When Does Visual Generation Help Visual Understanding in Unified Multimodal Models?

Yubo Zhu[1,3*], **Zhehan Kan**[2*], **Jingyi Yang**[4*], **Miaolin Chen**[1], **Jinbo Xing**[3], **Kai Zhu**[3†], **Zijian Wang**[1], **Sheng Zhong**[1], **Wei Tong**[1†]

[1] Nanjing University, [2] Tsinghua University, [3] TongYi Lab, [4] Fudan University
zyb@smail.nju.edu.cn zkzy@mail.ustc.edu.cn wtong@nju.edu.cn

Abstract

Unified multimodal models (UMMs) can perform both understanding and generation, raising a central question: can visual generation improve understanding? Existing evaluations provide mixed evidence, but confound task difficulty, reasoning paradigms, and the closed-loop interaction between generation and understanding. We introduce *VGAU-Diag*, a fine-grained evaluation framework for **v**ision **g**eneration-**a**ssisted **u**nderstanding. It stratifies samples by difficulty, enables unified evaluation of multiple reasoning paradigms, and uses Oracle-Assisted Reference Protocols. Our analysis shows that generated visual aids help on easier instances but become unreliable as reasoning complexity increases. Oracle-assisted diagnosis further reveals that the main bottleneck often lies on the visual-understanding side rather than the visual-generation side, as current UMMs struggle to leverage even faithful visual aids. We also show that effective visual generation should target visual-understanding bottlenecks rather than add more reasoning steps, and identify a three-stage transition from task-irrelevant noise, to misleading plausible guidance, and finally to useful assistance. These findings would be useful to guide the development of better UMMs. The code is available at https://github.com/zyb1029/VGAU-Diag.

1 Introduction

Recent advances in UMMs have enabled a single model to perform both visual understanding and visual generation (Deng et al., 2025; Wu et al., 2026; Meituan LongCat Team et al., 2026; Inclusion AI et al., 2026). This progress raises a fundamental question: can visual generation serve as an intermediate reasoning process that improves visual understanding? This question has been explored through several representative vision generation-assisted understanding paradigms. In Vision-CoT (VCoT), models reason through interleaved textual and visual chains of thought. In Generate-then-Answer (GtA), models first generate auxiliary visual content and then answer through textual reasoning. In Vision-Reasoning (VR), models directly produce visual outputs to solve the problem. Recent works (Shi et al., 2026; Zou et al., 2026; Wen et al., 2026) have tried these paradigms and suggested that visual generation can provide useful intermediate evidence for visual understanding.

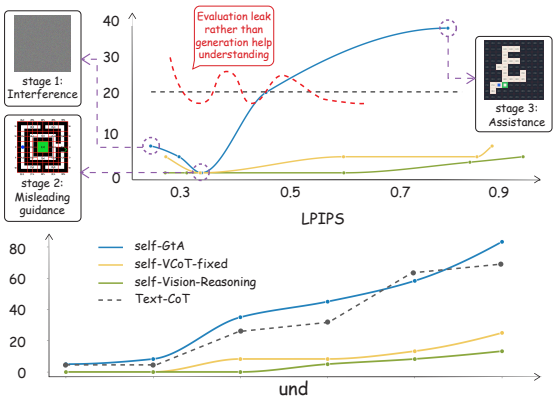

Figure 1: Solid curves show vision generation-assisted paradigms. **Top:** LPIPS serves as a proxy to distinguish interference, misleading guidance, and assistance. The dashed red trajectory denotes optimal-step prior leakage in Self-VCoT. **Bottom:** Final performance varies with visual understanding capability across different-sized Qwen-series models under the same visual aids.

