AMAZE: DIAGNOSING VISUAL-SPATIAL REASONING WITH MATCHED MAZE TWINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models (VLMs) often fail on visual problems that require extended spatial computation, even when they can solve an equivalent symbolic problem. We introduce AMAZE, a diagnostic benchmark of procedurally generated mazes varying in scale, geometry, and visual texture. Its central design is a matched-twin protocol: each solvable maze is paired with a minimally modified unsolvable twin, and a model receives credit only if it classifies both correctly. This exposes a strong bias toward predicting that a path exists. To distinguish failures of visual access from failures of evidence integration, we equip agents with image-processing tools and compare monolithic execution with an orchestrated system of context-isolated specialists using local programmatic verification. Across 16 maze layouts and 1,600 matched pairs evaluated with Gemini 3.5 Flash, the multi-step composition of low-level visual tools lifts paired accuracy from 12.9% to 31.6% for a monolithic agent. Orchestration with local verification further raises it to 67.6%. When a global reachability tool collapses connectivity reasoning to a single call, tools alone lift the monolithic agent to 75.8% and orchestration adds a smaller gain to 87.3%. The same pattern transfers to MentisOculi (1,000 puzzles), where accuracy rises from 58.0% without tools to 71.1% with a monolithic tool-using agent and 73.7% with orchestration. These results show that while visual tools yield large gains across visual reasoning tasks, tool access alone does not eliminate solvability bias: reliable integration of negative visual evidence is critical, and decentralization helps when that integration requires extended composition.

1 INTRODUCTION

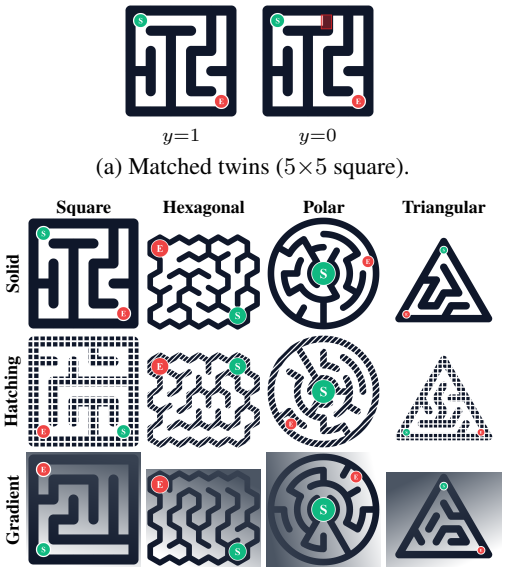
Despite strong performance on broad visual question-answering benchmarks, vision-language models remain unreliable on tasks requiring fine-grained visual perception, spatial relations, and multi-step spatial reasoning (Liu et al., 2023; Fu et al., 2024; Mayer et al., 2025; Li et al., 2026). Performance is particularly brittle when the answer depends on preserving exact spatial constraints over an extended sequence of decisions, rather than recognizing objects or matching local visual patterns (Yang et al., 2025). A maze provides a controlled instance of this problem: local corridor segments are visually repetitive, a single distant wall can change global reachability, and the correct answer can be verified deterministically. Mazes therefore allow us to separate three capabilities that tool-augmented evaluations often conflate: perceiving the relevant structure, performing global inference, and faithfully integrating evidence returned by tools. They also provide a setting in which to ask whether distributing these operations across independently verified specialists makes this reasoning more reliable.

Diagnosing failures of global visual-spatial reasoning is difficult because aggregate accuracy can conceal systematic answer biases. In our experiments, VLMs increasingly default to predicting that a maze is solvable as task difficulty grows; on a balanced dataset, this strategy achieves 50% accuracy without using the image at all. We therefore evaluate models on matched pairs comprising a solvable maze and a nearly pixel-identical counterpart in which a minimal connectivity intervention disconnects the start from the goal (see Figure 1a). A pair is counted as correct only if the model identifies both the path and its absence, directly testing whether its prediction is sensitive to the visual evidence that determines reachability. This distinction is important because finding a path

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.

requires identifying one successful route, whereas establishing that none exists requires maintaining and integrating negative evidence across alternative routes.

Tools provide a natural way to intervene on these failure modes, but prior evaluations report mixed results: visual scratchpads and guided search aid visual reasoning tasks (Wu & Xie, 2024; Hu et al., 2024; Qi et al., 2025), whereas passive preprocessing (`crop`, `resize`) on MentisOculi (Zeller et al., 2026) and open-loop visual programs (Fu et al., 2024; Khandelwal et al., 2023) yield little gain. This tension reflects that tools vary in how much of the task they perform. Low-level operations such as cropping, rotating, and thresholding make visual evidence easier to inspect without determining the answer. Converting a maze into a symbolic representation removes much of the perceptual burden but still leaves connectivity to be inferred, whereas operations such as flood fill or graph search effectively compute reachability. We therefore distinguish perceptual assistance, symbolic representation, and offloaded search (Section 4.1). Comparing these levels of assistance allows us to determine whether failures arise from access to visual evidence, spatial inference, or integration of evidence already available to the model.



(a) Matched twins (5×5 square).

(b) Design axes.

Figure 1: Benchmark overview. **(a)** An unsolvable twin ($y = 0$) is produced by severing critical corridor passages until no start-to-goal path remains (one passage for tree mazes, or multiple when cycles provide alternative routes); red boxes mark severed edges, not shown to the model. **(b)** Four geometries crossed with three wall textures. Maze size varies independently.

Our central question is therefore: *when do visual tools improve visual reasoning, and when should visual computation be distributed across specialist agents rather than performed by a single tool-using agent?* We compare a no-tools baseline, a monolithic agent, and a hierarchical multi-agent system in which a central orchestrator delegates visual operations to context-isolated specialists, each equipped with local programmatic verification. Importantly, the tool-using architectures receive the same underlying tools, allowing us to isolate the effect of how computation and verification are organized. Our results show that visual tools produce large gains across all architectures, while the value of decentralization depends on task difficulty: it offers little or no benefit on easy instances, but substantially improves reliability when agents must compose low-level visual operations over larger and more challenging maze geometries. Our contributions are:

- AMAZE, a controlled maze benchmark varying scale, geometry, and visual texture, together with a matched-pair metric that exposes model bias toward predicting solvability (Figure 1)[1];
- a hierarchy of interventions that distinguishes perceptual assistance, symbolic representation, and offloaded search;
- a controlled comparison between monolithic and hierarchically decentralized agents given the same visual operations; and
- extensive evaluation on AMAZE and out-of-domain visual reasoning tasks (from MentisOculi; Zeller et al. 2026) showing that visual tools yield large lifts even for a single agent while specialist agents with local verification yield further gains when multi-step composition of visual tools is required.

2 AMAZE: A Maze Benchmark for Visual Reasoning

AMAZE is a diagnostic benchmark for visual-spatial reasoning in which models must determine whether a path connects two marked locations in a maze. Each benchmark instance is a matched

[1]Our benchmark and source code are available at: `anonymous_url`

pair of rendered images, (I_i^+, I_i^-), with the same start location s_i and goal location g_i. In the solvable image I_i^+, an unobstructed path connects s_i and g_i, giving label $y_i^+ = 1$. Its nearly pixel-identical twin I_i^- contains a minimal additional barrier that disconnects the two locations, giving label $y_i^- = 0$ (Figure 1a). The model observes each image separately and predicts whether the maze can be solved. Because the intervention changes the correct answer while preserving almost all visual content, the pair tests whether model predictions are sensitive to the evidence that determines connectivity.

2.1 Benchmark Construction

Standard Suite (V1). Summarized in Table 1, V1 provides a standard baseline using familiar square mazes (Figure 1a) at six sizes, $N \in \{3, 5, 7, 9, 12, 16\}$. Colors and rendering are kept uniform, isolating how performance changes as maze size increases. We generate 500 matched pairs at each size, giving 3,000 pairs (6,000 images) in total. We use V1 to evaluate how zero-shot foundation model performance scales with maze size.

Table 1: AMAZE benchmark suites. Each pair contains a solvable and a matched unsolvable image.

	V1	V2
Description	Standard	Multi-geometry
Geometries	Square	4 families
Configurations	6 sizes	16 layouts
Pairs	3,000	1,600
Images	6,000	3,200

Multi-Geometry Suite (V2). V2 tests visual reasoning across a wider variety of mazes. It introduces four geometry families (square, hexagonal, polar, and triangular; Figure 1b) along with visual variations such as hatching and gradients. V2 contains 16 layout configurations: five square and five hexagonal layouts at sizes $N \in \{5, 7, 9, 12, 16\}$, three polar layouts with 4, 6, and 8 rings, and three triangular layouts of sizes 6, 10, and 14. Each layout contains 100 matched pairs, yielding 1,600 pairs (3,200 images; complete generation parameters and dataset split tables appear in Appendix A). We use V2 to evaluate tool-augmented agent architectures across varied geometries and visual styles (ablations on V2 are in Appendix B.4).

Each maze is generated procedurally and stored together with its adjacency graph, start and goal locations, rendering configuration, and graph-derived reachability label. The graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of maze cells identified by their coordinates in the layout, and $(u, v) \in \mathcal{E}$ whenever u and v are neighbors under the layout geometry and the wall between them is open. Neighborhood is fixed by the geometry: four neighbors for square layouts, six for hexagonal, three for triangular, and radial and circumferential neighbors for polar rings. An instance is solvable exactly when g_i is reachable from s_i in $\mathcal{G}$, which is the label we store. Standard maze generators produce perfect mazes containing a unique path between any two cells. We additionally generate braided variants (Buck, 2015) by reconnecting dead ends with ratio $r_{\text{braid}} \in \{0, 0.3, 0.6\}$, introducing loops and alternative routes. This prevents models from relying on simple structural heuristics tied to tree-like mazes. The graph is used only to construct and validate the benchmark and is not provided to models in the image-based evaluation.

2.2 Matched-Twin Evaluation

Each solvable maze I_i^+ is paired with an unsolvable twin I_i^- by inserting a barrier intervention that disconnects the start s_i from the goal g_i (Figure 1a). On the graph, we iteratively find a shortest remaining s_i–g_i path via breadth-first search and sever one uniformly sampled edge along it until s_i and g_i lie in different connected components, realizing each deleted edge in the image by closing the corresponding corridor. For perfect tree mazes ($r_{\text{braid}} = 0$), every edge on the unique s_i–g_i path is a bridge, so closing a single corridor ($k = 1$) suffices; for braided mazes ($r_{\text{braid}} > 0$), where cycles introduce alternative detours, $k \geq 1$ corridors are closed until all connecting paths are severed. We retain a pair only if g_i is reachable from s_i in I_i^+ but not in I_i^-. For a given twin, all the visual content, including the rendering, start and goal locations, and unaffected corridors, remains unchanged. The model receives the two images separately and predicts $f(I) \in \{0, 1\}$, where 1 denotes solvable and 0 denotes unsolvable.

$$\text{Acc}_{\text{pair}} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}[f(I_i^+) = 1 \ \wedge \ f(I_i^-) = 0]. \tag{1}$$

A pair is correct only if the model identifies both the original path and its absence after the intervention. Always-solvable and always-unsolvable classifiers therefore both score 0%, while independent random guessing scores 25% in expectation. We report accuracy on the solvable and unsolvable images separately to expose directional error patterns that paired accuracy alone does not distinguish.

3 Zero-Shot Diagnosis

We use V1 to measure how zero-shot performance changes with spatial horizon and how much difficulty comes from recovering maze structure from pixels. We evaluate three proprietary models (Gemini 3.5 Flash, Gemini 3.5 Flash-Lite, GPT-5.6 Sol) and two open-weights models (Gemma 4 31B, Qwen 3.8 27B) on 500 matched pairs for each maze scale using either the rendered image or an *oracle ASCII* text-only grid derived from the graph. ASCII removes perception but leaves connectivity inference to the model; it is a diagnostic upper bound, not a tool available to the agents. Figure 2 shows paired accuracy across maze scales for all models; full results and model cards are presented in Appendix B. Each experiment was run and averaged over 3 seeds (except GPT-5.6 Sol, which was run with one seed for budget constraints).

We observe that all open-weights and Gemini configurations fall below 3% paired accuracy by 9×9 (Figure 2a). For instance, Gemini 3.5 Flash drops from 99.3% at 3×3 to 1.6% at 9×9. GPT-5.6 Sol also confirms this ceiling: while achieving 100.0% accuracy on 3×3, its performance falls to 81.0% by 7×7, drops to 39.8% at 9×9, and falls below the 25.0% random baseline by 16×16 (20.8%), despite generating substantial (5,000+) reasoning tokens. Decoding parameters, prompts, and evaluation details are in Appendix B. We also find that errors are strongly asymmetric: with increasing maze scale, models default to predicting that a path exists. At 9×9, Gemini 3.5 Flash predicts "solvable" on 98.7% of images (Appendix Table B.2), yielding 98.3% false positives. Its per-image accuracy remains near 50%, but paired accuracy is only 1.6%.

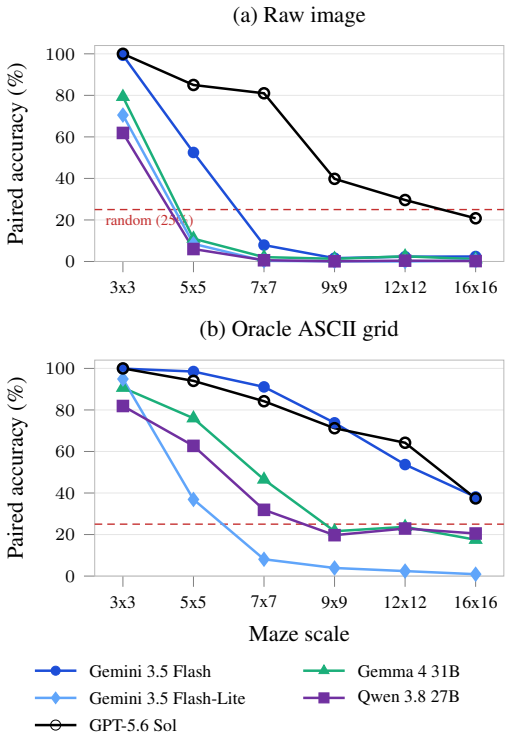


Figure 2: Zero-shot paired accuracy under raw-image and oracle-ASCII input ($M = 500$ matched pairs, 1,000 episodes per scale). The dashed line marks the random baseline (25%). Raw-image accuracy collapses with scale; ASCII raises performance but does not remove the decline. Full results and token counts appear in Appendix B.

Oracle ASCII yields large gains, confirming that visual structure recovery is a major bottleneck. At 9×9, Gemini 3.5 Flash rises from 1.6% to 73.8% paired accuracy. Yet its ASCII score still falls from 99.9% at 3×3 to 38.1% at 16×16, showing that long-horizon connectivity inference also remains difficult. Across all models, while oracle ASCII substantially improves performance, accuracy still declines toward or below the 25% random baseline as scale expands (GPT-5.6 Sol, Qwen 3.8 27B, Gemma 4 31B, and Flash-Lite fall to 37.4%, 20.5%, 17.5%, and 0.9% at 16×16), confirming that reasoning degrades even *without* perceptual limitations. Moreover, automatic image-to-ASCII transcription fails beyond small grids (Appendix B.3). Even the strongest transcription model (Gemini 3.5 Flash) recovers only 18.0% of 3×3 grids exactly, and 0.0% at every larger size; oracle ASCII is therefore a diagnostic, not a practical solution.

These findings motivate moving from passive visual inspection to active transformation of the image. Deterministic operations such as thresholding, morphology, and connected-component analysis can expose corridor structure, reduce visual clutter, and externalize intermediate state that would otherwise have to be maintained within the model's context. Related computational precedents in-

clude VisProg and ViperGPT, which use language models to compose executable vision modules for multi-step visual reasoning (Gupta & Kembhavi, 2023; Surís et al., 2023). Unlike these systems, however, we restrict agents to low-level image-processing primitives that do not solve reachability directly.

4 Tools and Agent Architectures

We compare two architectures that receive identical image-processing tools but organize their use differently: a monolithic agent executes all operations in a single context and a hierarchical multi-agent system with an orchestrator delegates operations to context-isolated specialists with local verification. Keeping the toolset fixed isolates how computation and verification are organized.

4.1 Low-Level Image Primitives

Each tool applies a deterministic operation f, parameterized by θ, to an input image I:

$$o = f(I; \theta). \tag{2}$$

Depending on the operation, the output o may be a transformed image, a binary mask, an integer label map, or a list of coordinates. We implement these tools with standard computer vision libraries (OpenCV; Bradski 2000, scikit-image; van der Walt et al. 2014, and Pillow; Clark et al. 2015) (detailed tool signatures in Appendix C). These primitives expose and simplify pixel-level structure, but they do not encode maze-specific knowledge. The agent remains responsible for selecting parameters, composing operations, locating the start and goal in the resulting representations, and deciding whether a traversable path connects them. We group the operations into five capabilities, each owned by one specialist in Section 4.3: (1) *transform*, localized sub-region cropping and magnification (`crop_image`) and orthogonal rotation (`rotate_image`); (2) *binarize*, isolating traversable corridors from gradients and wall textures via global or adaptive thresholding and smoothing (`threshold_image`, `adaptive_threshold`, `gaussian_blur`); (3) *topology*, morphological filtering and extracting 1-pixel centerlines and boundary contours (`erode`/`dilate`, `skeletonize_image`, `find_and_draw_contours`); (4) *annotate*, projecting waypoint markers and candidate paths onto the image canvas (`draw_line`, `draw_circle`, `draw_rectangle`, `draw_polyline`); and (5) *reachability*, evaluating connectivity of image components (`flood_fill`, `connected_components`). Unlike the other four, the image tools within (5) can possibly resolve maze connectivity in a single step when called with the right arguments, whereas the atomic suite from (1) to (4) requires composing topological masks and trace overlays over multiple steps.

