure 2 present a complete VSP-Grid case study: the skill-augmented harness converts an image into a structured grid and uses BFS to obtain a valid, optimal path, whereas direct reasoning enters a hole and the unskilled agent exhausts its execution budget.

4.3 GENERALIZATION ACROSS MULTIMODAL TASKS

Tables 3, 4, and 5 demonstrate that the skill-enhanced harness generalizes beyond image-based reasoning to video, audio, and audio-visual tasks. On Video-MME, it improves the average accuracy of Qwen3.6-35B and Qwen3.6-27B from 70.78% and 70.81% under direct inference to 71.22% and 71.74%, respectively. The improvements are mainly observed on medium- and long-duration videos. For example, Qwen3.6-27B improves from 61.78% to 63.22% on long videos. This indicates that the skill-enhanced harness is particularly beneficial for scenarios requiring structured temporal analysis and evidence aggregation.

On MMAU, the skill-enhanced harness improves the average accuracy of Qwen3-Omni-30B from 75.70% to 77.80%, with consistent gains across sound, music, and speech understanding. The overall improvement of Gemini 3.1 Pro is marginal (81.40% to 81.50%). However, the harness still improves speech recognition accuracy from 85.89% to 88.89%. On AVUT, the skill-enhanced harness improves Qwen3-Omni-30B from 66.24%/70.00% to 69.39%/70.95% on AV-Human and AV-Gemini, respectively. The largest improvements appear in AVOM, where the model benefits from better alignment of information across audio and visual modalities.

In contrast, the base harness consistently degrades performance across these benchmarks. Its average accuracy drops to 53.41% and 63.33% on Video-MME, 26.19% and 68.38% on MMAU, and 34.81%/28.25% on AVUT. These results further demonstrate that code execution alone are insufficient for reliable multimodal reasoning. Without task-specific guidance, the model may perform ineffective decomposition or select inappropriate processing strategies, leading to inferior performance. Overall, these results show that task-grounded skills enable the harness to effectively extend multimodal models, particularly for tasks involving long-context understanding, temporal reasoning, and cross-modal integration.

5 LIMITATIONS AND FUTURE DIRECTIONS

From fixed to self-improving skills. The current skills are manually designed and remain fixed during evaluation, potentially introducing substantial task-level engineering. Future agents could maintain an evolving skill library, distilling successful trajectories into reusable procedures and using failed executions to identify weak tool choices, missing preconditions, or brittle constraints.

Reducing reliance on task-specific skills. Our results show that code execution and tool access alone do not reliably produce effective multimodal workflows, and the profile-only harness may

Table 4: Performance (%) across MMAU audio categories. Avg. denotes the mean over all categories.

Model	Method	Sound	Music	Speech	Avg.
Qwen3-Omni-30B	DIRECT	81.38	72.46	73.27	75.70
	HARNESS	35.14	34.13	9.31	26.19
	HARNESS+SKILL	**83.18**	**74.55**	**75.68**	**77.80**
Gemini 3.1 Pro	DIRECT	**79.88**	**78.44**	85.89	81.40
	HARNESS	76.86	66.77	61.50	68.38
	HARNESS+SKILL	78.08	77.54	**88.89**	**81.50**

Table 5: Performance (%) comparison on AVUT. Avg. denotes the average accuracy across all categories. AV-Human accuracy is shown above AV-Gemini accuracy.

Model	Method	AVCM	AVOM	AVTM	Avg.
Qwen3-Omni-30B	DIRECT	68.11	60.20	70.42	66.24
		77.70	60.31	**71.98**	70.00
	HARNESS	28.57	43.59	32.26	34.81
		28.49	33.33	22.92	28.25
	HARNESS+SKILL	**68.59**	**66.58**	**72.99**	**69.39**
		77.76	**64.49**	70.60	**70.95**

underperform direct inference. Future work could post-train the planner through supervised fine-tuning or reinforcement learning on successful code trajectories, execution feedback, and verified outcomes. This may help the model internalize transferable orchestration strategies and reduce dependence on manually designed skills.

Heterogeneous model collaboration. The current system generally uses the same model for both the planner and sub-agents, without exploiting the complementary strengths of specialized models. Future systems could combine a strong text or code planner with dedicated vision, video, audio, document, and verification models. This would also allow a text-only LLM to solve multimodal tasks by routing selected artifacts to multimodal sub-agents.

Extension to open-world tasks. Currently, our framework focuses on multimodal tasks involving images, videos, audio, and audio-visual inputs. Nevertheless, the flexible design of our framework makes it straightforward to extend to a broader range of general-purpose tasks. Search- and document-intensive tasks are natural extensions of the variable-mediated workspace: retrieved pages, passages, tables, and metadata can remain outside the main context, while sub-agents filter or summarize them for the planner. Overall, generalizing CodeOmni to a wider variety of tasks is an interesting and promising avenue for future exploration.

6 CONCLUSION

In conclusion, CodeOmni demonstrates that executable code can serve as a shared harness substrate for extending frozen models across image, video, audio, and audio-visual tasks. Our results also reveal an important boundary: code execution and tool access alone are not sufficient, whereas task-grounded skills can transform an open-ended action space into a reliable and effective reasoning process. Looking forward, post-trained planners, self-improving skills, heterogeneous model collaboration, broader tool ecosystems, and extensions to search and open-world tasks may further reduce manual engineering and improve autonomy. We hope CodeOmni provides a practical step toward adaptive omni-modal agents that can actively acquire, preserve, and compose heterogeneous evidence through executable programs.

REFERENCES

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Zihui Cheng, Qiguang Chen, Jin Zhang, Hao Fei, Xiaocheng Feng, Wanxiang Che, Min Li, and Libo Qin. Comt: A novel benchmark for chain of multi-modal thought on large vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23678–23686, 2025.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. In *European Conference on Computer Vision*, pp. 148–166. Springer, 2024.

Google DeepMind. Gemini 3.1 Pro model card. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, February 2026. Google DeepMind model card. Accessed: 2026-07-27.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Ming Li, Jike Zhong, Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Yuxiang Lai, Chen Wei, Konstantinos Psounis, and Kaipeng Zhang. Tir-bench: A comprehensive benchmark for agentic thinking-with-images reasoning. *arXiv preprint arXiv:2511.01833*, 2025.

Jianglin Lu, Hailing Wang, Xu Ma, Qihua Dong, Mingyuan Zhang, Yizhou Wang, and Yun Fu. Muse: A unified agentic harness for mllms. *arXiv preprint arXiv:2606.03005*, 2026.

OpenAI. GPT-5.4 mini. `https://developers.openai.com/api/docs/models/gpt-5.4-mini`, 2026a. Accessed: 2026-07-27.

OpenAI. GPT-5.6 Luna. `https://developers.openai.com/api/docs/models/gpt-5.6-luna`, 2026b. Accessed: 2026-07-27.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, February 2026a. Accessed: 2026-07-27.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model. `https://qwen.ai/blog?id=qwen3.6-27b`, April 2026b. Accessed: 2026-07-27.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all. `https://qwen.ai/blog?id=qwen3.6-35b-a3b`, April 2026c. Accessed: 2026-07-27.

Maohao Ran, Zhenglin Wan, Cooper Lin, Yanting Zhang, Hongyu Xin, Hongwei Fan, Yibo Xu, Beier Luo, Yaxin Zhou, Wangbo Zhao, et al. Caveagent: Transforming llms into stateful runtime operators. *arXiv preprint arXiv:2601.01569*, 2026.

S Sakshi, Utkarsh Tyagi, Sonal Kumar, Ashish Seth, Ramaneswaran Selvakumar, Oriol Nieto, Ramani Duraiswami, Sreyan Ghosh, and Dinesh Manocha. Mmau: A massive multi-task audio understanding and reasoning benchmark. In *International Conference on Learning Representations*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Qi Song, Honglin Li, Yingchen Yu, Haoyi Zhou, Lin Yang, Song Bai, Qi She, Zilong Huang, and Yunqing Zhao. Codedance: A dynamic tool-integrated mllm for executable visual reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Didac Suris, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Qiucheng Wu, Handong Zhao, Michael Saxon, Trung Bui, William Yang Wang, Yang Zhang, and Shiyu Chang. Vsp: Diagnosing the dual challenges of perception and reasoning in spatial planning tasks for mllms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2270–2280, 2025.