However, these gains are not consistently reliable. Across different models and vision generation-assisted paradigms, the improvements brought by visual generation vary substantially. We argue that this variability arises from three key limitations in existing evaluations. *First*, current evaluations are commonly formulated as multiple-choice tasks and often mix instances with different difficulty levels, making it unclear whether observed gains reflect genuine capability improvement or stochastic success on easier samples. *Second*, each task is typically evaluated with only one visual rea-

* Equal contribution.
† Corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 8749–8788
October 24-29, 2026 ©2026 Association for Computational Linguistics

soning paradigm, without controlled comparison against other generation-assisted paradigms. As a result, it remains unclear whether the improvement truly comes from visual generation or from paradigm-specific evaluation. *Third*, existing evaluations largely treat UMM-based visual reasoning as a closed-loop black box, where visual generation and visual understanding are tightly entangled, making it difficult to diagnose whether failures mainly stem from insufficient visual understanding or unreliable visual generation.

Motivated by these limitations, we introduce a fine-grained evaluation framework for vision generation-assisted understanding. *First*, rather than evaluating mixed-difficulty samples with a single aggregate score, we organize instances by difficulty, which makes it possible to examine whether visual generation helps consistently across different levels of reasoning complexity. *Second*, our evaluation framework supports three representative vision generation-assisted understanding paradigms under the same setting: VCoT, GtA, and VR, enabling controlled comparison across different ways of using visual generation for reasoning. *Third*, we introduce Oracle-Assisted Reference Protocols to decouple visual generation from visual understanding, enabling us to diagnose whether failures mainly stem from weak understanding or unreliable generation. To support this evaluation, we use visual planning tasks, where models must infer task-relevant spatial states from images and produce executable action sequences. This task formulation provides a verifiable and fine-grained testbed for assessing whether visual generation can support visual understanding.

Under this evaluation framework, we compare multiple vision generation-assisted reasoning paradigms and obtain several key findings as illustrated in Figure 1. *First*, difficulty-aware evaluation is crucial for revealing the effect of visual generation: when samples are stratified by difficulty, visual generation yields observable gains on easier instances, but its benefits become much less stable on harder ones. This indicates that current unified models can exploit generated visual aids when the visual state is relatively simple, yet still struggle to translate visual generation into reliable understanding gains for more complex states. *Second*, visual understanding is the primary bottleneck for vision generation-assisted understanding. Although proprietary VLMs can substantially benefit from oracle visual aids, UMMs still perform poorly even when faithful aids are provided, suggesting that current UMMs often lack the visual understanding ability needed to effectively leverage such aids. *Third*, vision generation-assisted understanding exhibits a three-stage transition: task-irrelevant visuals mainly act as interference, real but incorrect visual aids can provide misleading guidance, and high-fidelity task-aligned visual aids eventually offer helpful assistance. *Finally*, effective visual generation should target the bottleneck of visual understanding rather than simply add more visual reasoning steps. We find a new GtA-based method offers targeted assistance by identifying decision-critical visual states and generating auxiliary visual representations for textual reasoning. By contrast, VCoT's unstable gains mainly stem from optimal-step prior leakage, after removing this prior, GtA remains more reliable, suggesting that targeted auxiliary generation is more effective than producing more intermediate visual states.

Our contributions are summarized as follows:

a) We introduce *VGAU-Diag*, a fine-grained evaluation framework for vision generation-assisted understanding in UMMs, enabling evaluation across different difficulty levels, unified comparison of representative vision generation-assisted understanding paradigms, and oracle-assisted diagnosis of understanding–generation bottlenecks.

b) We argue the view that unreliable visual generation is the main bottleneck of vision generation-assisted understanding. Through oracle-assisted analysis, we show that the dominant bottleneck lies in visual understanding: current UMMs struggle to use even faithful visual aids. Based on this diagnosis, we study a bottleneck-targeted GtA method that generates task-relevant visual intermediates for reasoning. After correcting optimal-step prior leakage in previous Vision-CoT evaluations, targeted GtA proves more reliable than generic step-wise visual reasoning, showing that effective visual generation should address visual-understanding bottlenecks rather than add more visual reasoning steps.

c) We argue that the benefit of vision generation-assisted understanding does not simply increase monotonically with stronger visual generation. Instead, we reveal a three-stage transition in vision generation-assisted understanding: generated intermediates first act as task-irrelevant noise, then become misleading guidance when they are visually plausible but incorrect, and finally provide stable assistance once both visual understanding and visual generation reach a sufficient capability level.