To evaluate multi-step visual reasoning without relying on reachability solvers, we construct two tool suites: the *atomic suite*, which excludes the reachability tools (5) and is our main condition, and the *flood-fill suite*, which contains all tools and serves as a control (detailed system prompt templates appear in Appendix G).

4.2 Monolithic Tool Use

The monolithic baseline follows a standard tool-use loop (Yao et al., 2023) summarized in Figure 3. A single agent selects every operation and its input arguments, inspects its output, and submits the final prediction. After t calls its history is

$$H_t = (I_0, a_1, o_1, \ldots, a_t, o_t), \tag{3}$$

where a_t is a tool call and o_t its returned artifact or structured output. Thus planning, parameter tuning, visual inspection, and error recovery all occur in one growing context. Failed thresholds, intermediate masks, and alternative contours remain available to later turns. This can be useful on short trajectories, but it also requires the agent to distinguish valid evidence from superseded or erroneous intermediate results.

4.3 Hierarchical Multi-Agent Execution

The multi-agent architecture decentralizes tool execution while retaining centralized planning and decision making. An orchestrator decomposes the problem into subgoals, delegates their execution

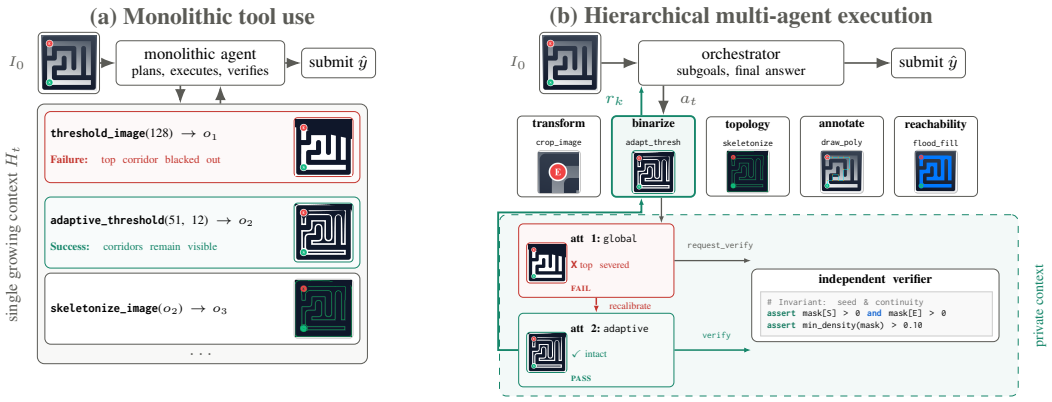


Figure 3: Both architectures receive the same input I_0 (here a gradient maze) and the same visual tool suite. **(a)** The monolithic agent accumulates all tool calls and outputs in a single history H_t: an initially poor global threshold (o_1) persists in context even after adaptive thresholding (o_2) and skeletonization (o_3) recover the corridors. **(b)** The orchestrator delegates subgoals (a_t) to context-isolated specialists (*transform*, *binarize*, *topology*, *annotate*, *reachability*). Each specialist verifies outputs locally against task-agnostic programmatic checks, retries when verification fails, and returns only the verified record r_k, discarding intermediate attempts.

to context-isolated specialists, integrates the returned evidence, and submits the final answer. Thus, decentralization refers specifically to where tool execution and verification take place, not to who determines the overall strategy or prediction. Each specialist owns one of the five capability groups defined in Section 4.1 and is invoked only for operations in that group.

Let k index the specialist selected by the orchestrator. At step t, a delegation specifies the specialist, an instruction, and the artifacts $\mathcal{A}_{\text{in}}$ required as input:

$$a_t = \text{Delegate}\left(k, \text{instruction}_t, \mathcal{A}_{\text{in}}\right). \tag{4}$$

The specialist executes the instruction within a private context and returns

$$r_k = \langle \texttt{status}_k, s_k, \mathcal{A}_{\text{out},k} \rangle, \qquad \texttt{status}_k \in \{\text{PASS}, \text{FAIL}\}. \tag{5}$$

Here, s_k is a compact textual summary of the operation performed, the evidence obtained, and the outcome of local verification, while $\mathcal{A}_{\text{out},k}$ contains references to any verified output artifacts.

Before this record is returned, the proposed output undergoes programmatic *verification*. The verifier checks properties appropriate to the artifact, using free-form code, including pixel-level checks, preservation of dark wall pixels, or the absence of threshold leakage across a barrier. If a check succeeds, the specialist returns PASS together with its summary and verified artifacts. A failed check triggers local recalibration: the orchestrator can make up to 20 specialist delegations per episode, each permitting up to 10 specialist–verifier interactions (matching the 200-step single-agent ceiling) with up to 3 tries for specialist tool calls or verifier code execution. If all attempts fail, the specialist returns FAIL, a concise failure report in s_k, and no output artifact. Importantly, PASS indicates that the delegated operation passed its local checks, not that the maze is solvable. Verifiers may inspect the candidate artifact and source pixels, but they cannot invoke a reachability solver or submit the task label.

Only the returned record enters the orchestrator's context (see Appendix C.2 for further details on artifact and multimodal context management). Intermediate images, parameter sweeps, and failed retries remain within the specialist's private context and are discarded when the sub-session ends. Relative to the monolithic loop, this architecture therefore introduces both context isolation and local verification. Coupling verification with context isolation allows specialists to detect and correct errors where they arise, while preventing failed attempts and unverified artifacts from entering the orchestrator's context. In Section 5, we also investigate the same setup without local verification for comparison.

5 EXPERIMENTS

We compare the monolithic and multi-agent architectures using Gemini 3.5 Flash on the V2 mazes. We select Gemini 3.5 Flash because it demonstrated strong capacity on the oracle ASCII grids while failing on raw images (Section 3), providing a clean testbed to see if visual tools can bridge this perceptual gap (we report scaling results with Flash-Lite in Appendix D.2). Both systems are evaluated on the same 16 layouts and receive the same image-processing tools. Our main setting uses only *atomic* image operations, so the agent must determine connectivity by composing multiple tool calls. As a control, we additionally provide the *flood-fill suite*, which can test reachability directly. For each setting, we compare a single tool-using agent with multi-agent execution both without and with local verification. We also report a no-tools baseline. Each layout contains 100 matched pairs. V2 was not used to design the prompts, specialist roles, or verification procedures. Full layout-level results appear in Appendix D.

Tools yield large lifts across architectures, while multi-agent execution helps when reasoning requires many tool calls. First, visual tools substantially improve accuracy over raw-image inference (no-tools: 12.9%): a single agent reaches 31.6% with atomic primitives and 75.8% with flood fill (Appendix D.3 shows that structure-exposing rather than cropping tools drive these gains).

Table 2: Paired accuracy (%) for Gemini 3.5 Flash across 16 Benchmark V2 layouts ($M = 100$ matched pairs per layout), by tool suite and architecture. The No tools baseline repeats in both blocks.

Tool suite	Layouts	No tools	Single	Multi-noV	Multi+Ver
Atomic (main)	Square (5)	13.6	41.6	51.6	**74.8**
	Hexagonal (5)	8.4	20.8	46.6	**58.2**
	Polar (3)	15.0	25.7	38.7	**68.7**
	Triangular (3)	17.0	38.7	48.3	**70.3**
	Overall (16)	**12.9**	**31.6**	**47.0**	**67.6**
Flood-fill (control)	Square (5)	13.6	89.6	89.6	**91.0**
	Hexagonal (5)	8.4	66.4	72.8	**83.2**
	Polar (3)	15.0	59.0	75.7	**85.0**
	Triangular (3)	17.0	85.3	85.7	**90.3**
	Overall (16)	**12.9**	**75.8**	**81.0**	**87.3**

Second, the benefit of the multi-agent architecture depends strongly on how much intermediate evidence the model must integrate. With the atomic tools (Table 2, top), solving a maze requires composing a long sequence of operations. Here, moving from a single agent to unverified multi-agent execution increases paired accuracy from 31.6% to 47.0%, and adding local verification further raises it to 67.6%. With the flood-fill suite (Table 2, bottom), by contrast, reachability is resolved over much shorter episodes. In this setting, unverified delegation moves accuracy from 75.8% to 81.0%, while verification raises it to 87.3%. Thus, decomposition is most useful when the task requires integrating evidence across an extended sequence of operations; when the relevant computation is resolved over a short trace, the additional architecture provides much less benefit.

Flood fill isolates tool-use errors from connectivity reasoning. Unlike the atomic tools, `flood_fill` called with the correct arguments can directly determine whether the start and goal lie in the same reachable region, largely removing the need for multi-step spatial reasoning. We therefore use it as a control for whether the agent can correctly ground, configure, and interpret a tool that performs the central connectivity computation. Even with flood fill, the tool is not a trivial shortcut. The monolithic agent reaches 75.8% paired accuracy, but still runs into errors such as misplacing seed points, selecting improper fill radius near sharp turns or hatched walls, or failing to set tolerances that work across gradients.

Verified multi-agent execution improves accuracy across all geometry families and scales. Table D.1 breaks down performance by maze geometry and scale. With the atomic tool suite, verified multi-agent execution improves paired accuracy over the monolithic agent on every geometry family, lifting all four from 20.8%–41.6% to 58.2%–74.8%. The advantage widens with maze size: on square layouts the monolithic agent falls from 72.0% at 5×5 to 27.0% at 16×16, while verified multi-agent execution still reaches 67.0% at the largest size, and hexagonal and triangular layouts show the same pattern. Context isolation alone does not ac-

Table 3: Accuracy (%) by tool suite and architecture for Gemini 3.5 Flash on matched pairs and by ground-truth label (Δ = Multi+Ver − Single).

Suite	Architecture	Paired	Solvable	Unsolv.
Baseline	No tools	12.9	**92.2**	15.2
Atomic (main)	Single	31.6	73.7	45.0
	Multi+Ver	**67.6**	**77.7**	**87.8**
	Diff. (Δ)	+36.0	+4.0	+42.8
Flood-fill (control)	Single	75.8	90.2	83.2
	Multi+Ver	**87.3**	**90.3**	**96.9**
	Diff. (Δ)	+11.5	+0.1	+13.7

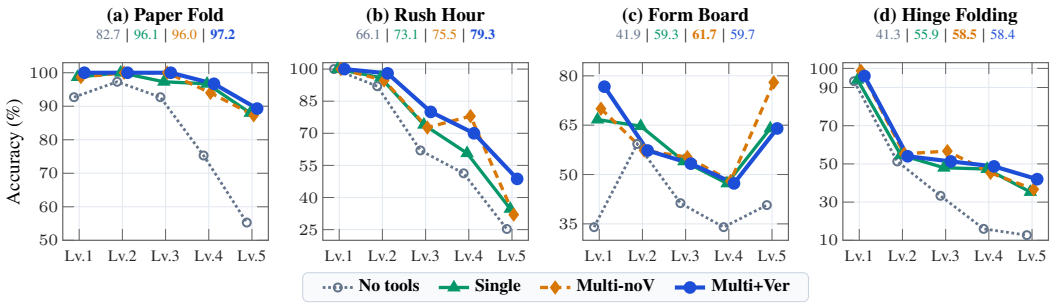


Figure 4: Out-of-domain transfer on MentisOculi across complexity Levels 1–5 with Gemini 3.5 Flash averaged over 3 seeds. Subplot headers list mean task accuracy (**No tools** | **Single** | **Multi-noV** | **Multi+Ver**).

count for this: without verification, multi-agent execution improves on the monolithic baseline in every family, reaching 47.0% overall.

Performance gains come from better detection of unsolvable mazes. Table 3 shows that the improvement in paired accuracy is highly asymmetric. Tools lift single-agent unsolvable accuracy from 15.2% to 45.0% and 83.2%. With the atomic tool suite, verified multi-agent execution slightly increases accuracy on solvable mazes from 73.7% to 77.7%, while greatly increasing accuracy on their unsolvable twins from 45.0% to 87.8%. The flood-fill control shows the same pattern in milder form: solvable accuracy holds steady from 90.2% to 90.3%, while unsolvable accuracy rises from 83.2% to 96.9%. Thus, the multi-agent system improves paired accuracy primarily by correcting the strong tendency of the monolithic agent to predict that a path exists.

Local verification preserves solvable accuracy by allowing specialists to retry when visual processing does not complete successfully. It substantially reduces false positives, where the monolithic agent often accepts spurious paths produced by overly permissive image processing or repeated relaxation of tool parameters. In paired evaluation, this reduction in false positives drives the +36.0 pp and +11.5 pp gains in paired accuracy. Detailed results for solvable and unsolvable mazes are provided in Appendix D.

Context isolation and local verification drive performance gains, not longer rollouts. Table 4 compares architectures under a matched 200-step ceiling (in the multi-agent case, this budget is split across up to 20 specialist delegations of 10 interactions; full details and a shorter-context $c = 50$ single-agent ablation in Appendix E). In all settings, input prefill accounts for over 98% of total token usage, while generated output scales linearly with tool calls. Despite its 200-step budget, Single stops after an average of 20 atomic tool calls when defaulting to an early prediction. Retaining every intermediate image in one history leads to quadratic prefill growth (1,086K tokens) and yields only 31.6% paired accuracy. In contrast, delegation introduces separate specialist contexts (279K specialist prefill), discarding intermediate trial images. This cuts the orchestrator's context to 129K and raises paired accuracy to 47.0%. Finally, local verification expends 632K prefill tokens on sandboxed Python checks (80 total operations across visual calls and Python sandbox runs). By leaving out invalid images before returning, it compresses orchestrator prefill to 65K and achieves 67.6% paired accuracy using fewer total tokens than Single (994K vs. 1,088K).

Table 4: Mean cumulative input prefill (in thousands of tokens), total generated output, and number of tool calls for Gemini 3.5 Flash by tool suite and architecture.

Tool suite	Arch.	Input prefill (K) Orch.	Sub.	Verif.	Out (K)	Vis. calls	Acc. pair
Atomic (main)	Single	1086	—	—	2	20	31.6
	Multi-noV	129	279	—	5	20	47.0
	Multi+Ver	65	282	632	15	14	**67.6**
Flood-fill (control)	Single	223	—	—	1	8	75.8
	Multi-noV	88	186	—	3	15	81.0
	Multi+Ver	54	197	531	13	11	**87.3**

Cross-task transfer reveals larger gains on tasks requiring multi-step spatial composition. To test whether our findings generalize beyond maze reachability, we evaluate the same specialist architecture and local verification protocol on MentisOculi (Zeller et al., 2026), a procedural benchmark of 1,000 puzzles spanning four multi-step visual reasoning tasks with 5 difficulty levels each: *Paper Fold*, *Rush Hour*, *Form Board*, and *Hinge Folding*. We evaluate each architecture with Gemini 3.5 Flash over 3 seeds (12,000 episodes in total); full results are in Appendix F. Figure 4 shows that the main pattern observed on AMA2E extends beyond mazes: from a no-tools baseline at 58.0%,

visual manipulation tools lift the single agent to 71.1% when averaged over all four tasks, but verified delegation is not uniformly better than a single tool-using agent. On *Paper Fold*, for example, the single agent already achieves 96.1% accuracy, compared with 97.2% for Multi+Ver. In contrast, consistent with Zeller et al. (2026, Section 4.4), who found cropping tools yielded no gain on *Rush Hour*, our single agent gives the smallest lift on *Rush Hour* compared to the no-tools baseline ($66.1\% \rightarrow 73.1\%$, stalling at 34.7% at Level 5), where verified delegation gives its clearest gains: Multi+Ver reaches 79.3% compared with 73.1% for Single (opening a $+14.0$ pp margin over Single at Level 5), while on *Hinge Folding* it reaches 58.4% vs. 55.9% (a $+6.7$ pp margin at Level 5). Averaged over all tasks, Multi+Ver achieves 73.7% accuracy, compared with 71.1% for Single and 72.9% for Multi-noV. These results suggest that the benefit of verified delegation transfers beyond mazes, but is largest when solving the task requires coordinating multiple intermediate spatial operations.

6 Related Work

Visual and spatial reasoning benchmarks. Controlled visual benchmarks use abstract images to isolate geometric, relational, and multi-step reasoning, including Bongard-LOGO (Nie et al., 2020), PuzzleVQA (Chia et al., 2024), AlgoPuzzleVQA (Ghosal et al., 2025), MentisOculi (Zeller et al., 2026), and JigShape (Li et al., 2026). AMAZE provides a complementary diagnostic: reachability is a non-local topological property where a minimal barrier reverses ground truth while leaving local visual evidence virtually unchanged. Unlike prior maze suites restricted to solvable grids or single-turn square lattices (Schwarzschild et al., 2021; Ivanitskiy et al., 2023; Einarsson, 2026; Rodriguez Salgado, 2026), our matched-twin protocol across four geometries exposes reachability bias (Appendix A.3).

External tools for visual reasoning. External representations can reduce visual-search and working memory demands (Kirsh & Maglio, 1994; Larkin & Simon, 1987), motivating active perception (Wu & Xie, 2024; Qi et al., 2025; Su et al., 2025; Zheng et al., 2026), visual scratchpads, and iterative transformations (Hu et al., 2024; Harvey & Wood, 2023; Hsu et al., 2023; Zhang et al., 2025a; Wiedemer et al., 2025). Visual-programming systems like VisProg (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023; Khandelwal et al., 2023) compose vision modules for multi-step reasoning. However, unverified tool composition or passive preprocessing (`crop`, `resize`) yields limited gains on fine-grained spatial tasks (Fu et al., 2024; Zeller et al., 2026). We pair deterministic operations with programmatic verification, distinguishing atomic tools that expose intermediate evidence from global operators like `flood_fill` that perform reachability computation.