Zhenghao Xing, Ruiyang Xu, Yuxuan Wang, Jinzheng He, Ziyang Ma, Qize Yang, Yunfei Chu, Jin Xu, Junyang Lin, Chi-Wing Fu, and Pheng-Ann Heng. Native active perception as reasoning for omni-modal understanding. *arXiv preprint arXiv:2606.19341*, 2026.

Jin Xu, Zhifang Guo, Haoyu Hu, Yunfei Chu, Xun Wang, Jinzheng He, Yuxuan Wang, Xingjian Shi, Tao He, Xizhou Zhu, et al. Qwen3-omni technical report. *arXiv preprint arXiv:2509.17765*, 2025.

Yudong Yang, Jimin Zhuang, Guangzhi Sun, Changli Tang, Yixuan Li, Peihan Li, Yifan Jiang, Wei Li, Zejun Ma, and Chao Zhang. Audio-centric video understanding benchmark without text shortcut. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

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 *International Conference on Learning Representations*, 2023.

Zhaoyang Yu, Jiayi Zhang, Huixue Su, Yufan Zhao, Yifan Wu, Mingyi Deng, Jinyu Xiang, Yizhang Lin, Lingxiao Tang, Yuyu Luo, Bang Liu, and Chenglin Wu. Recode: Unify plan and action for universal granularity control. *arXiv preprint arXiv:2510.23564*, 2025.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.

Junjie Zhang, Jingyi Xi, Zhuoyang Song, Junyu Lu, Yuhua Ke, Ting Sun, Yukun Yang, Jiaxing Zhang, Songxin Zhang, and Zejian Xie. L0: Reinforcement learning to become general agents. *arXiv preprint arXiv:2506.23667*, 2025b.

SUPPLEMENTARY MATERIAL

A DISCLOSURE OF AI USE

We used generative AI tools to assist with writing system code, searching for relevant literature, and drafting portions of the manuscript that describe experimental results. All AI-assisted work underwent human review and verification, as detailed below.

System code. AI tools assisted in writing the system code. All code was manually reviewed and validated through testing.

Literature search. AI tools assisted in searching for relevant literature. All relevant references were manually verified.

Presentation of experimental results. AI tools assisted in drafting portions of the text describing the experimental results. The numerical values in these passages were subsequently checked individually by human reviewers against the experimental results. Incorrect descriptions were corrected, and inappropriate wording was revised to improve accuracy and clarity.

The authors take responsibility for the final content of this work, including all code, text, and claims produced with AI assistance.

B Case Study: Verified Visual Planning

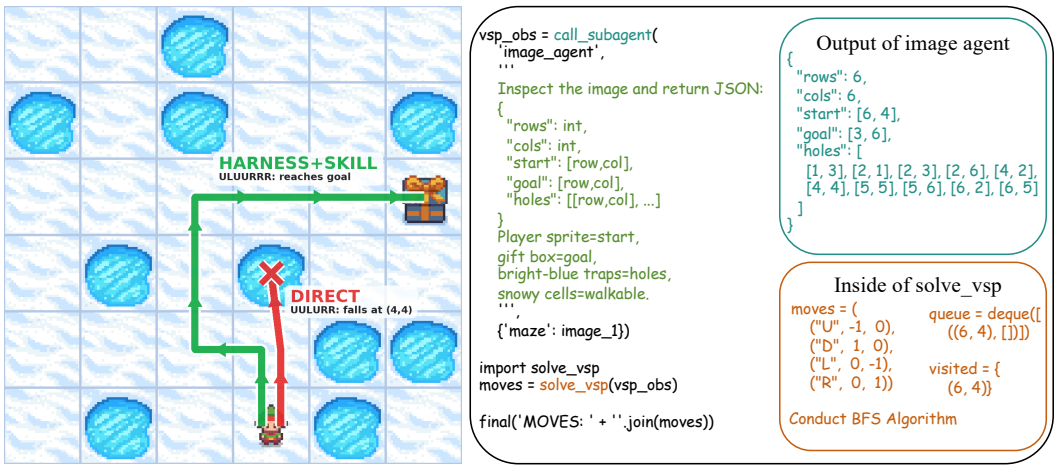


Figure 2: Case study on VSP-Grid. Direct reasoning enters the hole at $(4, 4)$, whereas the skill-augmented agent converts the image into a structured grid and uses BFS to find the shortest safe path ULUURRR.

Figure 2 compares direct reasoning with the skill-augmented harness on a 6×6 VSP-Grid instance. The player starts at $(6, 4)$ and the goal is located at $(3, 6)$. Direct reasoning predicts UULURR. Its second action enters the hole at $(4, 4)$; even if this collision were ignored, the sequence would end at $(3, 5)$ rather than the goal. The skill-augmented harness instead decomposes the task into visual state extraction and deterministic planning. An image agent converts the scene into JSON containing the grid dimensions, start, goal, and ten hole coordinates. The skill validates this representation and performs breadth-first search (BFS) over the four movement actions. BFS returns ULUURRR, corresponding to $(6, 4) \to (5, 4) \to (5, 3) \to (4, 3) \to (3, 3) \to (3, 4) \to (3, 5) \to (3, 6)$. The resulting seven-step trajectory is both valid and optimal. The unskilled agent correctly reads the image but exhausts its five-step budget while handling sub-agent selection, JSON state, and code indentation, ultimately failing to return a valid plan.

Overall, the case study demonstrates two distinct but complementary effects of skill augmentation. First, symbolic BFS replaces an unverified visual guess with an explicitly simulated shortest path. Second, the skill constrains the agent's workflow so that a correct visual observation is reliably converted into a valid final answer. The improvement therefore comes not only from better reasoning about the route, but also from more reliable execution of the complete perception–planning–verification pipeline.

Table 6: Evaluated benchmark scope. "Units" is the denominator used by the reported accuracy or validity metric.

Benchmark	Split/selection	Units	Primary metric
VSP-Grid	Levels 3–6, 100 per level	400	Valid-path rate (V) and optimal-length rate (O), reported per level and averaged over levels.
CoMT	First 200 `creation` items	200	Multiple-choice accuracy; calls count planner and model-backed sub-agent calls.
BLINK-Jigsaw	Labeled validation split	150	Multiple-choice accuracy; calls count planner and model-backed sub-agent calls.
TIR-Bench Word Search	Complete evaluated subtask	100	Exact match for an integer count or a 1-indexed row/column pair.
Video-MME	No subtitles; short/medium/long	2,700	Accuracy for 900 questions in each duration group and overall accuracy.
MMAU Test-mini	Sound/Music/Speech	1,000	Category accuracy and unweighted mean over the three categories.
AVUT	AVCM/AVOM/AVTM; AV-Human and AV-Gemini	5,888	Task accuracy and unweighted mean over the three task types, reported separately for the two partitions.

C Scope of the Supplement

This supplement provides the implementation details, evaluation protocols, complete experimental results, prompts, and task-grounded skills omitted from the main paper. Section D specifies the evaluated data, inference conditions, generation parameters, runtime budgets, and repair protocol. Section E documents the registered tools, variable and artifact stores, sub-agent roles, and profile-specific runtime constraints. Section F reports the complete experimental results, including the available verifier-guided repair runs. Sections G and H disclose the main-agent prompts and task-grounded skills, respectively.

No model is trained or fine-tuned in any experiment. The model weights remain frozen throughout evaluation. The terms "skill" and "repair" refer only to inference-time prompting, executable procedures, and verification mechanisms implemented outside the model.

D Detailed Experimental Protocol

D.1 Benchmarks, Splits, and Evaluation Units

Table 6 gives the exact evaluated scope. VSP-Grid uses 100 instances from each of Levels 3–6. CoMT uses the first 200 examples of the `creation` split in dataset order. BLINK-Jigsaw uses all 150 labeled validation examples. Word Search uses all 100 examples in the evaluated TIR-Bench subtask. Video-MME is evaluated without subtitles on 2,700 question–video pairs, equally divided into short, medium, and long duration groups. MMAU Test-mini contains 1,000 questions: 333 Sound, 334 Music, and 333 Speech. AVUT contains 5,888 scorable questions over 2,621 unique videos. Its AV-Gemini partition contains 4,768 questions and its AV-Human partition contains 1,120 questions.