2 Related Work

Visual Generation for Visual Understanding. Recent VLM research shows that visual generation can improve visual understanding during training through visual supervision (Wang et al., 2025b,a; Wei et al., 2026; Zhu et al., 2025; Dai et al., 2025; Shao et al., 2025; Shen et al., 2026; Shao et al., 2026; Kan et al., 2026), or during inference through external visual aids that assist grounding (Cheng et al., 2025; Li et al., 2026; Leng et al., 2025; Wu et al., 2025b; Fu et al., 2025b,a; Li et al., 2025).

Unified Multimodal Models. Unified multimodal models (UMMs) aim to integrate visual understanding and visual generation within a single model. Recent UMMs, including Show-o2, Janus-Pro, BAGEL, OmniGen2, LongCat-Next, have demonstrated increasingly unified capabilities across image-text understanding and visual generation (Xie et al., 2025; Chen et al., 2025; Deng et al., 2025; Yu et al., 2025, 2026; Wu et al., 2026; Meituan LongCat Team et al., 2026; Inclusion AI et al., 2026). This integration makes UMMs a natural setting for studying whether self-generated visual aids can improve visual understanding.

Vision Generation-Assisted Understanding in UMMs. Recent works have begun to evaluate whether vision generation can assist understanding in UMMs, such as (Xie et al., 2026; Shi et al., 2026; Liang et al., 2026; Zou et al., 2026; Zeller et al., 2026; Wen et al., 2026). MME-Unify, RealUnify, and Rover (Xie et al., 2026; Shi et al., 2026; Liang et al., 2026) address this question at the unified multimodal level, Uni-MMMU (Zou et al., 2026) examines bidirectional understanding–generation synergy across multi-discipline reasoning tasks, MentisOculi (Zeller et al., 2026) probes whether UMMs possess vision-reasoning capabilities, and UniG2U-Bench (Wen et al., 2026) reports that most UMMs obtain gains under the Vision-CoT paradigm on step-level accuracy. However, existing evaluations mix task difficulties or are confined to a single evaluation paradigm, rely on weakly verifiable answers, and treat generation–understanding interactions as a closed-loop black box. In contrast, we provide a controlled and verifiable testbed for vision generation-assisted understanding, enabling difficulty-aware comparison across reasoning paradigms and fine-grained diagnosis of generation–understanding interactions, thereby revealing when visual generation helps and what limits its effectiveness in current UMMs.

3 Evaluation Framework

We evaluate vision generation-assisted understanding in UMMs across six verifiable visual planning task families and three difficulty levels, adopt both closed-loop self-generation and oracle-assisted reference protocols, enabling controlled comparison across visual-aid paradigms and fine-grained diagnosis of generation–understanding bottlenecks.

3.1 Task Design

We instantiate the framework with visual planning tasks, whose image-grounded spatial reasoning and executable action outputs provide a verifiable and fine-grained testbed for generation-assisted visual understanding. The suite covers six families: Maze, Sokoban, Path, Onet, Parking, and Klotski, spanning path search, object manipulation, route tracing, pair matching, constrained motion, and sliding-block planning. **Maze:** Navigate from the start to the goal on a grid while avoiding obstacles. **Sokoban:** Push all boxes onto their target locations through agent-object interaction. **Path:** Identify the ordered letters passed along a path from the start to the end. **Onet:** Connect and eliminate all matching pairs under rule-based path constraints. **Parking:** Move vehicles in a constrained layout to guide the target vehicle to the exit. **Klotski:** Slide blocks within a confined board to move the target block to the goal position.

Many existing evaluations of vision generation-assisted understanding rely on multiple-choice questions, where UMMs perform close to random guessing, leading to unstable results. To ensure more reliable evaluation, we require models to generate complete executable solutions instead of selecting from predefined options. For each task family, we construct 100 instances and split them into easy, medium, and hard levels. Detailed settings are provided in Appendix A.1.