Multi-agent decomposition and verification. Specialized agents can distribute perception, execution, and criticism across separate contexts (Ma et al., 2026; Zarei et al., 2025; Yang et al., 2026), reducing interference between heterogeneous subtasks. However, multi-agent systems introduce inference and communication overhead and do not reliably outperform strong single agents (Cemri et al., 2025; Zhang et al., 2025b). Moreover, self-verification often reproduces rather than corrects errors (Stechly et al., 2025). Execution-grounded verification offers a stronger alternative: candidate programs or artifacts are checked against testable invariants (Chen et al., 2023; Kambhampati et al., 2024). Our architecture combines hierarchical delegation, context isolation, and local programmatic verification under a central orchestrator. We study when this scaffold improves reliability enough to justify its cost, rather than assuming more agents are uniformly better.

7 Conclusion

This work investigated why VLMs fail on extended visual-spatial reasoning and when distributing computation across specialist agents improves performance. We introduced AMAZE, a controlled maze benchmark whose matched-twin design exposes a strong reachability bias, paired with a tool hierarchy separating visual access, spatial inference, and evidence integration. While visual tools substantially improve single-agent accuracy, tool access alone does not eliminate reachability bias: gains are largest when multi-step visual operations are distributed across context-isolated specialists with local verification. Evaluations on MentisOculi show that this advantage extends to complex spatial reasoning. Future work should investigate whether these findings generalize across broader VLM families and develop adaptive systems that invoke specialist decomposition only when expected reliability gains justify the inference cost.

Reproducibility Statement

The AMAZE benchmark and source code are available at `anonymous_url`. Appendix A documents the details of the data generation; Appendix B provides full zero-shot diagnostic results on Benchmark V1 and specifies evaluation setup and exact model versions; Appendix C specifies tool interfaces and execution hyperparameters; Appendices D–E provide complete Benchmark V2 ablation results and compute profiling; Appendix F reports details on the out-of-domain transfer evaluations on MentisOculi; and Appendix G provides complete prompt templates.

AI Use Statement

In this work, we used generative AI tools for implementing methods (specifically, the agent evaluation harness and tool wrappers), cleaning and reformatting experimental logs, and supporting qualitative trace analysis of failure modes. Formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and assisting with translation are not applicable to this work. We have not used generative AI tools for generating synthetic datasets (all benchmark mazes are procedurally generated via deterministic graph algorithms), helping develop conceptual frameworks, proposing or refining hypotheses, designing experiments, or interpreting results.

Additionally, we used generative AI tools for drafting and editing parts of the research paper to improve readability, suggesting paper structure, creating auxiliary scripts for data aggregation and plotting, modifying scientific figures, and searching related literature. We have reviewed all AI-assisted work. Specifically, LLM-generated code and analysis scripts were verified and tested for correctness against raw experimental logs across all benchmark evaluations. All AI-assisted text, tables, and figures were rigorously reviewed, edited, and validated by the authors to ensure scientific accuracy. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242. Curran Associates, Inc., 2022. doi: 10.52202/068431-1471.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

Jamis Buck. *Mazes for Programmers: Code Your Own Twisty Little Passages*. The Pragmatic Bookshelf, 2015. ISBN 978-1-68050-055-4.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A Zaharia, Joseph Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 135676–135729, 2025. doi: 10.52202/085713-4082. Datasets and Benchmarks Track.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations (ICLR)*, 2023.

Yew Ken Chia, Vernon Toh, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. PuzzleVQA: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16259–16273, 2024. doi: 10.18653/v1/2024.findings-acl.962.

Alex Clark et al. Pillow (PIL Fork) documentation. `https://python-pillow.org`, 2015.

Hafsteinn Einarsson. MazeEval: A benchmark for testing sequential decision-making in language models. In *Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC)*, pp. 407–418. ELRA Language Resource Association, May 2026. doi: 10.63317/4nm93hckcaf2.

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 (ECCV)*, pp. 148–166, 2024. doi: 10.1007/978-3-031-73337-6_9.

Gemini Team, Google DeepMind. Gemini 3.5 Flash model card. Technical report, Google DeepMind, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Gemini Team, Google DeepMind. Gemini 3.5 Flash-Lite model card. Technical report, Google DeepMind, 2026b. URL https://deepmind.google/models/model-cards/gemini-3-5-flash-lite/.

Gemma Team, Google DeepMind. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Deepanway Ghosal, Vernon Toh, Yew Ken Chia, and Soujanya Poria. AlgoPuzzleVQA: Diagnosing multimodal reasoning challenges of language models with algorithmic multimodal puzzles. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pp. 9615–9632, 2025. doi: 10.18653/v1/2025.naacl-long.486.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962, 2023. doi: 10.1109/CVPR52729.2023.01436.

William Harvey and Frank Wood. Visual chain-of-thought diffusion models. In *ICML Workshop on Structured Probabilistic Inference and Generative Modeling (SPIGM)*, 2023. URL https://arxiv.org/abs/2303.16187.

Joy Hsu, Gabriel Poesia, Jiajun Wu, and Noah Goodman. Can visual scratchpads with diagrammatic abstractions augment LLM reasoning? In *Proceedings on "I Can't Believe It's Not Better: Failure Modes in the Age of Foundation Models" at NeurIPS 2023 Workshops*, volume 239 of *Proceedings of Machine Learning Research*, pp. 21–28. PMLR, 16 Dec 2023. URL https://proceedings.mlr.press/v239/hsu23a.html.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 139348–139379, 2024. doi: 10.52202/079017-4423.

Michael Igorevich Ivanitskiy, Rusheb Shah, Alex F. Spies, Tilman Räuker, Dan Valentine, Can Rager, Lucia Quirke, Chris Mathwin, Guillaume Corlouer, Cecilia Diniz Behn, and Samy Wu Fung. A configurable library for generating and manipulating maze datasets. *arXiv preprint arXiv:2309.10498*, 2023.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil B. Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235, pp. 22895–22907, 2024.

Apoorv Khandelwal, Ellie Pavlick, and Chen Sun. Analyzing modular approaches for visual question decomposition. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2590–2603. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.157.

David Kirsh and Paul P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, 1994. doi: 10.1207/s15516709cog1804_1.

Jill H. Larkin and Herbert A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65–100, 1987. doi: 10.1111/j.1551-6708.1987.tb00863.x.

Shawn Li, Wei Yang, Jike Zhong, Jiate Li, Jiawei Yang, You Qin, Ryan Rossi, Franck Dernoncourt, Roger Zimmermann, Yue Wang, Zhengzhong Tu, Vicente Ordonez, Mohit Bansal, and Yue Zhao. JigShape: Evaluating visual-geometric reasoning in VLMs through jigsaw puzzles. *arXiv preprint arXiv:2607.27670*, 2026.

Fangyu Liu, Guy Emerson, and Nigel Collier. Visual spatial reasoning. *Transactions of the Association for Computational Linguistics (TACL)*, 11:635–651, 2023. doi: 10.1162/tacl_a_00566.

Shichao Ma, Yunhe Guo, Jiahao Su, Qihe Huang, Zhengyang Zhou, and Yang Wang. Talk2Image: A multi-agent system for multi-turn image generation and editing. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 32437–32445, 2026. doi: 10.1609/aaai.v40i38.40519.

Julius Mayer, Mohamad Ballout, Serwan Jassim, Farbod Nosrat Nezami, and Elia Bruni. iVISPAR: An interactive visual-spatial reasoning benchmark for VLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 26757–26781. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1359.

Weili Nie, Zhiding Yu, Lei Mao, Ankit B. Patel, Yuke Zhu, and Animashree Anandkumar. Bongard-LOGO: A new benchmark for human-level concept learning and reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 16468–16480, 2020.

OpenAI. GPT-5.6 system card. Technical report, OpenAI, 2026. URL `https://deploymentsafety.openai.com/gpt-5-6`.

Ji Qi, Ming Ding, Weihan Wang, Yushi Bai, Qingsong Lv, Wenyi Hong, Bin Xu, Lei Hou, Juanzi Li, Yuxiao Dong, and Jie Tang. CogCoM: A visual language model with chain-of-manipulations reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Qwen Team. Qwen3.8: Towards native multimodal agents, August 2026. URL `https://qwen.ai/blog?id=qwen3.8`.

Alberto Gonzalo Rodriguez Salgado. From pixels to BFS: High maze accuracy does not imply visual planning. In *Proceedings of the 4th Workshop on Advances in Language and Vision Research (ALVR)*, pp. 164–179. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.alvr-main.13.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 6695–6706, 2021.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations (ICLR)*, 2025.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel Reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 8222–8251, 2025. doi: 10.52202/085713-0280.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via Python execution for reasoning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11854–11864, 2023. doi: 10.1109/ICCV51070.2023.01092.

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, et al. scikit-image: image processing in Python. *PeerJ*, 2:e453, 2014. doi: 10.7717/peerj.453.

Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.

Penghao Wu and Saining Xie. V^*: Guided visual search as a core mechanism in multimodal LLMs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13084–13094, 2024. doi: 10.1109/CVPR52733.2024.01243.

Bangji Yang, Ruihan Guo, Jiajun Fan, Chaoran Cheng, and Ge Liu. M3: High-fidelity text-to-image generation via multi-modal, multi-agent and multi-round visual reasoning. *arXiv preprint arXiv:2602.06166*, 2026.

Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in space: How multimodal large language models see, remember, and recall spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10632–10643, 2025. doi: 10.1109/CVPR52734.2025.00994.

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 (ICLR)*, 2023.

Arman Zarei, Jiacheng Pan, Matthew Gwilliam, Soheil Feizi, and Zhenheng Yang. AgentComp: From agentic reasoning to compositional mastery in text-to-image models. *arXiv preprint arXiv:2512.09081*, 2025.

Jana Zeller, Thaddäus Wiedemer, Fanfei Li, Thomas Klein, Prasanna Mayilvahanan, Matthias Bethge, Felix A. Wichmann, Ryan Cotterell, and Wieland Brendel. MentisOculi: Revealing the limits of reasoning with mental imagery. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL `https://jana-z.github.io/mentis-oculi`.

Huanyu Zhang, Wenshan Wu, Chengzu Li, Ning Shang, Yan Xia, Yangyu Huang, Yifan Zhang, Li Dong, Zhang Zhang, Liang Wang, Tieniu Tan, and Furu Wei. Latent Sketchpad: Sketching visual thoughts to elicit multimodal reasoning in MLLMs. *arXiv preprint arXiv:2510.24514*, 2025a.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267, pp. 76583–76599. PMLR, 2025b. URL `https://proceedings.mlr.press/v267/zhang25cq.html`.

Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. DeepEyes: Incentivizing thinking with images via reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

A BENCHMARK DETAILS

In this section, we describe the maze generation algorithm, the benchmark splits for AMAZE, and the related maze literature.

A.1 MAZE GENERATION

We generate mazes using an underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where vertices represent maze cells and edges represent open passages. We generate graphs using randomized depth-first search (recursive backtracker) to produce long, winding corridors. We introduce cycles in the graph via dead-end braiding with reconnection ratio $r_{\text{braid}} \in \{0, 0.3, 0.6\}$ (Buck, 2015). Mazes with $r_{\text{braid}} = 0$ have a unique start-to-exit path and zero cycles, whereas $r_{\text{braid}} \in \{0.3, 0.6\}$ re-links that fraction of dead-ends back to adjacent corridors. Because each corridor loop encloses a detached wall segment, both solvable and unsolvable mazes contain multiple disconnected wall components, preventing models from predicting maze solvability using wall-island counting shortcuts.

We implement four geometry families (Figure 1b; Section 2.1): Square (4-connected orthogonal lattice), Hexagonal (6-connected lattice), Polar (concentric rings subdivided outward to preserve corridor width), and Triangular (3-connected lattice). Figure A.1 shows illustrative examples of large-scale mazes. To maximize path length and search depth, start and exit locations are placed at random on opposing boundaries: outer corners for square and hexagonal grids, center-to-perimeter for polar, and outer vertices for triangular layouts.

A.2 DATASET SPLITS

Across V1 and V2, the benchmark comprises $4{,}600$ matched pairs ($9{,}200$ images) rendered at 1024×1024 resolution across three visual styles: solid, hatching, and gradient. Mazes are grouped by `pair_id` so that maze pairs are evaluated under identical conditions and never separated across dataset partitions.

Table A.1 details the configuration breakdown across both evaluation Benchmarks:

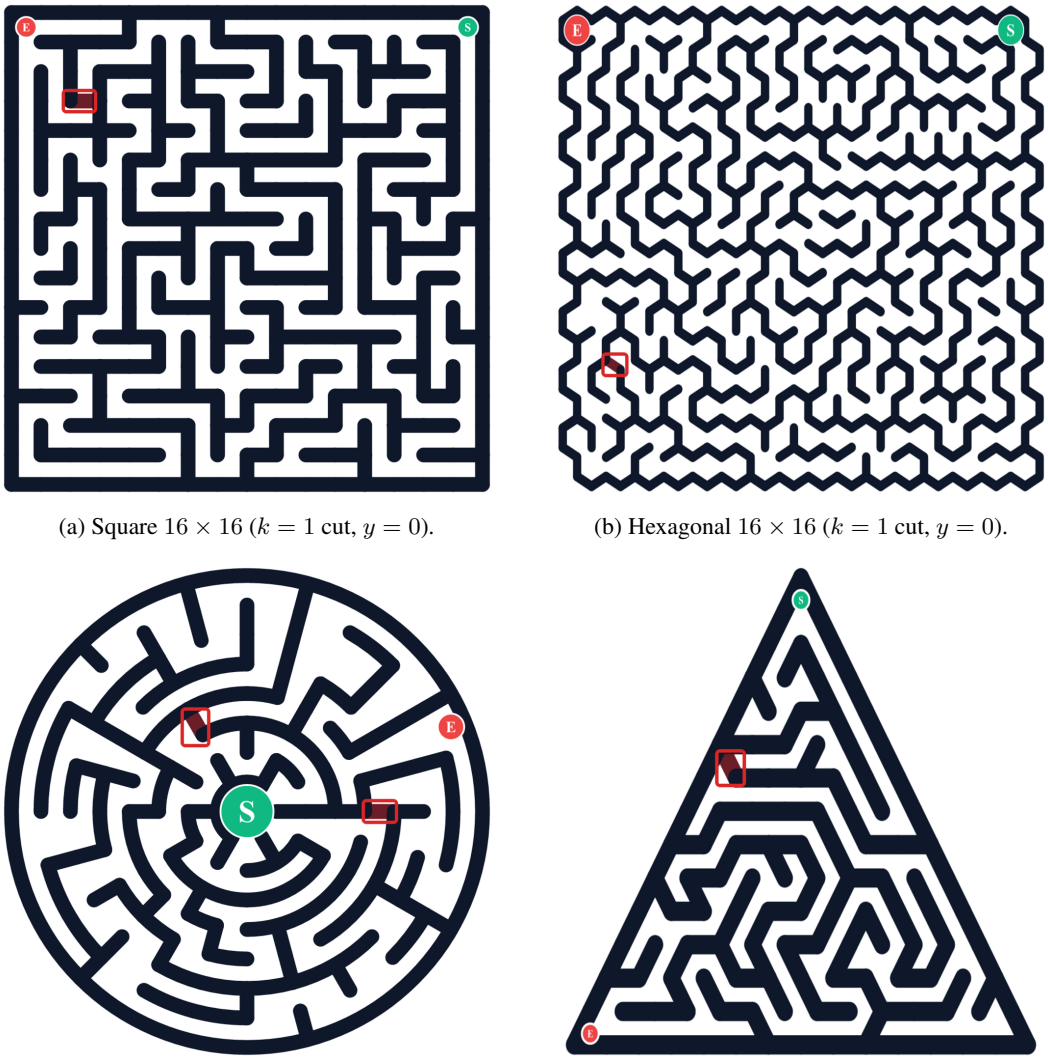
- **Benchmark V1 (3,000 pairs / 6,000 images)**: Evaluates scale progression on square mazes across six grid sizes (3×3 to 16×16), with 500 pairs per size.
- **Benchmark V2 (1,600 pairs / 3,200 images)**: Evaluates generalization across four geometries over 16 distinct layouts, with 100 pairs per layout.

Benchmark	Geometry	Scales	Pairs
V1	Square	$N \in \{3, 5, 7, 9, 12, 16\}$	3,000
V2	Square	$N \in \{5, 7, 9, 12, 16\}$	500
	Hexagonal	$N \in \{5, 7, 9, 12, 16\}$	500
	Polar	Rings $r \in \{4, 6, 8\}$	300
	Triangular	Heights $h \in \{6, 10, 14\}$	300
Total	4 Geometries	22 Unique Configurations	4,600

Table A.1: **Benchmark dataset composition across V1 and V2.** Each pair contains one solvable maze and one minimally modified unsolvable counterpart.

A.3 COMPARISON WITH EXISTING MAZE BENCHMARKS

We compare AMAZE with existing maze generators and evaluation suites. Schwarzschild et al. (2021), Bansal et al. (2022), and Ivanitskiy et al. (2023) create datasets for training recurrent networks and sequence models. They represent 4-connected square grids as low-resolution binary rasters (1–2 pixels per cell) or token sequences and generate only solvable mazes ($y = 1$). In the language-model setting, MazeEval (Einarsson, 2026) evaluates multi-turn navigation across 110 solvable square tree mazes using text-only coordinate and wall-distance tool calls without visual input, finding that most models fail beyond 9×9–12×12 grids due to repetitive looping.