D.2 Compared Inference Conditions

Direct. The raw multimodal input and public question are sent to the evaluated model in one call. There is no generated Python, tool call, variable store, or sub-agent. For multiple-choice tasks the model is asked to return only the option letter.

Harness. The model receives the shared code-agent system prompt, tool documentation, the benchmark runtime profile, and compact variable metadata. It does not receive the additional frozen

Table 7: Runtime budgets and input policies for the seven evaluated profiles. Steps denotes the maximum number of planner–executor iterations in one attempt, and Repairs denotes the maximum number of fresh verifier-guided attempts.

Benchmark	Profile	Steps	Repairs	Input policy
VSP-Grid	`muse_vsp`	5	3	One staged image.
BLINK-Jigsaw	`muse_jigsaw`	5	3	One incomplete image and two candidate patches.
CoMT	`muse_comt`	5	3	The complete problem statement and one diagram.
Word Search	`muse_tir`	3	3	One grid image.
Video-MME	`video_mme`	7	1	No subtitles; at most 64 temporally ordered frames in one model call.
MMAU	`audio_bench`	3	1	The complete audio input.
AVUT	`music_avqa`	7	3	Synchronized video and audio, with at most 120 frames sampled at 1 fps.

benchmark skill. The model generates Python at temperature zero, observes execution output, and may generate another program until the step budget is exhausted.

HARNESS+SKILL. This condition has the same model, public inputs, runtime profile, and execution budget as HARNESS, but adds the frozen task-grounded skill. A prompt skill is concatenated to the system message. If the skill contains the reserved marker `CODE_OMNI_EXECUTE_EXACT_SKILL_CODE`, the fenced Python block is executed as the first action without an additional planner call.

HARNESS+REPAIR. The initial attempt is identical to HARNESS+SKILL. If deterministic verification fails, the harness starts a fresh attempt with the original public task, a compressed failure category, the previous response, the number of code errors, and benchmark-safe repair hints. Unless otherwise stated, visual repair experiments allow up to three fresh attempts ($K = 3$). The correct answer, reference rationale, hidden annotations, and exact task state are not included in the repair prompt.

D.3 GENERATION AND RUNTIME PARAMETERS

The main code planner is decoded greedily with temperature 0.0. Its default maximum generation length is 8,192 tokens. Sub-agent calls default to temperature 0.0; a frozen skill may explicitly override temperature, thinking mode, and its local token budget. Direct visual baselines use temperature 0.0. The official-style direct answer budgets are 256 tokens for VSP-Grid, 2,048 tokens for BLINK-Jigsaw, 768 tokens for CoMT, and 8,192 tokens for Word Search. Video-MME is visual-only and does not use subtitles or audio. The direct anchor receives at most 64 uniformly sampled ordered frames. For MMAU, all 1,000 Test-mini examples are evaluated in dataset order without shuffling. Its direct-inference condition uses temperature 0.0. The Qwen MMAU skill sub-agent uses thinking mode, temperature 0.6, and at most 8,192 tokens. For AVUT, direct inference follows the benchmark multiple-choice prompt, samples at most 120 frames at 1 fps, and uses the complete audio. It follows its closed-source direct protocol with temperature 1.0 and a 256-token answer budget; skill sub-agent calls use the temperatures and budgets written in the frozen skill.

Table 7 summarizes the runtime profile, planner–executor step budget, repair budget, and input policy used for each benchmark. The step budget applies independently to each attempt. A repair creates a fresh attempt with a new planner state and variable store rather than continuing the failed trajectory. Exact executable skills typically terminate during the first planner–executor step.

The Python package requires Python 3.10 or newer. The core dependencies are `openai`, `pydantic`, `python-dotenv`, `Pillow`, and `requests`. Media experiments additionally use FFmpeg and, when required by a skill, OpenCV, `librosa`, `soundfile`, `matplotlib`, `pydub`, and `faster-whisper`. Tesseract is available to the OCR tool. Parallel workers and GPU sharding change throughput only; they do not change an example's prompt, evidence cap, or score.

Table 8: Complete inventory of the 32 registered Python tools, grouped by their primary runtime roles.

Group	Registered tool names
Execution	`run_cli`, `run_cli_json`.
Metadata and conversion	`media_info`, `video_info`, `audio_info`, `video_transcode`, `audio_to_wav`, `image_info`, and `image_convert`.
Video and frame processing	`video_extract_frames`, `video_extract_frames_by_timestamps`, `video_trim`, `video_probe_cv2`, `video_split_segments_cv2`, and `make_contact_sheet`.
Video/audio evidence packs	`video_extract_audio`, `video_extract_audio_compressed`, `prepare_video_for_omni`, `prepare_omnivideo_evidence`, `prepare_omnivideo_temporal_index`, `prepare_omnivideo_targeted_evidence`, `prepare_omnivideo_focus_evidence`, and `prepare_avut_alignment_evidence`.
Audio analysis	`audio_transcribe`, `audio_waveform`, and `audio_spectrogram`.
Image and document	`image_ocr` and `xlsx_preview`.
Search	`web_search`, `access_full_text`, `save_search_result`, and `comprehensive_search`.

E Tool, State, and Sub-Agent Configuration

E.1 Registered Python Tools

The runtime registry contains 32 ordinary Python functions. Four additional runtime objects—`store`, `artifacts`, `call_subagent`, and `final`—are always available to generated programs but are not counted among the 32 registered tools.

Table 8 groups the registered functions by their primary roles. The inventory covers local command execution, media inspection and conversion, bounded video and audio preprocessing, evidence-pack construction, audio analysis, image and document processing, and web retrieval.

Tool documentation is generated directly from the registry at runtime and appended to the main-agent system message. Each declaration specifies the exact Python function name, its purpose, accepted arguments, and output form. Runtime profiles enforce allow-lists by removing disallowed functions before both prompt construction and program execution.

The image profiles expose only the execution, image-processing, and contact-sheet tools required by their corresponding skills. The audio profile exposes execution, audio metadata, conversion, transcription, waveform, and spectrogram operations. Video and audio-visual profiles expose the bounded media-processing and evidence-pack tools needed for temporal or cross-modal analysis.

E.2 Profile-Specific Sub-Agent and Evidence Policies

The runtime profile controls not only the visible Python tools but also the sub-agent roles that generated programs may invoke. Table 9 reports the allowed sub-agents and the minimum evidence operations required by each evaluated profile.

The image benchmarks do not impose an additional hard evidence group because their exact or prompt-based skills already invoke the required image operation. In contrast, the media profiles require explicit evidence preparation before a final answer is accepted. Video-MME requires at least one bounded video operation, MMAU requires at least one audio inspection or analysis operation, and AVUT requires evidence from both the visual/video and audio modalities.

E.3 Variable Store and Artifact Store

Inputs are registered both as Python path variables (`image_1`, `video_1`, `audio_1`, and `file_1`) and as entries in the variable store. A value placed in the store receives a stable name,

Table 9: Profile-specific sub-agent allow-lists and minimum evidence requirements. An evidence requirement is satisfied when any listed alternative is actually executed.