3.2 Evaluation Protocols

Prior evaluations typically examine only a single visual-aid paradigm, treating one specific implementation as evidence for vision generation-assisted understanding as a whole. Moreover, evaluations based only on UMMs' self-generated visuals cannot disentangle whether failures arise from poor generation fidelity, weak visual understanding, or the vision generation-assisted understanding paradigm itself. To address these limitations, we use two groups of protocols: closed-

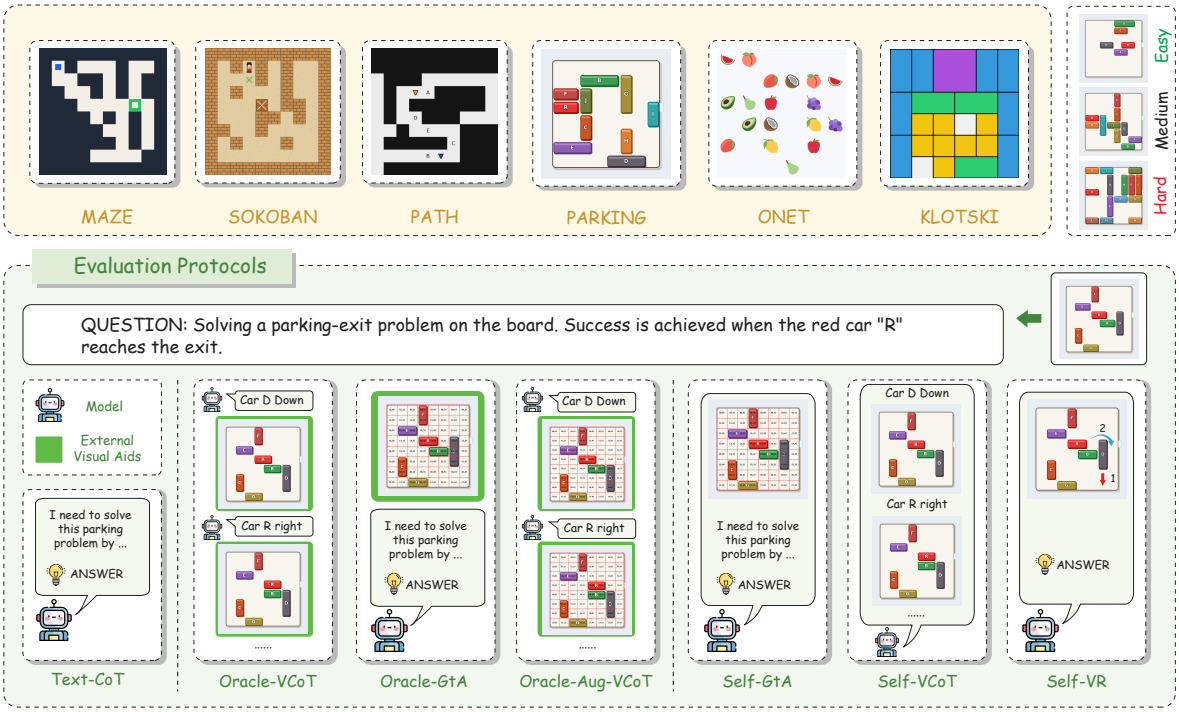

Figure 2: Overview of proposed evaluation framework.

loop self-generation protocols, including Self-GtA, Self-VCoT, and Self-VR, and oracle-assisted reference protocols for both VLMs and UMMs, where externally constructed visual aids replace model-generated ones. Together, these protocols enable controlled comparison across vision generation-assisted understanding paradigms and fine-grained diagnosis of generation fidelity, visual understanding, and paradigm-level bottlenecks.

3.2.1 Self-Generation Protocols for UMMs

For UMMs, we evaluate three self-generation protocols: Self Generate-then-Answer, Self Vision CoT, and Self Visual Reasoning.