(a) Square 16×16 ($k = 1$ cut, $y = 0$).

(b) Hexagonal 16×16 ($k = 1$ cut, $y = 0$).

(c) Polar 8-ring ($k = 2$ braided cuts, $y = 0$).

(d) Triangular $h = 14$ ($k = 1$ cut, $y = 0$).

Figure A.1: **Example large-scale mazes across all four geometry families.** These examples are unsolvable mazes ($y = 0$) with solid walls; the red boxes mark the locations of the path cut-offs with respect to their solvable counterpart ($k = 1$ for unique solution path, $k = 2$ when loop braiding creates alternative detours; these red box annotations are not shown to evaluated models). At these scales, confirming unsolvability requires verifying that every corridor across hundreds of maze cells is blocked, and overlooking a single wall segment may produce a false positive.

Most related to our zero-shot baseline, concurrent work on MazeBench (Rodriguez Salgado, 2026) evaluates VLMs tool-free on 110 square tile mazes (79 solvable, 31 unsolvable). Consistent with our transcription findings (Section 3, Appendix B.3), they observe that models attempt to transcribe pixel rows into text matrices to run search in text, falling back on strong solvability biases when an instance is unsolvable. AMAZE extends beyond this setting in four key ways:

- **Matched-twin protocol and scale:** Every solvable maze is paired with a minimally modified unsolvable counterpart across 4,600 pairs (9,200 images), awarding credit only when a model correctly classifies both instances rather than rewarding a bias toward predicting paths.
- **Varied geometries and textures:** Beyond orthogonal square grids (which also support oracle ASCII evaluation in V1), Benchmark V2 introduces hexagonal, polar, and triangular layouts rendered with solid, hatched, and gradient walls that cannot be mapped onto simple row–column matrices.
- **Loop braiding against simple heuristics:** In pure tree mazes ($r_{\text{braid}} = 0$), inserting a blocking wall splits all corridors into exactly two components, allowing agents to classify solvability by simply counting connected regions without locating the start or exit. Reconnecting dead ends ($r_{\text{braid}} \in \{0.3, 0.6\}$) introduces corridor cycles and detached wall islands, breaking this component-counting shortcut and preventing dead-end pruning.
- **Multi-step visual tools with verification:** Rather than testing single-turn visual inspection alone, we evaluate agents that compose low-level vision tools, comparing monolithic agents with context-isolated specialist architectures equipped with programmatic verification.

B Zero-Shot Diagnosis Details

B.1 Evaluation Setup

We evaluate zero-shot models on Benchmark V1 across raw images and text-only oracle ASCII.

The square mazes of V1 can be converted directly into ASCII text representations. For an $N \times N$ square maze, we implement the oracle ASCII representation as a $(2N + 3) \times (2N + 3)$ character grid using `#` for walls, `.` for open passages, `S` for start, and `E` for exit. This encompasses the $(2N - 1) \times (2N - 1)$ cell-and-wall core, a one-character perimeter wall on which the start and exit are located, and a one-character outer perimeter wall (yielding 9×9 characters for a 3×3 maze).

Zero-shot raw-image evaluation uses the following prompt:

> *You are looking at a maze. The paths (corridors) are white and the walls and background are black. There is a green circle representing the start and a red circle representing the end. Do not change the maze or its structure in your reasoning. Is there a valid path from the green circle to the red circle without crossing any black walls? Answer Yes or No.*

For text-only Oracle ASCII evaluation, no image is provided and the model receives the following prompt:

> *You are given the ASCII representation of a maze. The paths (corridors) are represented by* `.` *and the walls and borders are represented by* `#`*. There is an* `S` *representing the start and an* `E` *representing the end. Do not change the maze or its structure in your reasoning. Is there a valid path from* `S` *to* `E` *without crossing any* `#` *walls?*
>
> *Here is the ASCII representation of the maze:*
> `{{ascii_grid}}`
>
> *Answer Yes or No.*

We extract the *Yes / No* answers via regular expressions.

We evaluate five models:

- Gemini 3.5 Flash (Gemini Team, Google DeepMind, 2026a),
- Gemini 3.5 Flash-Lite (Gemini Team, Google DeepMind, 2026b),
- GPT-5.6 Sol (OpenAI, 2026),
- Gemma 4 31B (Gemma Team, Google DeepMind, 2026), and

Maze Scale	Gemini 3.5 Flash	Gemini 3.5 Flash-Lite	Gemma 4 31B	Qwen 3.8 27B	GPT-5.6 Sol*
Raw Image					
3×3	99.3 ± 0.2 (2.8K)	70.5 ± 1.5 (2.0K)	79.3 ± 0.8 (3.9K)	61.9 ± 2.4 (1.0K)	**100.0** (1.4K)
5×5	52.5 ± 2.8 (8.7K)	8.5 ± 0.8 (2.5K)	11.0 ± 3.3 (8.0K)	6.0 ± 1.4 (1.9K)	**85.0** (1.7K)
7×7	7.9 ± 1.0 (10.8K)	0.3 ± 0.3 (2.9K)	2.1 ± 0.6 (8.8K)	0.6 ± 0.5 (1.8K)	**81.0** (2.3K)
9×9	1.6 ± 0.2 (10.6K)	0.1 ± 0.1 (3.5K)	1.3 ± 0.9 (10.2K)	0.1 ± 0.1 (1.6K)	**39.8** (4.3K)
12×12	2.3 ± 0.3 (13.6K)	0.0 ± 0.0 (2.0K)	2.5 ± 0.7 (4.9K)	0.4 ± 0.2 (1.6K)	**29.6** (5.0K)
16×16	2.4 ± 0.6 (12.8K)	0.4 ± 0.3 (2.0K)	1.1 ± 0.4 (4.4K)	0.2 ± 0.4 (1.4K)	**20.8** (5.0K)
Oracle ASCII					
3×3	**99.9 ± 0.1** (2.5K)	94.9 ± 0.8 (2.9K)	90.7 ± 3.2 (1.7K)	81.9 ± 1.7 (891)	**100.0** (283)
5×5	**98.5 ± 0.5** (6.2K)	36.9 ± 1.8 (3.0K)	76.0 ± 11.2 (2.6K)	62.7 ± 1.7 (2.1K)	94.0 (597)
7×7	**91.1 ± 1.0** (12.5K)	8.1 ± 0.6 (3.0K)	46.5 ± 8.1 (2.8K)	31.9 ± 2.5 (1.3K)	84.2 (1.1K)
9×9	**73.8 ± 1.3** (21.9K)	3.9 ± 1.1 (2.9K)	21.6 ± 5.6 (2.8K)	19.7 ± 2.2 (730)	71.2 (1.9K)
12×12	53.7 ± 1.7 (30.0K)	2.4 ± 0.9 (1.9K)	23.7 ± 2.6 (14.6K)	22.9 ± 2.2 (690)	**64.2** (3.3K)
16×16	**38.1 ± 2.4** (36.2K)	0.9 ± 0.2 (1.9K)	17.5 ± 2.2 (14.2K)	20.5 ± 1.3 (4.2K)	37.4 (3.8K)

Table B.1: **Zero-shot baseline performance across maze scales on raw images and oracle ASCII on Benchmark V1.** Values report paired accuracy (%), with mean generated reasoning tokens shown in parentheses. Gemini, Gemma, and Qwen models report sample mean ± standard deviation across 3 seeds, *GPT-5.6 Sol uses 1 seed. Bold indicates top paired accuracy per scale within each modality.

- Qwen 3.8 27B (Qwen Team, 2026).

Generation was performed at default temperature $T = 1.0$ with reasoning enabled (e.g., `reasoning_effort=medium` or `include_thoughts=True`). Gemini, Gemma, and Qwen are run with 3 seeds (Seed 42, 123, 456) over $M = 500$ pairs / 1,000 episodes per maze scale and per seed, totaling 144,000 multi-seed episodes. GPT-5.6 Sol uses 1 seed for budget constraints.

B.2 Scaling Behavior and Solvability Bias

Detailed results and token counts per maze scale over raw images and oracle ASCII for Benchmark V1 are reported in Table B.1. Table B.2 reports false-positive and false-negative rates on the images of Benchmark V1, revealing strong biases in model behavior. We see that false-negative rates (FNR) stay at or below 4.7% across all scales for Gemini, Gemma, and Qwen, whereas false-positive rates approach 100% on medium grids: Flash-Lite, Gemma, and Qwen reach 87.9%–93.8% FPR by 5×5, and Gemini 3.5 Flash rises from 45.3% at 5×5 to 91.9% at 7×7. By contrast, GPT-5.6 Sol maintains an FPR $\leq 5.2\%$ through 7×7; on 9×9 grids, its errors are mostly false negatives (51.6% FNR), before shifting toward false positives on 16×16 (68.0% FPR). This asymmetry drives the low paired accuracy in Table B.1.

Longer reasoning traces do not fix early errors. All models generate longer reasoning traces as maze scale increases (Table B.1). Gemini 3.5 Flash expands token usage from 2.8K on 3×3 to 13.6K on 12×12, while GPT-5.6 Sol scales from 1.4K to 5.0K. Yet despite generating 4–5× more reasoning tokens, accuracy drops on both images and ASCII grids. Inspecting the traces shows why: once a model records an invalid step across a wall, subsequent tokens build on that faulty step instead of backtracking, extending the invalid path to the exit of the maze and driving false-positive rates.

B.3 Image-to-ASCII Transcription

Oracle ASCII tests pathfinding without visual parsing, but ground-truth text grids are not available at test time. A natural alternative pipeline is two-stage: first transcribing the maze image into an ASCII grid, then solving the text representation symbolically. To test the feasibility of this approach, we evaluate three of our benchmarked models (Gemini 3.5 Flash, Gemini 3.5 Flash-Lite, and Gemma 4 31B) on image-to-ASCII transcription by prompting them to convert the input images from Benchmark V1 into an ASCII representation following the conventions from Section B.1.

Maze Scale	Gemini 3.5 Flash	Gemini 3.5 Flash-Lite	Gemma 4 31B	Qwen 3.8 27B	GPT-5.6 Sol*
False Positive Rate (FPR, %)					
3×3	0.2 ± 0.2	29.3 ± 1.7	19.5 ± 0.3	36.1 ± 2.9	0.0
5×5	45.3 ± 3.3	91.5 ± 0.9	87.9 ± 4.6	93.8 ± 1.4	5.2
7×7	91.9 ± 0.8	99.7 ± 0.3	97.8 ± 0.9	99.4 ± 0.5	4.2
9×9	98.3 ± 0.2	99.9 ± 0.1	98.7 ± 0.9	99.9 ± 0.1	16.6
12×12	97.7 ± 0.3	100.0 ± 0.0	97.5 ± 0.7	99.6 ± 0.2	41.2
16×16	97.6 ± 0.6	99.6 ± 0.3	98.9 ± 0.4	99.8 ± 0.3	68.0
False Negative Rate (FNR, %)					
3×3	0.5 ± 0.4	1.1 ± 0.3	1.6 ± 0.8	3.4 ± 0.5	0.0
5×5	3.1 ± 0.8	0.7 ± 0.1	4.7 ± 2.9	2.0 ± 0.3	10.0
7×7	2.0 ± 0.9	0.3 ± 0.3	1.6 ± 1.4	0.2 ± 0.0	16.2
9×9	1.0 ± 0.3	0.1 ± 0.1	1.8 ± 1.1	0.2 ± 0.2	51.6
12×12	1.9 ± 0.6	0.1 ± 0.2	3.1 ± 0.8	0.3 ± 0.1	43.0
16×16	1.4 ± 0.4	0.3 ± 0.5	1.3 ± 0.4	0.2 ± 0.2	26.0

Table B.2: **Zero-shot solvability bias across maze scales on Benchmark V1.** False positive rate (FPR, unsolvable mazes predicted solvable; *upper*) and false negative rate (FNR, solvable mazes predicted unsolvable; *lower*). Gemini, Gemma, and Qwen models report sample mean $\pm$ standard deviation across 3 random seeds, *GPT-5.6 Sol uses 1 seed.

We evaluate each parsed ASCII grid $\hat{G}$ against the ground-truth grid G across three metrics (Table B.3): *Dimension Match*, which requires $\hat{G}$ and G to have identical dimensions; *Exact Grid Match*, which requires element-wise match across the full grid; and *Mean Cell Accuracy*, the fraction of matching characters between $\hat{G}$ and G. So that missing outer # borders are not penalized as complete failures (for instance, when a model omits the 1-character outer # perimeter), when $\hat{G}$ and G differ in outer dimensions, we extract the inner cell-and-walls in both grids and compute element-wise cell agreement if they match in shape (scoring 0% otherwise).

As shown in Table B.3, current models cannot reliably transcribe maze images into text grids. Even on small 3×3 instances, the strongest model (Gemini 3.5 Flash) achieves only an 18.0% exact grid match rate. From 5×5 onward, exact grid recovery drops to 0.0% for all three models. Cell-level accuracy falls from 42.8% on 3×3 to 0.4% on 16×16 for Gemini 3.5 Flash (0.0% dimension match at 12×12 and 16×16), while neither Flash-Lite nor Gemma 4 31B exceeds 9.2% cell accuracy at any scale.

Prompting a VLM to transcribe the image into ASCII fails just as badly as solving the image directly. Oracle ASCII is therefore useful only as an upper-bound baseline, not as a practical preprocessing step in place of image tools.

Model	Metric	3×3	5×5	7×7	9×9	12×12	16×16
Gemini 3.5 Flash	Exact Grid Match (%)	18.0	0.0	0.0	0.0	0.0	0.0
	Mean Cell Accuracy (%)	42.8	36.3	8.2	7.9	2.1	0.4
	Dimension Match (%)	31.0	36.0	6.0	10.0	0.0	0.0
Gemini 3.5 Flash-Lite	Exact Grid Match (%)	0.0	0.0	0.0	0.0	0.0	0.0
	Mean Cell Accuracy (%)	1.3	7.0	6.9	9.2	0.3	0.0
	Dimension Match (%)	2.0	11.2	12.0	14.0	1.0	0.0
Gemma 4 31B	Exact Grid Match (%)	0.0	0.0	0.0	0.0	0.0	0.0
	Mean Cell Accuracy (%)	0.8	1.6	0.5	0.0	0.0	0.0
	Dimension Match (%)	0.0	0.0	1.0	0.0	0.0	0.0

Table B.3: Direct image-to-ASCII transcription performance on Benchmark V1

Evaluation Condition	Paired Acc. (%)	FPR (%)	FNR (%)
Visual Style (Square 5×5, identical graphs)			
Solid	27.0	72.0	8.0
Gradient	22.7	77.3	2.1
Diagonal hatch	7.1	92.9	0.0
Cross hatch	5.2	94.8	0.0
Geometry (Smallest scale, solid walls)			
Square (5×5)	27.0	72.0	8.0
Triangular	23.0	77.0	0.0
Polar	21.9	76.0	9.4
Hexagonal	14.0	82.0	13.0

Table B.4: **Ablation of visual style and geometry on Benchmark V2 for Gemini 3.5 Flash** ($M = 100$ pairs per row). Each block varies one factor while holding the others fixed. In the visual style block, all four styles are rendered on the same 100 underlying 5×5 mazes; in the geometry block, all mazes use solid walls.

B.4 Zero-Shot Evaluation on V2

Benchmark V1 and V2 target different aspects of spatial reasoning. V1 isolates the effect of scale, keeping the mazes to a single geometry and visual style. V2 introduces visual textures and non-square geometries to evaluate models under varied perceptual conditions.

When agents are equipped with vision tools (Section 4.1), uniform square mazes can often be solved by a fixed strategy, such as thresholding the image and running flood-fill, without inspecting intermediate states or adapting parameters. Textured walls break these fixed scripts: hatched textures can cause flood fills to leak across walls, and gradients can cause flood fills with small tolerances to fail. Similarly, triangular, polar, and hexagonal layouts cannot be mapped onto simple row–column matrices, requiring models to adapt their strategy.

Table B.4 shows how each of these factors affects no-tools zero-shot evaluation on Gemini 3.5 Flash on Benchmark V2. Note that on 5×5 solid square mazes, paired accuracy is lower on V2 (27.0%) than on V1 (52.5% in Table B.1): V1 renders walls in a thick blocky style, whereas V2 uses thin vector walls (16px walls with wider 32px corridors), making thin walls easier to overlook. On 5×5 square mazes sharing the same underlying graphs, replacing solid walls with smooth gradients causes only a slight drop in paired accuracy (27.0% vs. 22.7%). In contrast, diagonal and cross hatching degrade paired accuracy to 7.1% and 5.2%.

When evaluating solid-wall layouts at their smallest scales across geometry families, triangular (23.0%) and polar (21.9%) configurations exhibit similar difficulty to square mazes (27.0%), whereas hexagonal grids fall to 14.0%.

C Tool Interfaces and Runtime Environment

This section describes the Python image-processing tools, session storage, and decoding parameters used in our evaluations. Instead of answering directly from the raw image in one pass, agents call these tools and store intermediate images in an in-memory dictionary.

C.1 Tool Suites and Exposed Signatures

Table C.1 lists the Python function signatures, default arguments, specialist role assignments (`{role}` and `{tool_names}` in Listing G.2), and underlying library calls for the **atomic** and **flood-fill** tool suites (Section 4.1).