Profile	Allowed sub-agents	Required evidence
`muse_vsp`	`image_agent`, `verification_agent`	No additional hard evidence group. The exact skill invokes image observation and the BFS wrapper.
`muse_jigsaw`	`image_agent`, `verification_agent`	No additional hard evidence group. The exact skill invokes the deterministic seam scorer.
`muse_comt`	`image_agent`, `verification_agent`	No additional hard evidence group. Diagram inspection is optional when the textual statement is sufficient.
`muse_tir`	`image_agent`, `verification_agent`	No additional hard evidence group. The exact skill invokes the pixel-grid solver.
`video_mme`	`image_agent`, `audio_agent`, `video_agent`, `omni_agent`, `verification_agent`	At least one bounded video-evidence operation: a packed overview, uniformly sampled frames, timestamped frames, or segmented frames. The benchmark runner disables audio under the visual-only Video-MME protocol.
`audio_bench`	`audio_agent`, `verification_agent`	At least one audio operation: metadata inspection, transcription, spectrogram generation, or waveform generation.
`music_avqa`	`image_agent`, `audio_agent`, `video_agent`, `omni_agent`, `verification_agent`	At least one visual/video preparation operation and one audio preparation/analysis operation. The combined AVUT evidence-preparation tool satisfies both requirements.

type, compact summary, creation time, and preview. `store.list()` returns only this compact inventory; `store.inspect(name)` returns bounded metadata; and `store.get(name)` materializes the value for generated code or a sub-agent. Large frames, clips, audio, and derived files are written under the task-local artifact directory rather than pasted into the planner context.

Python globals persist across steps within one attempt. Repair starts a fresh agent and therefore a fresh variable store. This distinction prevents a failed attempt from silently leaking hidden state into a repair attempt.

E.4 Sub-Agent Roles

The registry defines six canonical roles, while each runtime profile exposes only the subset relevant to that benchmark:

- `image_agent`: direct image evidence extraction;
- `audio_agent`: speech, sound, music, and audio timing;
- `video_agent`: visual, audio, and temporal evidence from clips, frames, contact sheets, and transcripts;
- `omni_agent`: fusion of text, image, audio, and video evidence;
- `document_agent`: document/file evidence;
- `verification_agent`: independent evidence checking.

By default, sub-agents reuse the planner model. A run may provide a separate backend configuration for any role. `call_subagent` materializes selected store values, extracts bounded local media, invokes the role, writes the response back to the store, and returns the string key of that stored response.

F Complete Experimental Results

This section reports the complete experimental matrices corresponding to the main-paper evaluations. In addition to Direct, Harness, and Harness+Skill, the tables include the avail-

Table 10: Complete VSP-Grid results in the main-paper format. V: valid-path rate; O: optimal-length rate; Avg.: average across Levels 3–6.

Model	Method	Level 3		Level 4		Level 5		Level 6		Avg.	
		V	O	V	O	V	O	V	O	V	O
Qwen3.5-27B	DIRECT	0.91	0.90	0.82	0.80	0.69	0.67	0.44	0.44	0.72	0.70
	HARNESS	0.89	0.88	0.68	0.65	0.61	0.58	0.50	0.48	0.67	0.65
	HARNESS+SKILL	**1.00**	0.98	0.93	0.89	0.90	0.89	0.77	0.74	0.90	0.88
	HARNESS+REPAIR	**1.00**	**0.99**	**0.96**	**0.92**	**0.97**	**0.91**	**0.90**	**0.86**	**0.96**	**0.92**
Qwen3.5-35B	DIRECT	0.21	0.17	0.10	0.04	0.06	0.01	0.04	0.03	0.10	0.06
	HARNESS	0.37	0.34	0.31	0.27	0.36	0.32	0.26	0.21	0.33	0.29
	HARNESS+SKILL	0.99	0.98	0.98	**0.98**	0.94	0.94	**1.00**	**1.00**	0.98	0.98
	HARNESS+REPAIR	**1.00**	**1.00**	**0.99**	**0.98**	**0.98**	**0.97**	**1.00**	**1.00**	**0.99**	**0.99**
Qwen3.6-27B	DIRECT	0.56	0.46	0.43	0.39	0.24	0.20	0.12	0.08	0.34	0.28
	HARNESS	0.33	0.32	0.45	0.45	0.45	0.44	0.46	0.46	0.42	0.42
	HARNESS+SKILL	**1.00**	**0.98**	0.94	0.92	0.92	0.90	0.88	0.88	0.94	0.92
	HARNESS+REPAIR	**1.00**	**0.98**	**0.96**	**0.94**	**0.98**	**0.96**	**0.92**	**0.92**	**0.97**	**0.95**
Qwen3.6-35B	DIRECT	0.69	0.69	0.51	0.49	0.45	0.43	0.24	0.24	0.47	0.46
	HARNESS	0.30	0.29	0.34	0.31	0.41	0.39	0.44	0.43	0.37	0.36
	HARNESS+SKILL	0.90	0.86	0.89	0.78	0.85	0.82	0.86	0.85	0.88	0.83
	HARNESS+REPAIR	**0.96**	**0.94**	**0.90**	**0.81**	**0.88**	**0.86**	**0.89**	**0.87**	**0.91**	**0.87**
Qwen3-Omni-30B	DIRECT	0.13	0.02	0.02	0.00	0.04	0.02	0.02	0.00	0.05	0.01
	HARNESS	0.19	0.13	0.17	0.14	0.19	0.14	0.06	0.03	0.15	0.11
	HARNESS+SKILL	0.70	0.63	0.60	0.57	0.56	0.49	0.31	0.28	0.54	0.49
	HARNESS+REPAIR	**0.95**	**0.92**	**0.88**	**0.84**	**0.85**	**0.80**	**0.80**	**0.72**	**0.87**	**0.82**
GPT-5.4-mini	DIRECT	0.44	0.35	0.32	0.23	0.19	0.12	0.17	0.14	0.28	0.21
	HARNESS	0.64	0.58	0.58	0.53	0.46	0.43	0.39	0.36	0.52	0.47
	HARNESS+SKILL	0.58	0.55	0.45	0.41	0.49	0.49	0.52	0.52	0.51	0.49
	HARNESS+REPAIR	**0.93**	**0.87**	**0.91**	**0.86**	**0.90**	**0.89**	**0.89**	**0.86**	**0.91**	**0.87**
GPT-5.6-luna	DIRECT	0.98	0.93	0.96	0.93	0.96	0.94	0.93	0.88	0.96	0.92
	HARNESS	0.96	0.95	0.97	0.96	0.97	0.96	0.98	0.94	0.97	0.95
	HARNESS+SKILL	0.99	0.99	0.99	0.99	0.99	0.97	**1.00**	0.99	0.99	0.98
	HARNESS+REPAIR	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**
Gemini 3.1 Pro	DIRECT	0.88	0.88	0.78	0.74	0.67	0.64	0.38	0.36	0.68	0.66
	HARNESS	0.80	0.79	0.82	0.81	0.88	0.88	0.80	0.80	0.82	0.82
	HARNESS+SKILL	0.99	0.99	**1.00**	**1.00**	0.98	0.98	0.98	0.98	0.99	0.99
	HARNESS+REPAIR	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

able HARNESS+REPAIR results. Throughout this supplement, HARNESS+REPAIR denotes Harness+Skill+Repair: the first attempt uses exactly the same skill-guided configuration as HARNESS+SKILL, and additional attempts are launched only after verifier rejection.

The repair results do not replace the main comparison between direct inference, the unskilled harness, and the skill-guided harness. Instead, they measure whether deterministic verification and fresh attempts can recover failures that remain after task-grounded skill guidance. Unless otherwise stated, repair allows at most three fresh attempts. Within each model block, the best value in each performance column is shown in bold, with all tied best values bolded.

F.1 COMPLETE IMAGE-BENCHMARK RESULTS

Table 10 and Tables 11–13 report the full results of VSP-Grid, BLINK-Jigsaw, CoMT, and Word Search matrices with HARNESS+REPAIR. Table 14 summarizes the same results at two aggregation levels. Panel (a) reports the eight-model mean for each image benchmark, whereas Panel (b) aggregates all 3,600 paired model–question instances from BLINK-Jigsaw, CoMT, and Word Search.

Overview of repair results. Across the three paired non-VSP image experiments, HARNESS+REPAIR improves the aggregate accuracy from 87.69% to 94.75%, recovering 254 additional correct answers over 3,600 model–question pairs. The improvement is accompanied by an increase from 1.65 to 2.73 model-backed calls per item, indicating that repair trades additional inference cost for higher reliability.

Effect of task-grounded skills. From Table 14, relative to direct inference, HARNESS improves the aggregate VSP score by 8.88 points and CoMT by 7.69 points, but loses 14.92 points on BLINK-Jigsaw and 11.38 points on Word Search; its four-task macro is consequently 2.43 points lower than direct inference. HARNESS+SKILL instead improves all four task metrics: +39.94 points on

Table 11: Complete BLINK-Jigsaw results, including verifier-guided repair. Valid counts correct answers; Calls counts all model calls; Repairs counts repaired items; ΔValid is relative to DIRECT.