Self Generate-then-Answer (Self-GtA). Self-GtA follows a two-stage generation-to-understanding paradigm: the UMM first generates an auxiliary visual output, then performs textual reasoning conditioned on both the auxiliary visual and the original task. For vision planning tasks, we instantiate this paradigm as grid augmentation, where UMMs add grids and coordinates to the original visual input to clarify spatial layouts.

Self Vision CoT (Self-VCoT). Self-VCoT requires the model to solve the task through step-by-step visual interaction rather than relying on a single generated aid. Following prior vision-planning evaluations (Xie et al., 2026; Zou et al., 2026; Wen et al., 2026), the model predicts one action at each step, after which the next visual state is generated and fed back for the next prediction. This process continues until an optimal-length solution is completed, an invalid action is produced, or the optimal step budget is exceeded.

Self Visual Reasoning (Self-VR). Self-VR evaluates whether UMMs can reason directly through visual generation. Instead of using generated visuals as intermediate aids, the model produces a visual output that encodes its predicted solution. For vision planning tasks, such outputs may include a route from the start to the target, a sequence of movement directions, or selected object pairs.

3.2.2 Oracle-Assisted Reference Protocols

Self-generation protocols evaluate UMMs in a closed loop, but their failures entangle generation fidelity, visual understanding, and paradigm effectiveness. To disentangle these factors, we introduce oracle-assisted reference protocols, where externally provided visual aids replace model-generated outputs. For VLMs, these aids estimate the attainable performance of each paradigm under strong visual understanding, providing an upper reference under the same visual-aid design and evaluation protocol. For UMMs, oracle conditions control for generation fidelity, so the gap from self-generation reveals whether the bottleneck lies in producing the aid, using it for plan-

ning, or both. We consider four reference conditions: **Text-CoT.** The model solves the task with textual chain-of-thought reasoning only, without auxiliary visual aids. **Oracle-Vision-CoT (Oracle-VCoT).** Oracle-VCoT follows Self-VCoT but replaces model-generated step-wise visual states with externally constructed ones. **Oracle-Generate-then-Answer (Oracle-GtA).** Oracle-GtA follows Self-GtA but replaces the model-generated auxiliary visual with an externally constructed aid of the same format. **Oracle-Augmented-Vision-CoT (Oracle-Aug-VCoT).** Oracle-Aug-VCoT extends Oracle-VCoT by applying the same auxiliary-visual augmentation used in Oracle-GtA to each externally constructed step-wise visual state.

These four conditions enable controlled comparisons across vision generation-assisted paradigms and visual-aid designs. Beyond comparing text-only reasoning with generation-assisted reasoning, the oracle protocols support two targeted analyses: Oracle-VCoT vs. Oracle-Aug-VCoT measures the effect of adding task-relevant structural cues to step-wise visual states, while Oracle-GtA vs. Oracle-Aug-VCoT compares global GtA with step-wise VCoT under the same augmented visual-aid design. Together, these comparisons help disentangle the effect of reasoning paradigm from explicit visual augmentation. The details are provided in A.2.

3.3 Solver-based Verification

Previous evaluations of generation-assisted understanding in UMMs (Xie et al., 2026; Zou et al., 2026; Wen et al., 2026) often rely on exact-match or step-wise accuracy metrics. Exact-match evaluation rejects valid alternative solutions, while step-wise accuracy can bias results by rewarding action overlap rather than task success. To address this, we adopt solver-based task-level evaluation and separately report optimal/feasible solution accuracy. The details are provided in A.1.

4 Experiment

4.1 Experiment Setup

Models. We categorize the evaluated models into open-source VLMs, closed-source VLMs and UMMs. The details are provided in Appendix A.3.
Evaluation Details. We set `max_new_tokens` to 8192. For open-source UMMs, we run each experiment independently three times and report the average performance. Additional implementation details are provided in Appendix A.4.