Each tool is a short Python wrapper around `OpenCV`, `Pillow`, or `scikit-image`. Instead of passing raw pixel arrays, agents pass an `image_key`; the wrapper loads the image from the session inventory (Section C.2), runs the operation, saves the output PNG under a new `image_id`, and returns that ID. These wrappers also standardize argument names and coordinate order (such as `ymin, xmin,`

Exposed Tool Signature	Underlying Library API
Image Transformation (image_transformation_specialist)	
crop_image(ymin, xmin, ymax, xmax)	PIL.Image.crop
rotate_image(degrees=90)	PIL.Image.rotate
resize_image(scale)	cv2.resize
flip_image(direction)	cv2.flip
pad_image(pad)	cv2.copyMakeBorder
Filter and Binarization (filter_binarize_specialist)	
threshold_image(thresh=127, maxval=255, method="otsu")	cv2.threshold
adaptive_threshold(block_size=11, c=2.0)	cv2.adaptiveThreshold
gaussian_blur(ksize=5, sigma=1.0)	cv2.GaussianBlur
median_blur(ksize=5)	cv2.medianBlur
Structural Topology (structural_topology_specialist)	
morphological_operation(operation="dilate", ksize=3)	cv2.morphologyEx
skeletonize_image()	skimage.skeletonize
find_and_draw_contours()	cv2.findContours
canny_edge_detection(low, high)	cv2.Canny
Canvas Annotation (canvas_annotate_specialist)	
draw_line(x1, y1, x2, y2)	cv2.line
draw_rectangle(xmin, ymin, xmax, ymax)	cv2.rectangle
draw_circle(x, y, radius)	cv2.circle
draw_point(x, y, color)	cv2.circle
draw_polyline(points_json, color, thickness)	cv2.polylines
create_canvas(w, h)	PIL.Image.new
paste_region(src, dst)	PIL.Image.paste
fill_region(box, color)	cv2.rectangle
swap_regions(box1, box2)	np.ndarray
Measurement and Reachability (measurement_analysis_specialist / flood_fill_specialist†)	
distance_transform()	cv2.distanceTransform
measure_regions()	skimage.regionprops
flood_fill(start_y, start_x, fill_radius, threshold)†	cv2.floodFill
connected_components()†	cv2.connectedComponents

All tools accept image_key="maze_image" and return a registered image_id. †Available only in the flood-fill suite; withheld in the atomic suite. morphological_operation supports erosion and dilation via cv2.morphologyEx.

Table C.1: **Exposed tool signatures, specialist role assignments, and underlying library APIs.** Clicking any library API name opens its official documentation (Pillow, OpenCV, or scikit-image). Category subheaders indicate the specialist prompt role ({role} in Listing G.2) assigned to each set of tools. Tools marked with † are available only in the flood-fill suite.

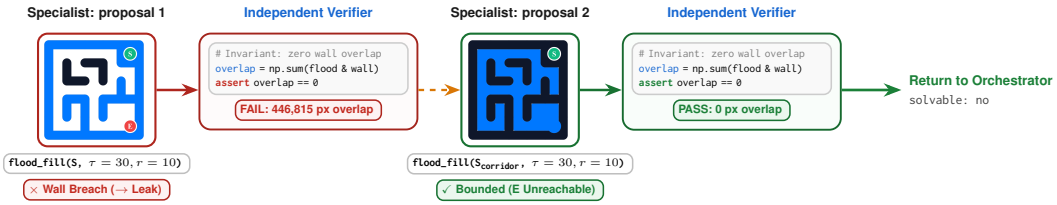


Figure C.1: **Specialist-verifier retry loop on an unsolvable 5×5 maze. (Proposal 1)** The reachability specialist starts a flood fill on the outer border (τ: color `threshold`, r: `fill_radius`), causing the mask to spill across wall boundaries (446,815 overlapping wall pixels). **(Verification)** The independent verifier runs a Python assertion checking whether the fill overlaps dark wall pixels, catches the leak (`FAIL`), and reports the overlap count. **(Proposal 2)** The specialist re-runs `flood_fill` from inside the open corridor with zero wall overlap (`PASS`), showing that exit E remains unreachable. Only the verified image is returned to the orchestrator, yielding the correct prediction (`solvable: no`).

`ymax`, `xmax` for bounding boxes). In addition, our `flood_fill` tool adds a minimum clearance radius (`fill_radius`, Section 5) to `cv2.floodFill`. Any gap narrower than `fill_radius` pixels blocks the fill.

C.2 Asset Inventory and Visual Feedback

After every tool call, the output image is appended to the agent's prompt so that it can see the result on the next turn. The single-agent (`Single`) keeps every intermediate image and failed try in one growing prompt. In contrast, the multi-agent setups (`Multi-noV` and `Multi+Ver`) give each delegated specialist a temporary workspace and store only its final output image ($\mathcal{A}_{\text{out},k}$) in the orchestrator's main inventory, which starts with the original maze stored as a read-only entry. Because only $\mathcal{A}_{\text{out},k}$ is returned and appended to the orchestrator's prompt, the orchestrator's visual context grows by at most one image per delegation.

Saving an image in the main inventory does not automatically expose it to later sub-agents. Instead, the orchestrator chooses which stored images $\mathcal{A}_{\text{in}}$ to pass when calling $\text{Delegate}(k, \text{instruction}_t, \mathcal{A}_{\text{in}})$, omitting any unhelpful or outdated images from earlier steps. The same rule applies at the final submission step: earlier sub-agent chat histories are withheld, and the submitter agent sees only the orchestrator's summary instruction and selected images, so failed intermediate tries cannot bias the final answer.

C.3 Programmatic Verification Sandbox

When a specialist calls `request_verification`, the harness runs an isolated verifier turn equipped only with a restricted Python tool (`execute_python_code`). To prevent earlier failed tries from interfering with the check, the verifier receives only `"maze_image"`, the specialist's initial inputs $\mathcal{A}_{\text{in}}$, and the single candidate image being tested. The verifier has no maze-solving or image-editing tools. Instead, it runs short Python checks using `numpy`, `cv2`, `PIL`, and `skimage` (10-second timeout, 512 MB memory limit, no file or network access) to test pixel-level constraints, such as verifying that a binary or flood-fill mask does not cross wall pixels (Figure C.1). If an assertion fails, the verifier returns an error summary explaining the failure. The specialist then adjusts its tool arguments (such as lowering a threshold or moving a seed point) and submits a new candidate image until the check passes or the retry limit is reached (Section C.4).

C.4 Execution Horizons, Iteration Budgets, and Decoding Parameters

We evaluate all agent setups (Single, Multi-noV, and Multi+Ver) with Gemini 3.5 Flash and Gemini 3.5 Flash-Lite at temperature $T = 0.2$ using strict JSON tool-call schemas. Each turn allows up to 2,048 thinking tokens before the model makes a tool or delegation call. Full system prompts for every role appear in Appendix G.

On AMAZE, both architectures operate under a matched 200-step default ceiling: Single has a limit of 200 steps (with a wrap-up reminder at step 100), while the multi-agent orchestrator can make up to 20 specialist delegations, each allowing up to 10 specialist–verifier interactions ($20 \times 10 = 200$). Within each interaction, both the specialist and the verifier have up to 3 tries to execute valid tool calls or Python verification code. To test whether restricting the single-agent context length reduces context bloat, we also run a controlled $c = 50$ single-agent ablation (Single ($c = 50$), 50-step limit with a wrap-up reminder at step 25; Appendix E). On MENTISOCULI, the orchestrator has a 20-step outer limit, and Single receives a budget of 150 steps.

As shown in Table C.2, Single never hits its step limit (0.0% across all runs): it stops calling tools and submits an answer after 8.9–21.4 turns under the default 200-step cap (8.8–11.8 turns under $c = 50$), and 17.8 turns on MENTISOCULI. Consequently, the accuracy gap between single- and multi-agent setups is not caused by running out of steps. Instead, as failed intermediate images pile up in one growing prompt, Single submits an answer early. In contrast, Multi+Ver retries failed checks inside sub-agent workspaces, keeping the orchestrator's trajectory short (3.8–8.3 orchestrator turns) and reducing specialist delegations relative to Multi-noV (3.5 vs. 6.1 on AMAZE *atomic*; 1.7 vs. 2.8 on MENTISOCULI; see Section D.3 for the full function-call breakdown). Across our benchmark traces in Multi+Ver, basic image tools (thresholding, cropping, and flood-filling) pass verification in 1.3–1.8 specialist–verifier interactions on average ($> 75\%$ passing on the first or second check). Only tasks involving exact pixel coordinates, such as drawing multi-segment polylines through narrow curved corridors or placing rotated polygon pieces, commonly require additional recalibration interactions.

Benchmark & Suite	Architecture	Orchestrator Turns (Mean ± SD)	Specialist Delegations (Mean ± SD)
AMAZE V2 (Atomic Suite, $N = 3{,}200$/arm)	Single ($c = 200$)	21.4 ± 15.5	—
	Single ($c = 50$)	11.8 ± 4.9	—
	Multi-noV	11.5 ± 4.1	6.1 ± 2.9
	Multi+Ver	8.3 ± 3.6	3.5 ± 1.7
AMAZE V2 (Flood-Fill Suite, $N = 3{,}200$/arm)	Single ($c = 200$)	8.9 ± 6.5	—
	Single ($c = 50$)	8.8 ± 4.4	—
	Multi-noV	10.1 ± 3.6	4.9 ± 2.5
	Multi+Ver	8.0 ± 2.6	3.4 ± 1.6
MENTISOCULI	Single	17.8 ± 14.6	—
	Multi-noV	5.1 ± 2.0	2.8 ± 1.7
	Multi+Ver	3.8 ± 1.1	1.7 ± 0.9

Table C.2: **Outer turn utilization and specialist delegations for Gemini 3.5 Flash across benchmarks. Orchestrator Turns** count direct turns for Single (200-step default and $c = 50$ shorter-context ablation on AMAZE; 150-step cap on MENTISOCULI) and orchestrator turns for multi-agent setups (up to 20 delegations of 10 interactions on AMAZE; 20-turn cap on MENTISOCULI). **Specialist Delegations** count orchestrator dispatches to visual specialists. Across all runs, Single stops well below its step limit (0.0% cap rate), and local verification (Multi+Ver) reduces both orchestrator turns and specialist delegations relative to Multi-noV.

D ADDITIONAL BENCHMARK V2 RESULTS

Section 5 reports aggregate Benchmark V2 results for Gemini 3.5 Flash across the four geometries. This appendix provides layout-level results across all 16 layouts for Gemini 3.5 Flash (Section D.1) and Gemini 3.5 Flash-Lite (Section D.2), followed by a fine-grained analysis of tool utility and confirmation bias (Section D.3).

We omit a single-agent baseline with Python verification (**Single+Ver**) because giving one agent access to both `execute_python_code` and `submit_solution` allows it to bypass the visual toolkit entirely: the agent simply writes a Python breadth-first search (BFS) script over the raw image pixels and submits the answer directly (which would not generalize to other visual reasoning puzzles). In **Multi+Ver**, role separation prevents this shortcut: the orchestrator (which calls `submit_solution`)

has no Python execution tool, while verifier sub-agents (which run Python checks in isolated contexts) cannot submit solutions and only return pass/fail verdicts and feedback on a specialist's visual tool output.

All configurations are evaluated under the two tool suites described in Section 5: the **atomic suite** (which excludes `flood_fill` and `connected_components`, requiring the agent to chain multiple visual operations) and the **flood-fill suite** (which includes `flood_fill` to test reachability directly in a few steps). Each layout cell reports paired accuracy across $M = 100$ matched counterfactual pairs (200 episodes per layout; $M = 1{,}600$ pairs / $N = 3{,}200$ episodes across all 16 layouts). All p-values reported in the text correspond to two-sided exact McNemar tests over the $M = 1{,}600$ pairs.

D.1 Gemini 3.5 Flash Evaluation

Tables D.1 and D.2 report layout-level paired accuracy for Gemini 3.5 Flash on the atomic and flood-fill suites, and Table D.3 breaks down accuracy by ground-truth solvability label. On the atomic suite (Table D.1), moving from **Single** to **Multi-noV** raises overall paired accuracy from 31.6% to 47.0% ($+15.4$ pp), with the largest gains on large grids that require long sequences of tool calls (e.g., $27.0\% \rightarrow 48.0\%$ on `square_16x16` and $10.0\% \rightarrow 51.0\%$ on `hex_16x16`). Adding programmatic verification (**Multi+Ver**) raises paired accuracy by another $+20.6$ pp to 67.6%. In the flood-fill suite (Table D.2), where a single `flood_fill` call usually resolves reachability in a few steps, unverified delegation yields a smaller $+5.2$ pp gain over a single agent ($75.8\% \rightarrow 81.0\%$), concentrated on hexagonal and polar grids. This confirms that isolating sub-agent contexts mainly helps when solving a maze requires chaining many tool calls. Even in this short-horizon setting, programmatic verification (**Multi+Ver**) raises paired accuracy by another $+6.3$ pp to 87.3%. As Table D.3 shows, verification catches cases where `flood_fill` bleeds across thin walls on hexagonal and polar grids, cutting the false-positive rate on unsolvable mazes from 16.8% to 3.1%.

Geometry	Scale	Paired Accuracy (%)			
		No tools	Single	Multi-noV	Multi+Ver
Square	5×5	24.0	72.0	72.0	**83.0**
	7×7	16.0	48.0	55.0	**75.0**
	9×9	11.0	40.0	47.0	**72.0**
	12×12	12.0	21.0	36.0	**77.0**
	16×16	5.0	27.0	48.0	**67.0**
	Subtotal	*13.6*	*41.6*	*51.6*	***74.8***
Hexagonal	5×5	8.0	31.0	43.0	**62.0**
	7×7	5.0	24.0	45.0	**57.0**
	9×9	8.0	22.0	49.0	**68.0**
	12×12	7.0	17.0	45.0	**49.0**
	16×16	14.0	10.0	51.0	**55.0**
	Subtotal	*8.4*	*20.8*	*46.6*	***58.2***
Polar	4 rings	23.0	34.0	38.0	**68.0**
	6 rings	10.0	24.0	39.0	**71.0**
	8 rings	12.0	19.0	39.0	**67.0**
	Subtotal	*15.0*	*25.7*	*38.7*	***68.7***
Triangular	$h = 6$	28.0	61.0	60.0	**80.0**
	$h = 10$	12.0	36.0	44.0	**73.0**
	$h = 14$	11.0	19.0	41.0	**58.0**
	Subtotal	*17.0*	*38.7*	*48.3*	***70.3***
OVERALL	**All 16 Layouts**	**12.9**	**31.6**	**47.0**	**67.6**

Table D.1: **Gemini 3.5 Flash on the atomic suite.** Paired accuracy (%) across all 16 Benchmark V2 layouts ($M = 100$ matched pairs per layout) and architectures.

Geometry	Scale	Paired Accuracy (%)			
		No tools	Single	Multi-noV	Multi+Ver
Square	5×5	24.0	94.0	96.0	**97.0**
	7×7	16.0	89.0	**92.0**	91.0
	9×9	11.0	91.0	88.0	**94.0**
	12×12	12.0	88.0	**92.0**	86.0
	16×16	5.0	86.0	80.0	**87.0**
	Subtotal	*13.6*	*89.6*	*89.6*	***91.0***
Hexagonal	5×5	8.0	69.0	74.0	**90.0**
	7×7	5.0	71.0	74.0	**81.0**
	9×9	8.0	72.0	81.0	**84.0**
	12×12	7.0	65.0	60.0	**84.0**
	16×16	14.0	55.0	75.0	**77.0**
	Subtotal	*8.4*	*66.4*	*72.8*	***83.2***
Polar	4 rings	23.0	67.0	78.0	**86.0**
	6 rings	10.0	53.0	74.0	**85.0**
	8 rings	12.0	57.0	75.0	**84.0**
	Subtotal	*15.0*	*59.0*	*75.7*	***85.0***
Triangular	$h = 6$	28.0	90.0	**92.0**	90.0
	$h = 10$	12.0	83.0	80.0	**93.0**
	$h = 14$	11.0	83.0	85.0	**88.0**
	Subtotal	*17.0*	*85.3*	*85.7*	***90.3***
OVERALL	**All 16 Layouts**	**12.9**	**75.8**	**81.0**	**87.3**

Table D.2: **Gemini 3.5 Flash on the flood-fill suite.** Paired accuracy (%) across all 16 Benchmark V2 layouts ($M = 100$ matched pairs per layout) and architectures.

Architecture	Paired Acc. (%)	Solvable Acc. (%)	Unsolvable Acc. (%)	FPR (%)
Atomic suite				
No tools	12.9	92.2	15.2	84.8
Single	31.6	73.7	45.0	55.0
Multi-noV	47.0	66.9	73.3	26.7
Multi+Ver	**67.6**	77.7	**87.8**	**12.2**
Flood-fill suite				
No tools	12.9	92.2	15.2	84.8
Single	75.8	90.2	83.2	16.8
Multi-noV	81.0	91.9	87.9	12.1
Multi+Ver	**87.3**	90.3	**96.9**	**3.1**

Table D.3: **Accuracy by ground-truth label for Gemini 3.5 Flash on Benchmark V2** across architectures and both tool suites.

D.2 Gemini 3.5 Flash-Lite Evaluation

Tables D.4 and D.5 report layout-level paired accuracy for Gemini 3.5 Flash-Lite on the atomic and flood-fill suites, and Table D.6 breaks down accuracy by ground-truth solvability label.