Model	Method	Valid	Calls	Repairs	ΔValid
Qwen3.5-27B	DIRECT	113/150	150	–	+0
	HARNESS	118/150	363	–	+5
	HARNESS+SKILL	**150**/150	150	–	+37
	HARNESS+REPAIR	**150**/150	150	0	+37
Qwen3.5-35B	DIRECT	115/150	150	–	+0
	HARNESS	72/150	204	–	−43
	HARNESS+SKILL	**150**/150	150	–	+35
	HARNESS+REPAIR	**150**/150	151	1	+35
Qwen3.6-27B	DIRECT	115/150	150	–	+0
	HARNESS	83/150	815	–	−32
	HARNESS+SKILL	**150**/150	150	–	+35
	HARNESS+REPAIR	**150**/150	150	0	+35
Qwen3.6-35B	DIRECT	110/150	150	–	+0
	HARNESS	72/150	654	–	−38
	HARNESS+SKILL	**150**/150	150	–	+40
	HARNESS+REPAIR	**150**/150	150	0	+40
Qwen3-Omni-30B	DIRECT	82/150	150	–	+0
	HARNESS	18/150	1291	–	−64
	HARNESS+SKILL	144/150	150	–	+62
	HARNESS+REPAIR	**150**/150	157	7	+68
GPT-5.4-mini	DIRECT	83/150	150	–	+0
	HARNESS	91/150	360	–	+8
	HARNESS+SKILL	79/150	150	–	−4
	HARNESS+REPAIR	**150**/150	248	72	+67
GPT-5.6-luna	DIRECT	110/150	150	–	+0
	HARNESS	114/150	390	–	+4
	HARNESS+SKILL	138/150	238	–	+28
	HARNESS+REPAIR	**150**/150	310	78	+40
Gemini 3.1 Pro	DIRECT	117/150	150	–	+0
	HARNESS	98/150	356	–	−19
	HARNESS+SKILL	**150**/150	151	–	+33
	HARNESS+REPAIR	**150**/150	153	1	+33

Table 12: Complete CoMT results, including verifier-guided repair. Valid counts correct answers; Calls counts all model calls; Repairs counts repaired items; ΔValid is relative to DIRECT.

Model	Method	Valid	Calls	Repairs	ΔValid
Qwen3.5-27B	DIRECT	172/200	200	–	+0
	HARNESS	192/200	576	–	+20
	HARNESS+SKILL	194/200	506	–	+22
	HARNESS+REPAIR	**198**/200	939	149	+26
Qwen3.5-35B	DIRECT	169/200	200	–	+0
	HARNESS	183/200	464	–	+14
	HARNESS+SKILL	179/200	455	–	+10
	HARNESS+REPAIR	**191**/200	883	176	+22
Qwen3.6-27B	DIRECT	167/200	200	–	+0
	HARNESS	197/200	759	–	+30
	HARNESS+SKILL	197/200	684	–	+30
	HARNESS+REPAIR	**200**/200	1075	159	+33
Qwen3.6-35B	DIRECT	161/200	200	–	+0
	HARNESS	177/200	559	–	+16
	HARNESS+SKILL	173/200	618	–	+12
	HARNESS+REPAIR	**190**/200	1191	100	+29
Qwen3-Omni-30B	DIRECT	143/200	200	–	+0
	HARNESS	174/200	550	–	+31
	HARNESS+SKILL	186/200	558	–	+43
	HARNESS+REPAIR	**190**/200	1086	135	+47
GPT-5.4-mini	DIRECT	97/200	200	–	+0
	HARNESS	110/200	268	–	+13
	HARNESS+SKILL	116/200	314	–	+19
	HARNESS+REPAIR	**168**/200	814	124	+71
GPT-5.6-luna	DIRECT	199/200	200	–	+0
	HARNESS	196/200	356	–	−3
	HARNESS+SKILL	190/200	351	–	−9
	HARNESS+REPAIR	**200**/200	509	66	+1
Gemini 3.1 Pro	DIRECT	193/200	200	–	+0
	HARNESS	195/200	305	–	+2
	HARNESS+SKILL	195/200	248	–	+2
	HARNESS+REPAIR	**200**/200	282	14	+7

Table 13: Complete Word Search results, including verifier-guided repair. Valid counts correct answers; Calls counts all model calls; Repairs counts repaired items; ΔValid is relative to DIRECT.

Model	Method	Valid	Calls	Repairs	ΔValid
Qwen3.5-27B	DIRECT	20/100	100	–	+0
	HARNESS	3/100	422	–	−17
	HARNESS+SKILL	**91**/100	117	–	+71
	HARNESS+REPAIR	**91**/100	184	9	+71
Qwen3.5-35B	DIRECT	22/100	100	–	+0
	HARNESS	4/100	376	–	−18
	HARNESS+SKILL	91/100	117	–	+69
	HARNESS+REPAIR	**92**/100	171	9	+70
Qwen3.6-27B	DIRECT	21/100	100	–	+0
	HARNESS	7/100	399	–	−14
	HARNESS+SKILL	**91**/100	117	–	+70
	HARNESS+REPAIR	**91**/100	180	10	+70
Qwen3.6-35B	DIRECT	22/100	100	–	+0
	HARNESS	3/100	267	–	−19
	HARNESS+SKILL	**91**/100	117	–	+69
	HARNESS+REPAIR	**91**/100	164	10	+69
Qwen3-Omni-30B	DIRECT	8/100	100	–	+0
	HARNESS	0/100	260	–	−8
	HARNESS+SKILL	88/100	117	–	+80
	HARNESS+REPAIR	**89**/100	183	11	+81
GPT-5.4-mini	DIRECT	12/100	100	–	+0
	HARNESS	8/100	169	–	−4
	HARNESS+SKILL	4/100	104	–	−8
	HARNESS+REPAIR	**27**/100	381	98	+15
GPT-5.6-luna	DIRECT	46/100	100	–	+0
	HARNESS	47/100	223	–	+1
	HARNESS+SKILL	63/100	111	–	+17
	HARNESS+REPAIR	**96**/100	190	45	+50
Gemini 3.1 Pro	DIRECT	31/100	100	–	+0
	HARNESS	19/100	313	–	−12
	HARNESS+SKILL	**97**/100	118	–	+66
	HARNESS+REPAIR	**97**/100	142	5	+66

Table 14: Cross-model aggregates. VSP is the mean of the valid-path and optimal-length rates, and Image Macro is the unweighted mean of VSP, BLINK, CoMT, and Word Search. Panel (b) aggregates the 3,600 paired CoMT, BLINK-Jigsaw, and Word Search model–question pairs.

(a) Eight-model image means (%)

Method	VSP	BLINK	CoMT	Word	Image Macro
DIRECT	43.13	70.42	81.31	22.75	54.40
HARNESS	52.01	55.50	89.00	11.38	51.97
HARNESS+SKILL	83.07	92.58	89.38	77.00	85.51
HARNESS+REPAIR	**93.94**	**100.00**	**96.06**	**84.25**	**93.56**

(b) Paired three-task aggregate

Method	Correct	Accuracy	Calls/item
DIRECT	2,328/3,600	64.67	**1.00**
HARNESS	2,181/3,600	60.58	2.97
HARNESS+SKILL	3,157/3,600	87.69	1.65
HARNESS+REPAIR	**3,411**/3,600	**94.75**	2.73

VSP, +8.06 on CoMT, +22.17 on BLINK-Jigsaw, and +54.25 on Word Search. The result supports the paper's central distinction: code provides an execution substrate, whereas the skill determines whether the enlarged action space is used productively.

Scaling with visual difficulty. VSP further separates perception-and-planning quality as problem difficulty increases. Averaged across models, direct valid-path rate falls from 60.00% at Level 3 to 29.25% at Level 6. The corresponding HARNESS+SKILL rates are 89.38% and 79.00%, so the skill advantage grows from 29.38 to 49.75 points. The unskilled harness remains nearly flat (56.00% to 48.62%), but at a much lower level than the skill-guided method. This pattern is consistent with the skill's decomposition: visual extraction is performed once, while BFS removes the combinatorial burden of mentally extending longer paths and checking every intermediate collision.