On the atomic suite (Table D.4), **Single** Flash-Lite reaches only 5.7% overall paired accuracy (93.6% false-positive rate), scoring 1.0%–4.0% on square and hexagonal grids of size 7×7 or larger. Moving to **Multi-noV** raises paired accuracy to 9.6% ($+3.9$ pp), and adding programmatic verification (**Multi+Ver**) raises it to 29.4% ($+19.8$ pp over **Multi-noV** and $+23.7$ pp over **Single**), with gains across all four geometries.

On the flood-fill suite (Table D.5), unverified delegation performs on par with the single-agent baseline ($59.8\% \rightarrow 59.1\%$, -0.7 pp, $p = 0.63$). Programmatic verification (**Multi+Ver**) raises paired accuracy to 71.5% ($+12.4$ pp over **Multi-noV** and $+11.7$ pp over **Single**), cutting the false-positive rate on unsolvable mazes from 35.8% to 19.4% (Table D.6). Comparing across model sizes, the gap between verified Flash and verified Flash-Lite is 38.2 pp on the atomic suite (67.6% vs. 29.4%), but narrows to 15.8 pp when both models have access to `flood_fill` (87.3% vs. 71.5%).

Geometry	Scale	Paired Accuracy (%)			
		No tools	Single	Multi-noV	Multi+Ver
Square	5×5	3.0	5.0	19.0	**46.0**
	7×7	0.0	3.0	9.0	**31.0**
	9×9	1.0	3.0	10.0	**34.0**
	12×12	0.0	2.0	7.0	**23.0**
	16×16	1.0	1.0	0.0	**24.0**
	Subtotal	*1.0*	*2.8*	*9.0*	***31.6***
Hexagonal	5×5	0.0	4.0	9.0	**29.0**
	7×7	0.0	3.0	10.0	**27.0**
	9×9	0.0	1.0	2.0	**29.0**
	12×12	0.0	4.0	5.0	**27.0**
	16×16	0.0	2.0	9.0	**22.0**
	Subtotal	*0.0*	*2.8*	*7.0*	***26.8***
Polar	4 rings	5.0	11.0	13.0	**33.0**
	6 rings	1.0	6.0	13.0	**24.0**
	8 rings	0.0	7.0	7.0	**29.0**
	Subtotal	*2.0*	*8.0*	*11.0*	***28.7***
Triangular	$h = 6$	6.0	21.0	19.0	**33.0**
	$h = 10$	1.0	11.0	11.0	**27.0**
	$h = 14$	1.0	7.0	10.0	**33.0**
	Subtotal	*2.7*	*13.0*	*13.3*	***31.0***
OVERALL	**All 16 Layouts**	**1.2**	**5.7**	**9.6**	**29.4**

Table D.4: **Gemini 3.5 Flash-Lite on the atomic suite.** Paired accuracy (%) across all 16 Benchmark V2 layouts ($M = 100$ matched pairs per layout) and architectures.

Geometry	Scale	Paired Accuracy (%)			
		No tools	Single	Multi-noV	Multi+Ver
Square	5×5	3.0	65.0	66.0	**82.0**
	7×7	0.0	40.0	55.0	**76.0**
	9×9	1.0	66.0	71.0	**79.0**
	12×12	0.0	59.0	63.0	**80.0**
	16×16	1.0	62.0	60.0	**75.0**
	Subtotal	*1.0*	*58.4*	*63.0*	***78.4***
Hexagonal	5×5	0.0	51.0	48.0	**70.0**
	7×7	0.0	58.0	65.0	**77.0**
	9×9	0.0	64.0	71.0	**82.0**
	12×12	0.0	70.0	55.0	**74.0**
	16×16	0.0	62.0	54.0	**69.0**
	Subtotal	*0.0*	*61.0*	*58.6*	***74.4***
Polar	4 rings	5.0	69.0	**71.0**	51.0
	6 rings	1.0	55.0	62.0	**72.0**
	8 rings	0.0	58.0	58.0	**68.0**
	Subtotal	*2.0*	*60.7*	***63.7***	***63.7***
Triangular	$h = 6$	6.0	**61.0**	51.0	**61.0**
	$h = 10$	1.0	56.0	56.0	**63.0**
	$h = 14$	1.0	61.0	39.0	**65.0**
	Subtotal	*2.7*	*59.3*	*48.7*	***63.0***
OVERALL	**All 16 Layouts**	**1.2**	**59.8**	**59.1**	**71.5**

Table D.5: **Gemini 3.5 Flash-Lite on the flood-fill suite.** Paired accuracy (%) across all 16 Benchmark V2 layouts ($M = 100$ matched pairs per layout) and architectures.

Architecture	Paired Acc. (%)	Solvable Acc. (%)	Unsolvable Acc. (%)	FPR (%)
Atomic suite				
No tools	1.2	99.0	1.4	98.6
Single	5.7	93.6	6.4	93.6
Multi-noV	9.6	90.6	11.1	88.9
Multi+Ver	**29.4**	52.2	**60.6**	**39.4**
Flood-fill suite				
No tools	1.2	99.0	1.4	98.6
Single	59.8	90.4	64.2	35.8
Multi-noV	59.1	90.0	63.6	36.4
Multi+Ver	**71.5**	86.6	**80.6**	**19.4**

Table D.6: **Accuracy by ground-truth label for Gemini 3.5 Flash-Lite on Benchmark V2** across architectures and both tool suites.

D.3 Quantifying Tool Utility and Confirmation Bias

To examine how multi-agent delegation and verification change tool usage, we analyze all $9{,}600$ trajectories of Gemini 3.5 Flash on the Benchmark V2 **atomic suite** across **Single**, **Multi-noV**, and **Multi+Ver**. Across all three configurations, 99.4% of visual tool calls use the 11 image-processing operations listed in Table D.7, whereas the 7 orientation and shape-drawing tools (`flip_image`, `rotate_image`, `resize_image`, `pad_image`, `draw_circle`, `draw_rectangle`, `apply_colormap`) are called in fewer than 1% of episodes ($|\mathrm{ENAC}| < 0.1\%$).

Decomposing Tool Utility on Paired Twins. Because an agent chooses which tools to call based on its intermediate beliefs (for example, calling `draw_polyline` only when it already suspects a path exists), simple accuracy on the subset of episodes where a tool is called can be misleading. To evaluate each tool T fairly across Benchmark V2's balanced $50/50$ split of solvable ($Y = +1$) and unsolvable ($Y = -1$) mazes, we report four complementary metrics in Tables D.7 and D.8:

- **Expected Net Accuracy Contribution (ENAC, Table D.7)**: Measures the net percentage-point impact of tool T on overall benchmark accuracy, weighting its effect on solvable and unsolvable mazes by how often the agent calls it on each class:

$$\mathrm{ENAC}(T) \;=\; \underbrace{\frac{1}{2}\,P(T{=}1 \mid Y{=}{+}1)\,\Delta\mathrm{Sens}(T)}_{\text{Solvable Recall Contribution}} \;+\; \underbrace{\frac{1}{2}\,P(T{=}1 \mid Y{=}{-}1)\,\Delta\mathrm{Spec}(T)}_{\text{Unsolvable Specificity Contribution}}, \tag{6}$$

  where $\Delta\mathrm{Sens}(T)$ is the difference in solvable accuracy between episodes that call tool T ($T = 1$) and episodes that do not ($T = 0$), and $\Delta\mathrm{Spec}(T)$ is the corresponding difference in unsolvable accuracy.

- **Selectivity Ratio ($\mathcal{S}$) vs. Hazard Ratio ($\mathcal{H}$, Table D.8)**: For operations used to confirm candidate paths (Groups II and III), a tool helps establish valid connections on solvable mazes ($\Delta\mathrm{Sens} > 0$) but risks inducing false positives when drawn across walls on unsolvable mazes ($\Delta\mathrm{Spec} < 0$). Rearranging Eq. 6 shows that such a tool yields a net accuracy gain ($\mathrm{ENAC} > 0$) if and only if its **Selectivity Ratio** $\mathcal{S}(T)$ (how many times more often the agent calls T on solvable mazes than on unsolvable ones) exceeds its **Hazard Ratio** $\mathcal{H}(T)$ (the false-positive specificity loss per unit of solvable recall gain):

$$\underbrace{\frac{P(T{=}1 \mid Y{=}{+}1)}{P(T{=}1 \mid Y{=}{-}1)}}_{\text{Selectivity Ratio } \mathcal{S}(T)} \;>\; \underbrace{\frac{\max(0, -\Delta\mathrm{Spec}(T))}{\Delta\mathrm{Sens}(T)}}_{\text{Hazard Ratio } \mathcal{H}(T)}. \tag{7}$$

  Table D.8 reports these diagnostics specifically for the candidate path-finding operations (Groups II and III), where route confirmation directly risks false alarms: unsolvability detectors (Group I) detect disconnection rather than asserting paths ($\Delta\mathrm{Spec} \geq 0$, yielding $\mathcal{H} = 0.00^{\dagger}$), while cropping and smoothing filters (Group IV) reduce accuracy across both

classes ($\Delta\text{Sens} < 0$) due to corridor distortion rather than confirmation bias. When calling a tool does not lower unsolvable accuracy ($\Delta\text{Spec} \geq 0$), its false-positive hazard is zero ($\mathcal{H}(T) = 0.00^{\dagger}$); any negative ENAC in that case comes from lower solvable recall ($\Delta\text{Sens} < 0$) rather than false positives.

- **Youden's Index (J_T, Table D.8)**: Defined on the episodes where tool T is called as

$$J_T = \text{Sens}(T{=}1) + \text{Spec}(T{=}1) - 1 = \text{Sens}(T{=}1) - \text{FPR}(T{=}1) \in [-1, +1],$$

i.e., *solvable accuracy minus false-positive rate* when tool T is used. A value near $J_T \approx 0$ indicates strong confirmation bias: for example, if an agent predicts "Solvable" on 96% of episodes where it calls `draw_polyline` regardless of whether a wall blocks the route (Sens = 98%, FPR = 96%), it achieves high solvable recall but $J_T = +0.022$, whereas genuine discrimination between solvable and unsolvable mazes drives $J_T \to +1$.

	Single		Multi-noV		Multi+Ver	
Tool / Operation	**Call (%)**	ENAC **(%)**	**Call (%)**	ENAC **(%)**	**Call (%)**	ENAC **(%)**
I. Unsolvability Detectors						
`measure_regions`	27.6	**+6.12**	89.0	**+3.52**	57.5	-1.34
`threshold_image`	78.3	**+3.19**	83.5	**+0.33**	64.3	-1.32
II. Path Tracers						
`draw_polyline`	34.6	-4.07	41.1	**+0.67**	16.8	**+1.76**
`draw_line`	3.0	+0.10	6.8	**+0.41**	3.1	**+0.27**
III. Topological Primitives						
`skeletonize_image`	70.5	-3.90	98.3	-4.91	94.8	**+11.23**
`find_and_draw_contours`	34.4	-0.33	89.1	-7.21	90.7	**+6.16**
IV. Cropping & Smoothing Filters						
`crop_image`	85.1	-1.31	12.5	-1.21	1.7	-0.12
`morphological_operation`	44.2	-0.42	33.2	-3.72	36.0	-1.60
`gaussian_blur`	1.2	+0.02	47.2	-4.94	35.6	-3.01
`median_blur`	9.6	-0.43	51.9	-5.58	41.0	-3.19
`adaptive_threshold`	24.3	-1.61	69.2	-7.27	54.8	-3.97

Table D.7: **Tool call rates and Expected Net Accuracy Contribution** (ENAC) **on the Benchmark V2 atomic suite (Gemini 3.5 Flash; N=9,600 episodes,** 3,200 **per configuration).** Call frequency (**Call (%)**, percentage of episodes invoking each tool at least once) and ENAC (%; Eq. 6). **Green bold** indicates net accuracy gains (ENAC $> +0.1\%$); red indicates accuracy drops (ENAC $\leq -1.0\%$).

Unsolvability detectors vs. path tracers (Groups I–II). The tools in Groups I and II serve opposite roles: *Unsolvability Detectors* (`measure_regions`, +6.12% ENAC in **Single**) help verify that start and exit lie in disconnected regions on unsolvable mazes, whereas *Path Tracers* (`draw_polyline`, `draw_line`) help trace candidate routes on solvable mazes (ΔSens up to $+25.9\%$). In **Single**, however, `draw_polyline` hurts overall accuracy (ENAC $= -4.07\%$, Table D.8): because the agent calls it nearly as often on unsolvable mazes as on solvable ones ($\mathcal{S} = 1.42 < \mathcal{H} = 3.50$), drawing polylines frequently crosses thin walls and reinforces false-positive predictions. In **Multi-noV**, context isolation reduces this false-positive rate enough for `draw_polyline` to become slightly net-positive ($\mathcal{S} = 1.72 > \mathcal{H} = 1.54$, ENAC $= +0.67\%$). In **Multi+Ver**, programmatic verifiers reject blocked paths before the orchestrator draws final annotations ($\mathcal{S} = 5.64 \gg \mathcal{H} = 1.75$), raising `draw_polyline` to ENAC $= +1.76\%$ ($J_T = +0.484$).

Why verification reverses the impact of topological primitives (Group III). Without verification, topological operations (`skeletonize_image` and `find_and_draw_contours`) lower overall accuracy (ENAC from -0.33% to -7.21%). Note that in Table D.8, these operations have $\mathcal{H} = 0.00^{\dagger}$ in **Multi-noV** because they are called on nearly every episode (89.1%–98.3% call rate) and slightly increase unsolvable specificity ($\Delta\text{Spec} \geq 0$), but they reduce solvable recall

($\Delta\text{Sens} < 0$) because the VLM misreads 1-pixel skeleton gaps or contour boundaries as blocked passages. Pairing these same operations with programmatic verification in **Multi+Ver** eliminates these visual misreadings, turning `find_and_draw_contours` ($+6.16\%$) and `skeletonize_image` ($+11.23\%$) into the largest visual contributors to accuracy.

	Single			Multi-noV			Multi+Ver		
Tool / Operation	$\mathcal{S}$	$\mathcal{H}$	J_T	$\mathcal{S}$	$\mathcal{H}$	J_T	$\mathcal{S}$	$\mathcal{H}$	J_T
II. Path Tracers									
`draw_polyline`	1.42	3.50	+0.022	**1.72**	**1.54**	+0.248	**5.64**	**1.75**	**+0.484**
`draw_line`	**2.31**	**1.17**	+0.266	**1.88**	**0.54**	+0.454	**3.16**	**1.03**	**+0.643**
III. Topological Primitives									
`skeletonize_image`	1.05	2.90	+0.245	1.01	$0.00^{\dagger}$	+0.401	**1.01**	$0.00^{\dagger}$	**+0.666**
`find_and_draw_contours`	0.92	$0.00^{\dagger}$	+0.266	1.00	$0.00^{\dagger}$	+0.384	**0.97**	$0.00^{\dagger}$	**+0.661**

Table D.8: **Selectivity Ratio ($\mathcal{S}$), Hazard Ratio ($\mathcal{H}$), and Youden's Index (J_T) on the Benchmark V2 atomic suite (Gemini 3.5 Flash).** Diagnostics are reported for candidate path-finding operations (Groups II and III), where agents attempt route confirmation and risk inducing false positives. By Eq. 7, tools with $\Delta\text{Sens} > 0$ and $\Delta\text{Spec} < 0$ improve overall accuracy when **Selectivity exceeds Hazard ($\mathcal{S} > \mathcal{H}$, green bold)** and hurt accuracy when Hazard exceeds Selectivity ($\mathcal{S} < \mathcal{H}$, red). Here $^{\dagger}\mathcal{H} = 0.00$ indicates $\Delta\text{Spec} \geq 0$ (calling the tool does not lower unsolvable accuracy; negative ENAC in those cases comes from lower solvable recall when $\Delta\text{Sens} < 0$).

Why cropping and smoothing filters hurt accuracy (Group IV). Cropping, smoothing, and morphological filters (`crop_image`, `morphological_operation`, `gaussian_blur`, `median_blur`, and `adaptive_threshold`) consistently lower accuracy across all three configurations (ENAC between -0.12% and -7.27%). Zooming into local regions via `crop_image` loses global start-to-exit connectivity context, while blurring or eroding thin-walled mazes either seals narrow 1-pixel corridors on solvable instances or bridges 1-pixel barriers on unsolvable ones.

E Inference Token and Tool Call Profiling

This section details the token accounting methodology and presents the shorter-horizon single-agent experiment ($c = 50$), single-turn active context lengths, visual prefill breakdown, and Gemini 3.5 Flash-Lite profiles deferred from Section 5 (Tables E.1 and E.2).

E.1 Token and Step-Budget Accounting

For each episode, we log token counts at every model turn and track three quantities: generated output tokens, single-turn active prompt length $|H_t|$ (reported as the per-call mean and final-turn peak in Table E.1), and cumulative input prefill summed across all turns of the episode. Because inference prefix-cache hit rates depend on serving load and hardware routing, we report raw, uncached input prefill throughout. Recall from Section 4.2 that at turn t of a session, the model reads the full history $H_t = (I_0, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$, where I_0 ($\approx$ 1,089 visual tokens) is the initial prompt and maze image and (a_j, o_j) are prior tool calls and returned artifacts. Re-reading this history at every turn yields cumulative session prefill

$$\mathrm{Prefill}_{\mathrm{session}}(T) \;=\; \sum_{t=1}^{T} |H_t| \;=\; T\,|I_0| \;+\; \sum_{j=1}^{T-1} (T-j)\,\big(|a_j| + |o_j|\big). \tag{8}$$

In a single-agent session, retaining every intermediate image in one history causes cumulative prefill to grow quadratically with T. Delegating subgoals to isolated sub-agents resets the turn history at the end of each specialist sub-session ($T_s \ll T$), trading quadratic within-session growth for linear context-reloading overhead across sub-sessions (each specialist or verifier loads its role instructions, I_0, and selected input artifacts).

By default, both architectures operate under a matched 200-step ceiling: Single is capped at $c = 200$ steps (with a mid-budget wrap-up reminder at step 100), while the multi-agent orchestrator can make up to 20 specialist delegations, each allowing up to 10 specialist–verifier interactions ($20 \times 10 = 200$), with up to 3 tries for specialist tool calls or verifier code execution. Alongside the Single vs. Multi-noV comparison in Section 5, we run a second controlled experiment to measure the impact of context bloat on the single agent directly: restricting the single-agent step cap to $c = 50$ (Single ($c = 50$), wrap-up reminder at step 25) to test whether forcing shorter trajectories improves single-agent accuracy and token efficiency.

E.2 Shorter-Context Ablation, Active Context Windows, and Role Profiles

Shorter contexts benefit the single agent ($c = 50$ vs. default $c = 200$). In Section 5, comparing Single ($c = 200$) with Multi-noV at an identical average of 20 visual tool calls showed that isolating sub-agent contexts raises paired accuracy from 31.6% to 47.0%. Comparing Single ($c = 200$) against the shorter-horizon Single ($c = 50$) configuration in Table E.1 provides a second, complementary test of context bloat within the monolithic architecture itself. Although Single never exhausts either budget (0.0% cap hit rate; Table C.2), lowering the ceiling from 200 to 50 triggers the mid-budget wrap-up reminder earlier (step 25 vs. step 100), causing the agent to submit after an average of 10 visual calls on the atomic suite instead of 20 (7 vs. 8 on `flood_fill`). Crucially, stopping earlier *improves* single-agent paired accuracy from 31.6% to 35.6% on the atomic suite (75.8% $\rightarrow$ 80.6% on `flood_fill`) while cutting mean tokens per call in half (51K $\rightarrow$ 23K) and reducing cumulative prefill by 75% (1,086K $\rightarrow$ 270K, with visual prefill dropping from 692K to 152K in Table E.2). Without a verifier or sub-session isolation, extra single-agent turns simply accumulate flawed intermediate masks in a single shared context and increase false-positive wall crossings.

Single-turn active context window requirements. Beyond cumulative episode tokens, the single-turn prompt length $|H_t|$ determines the context window and KV-cache memory required at serving time (Table E.1). In Single, the final-turn prompt averages 100K tokens under the default $c = 200$ budget (and 43K even under $c = 50$) on the atomic suite, requiring a 128K–256K context window to avoid truncation. By contrast, because specialist sub-sessions reset after each delegation, no model call in either multi-agent architecture exceeds 23K tokens at peak: the orchestrator peaks at 15K in Multi-noV and 13K in Multi+Ver (13K and 10K on `flood_fill`), allowing every role to run within a 32K context window.

Tool Suite	Arch.	Tools Visual	Episode Tokens (K) Prefill	Gen.	Active Turn Context (K) Orch. (Mean / Peak)	Tok/C	Acc. Pair
Atomic (main)	Single ($c = 200$)	20	1086	2	51 / 100	51	31.6
	Single ($c = 50$)	10	270	1	23 / 43	23	35.6
	Multi-noV	20	409	5	11 / 15	10	47.0
	Multi+Ver	14	979	15	8 / 13	12	**67.6**
Flood-fill (control)	Single ($c = 200$)	8	223	1	25 / 30	25	75.8
	Single ($c = 50$)	7	165	1	19 / 28	19	80.6
	Multi-noV	15	274	3	9 / 13	9	81.0
	Multi+Ver	11	782	13	7 / 10	11	**87.3**

Table E.1: **Visual tool calls, cumulative episode tokens, and active turn context on Benchmark V2 for Gemini 3.5 Flash** ($M = 1{,}600$ matched pairs across 16 layouts). **Single ($c = 200$)** is the default 200-step single agent reported in Section 5; **Single ($c = 50$)** is the shorter-context ablation. **Orch. (Mean / Peak)** reports the orchestrator's mean and final-turn prefill per step, and **Tok/C** reports mean tokens per tool call across all active roles.

Tool Suite	Arch.	Input Prefill by Role (K) Orch.	Sub.	Verif.	Visual (K) Img.	Out & Total (K) Gen.	Total	Acc. Pair
Gemini 3.5 Flash								
Atomic (main)	Single ($c = 200$)	1086	—	—	692	2	1088	31.6
	Single ($c = 50$)	270	—	—	152	1	270	35.6
	Multi-noV	129	279	—	160	5	413	47.0
	Multi+Ver	65	282	632	379	15	994	**67.6**
Flood-fill (control)	Single ($c = 200$)	223	—	—	118	1	224	75.8
	Single ($c = 50$)	165	—	—	89	1	166	80.6
	Multi-noV	88	186	—	117	3	277	81.0
	Multi+Ver	54	197	531	300	13	796	**87.3**
Gemini 3.5 Flash-Lite								
Atomic (main)	Single ($c = 200$)	117	—	—	60	1	117	5.7
	Single ($c = 50$)	152	—	—	96	0	153	5.5
	Multi-noV	41	129	—	86	2	172	9.6
	Multi+Ver	39	145	238	206	7	428	**29.4**
Flood-fill (control)	Single ($c = 200$)	97	—	—	47	1	97	59.8
	Single ($c = 50$)	133	—	—	84	1	134	62.0
	Multi-noV	34	69	—	53	1	104	59.1
	Multi+Ver	28	88	154	130	4	275	**71.5**

Table E.2: **Role-level input prefill, visual prefill, generated output, and paired accuracy on Benchmark V2** ($M = 1{,}600$ matched pairs across 16 layouts per row). Input prefill (in thousands of tokens) is partitioned across the orchestrator (**Orch.**), task specialists (**Sub.**), and independent verifiers (**Verif.**); **Img.** reports visual prefill tokens.

Visual prefill and Gemini 3.5 Flash-Lite scaling. Table E.2 extends the role-level breakdown of Table 4 with visual prefill counts (**Img.**) and Gemini 3.5 Flash-Lite profiles. Despite executing the same average number of visual tool calls (20) as Single ($c = 200$) on the atomic suite, Multi-noV processes 77% fewer visual prefill tokens (160K vs. 692K) because intermediate trial masks are not re-read across subsequent turns. Adding local verification in Multi+Ver catches flawed masks inside the specialist loop before they reach the orchestrator, reducing visual tool calls by 30% relative to Multi-noV ($20 \rightarrow 14$ on atomic; $15 \rightarrow 11$ on `flood_fill`) while shifting compute into isolated verifier sandboxes (where writing short `execute_python_code` test scripts accounts for most of the 13K–15K generated output tokens). Gemini 3.5 Flash-Lite follows the same pattern under both the default $c = 200$ and shorter-context $c = 50$ single-agent baselines: delegation cuts orchestrator prefill from 117K (152K at $c = 50$) to 41K on the atomic suite and from 97K (133K at $c = 50$) to

34K on `flood_fill`, while verification raises paired accuracy from 5.7% to 29.4% on atomic and from 59.8% to 71.5% on `flood_fill`.

F Out-of-Domain Transfer to MentisOculi

Do the gains from visual tools (**Single**), isolated specialists (**Multi-noV**), and independent verification (**Multi+Ver**) rely on maze-specific assumptions, or do they transfer to other visual puzzles? To test this without changing prompts or adding task solvers, we evaluate all four setups zero-shot on the four non-maze tasks of the MentisOculi benchmark (Zeller et al., 2026): *Paper Fold*, *Rush Hour*, *Form Board*, and *Hinge Folding* (Figure F.1).

F.1 Tasks, Tooling, and Evaluation Setup

Each task spans five difficulty levels (50 puzzles per level, 250 per task; Figure F.1 and Listing G.6), targeting distinct spatial and geometric operations. **Paper Fold** tests sequential reflective symmetry by predicting unfolded hole patterns after 1–5 orthogonal or diagonal creases. **Rush Hour** evaluates constrained multi-vehicle path planning, requiring cars to slide along dashed tracks to unblock the target vehicle as move counts grow from shallow maneuvers to deep deadlocks. **Form Board** evaluates 2D silhouette tiling by selecting piece subsets that fill a target template without rotation or overlap among decoys. **Hinge Folding** tests planar kinematic reasoning by predicting orthogonal joint rotations that fold a connected polygon chain into a goal silhouette across 1–5 coupled hinges.

Tooling and Specialist Roster. The tooling and specialist setup is identical to the maze experiments in Section 4.3 and Appendix C.1. We provide no puzzle solvers, path planners, or geometry engines. Each puzzle image is loaded into the shared `OrchestratorInventory` as the initial state, and agents access the same five specialist teams from Table C.1 (`transform`, `binarize`, `topology`, `annotate`, and `measurement`). Agents freely call these general image-processing primitives to crop, rotate, flip, measure areas, and composite regions on canvases, testing intermediate configurations across all four tasks without task-specific code or prompt tuning.

Evaluation Protocol. We evaluate across 3 random seeds per tier using **Gemini 3.5 Flash**. With 50 puzzles/level $\times$ 5 levels $\times$ 3 seeds, this yields 150 episodes per level, 750 per task, and 3,000 per tier across the four architectures ($N = 12{,}000$ total episodes). All 750 episodes per task-tier are fully evaluated and scored against ground truth.

F.2 Results by Architecture Tier

Table F.1 compares the four architecture tiers on Gemini 3.5 Flash across all 3,000 episodes (750 per task), evaluated with paired two-sided McNemar tests. While verified multi-agent orchestration achieves the highest overall accuracy (73.67%), the performance delta across tiers depends on the underlying computational demands of each visual problem.

Grounding reasoning in visual manipulation primitives. Without tools, the model must solve puzzles entirely through zero-shot visual inspection. While unaugmented perception suffices for shallow transformations with minimal occlusions, it degrades sharply on tasks requiring multi-step spatial composition: zero-shot accuracy falls to approximately 41% on both Form Board and Hinge Folding. Providing a single agent with basic image-manipulation tools (crop, rotate, flip, and canvas composition) yields substantial, double-digit gains across all four task domains (pooled +393 net pairs won, $b/c = 608/215, \chi^2 = 186.7, p < 0.0001^{***}$; Table F.1), with the largest improvements occurring on multi-piece tiling (Form Board, +131 net pairs) and rotational kinematics (Hinge Folding, +109 net pairs). Rather than mentally projecting high-dimensional geometric transformations, the agent uses the shared canvas as an external workspace, physically isolating candidate regions, testing rotations, and verifying fit before finalizing its response.

Specialist delegation shields coordinator context. While visual primitives provide necessary grounding, concentrating multi-turn manipulation inside a single agent creates a severe prompt bottleneck: every intermediate crop, trial rotation, and coordinate error accumulates in the conversation history. On search-heavy tasks requiring iterative hypothesis testing, this growing history clutters the context and causes the model to fixate on stale errors. Delegating manipulation to isolated specialist sub-agents (**Multi-noV**) keeps intermediate scratchpad edits off the coordinator's primary history, significantly improving pooled accuracy over the single agent by +55 net pairs ($b/c = 334/279, \chi^2 = 4.76, p = 0.0291^{*}$). This context isolation benefit concentrates on combinatorial search and navigation, where intermediate exploration is heavy. Conversely, on shallow

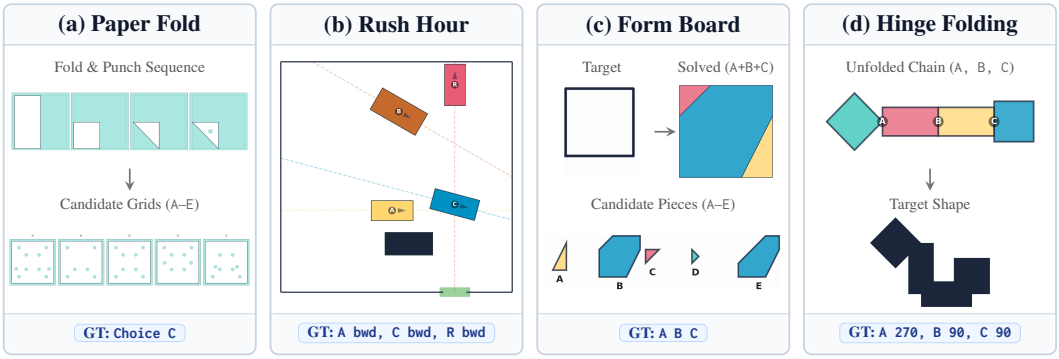


Figure F.1: **Example Level 3 puzzles from the four MENTISOCULI tasks.** Bottom pill tags indicate the ground-truth solution for each representative instance.

reflective tasks like Paper Fold, where solutions require only a few simple folds and the single agent already scores 96.13%, delegation provides no measurable lift ($b/c = 20/21, p = 1.0000$).

Verification enforces validity constraints but cannot guide search. Adding independent programmatic verifiers to specialist teams (**Multi+Ver**) yields the highest pooled accuracy (73.67%), but its efficacy reveals a sharp dichotomy between sequential navigation and combinatorial assembly. In Rush Hour, where moves are strictly ordered and a single invalid slide into an occupied lane corrupts all subsequent reasoning, verification provides a substantial boost, significantly outperforming unverified delegation by +29 net pairs ($b/c = 95/66, \chi^2 = 4.87, p = 0.0270^{*}$) and the single agent by +47 net pairs ($b/c = 104/57, \chi^2 = 13.14, p = 0.0003^{***}$). By inspecting candidate board states, the verifier intercepts collisions immediately, preventing the agent from pursuing dead ends.

In geometric tiling and kinematic chain folding, however, verification offers no advantage over unverified delegation (Form Board $b/c = 95/110, p = 0.3282$; Hinge Folding $b/c = 93/94, p = 1.0000$; pooled across all tasks +22 net pairs, $b/c = 308/286, \chi^2 = 0.74, p = 0.3889$). Detailed trace inspections illuminate the mechanistic cause: errors in Form Board and Hinge Folding rarely arise from illegal physical maneuvers, but from combinatorial search exhaustion, where the agent depletes its step budget without finding the matching subset of pieces or angles. While a verifier can confirm that a candidate shape fails to match the target silhouette, it cannot provide constructive guidance on which alternative subset or rotation to explore. Programmatic verification reliably prevents illegal actions, but cannot substitute for heuristic search guidance.

Table F.1: **Accuracy across architecture tiers on MentisOculi with Gemini 3.5 Flash** (3,000 episodes per tier across Levels 1–5 and 3 seeds; $N = 12{,}000$ total). Entries report accuracy (%) and 3-seed Mean ± SD.

Task Domain (N)	**No tools**	**Single**	**Multi-noV**	**Multi+Ver**
Paper Fold (750)	82.67	96.13	96.00	**97.20**
Rush Hour (750)	66.13	73.07	75.47	**79.33**
Form Board (750)	41.87	59.33	**61.73**	59.73
Hinge Folding (750)	41.33	55.87	**58.53**	58.40
Pooled Overall (3,000)	58.00	71.10	72.93	**73.67**
3-Seed Mean ± SD	58.00±0.40	71.10±0.66	72.93±0.71	**73.67±0.49**

F.3 ACCURACY ACROSS DIFFICULTY LEVELS (LEVELS 1–5)

Table F.2 examines performance across difficulty levels (Levels 1–5, 150 episodes per cell across 3 seeds), illuminating how agent architectures interact with expanding problem depth.

High baseline competence on shallow horizons (Levels 1–2). At the lowest difficulty tiers, spatial reasoning horizons are short: Paper Fold requires only 1–2 orthogonal creases, Rush Hour requires 2–4 unconstrained vehicle slides, and Hinge Folding involves only 1–2 pivot rotations. On these shallow instances, zero-shot visual inspection demonstrates high competence, solving 92.7%–100.0% of puzzles across tasks (Table F.2). Tool-augmented agents reliably attain near-ceiling accuracy (94.0%–100.0%). The solitary outlier is Level 1 Form Board (34.0% without tools), where the model must distinguish fine metric shape variations between individual candidate pieces. Here, tool grounding provides an immediate perceptual scaffold: measuring piece areas and testing silhouette alignment lifts accuracy to 66.7% for the single agent and 76.7% with verification ($+42.7$ pp over zero-shot).

Compounding transformation depth drives unaugmented collapse (Levels 4–5). As puzzle complexity advances to Levels 4 and 5, unaugmented visual inspection breaks down across all domains. In Rush Hour, trajectory requirements expand into dense multi-car deadlocks, driving zero-shot accuracy down from 100.0% at Level 1 to 25.3% at Level 5 (a 74.7 pp collapse). An even steeper drop occurs in Hinge Folding, where tracking five sequential 90° joint rotations pushes unaugmented accuracy down to 12.7% (-80.6 pp), and in Paper Fold, where multi-fold diagonal creases cut baseline accuracy to 55.3%. Without external storage, mental simulation carries no persistent intermediate state; orientation and coordinate errors compound exponentially over successive operations, rendering unaugmented reasoning brittle on deep trajectories.

External scratchpads and verification buffer against depth collapse. Tool manipulation fundamentally alters this degradation curve. On Paper Fold, testing intermediate folds on a digital canvas preserves 88.0%–89.3% accuracy at Level 5 ($+34.0$ pp over the zero-shot baseline). On Rush Hour Level 5, where losing track of car positions causes the single agent to stall at 34.7%, programmatic state verification (**Multi+Ver**) maintains 48.7% accuracy ($73/150$), significantly outperforming the single agent by $+14.0$ pp (McNemar $b/c = 41/20, p = 0.0099^{**}$) and unverified delegation by $+16.7$ pp. Similarly, physically rotating chain segments on the canvas triples Level 5 Hinge Folding accuracy to 42.0% ($+29.3$ pp over zero-shot). External visual scratchpads thus prevent compounding simulation drift, while programmatic verifiers catch invalid intermediate states before they derail multi-step trajectories.