Repair benefit. As shown in Table 14, relative to HARNESS+SKILL, repair recovers 254 additional answers and raises weighted accuracy from 87.69% to 94.75% (+7.06 points). The task-balanced three-task macro rises from 86.32% to 93.44% (+7.12 points). The improvement is broad rather than driven by one benchmark: CoMT gains 6.68 points, BLINK-Jigsaw gains 7.42 points and reaches 100%, and Word Search gains 7.25 points. With the final VSP runs included, the aggregate VSP score rises from 83.07% to 93.94% (+10.87 points), and the four-task Image Macro rises from 85.51% to 93.56% (+8.05 points). Repair is selective but not free. It is activated on 1,279 of the 3,600 model–question pairs (35.53%). Calls rise from 1.65 to 2.73 per item (+65.68%) relative to HARNESS+SKILL, although this remains below the 2.97 calls of the less accurate unskilled harness. The cost is benchmark dependent. On BLINK-Jigsaw, repair adds 89 correct answers while increasing calls by only 180; on CoMT it adds 107 correct answers but increases calls by 3,045 because 57.69% of items trigger repair. This difference reflects the benchmark verifier: deterministic seam rejection often identifies a recoverable local failure, whereas geometry failures more often require a complete new reasoning attempt.

Where the repair gain comes from. The aggregate gain is concentrated in models for which the first skill-guided attempt remains brittle. Across the three paired tasks, GPT-5.4-mini improves from 199/450 to 345/450 (+32.44 points), and GPT-5.6-luna improves from 391/450 to 446/450 (+12.22 points). Together they contribute 201 of the 254 additional correct answers (79.13%). By contrast, the other six models start between 92.00% and 98.22% and gain only 0.67–3.78 points. This is a ceiling effect rather than evidence that verification is model independent: when almost every first attempt already passes, the verifier has few opportunities to change the outcome.

Cost per recovered answer. The marginal efficiency also differs sharply across models. GPT-5.4-mini and GPT-5.6-luna require 5.99 and 5.62 additional calls per recovered answer, respectively, while the five Qwen variants require between 36.47 and 151.33. Gemini requires 12.00. A small accuracy gain can therefore be expensive when a model is already near saturation or repeatedly

produces a failure that the coarse verifier signal cannot localize. This motivates adaptive stopping and model-specific repair budgets rather than applying all three repair rounds uniformly.

Model-level behavior also shows the boundary of repair. GPT-5.4-mini improves from 79/150 to 150/150 on BLINK-Jigsaw, but only from 4/100 to 27/100 on Word Search even though 98 Word Search items trigger repair. Conversely, models whose first skill-guided attempt is already deterministic and correct incur almost no repair overhead: five BLINK rows require at most one repaired item. Repair therefore works best as a verifier-gated recovery mechanism, not as a replacement for a task-grounded skill or reliable perception.

Table 15: Results of Video-MME, MMAU, and AVUT. Panel (c) explicitly identifies the AV-Human and AV-Gemini partitions.

(a) Video-MME

Model	Method	Short	Med.	Long	Avg.
Qwen3.6-35B	DIRECT	81.44	70.00	60.89	70.78
	HARNESS	55.56	54.89	49.78	53.41
	HARNESS+SKILL	81.22	70.67	61.78	71.22
	HARNESS+REPAIR	**82.89**	**72.89**	**67.67**	**74.48**
Qwen3.6-27B	DIRECT	81.11	69.56	61.78	70.81
	HARNESS	73.56	61.11	55.33	63.33
	HARNESS+SKILL	81.44	70.56	63.22	71.74
	HARNESS+REPAIR	**85.33**	**74.44**	**67.11**	**75.63**

(b) MMAU

Model	Method	Sound	Music	Speech	Avg.
Qwen3-Omni-30B	DIRECT	81.38	72.46	73.27	75.70
	HARNESS	35.14	34.13	9.31	26.19
	HARNESS+SKILL	83.18	74.55	75.68	77.80
	HARNESS+REPAIR	**87.99**	**79.04**	**79.88**	**82.30**
Gemini 3.1 Pro	DIRECT	79.88	78.44	85.89	81.40
	HARNESS	76.86	66.77	61.50	68.38
	HARNESS+SKILL	78.08	77.54	88.89	81.50
	HARNESS+REPAIR	**82.58**	**83.23**	**91.29**	**85.70**

(c) AVUT

Model	Method	Partition	AVCM	AVOM	AVTM	Avg.
Qwen3-Omni-30B	DIRECT	AV-Human	68.11	60.20	70.42	66.24
		AV-Gemini	77.70	60.31	71.98	70.00
	HARNESS	AV-Human	28.57	43.59	32.26	34.81
		AV-Gemini	28.49	33.33	22.92	28.25
	HARNESS+SKILL	AV-Human	68.59	66.58	72.99	69.39
		AV-Gemini	77.76	64.49	70.60	70.95
	HARNESS+REPAIR	AV-Human	**79.64**	**71.57**	**74.88**	**75.36**
		AV-Gemini	**80.90**	**72.71**	**75.10**	**76.24**

F.2 VIDEO-MME

As shown in Table 15, all eight rows contain 2,700 examples. Before repair, the corresponding overall counts are 1,911, 1,442, and 1,923 correct for Qwen3.6-35B, and 1,912, 1,710, and 1,937 for Qwen3.6-27B. Skill guidance alone therefore adds only 12 and 25 correct answers over direct inference. The gain is concentrated in medium and long videos, supporting the interpretation that structured temporal aggregation helps most when a uniform native view is insufficient. For Qwen3.6-35B, the skill changes short/medium/long accuracy by $-0.22/+0.67/+0.89$ points; for Qwen3.6-27B the changes are $+0.33/+1.00/+1.44$. The monotonic tendency toward larger gains at longer durations is consistent with an evidence-selection benefit rather than a broad improvement in frame-level recognition. At the same time, the small overall gain and one short-video regression

show that extra orchestration can discard useful native context or introduce an unnecessary fusion decision.

One verifier-gated repair raises Qwen3.6-35B to 2,011/2,700 (74.48%) and Qwen3.6-27B to 2,042/2,700 (75.63%). Relative to HARNESS+SKILL, this recovers 88 and 105 additional answers, or +3.26 and +3.89 points. The 35B gain grows with duration: +1.67, +2.22, and +5.89 points for short, medium, and long videos. In contrast, the 27B gain is nearly duration-invariant (+3.89/+3.88/+3.89). Thus repair is not merely correcting a fixed global formatting rate. For the 35B model it disproportionately helps the long-context regime, whereas the 27B model benefits from a broader second-pass reconsideration.

F.3 MMAU

The category denominators are 333 Sound, 334 Music, and 333 Speech. For Qwen3-Omni-30B, HARNESS+SKILL improves the overall result from 757/1,000 to 778/1,000, with consistent gains across Sound, Music, and Speech of 1.80, 2.09, and 2.41 percentage points, respectively. HARNESS+REPAIR further increases the score to 823/1,000 (82.30%), adding 45 correct answers over HARNESS+SKILL. The corresponding gains are also consistent across the three categories: +4.81 points on Sound, +4.49 on Music, and +4.20 on Speech. Thus, repair provides a broad improvement rather than addressing only a single type of audio content.

Gemini 3.1 Pro exhibits a different pattern under HARNESS+SKILL. Its overall result changes only slightly from 814/1,000 (81.40%) to 815/1,000 (81.50%), as the 3.00-point improvement on Speech offsets regressions of 1.80 points on Sound and 0.90 points on Music. Nevertheless, HARNESS+REPAIR substantially improves all three categories, reaching 857/1,000 (85.70%). Relative to HARNESS+SKILL, repair adds 4.50 points on Sound, 5.69 points on Music, and 2.40 points on Speech, resulting in a 4.20-point aggregate improvement. The consistent repair gains for both models suggest that verifier-guided additional attempts can correct audio interpretation, reasoning, and answer-selection failures that remain after applying the task skill.