Table F.2: **Accuracy across difficulty levels (Levels 1–5) on MentisOculi with Gemini 3.5 Flash** (150 episodes/level per task, 600 pooled across 3 seeds; $N = 12{,}000$ total). Entries report 3-seed Mean $\pm$ SD (%). Bold indicates top score per level.

Task Domain	Architecture	Level 1	Level 2	Level 3	Level 4	Level 5	Overall
Paper Fold	No tools	92.7 ±1.2	97.3 ±1.2	92.7 ±4.2	75.3 ±7.6	55.3 ±10.3	82.7 ±0.5
	Single	98.7 ±1.2	**100.0** ±0.0	97.3 ±1.2	**96.7** ±4.2	88.0 ±3.5	96.1 ±1.0
	Multi-noV	98.7 ±1.2	**100.0** ±0.0	**100.0** ±0.0	94.0 ±3.5	87.3 ±7.0	96.0 ±1.8
	Multi+Ver	**100.0** ±0.0	**100.0** ±0.0	**100.0** ±0.0	**96.7** ±1.2	**89.3** ±8.3	**97.2** ±1.8
Rush Hour	No tools	**100.0** ±0.0	92.0 ±2.0	62.0 ±7.2	51.3 ±4.2	25.3 ±6.4	66.1 ±3.0
	Single	**100.0** ±0.0	96.0 ±2.0	74.0 ±3.5	60.7 ±2.3	34.7 ±8.1	73.1 ±0.2
	Multi-noV	**100.0** ±0.0	94.7 ±2.3	72.7 ±3.1	**78.0** ±3.5	32.0 ±3.5	75.5 ±0.5
	Multi+Ver	**100.0** ±0.0	**98.0** ±2.0	**80.0** ±2.0	70.0 ±4.0	**48.7** ±7.0	**79.3** ±1.9
Form Board	No tools	34.0 ±2.0	59.3 ±8.1	41.3 ±1.2	34.0 ±7.2	40.7 ±4.2	41.9 ±3.6
	Single	66.7 ±5.0	**64.7** ±4.2	54.0 ±4.0	47.3 ±3.1	64.0 ±5.3	59.3 ±1.6
	Multi-noV	70.0 ±8.0	57.3 ±3.1	**55.3** ±5.0	**48.0** ±2.0	**78.0** ±3.5	**61.7** ±2.3
	Multi+Ver	**76.7** ±1.2	57.3 ±4.2	53.3 ±8.1	47.3 ±1.2	64.0 ±0.0	59.7 ±1.6
Hinge Folding	No tools	93.3 ±5.0	51.3 ±2.3	33.3 ±5.8	16.0 ±2.0	12.7 ±5.0	41.3 ±1.2
	Single	94.0 ±2.0	54.7 ±6.1	48.0 ±5.3	47.3 ±6.4	35.3 ±9.5	55.9 ±4.0
	Multi-noV	**98.7** ±1.2	**55.3** ±4.6	**56.7** ±1.2	45.3 ±4.6	36.7 ±1.2	**58.5** ±1.0
	Multi+Ver	96.0 ±2.0	54.0 ±2.0	51.3 ±5.8	**48.7** ±8.1	**42.0** ±9.2	58.4 ±1.6
All Tasks Pooled ($N = 3{,}000$ episodes/tier)							
	No tools	80.0 ±1.3	75.0 ±2.0	57.3 ±0.8	44.2 ±3.3	33.5 ±2.3	58.0 ±0.4
	Single	89.8 ±1.0	**78.8** ±2.4	68.3 ±0.8	63.0 ±2.2	55.5 ±3.1	71.1 ±0.7
	Multi-noV	91.8 ±2.0	76.8 ±0.8	**71.2** ±0.8	**66.3** ±1.5	58.5 ±3.6	72.9 ±0.7
	Multi+Ver	**93.2** ±0.3	77.3 ±1.3	**71.2** ±3.3	65.7 ±2.0	**61.0** ±3.5	**73.7** ±0.5

F.4 Token Usage and Tool Call Profiling

Table F.3 breaks down token counts and tool calls across all 12,000 **Gemini 3.5 Flash** episodes, matching the maze cost analysis in Appendix E. Giving the single agent tools (**Single**) raises average usage from 10.1K tokens (**No tools**, 2.3 calls) to 579.3K tokens (18.6 calls) because every crop, rotation, and canvas edit stays in one growing prompt history. Moving tool work into separate specialists (**Multi-noV**) raises total tokens to 1,148.2K (53.4 calls), but cuts the orchestrator's context by 78.3% (from 579.3K to 125.7K tokens, with 1,022.6K kept inside specialist sessions). Adding verification (**Multi+Ver**) averages 2.3 checks per episode (624.0K verifier tokens; 1,456.4K total) and shrinks the orchestrator's context further to 81.3**K tokens** (86.0% smaller than **Single**). The modular multi-agent architecture thus exchanges parallel background tokens for compact, high-signal coordinator context.

Table F.3: **Token usage and tool call profiling on MentisOculi with Gemini 3.5 Flash** ($N = 12{,}000$ episodes across 3 seeds). Token quantities are reported as mean cumulative tokens per episode in thousands (K); tool calls are mean calls per episode.

		Input Tokens by Role (K)			Total	Tool	Verif.
Task Domain	**Architecture**	**Orch.**	**Spec.**	**Verif.**	**Tok (K)**	**Calls**	**Checks**
Paper Fold	No tools	11.4	–	–	11.4	2.2	–
	Single	546.6	–	–	546.6	19.5	–
	Multi-noV	149.4	1147.3	–	1296.7	58.0	–
	Multi+Ver	77.4	737.0	543.0	1357.4	57.6	1.8
Rush Hour	No tools	5.3	–	–	5.3	1.8	–
	Single	145.3	–	–	145.3	8.0	–
	Multi-noV	55.0	608.9	–	663.9	31.0	–
	Multi+Ver	42.2	376.8	383.3	802.3	40.8	1.6
Form Board	No tools	10.5	–	–	10.5	2.6	–
	Single	1289.4	–	–	1289.4	33.1	–
	Multi-noV	166.9	1481.4	–	1648.3	77.8	–
	Multi+Ver	87.5	1033.9	796.2	1917.6	79.3	2.5
Hinge Folding	No tools	13.0	–	–	13.0	2.5	–
	Single	335.7	–	–	335.7	14.0	–
	Multi-noV	131.3	852.7	–	983.9	47.0	–
	Multi+Ver	118.3	856.5	773.7	1748.4	81.1	3.4
All Domains Pooled ($N = 3{,}000$ episodes/tier)							
	No tools	10.1	–	–	10.1	2.3	–
	Single	579.3	–	–	579.3	18.6	–
	Multi-noV	**125.7**	1022.6	–	1148.2	53.4	–
	Multi+Ver	**81.3**	751.0	624.0	1456.4	64.7	2.3

G Prompt Templates and System Instructions

We use the same system prompt templates across all architectural tiers and benchmarks. Section G.1 presents the domain-agnostic instructions for the coordinator, task specialist, independent verifier, and single-agent baseline, where highlighted placeholders (such as `{task_description}`) are populated at runtime with the task specifications for AMAZE (Section G.2) or MENTISOCULI (Section G.3). In the tool-free baseline, the inventory and tool instructions are omitted from Listing G.4; in unverified delegation, the verification workflow is omitted from Listing G.2.

G.1 Common Agent Architecture Templates

Listing G.1: Multi-Agent Coordinator (Orchestrator) System Prompt

Role & Task Setup: You are the coordinator of a team of `{domain_role}` specialists. `{task_description}` You have a shared inventory that already contains the input image under the key `{input_image_key}` (pre-loaded, read-only; do NOT overwrite it) where you can store and retrieve derived data.

Domain Visual Properties & Rules: `{domain_visual_rules}`

Available Specialists: `{skills_summary}` Specialists are sub-agents with isolated context windows: delegate a task by calling a specialist with an `instruction` and the `context_keys` they need from your inventory.

Coordination & Evidence Synthesis Principles:

1. **Rely on Verified Specialist Findings**: Base your final conclusion on verified structural data and visual artifacts produced by your specialists; do not override verified specialist evidence with ungrounded visual guesses.
2. **Specialist Roles**: Use `image_transformation_specialist` and `filter_binarize_specialist` to crop sub-regions and isolate foreground structure from wall textures/gradients; use `structural_topology_specialist` and `canvas_annotate_specialist` to extract skeletons/contours and composite candidate states onto visual overlays; use `measurement_analysis_specialist` to compute geometric properties and connectivity.

Interaction & Termination Protocol: Before each tool call, state your reasoning in `<|channel>thought...<channel|>` and invoke the tool. Required answer format: `{answer_format}`. Conclude by calling `{termination_tool}`.

Listing G.2: Task Specialist Instruction Template

Role & Goal: You are a `{role}`. `{goal}` | **Available Tools:** `{tool_names}`

Domain Visual Properties: `{domain_visual_rules}`

Tool Execution & Parameter Verification: (1) Select tool arguments carefully based on visual inspection and analysis context. (2) Inspect the tool return payload and generated artifacts immediately; if the payload indicates an error, partial result, or unexpected outcome, adjust your parameters and re-invoke the tool before completing your task.

Verification Workflow (Multi+Ver): After producing your output, call `request_verification` with a concise summary of what you produced. The independent verifier will test your artifact and return a verdict:

- **`pass`**: Artifact is valid and accurate; immediately call `finish_task(result=...)` with your full data output and summary.
- **`fail`**: Errors or discrepancies detected; read the verifier's feedback, inspect output images in your context, revise your parameters, and call `request_verification` again (up to the verification budget).

Task Completion & Missing Prerequisites: `finish_task(result=...)` is the ONLY way your output reaches the coordinator. If a required key is missing from the shared inventory, do NOT fabricate it; call `finish_task` immediately stating which prerequisite is missing.

LISTING G.3: INDEPENDENT VERIFIER INSTRUCTION TEMPLATE

Role: You are an independent `{role}` verification specialist. Your job is to verify whether the specialist's output/artifact accurately reflects the input image (`{input_image_key}`) and satisfies the task constraints.

Step 1 — Inspect Output & Parameter Validity: Examine the specialist's tool calls, chosen arguments, return payloads, and generated artifacts. Check whether the parameters were appropriate and whether the resulting artifact is consistent with the source image.

Step 2 — Programmatic Code Verification (`execute_python_code`): Write and run Python checks via `execute_python_code` to quantitatively test the specialist's artifact against the shared inventory (`_context`, inspectable via `read_shared_context(key)`). Pre-loaded sandbox modules: `numpy` (`np`), `cv2`, `PIL.Image`, `json`, `collections`, `math`, `re`, `heapq`, `itertools`. You MUST execute at least one programmatic check before calling `finish_task`.

Output Format & Critical Rules:

- Return structured JSON via `finish_task(result=...)` with keys `verdict` (`'pass'` or `'fail'`), `summary`, and `feedback`.
- Issue `verdict='fail'` ONLY for genuine errors in the artifact itself (e.g., crossing through solid/hatched walls, misidentifying open regions, or invalid coordinates). A verified proof of unreachability/unsolvability is a valid outcome and MUST receive `verdict='pass'`.
- Do NOT solve the task yourself; only write code to TEST and VALIDATE the specialist's artifact.

LISTING G.4: MONOLITHIC SINGLE-AGENT SYSTEM PROMPT

Role & Task Setup: You are a visual `{domain_role}` agent. `{task_description}` You have a shared inventory containing the input image under `{input_image_key}` (read-only) and direct access to all visual tools (`source_asset_id=`{input_image_key}).

Domain Visual Properties & Rules: `{domain_visual_rules}`

Interaction & Termination Protocol: Reason step-by-step in `<|channel>thought...<channel|>` and invoke tools to inspect and verify spatial constraints. Required answer format: `{answer_format}`. Conclude by calling `{termination_tool}`.

G.2 MAZE BENCHMARK TASK SPECIFICATION

For all AMAZE evaluations, the common templates in Section G.1 are instantiated with the task specification in Listing G.5.

LISTING G.5: MAZE BENCHMARK TASK SPECIFICATION (AMAZE V1 & V2)

Shared Wrapper Slots: `{domain_role}` = `"maze analysis"` | `{input_image_key}` = `"maze_image"` | `{termination_tool}` = `submit_solution(solvability="yes"|"no", path_trace="...")`

AMAZE Solvability & Path-Tracing Task

Task & Visual Rules (`{task_description}`, `{domain_visual_rules}`): *"Given a maze image, determine if it is solvable (i.e., whether a continuous open path exists from the green start circle labeled 'S' to the red exit circle labeled 'E'). Rules: (1) Mazes may have solid walls, hatched patterns, textures, or shaded gradients; hatched or textured patterns within a wall boundary represent impassable wall barriers, not open corridors. (2) Corridors (paths) are open, traversable regions (white / light-colored); walls and outer boundaries are impassable barriers (dark, solid, or hatched). (3) If a verified graph, flood fill, or connectivity check shows that start 'S' and exit 'E' lie in disconnected components (or all corridors from 'S' terminate at dead ends), treat this as conclusive proof that the maze is unsolvable (`solvability="no"`), not a tool error."*

Output Format (`{answer_format}`): Binary solvability decision in {`"yes"`, `"no"`} and supporting corridor trace.

G.3 MENTISOCULI BENCHMARK TASK SPECIFICATIONS

For the out-of-domain transfer evaluation on MENTISOCULI (Zeller et al., 2026) (Appendix F), the common templates in Section G.1 are instantiated with the four unmodified task instruction strings from Zeller et al. (2026) shown in Listing G.6.

LISTING G.6: MENTISOCULI TASK SPECIFICATIONS

Shared Wrapper Slots: `{domain_role}` = `"visual-spatial reasoning"` | `{input_image_key}` = `"puzzle_image"` | `{termination_tool}` = `submit_answer(answer="...", rationale="...")`

---

1. Paper Fold

Task & Visual Rules (`{task_description}`, `{domain_visual_rules}`): *"In the first row, it shows a sequence of folds performed on a square paper. The final image in the sequence shows one or more holes being punched into the folded paper. The second row shows five unfolded square papers with different hole patterns labeled A through E. Rules: (1) The paper starts as a flat square. (2) Each step shows the paper being folded along a line (horizontal, vertical, or diagonal). (3) After all folds are complete, one hole is punched through all layers at the marked positions. (4) When the paper is unfolded completely, holes appear at multiple positions due to the layering. (5) One of the five options (A, B, C, D, E) shows the correct hole pattern. Task: Mentally follow each fold in sequence, track where the hole is punched through all folded layers, unfold in reverse order, and determine which unfolded pattern (A, B, C, D, or E) matches your result."*

Output Format (`{answer_format}`): Single option letter in {`A`, `B`, `C`, `D`, `E`} (e.g., `"C"`).

---

2. Rush Hour

Task & Visual Rules (`{task_description}`, `{domain_visual_rules}`): *"It shows the initial configuration of a congested parking lot. Each colored rectangle with a letter and arrow represents a vehicle. Black rectangles without a letter represent immovable obstacles. The light green area at the border indicates the exit. The goal is to move the red vehicle (marked with an R) to the exit. Rules: (1) Each vehicle can only move forward or backward along its own axis (indicated in the image as a dashed line)—no rotation is allowed. The arrow on each vehicle indicates the forward direction. (2) A vehicle continues to move in the chosen direction until it touches another vehicle, an immovable object, or the image's boundary. (3) Only one vehicle moves per action. (4) The red vehicle must reach the exit on the edge of the grid. Task: Plan the minimal sequence of moves needed to free the red car and allow it to exit the parking lot."*

Output Format (`{answer_format}`): Comma-separated list of `<VEHICLE> <forward|backward>` moves (e.g., `"A forward, C backward, E forward, R forward"`).

---

3. Form Board

Task & Visual Rules (`{task_description}`, `{domain_visual_rules}`): *"It is showing from left to right, a target shape outlined in black and five pieces labeled A through E in various colors. Rules: (1) The target shape can be assembled using* 1 *to* 5 *of the given pieces. (2) Pieces must fit together perfectly with no gaps or overlaps. (3) Some pieces are distractors and are not needed. (4) Pieces are shown in their correct orientation and size (no rotation or scaling needed). Task: Determine the subset of pieces from {A, B, C, D, E} necessary to assemble the target shape."*

Output Format (`{answer_format}`): Space- or comma-separated subset of piece labels from {`A`, . . . , `E`} (e.g., `"A C E"`).

---

4. Hinge Folding

Task & Visual Rules (`{task_description}`, `{domain_visual_rules}`): *"The left side shows several rigid shapes connected by labeled hinges (A, B, C, . . .). On the right side is a target folded configuration. Rules: (1) Shapes are connected in a kinematic chain; each hinge connects two adjacent shapes. (2) Rotating hinge* N *causes the shape on the right side of the hinge in the original configuration to rotate anti-clockwise; all shapes to the right of the rotated shape rotate with it, while shapes to the left remain fixed. (3) Rotations must be multiples of* 90° *(*$90^\circ, 180^\circ, 270^\circ$*). (4) Shapes maintain their connections throughout all rotations. Task: Determine the rotation angle (in degrees) for each numbered hinge to achieve the target configuration."*

Output Format (`{answer_format}`): Comma-separated list of `<HINGE> <ANGLE>` pairs in degrees (e.g., `"A 90, B 90, C 180"`).