The base harness provides the opposite diagnostic. Qwen falls from 75.70% under DIRECT to 26.19%, including only 9.31% on Speech, while Gemini decreases from 81.40% to 68.38%. These large reductions occur despite access to the same underlying model and input, indicating that performance cannot be explained by the number of model calls alone. Instead, the results show that unconstrained orchestration can select unsuitable decompositions or intermediate representations, whereas an audio-aware skill and a repair mechanism make the execution process substantially more reliable.

F.4 AVUT

On the AV-Human partition, DIRECT obtains 284/417 AVCM, 236/392 AVOM, and 219/311 AVTM, whereas HARNESS+SKILL obtains 286/417, 261/392, and 227/311, respectively. This raises the macro-average from 66.24% to 69.39%. On AV-Gemini, the corresponding scores change from 77.70%, 60.31%, and 71.98% to 77.76%, 64.49%, and 70.60%, increasing the average from 70.00% to 70.95%. For both partitions, AVOM receives the largest benefit from the skill: +6.38 points on AV-Human and +4.18 points on AV-Gemini. However, AV-Gemini AVTM decreases by 1.38 points, showing that targeted audio-visual alignment is helpful but not uniformly beneficial across all relation types.

Repair produces substantially larger and more consistent improvements. On AV-Human, HARNESS+REPAIR reaches 79.64% AVCM, 71.57% AVOM, and 74.88% AVTM, yielding an average of 75.36%. On AV-Gemini, it reaches 80.90%, 72.71%, and 75.10%, respectively, with an average of 76.24%. Relative to HARNESS+SKILL, these results correspond to average gains of 5.97 points on AV-Human and 5.29 points on AV-Gemini, and every task type improves on both partitions. The largest individual gains occur on AV-Human AVCM (+11.05 points) and AV-Gemini AVOM (+8.22 points), suggesting that repair is particularly effective when the initial execution selects an incorrect visual entity or an insufficiently aligned audio-visual interval.

Cross-modal interpretation. Across Video-MME, MMAU, and AVUT, HARNESS consistently produces the largest performance degradation, whereas HARNESS+SKILL matches or outperforms

DIRECT in every reported model or partition average. On Video-MME, the skill improves over direct inference by 0.44 and 0.93 points for Qwen3.6-35B and Qwen3.6-27B, respectively. The corresponding improvements are 2.10 and 0.10 points for the two MMAU models, and 3.15 and 0.95 points for the AV-Human and AV-Gemini partitions. These results indicate that task-grounded skills are necessary for converting the open code-execution space into a reliable task-specific workflow, although their direct benefit depends on how well native inference already matches the input structure.

Repair provides a further improvement in all six reported settings. Relative to HARNESS+SKILL, it adds 3.26 and 3.89 points for the two Video-MME models, 4.50 and 4.20 points for the two MMAU models, and 5.97 and 5.29 points for the two AVUT partitions, corresponding to a mean gain of 4.52 points. The gains are especially pronounced for long-video understanding, difficult audio interpretation, and audio-visual correspondence, where errors can arise from evidence selection, temporal alignment, or answer extraction. Overall, the results support a staged orchestration strategy: use task-grounded skills to impose an appropriate evidence-processing structure, and invoke verifier-guided repair to revisit executions that remain incorrect or incomplete.

G PROMPTS

G.1 MAIN AGENT SYSTEM PROMPT

Listing 1 is the complete static main-agent prompt. At runtime, the harness appends (1) generated documentation for the allowed tools and store API and (2) the selected profile and optional task skill.

Listing 1: Complete static main-agent system prompt.

```
You are Code-Omni-Harness, a code-driven omni-modal execution harness.

Core rules:
1. You MUST solve tasks by writing Python code that calls tools. Do not rely on guesses about images, audio, or video.
2. Tool outputs should be saved into variables or store.put(...) first. Do not print huge raw outputs.
3. Use store.inspect(...) and store.list() for lightweight context.
4. Use call_subagent(...) for model-based analysis of images, audio, video, or fused evidence. Canonical names include `image_agent`, `audio_agent`, `video_agent`, `omni_agent`, and `verification_agent`.
5. `call_subagent(...)` returns a string store key, not a result object. Read it with `raw = str(store.get(key).get("raw") or "")`. Never call `.get()` on `raw`, never request a `var_name`, and make every `output_schema` value a JSON-serializable description string such as `{"answer":"A/B/C/D"}` rather than a Python type such as `str`.
6. Use only documented tool fields and keyword arguments. When a prior step fails, read its traceback and change the code; do not repeat the identical failing program.
7. Call preprocessing tools directly as Python functions; do not ask a subagent to impersonate a tool. `store.get(name)` accepts only the variable name (no `default=` keyword). Pass `output_schema` as a Python dictionary, not a JSON string.
8. When the task is complete, call `final("...")`. It terminates the current program immediately, so do not add fallback code after it.
9. Return only one Python code block. No prose outside code.

For video tasks, choose evidence slices before model analysis. Inspect metadata when useful, then select a bounded overview, audio track, timestamped frames, or a targeted clip with the available preparation/extraction tools. Pass only those prepared artifacts to `call_subagent(...)`; never pass `video_1` or another raw registered video path directly. Always cap frame count/fps and clip duration. Choose the slice and modality from the question rather than following one fixed script, and widen or refine evidence only when the first bounded slice is insufficient.

The standard `json` module is already available as `json` in the execution runtime. If execution reports a NameError or other code error, the next code block must repair that exact error instead of rerunning the same block.
Registered inputs such as `video_1`, `audio_1`, and `image_1` are already local path strings: pass them directly to tools. `store.get("video_1")` returns the same path value, not an inspect-metadata dictionary.

Available variables and tools are shown in the runtime context.
```

The first user message has the following dynamic form:

```
User task:
{public question and choices}

```

```
Runtime context:
{
  "current_step": 1,
  "max_steps": ...,
  "available_variables": {...compact store inventory...}
}
```

After each program, the next observation contains the current step, captured standard output, standard error, traceback, and error string. All planner calls use temperature 0.0.

G.2 DIRECT PROMPT

Listing 2: Generic direct-inference prompt.

```
Answer the task directly. Return only the final answer; for
multiple-choice tasks, return only the option letter.

Question:
{question}
Choices:
A. {choice A}
B. {choice B}
...
```

Dataset runners may replace this generic form with the benchmark's official prompt while preserving the same public input and one-call condition.

G.3 SUB-AGENT PROMPTS

The image, audio, video, and omni roles use the following role-specific system prompts. Document and verification roles use the generic evidence-analysis fallback plus stage-specific instructions.

Listing 3: Sub-agent role prompts.

```
image_agent:
You are an image evidence-analysis sub-agent. Inspect the provided images directly. Follow
     the stage-specific task and return JSON matching its required output schema exactly. Do
      not produce a final answer when the requested schema is for localization or
     observation only.

audio_agent:
You are an audio evidence-analysis sub-agent. Analyze speech, sounds, music, and timing from
      audio evidence. When timestamped frames are also provided, use them to attribute a
     voice or sound to the correct visible subject and reject similar occurrences. Follow
     the stage-specific task and return JSON matching its required output schema exactly. Do
      not produce a final answer when the requested schema is for localization or
     observation only.

video_agent:
You are a video evidence-analysis sub-agent. Analyze visual, audio, and temporal evidence
     from videos, frames, segments, contact sheets, and transcripts. Follow the stage-
     specific task and return JSON matching its required output schema exactly; do not
     answer choices during a localization-only stage.

omni_agent:
You are an omni-modal evidence-fusion sub-agent. Combine text, image, audio, and video
     evidence without guessing. Follow the stage-specific task and return JSON matching its
     required output schema exactly. Do not produce a final answer when the requested schema
      is for localization or observation only.
```

G.4 REPAIR PROMPT

Listing 4: Template for a fresh verifier-guided repair attempt. Braced fields are populated with gold-free run information.

```
Below is the same user message you received on the first turn
(full task instructions):
{original public task}

---
Your previous attempts did not yield a valid answer. Recent history(oldest first):
Failure summary (compressed, for actionable correction):
```

```
- Failure pattern counts: {error categories and counts}
- Latest verifier signal: {gold-free detail}
- Repeated same output as previous attempt: {yes/no}
- Next actions:
  - {profile-specific repair hints}

Detailed recent attempts:
{up to two previous replies, verifier signals, and code-error counts}

Repair attempt {repair index} of {maximum repairs}.

Requirements for this repair:
1. Do not just repeat the previous answer.
2. If the failure was a code/tool error, fix the code and use the available variables correctly.
3. If the failure was evidence-related, call the relevant media/search/sub-agent tools again.
4. End with final("...") or set FINAL_ANSWER to the concise final answer.
```

H TASK-GROUNDED SKILLS

H.1 VSP-GRID EXACT SKILL

The VSP skill separates perception from planning. One image sub-agent returns rows, columns, start, goal, and holes as 1-indexed coordinates. The wrapper rejects out-of-range coordinates, start/goal collisions with holes, malformed JSON, and unsolvable layouts. It then runs BFS with actions U, D, L, R and returns the first shortest safe path.

Listing 5: Exact first-step VSP program.

```
from code_omni_agent.harness.vsp_skill import solve_vsp_image
maze = image_3 if "image_3" in globals() else image_1
moves = solve_vsp_image(
    maze, call_subagent=call_subagent, store=store
)
final("Action plan: " + ",".join(moves))
```

The image observation schema is:

```
{"rows": int, "cols": int, "start": [row, col],
 "goal": [row, col], "holes": [[row, col], ...]}
```

Only the public maze image is inspected. The wrapper does not read the source grid, goal coordinate from benchmark metadata, shortest path, or reference answer.

H.2 BLINK-JIGSAW EXACT SKILL

The incomplete board is `image_1`; option A and B patches are `image_2` and `image_3`. For a bottom-right missing patch, the helper computes mean RGB L1 discontinuity along the candidate's left and top adjoining seams:

$$L(c) = \frac{1}{N} \sum_{(p,q)\in\mathcal{S}} \|I_{\text{board}}(p) - I_c(q)\|_1. \tag{1}$$

The lower-loss candidate wins. A difference smaller than 10^{-3} is treated as a tie rather than silently selecting a label.

Listing 6: Exact first-step BLINK-Jigsaw program.

```
from code_omni_agent.harness.jigsaw_stitch import \
    score_bottom_right_candidates
seam = score_bottom_right_candidates(
    image_1, image_2, image_3
)
if seam["winner"] not in ("A", "B"):
    raise ValueError("ambiguous seam")
final(seam["winner"])
```

H.3 CoMT Prompt Skill

Listing 7: Complete task-specific CoMT guidance.

```
You are a precise CoMT plane-geometry solver operating inside the existing Code-Omni agent
    workflow.

Solve from the complete statement. Use image evidence or one image sub-agent only when
    labels, connectivity, marked relations, extensions, or angle-ray orientation are not
    fully specified in text; never infer metric facts from drawing scale. Derive the
    requested value, check interior/exterior and supplementary/vertical angle orientation,
    then compare the value with every option and verify that the final letter labels the
    matching option. Keep the final answer minimal.
```

H.4 Word Search Exact Skill

The Word Search solver does not use OCR on the standard generated grids. It infers the square grid pitch (25 or 35 pixels), estimates cell boundaries from ink projections, extracts binary glyph masks, forms a median glyph prototype, and computes each cell's normalized Hamming distance to that prototype. For count questions, the largest gap in the sorted distance distribution separates outlier target glyphs from the repeated background glyph. For location questions, the maximum-distance cell is returned with 1-indexed coordinates. Unsupported stylized grids allow one image-sub-agent fallback.

Listing 8: Exact first-step Word Search program.

```
import json
from code_omni_agent.harness.word_search_grid import \
    solve_word_search_task
answer = solve_word_search_task(
    image_1,
    mode="count",  # use "location" only for row/column questions
    question=USER_TASK,
    call_subagent=call_subagent,
    store=store,
)
final(json.dumps(answer, separators=(",", ":")))
```

H.5 Video-MME Dual-View Skill

The complete skill contains detailed rules for counting, numeric facts, synopsis, negation, evidence updates, and error fallback. Its invariant execution contract is reproduced below.

Listing 9: Video-MME invariant procedure.

```
1. Probe the video and classify the question route:
   COUNT, NUMERIC-FACT, NEGATION, or STANDARD.
2. Extract exactly 24 uniform frames over the complete video and construct a contact sheet.
3. Run image-agent pass 1 on contact sheet + frames[0:12].
4. Run an independent pass 2 on contact sheet + frames[12:24].
5. If the passes disagree, compare positive evidence item by item. A third call is allowed
     only for the disputed interval/object, using at most 12 targeted frames.
6. Finalize the best-supported A/B/C/D. Never replace a specific, evidenced answer with a
    later "cannot confirm" response.

Counting must distinguish STATIC_SIMULTANEOUS, UNIQUE_ENTITIES, EVENT_OCCURRENCES,
    CLIP_APPEARANCES, and NUMERIC_FACT. Negation requires True/False/Unknown status for
    every option; Unknown is not False.
```

H.6 MMAU Skills

The planner receives the following first-step contract. The skill runtime sets temperature 0.6, maximum 8,192 sub-agent tokens, and an empty sub-agent system override. The associated guidance is: "Listen to the complete audio, compare all supplied choices against the audible evidence, and return the best-supported option."

Listing 10: Skill of MMAU.

```
info = audio_info(audio_1)
from evaluation.code.MMAU.mmau_skill import solve_mmau_audio
```

```
answer = solve_mmau_audio(
    audio_1,
    prompt=USER_TASK,
    call_subagent=call_subagent,
    store=store,
    guidance="Listen to the complete audio, compare all supplied "
             "choices against the audible evidence, and return the "
             "best-supported option.",
    enable_thinking=True,
    temperature=0.6,
    max_tokens=8192,
    system="",
)
final(answer)
```

H.7 AVUT Alignment Skill

The full skill is an exact executable skill. Its routing logic is:

Listing 11: Skill of AVUT: routing and evidence contract.

```
q = USER_TASK.lower()
appearance = (
    "what do the people" in q or
    "what does the person" in q or
    "what do they look" in q
)
entirety = "does not appear" in q or "not appear" in q
after_text = "after the on-screen text" in q

if not (appearance or entirety or after_text):
    frames = video_extract_frames(video_1, fps=1.0, max_frames=120)
    audio = video_extract_audio_compressed(video_1, bitrate="32k")
    # One omni-agent compares all choices from full chronological evidence.
else:
    audio = video_extract_audio_compressed(video_1, bitrate="32k")
    # An audio-agent returns start_seconds, end_seconds, and  literal
    # transcript/translation without answering the MCQ.
    # prepare_omnivideo_targeted_evidence then constructs synchronized
    # frames and audio around the event.
    # Appearance keeps frames within 1.2 s of the located start.
    # Object-entirety keeps the final ten frames of the target window.
    # After-text starts at the located event end and looks ahead 25 s.
    # A final omni-agent compares every option and returns one letter.
```

Every branch returns only A/B/C/D. The skill explicitly forbids annotations, labels, and scorer state. Audio locator stages are prevented from seeing an answer schema, so their output cannot collapse into an early option guess.

H.8 Deterministic Verification

The default verifier first checks whether any executed step raised a code error. It then parses a nonempty final answer and, for multiple-choice tasks, normalizes it against the available option letters/text. When a reference answer is available, it performs exact or normalized choice matching. A failure passed to repair is reduced to one of four categories: *format*, *task logic*, *evidence mismatch*, or *execution*. The generic task-logic message is only: "answer did not pass acceptance checks; re-examine the evidence and revise." It does not reveal the correct option.

VSP replaces exact string matching with simulation of the submitted move sequence and separately checks path optimality. BLINK-Jigsaw uses the benchmark label for final scoring but its repair feedback contains only a coarse seam-stitch rejection, not the winning candidate or seam losses. Word Search and CoMT use strict task-specific parsing before exact-match scoring.