4.2 Main Results and Analysis

In this section, we analyze the main results across oracle-assisted and self-generation settings. Table 1 reports the performance of VLMs and UMMs under Text-CoT, Oracle-VCoT, Oracle-GtA and Oracle-Aug-VCoT across six tasks. Table 2 presents the results of UMMs under three self-generation protocols, including Self-GtA, Self-VCoT, and Self-VR. Given the generally low performance on harder cases under oracle-assisted settings, we evaluate UMM self-generation protocols only on easy cases. Feasible-solution accuracy under the corresponding oracle-assisted and self-generation settings is reported in Table 8 and Table 9 in Appendix.

Insight 1: *Difficulty-aware evaluation is essential for revealing when visual generation improves visual understanding.*

Previous evaluations of vision generation-assisted visual understanding usually mix samples of different difficulty levels into a single evaluation set, which makes the effect of visual generation difficult to observe: gains on easy cases can be masked by failures on harder ones, leading to an overly coarse conclusion about whether visual generation helps visual understanding. In contrast, as illustrated in Table 1, after dividing samples into easy, medium, and hard levels according to the optimal solution length, we can conduct a more fine-grained analysis of generation-assisted reasoning across different planning difficulties. Under this difficulty-aware setting, we find that for UMMs, visual generation can already bring observable improvements on easy instances, indicating that current unified models are able to use generated visual aids when the visual state is relatively simple and the planning horizon is short. However, as task difficulty increases, these gains become much less stable and often disappear, suggesting that current UMMs still struggle to convert visual generation into reliable understanding gains for more complex long-horizon tasks.

Insight 2: *Visual understanding side is the primary bottleneck for generation-assisted understanding.*

Under the oracle setting, proprietary VLMs obtain substantial gains from the same type of visual aids, indicating that these visual aids are effective when the model has sufficient visual understanding capability. In contrast, most open-source UMMs still achieve near-zero performance under Oracle-GtA, showing that their failures cannot be mainly

Model	Method	Maze			Sokoban			Path			Onet			Parking			Klotski		
		Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Open-source VLMs																			
Qwen2.5-VL-7B-Instruct	Direct	10.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**15.0**	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	**15.0**	0.0	0.0	0.0	0.0	0.0	**15.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Nemotron-nano-12b-v2-vl	Direct	5.0	0.0	0.0	0.0	0.0	0.0	**5.0**	**6.6**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	**10.0**	0.0	0.0	**5.0**	**6.6**	**2.0**	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	15.0	0.0	0.0	**10.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	65.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**35.0**	0.0	0.0	0.0	0.0	0.0	0.0	3.3	0.0	**40.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	15.0	0.0	0.0	**10.0**	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	**75.0**	0.0	0.0	0.0	0.0	0.0
Qwen3-VL-8B-Instruct	Direct	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	5.0	0.0	0.0	10.0	**3.3**	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	20.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	65.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**55.0**	0.0	0.0	0.0	0.0	0.0	**15.0**	0.0	**2.0**	55.0	**6.6**	0.0	25.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	45.0	0.0	0.0	**15.0**	0.0	0.0	0.0	0.0	0.0	**75.0**	3.3	0.0	**70.0**	0.0	0.0	0.0	0.0	0.0
Qwen3.5-9B	Direct	0.0	0.0	0.0	0.0	0.0	0.0	25.0	**13.3**	0.0	5.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	25.0	0.0	0.0	30.0	0.0	0.0	20.0	0.0	0.0	25.0	3.3	0.0	25.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	25.0	0.0	0.0	20.0	0.0	0.0	25.0	0.0	0.0	0.0	0.0	0.0	60.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**90.0**	**33.3**	**6.0**	**55.0**	**10.0**	**6.6**	**60.0**	**13.3**	**2.0**	**95.0**	**43.3**	0.0	**80.0**	**6.6**	0.0	**50.0**	0.0	0.0
	Oracle-Aug-VCoT	50.0	0.0	0.0	20.0	0.0	0.0	15.0	0.0	0.0	85.0	16.6	0.0	70.0	0.0	0.0	0.0	0.0	0.0
Closed-source VLMs																			
Qwen3.6-Plus	Text-CoT	70.0	3.3	0.0	30.0	3.3	2.0	60.0	23.3	4.0	75.0	70.0	46.0	70.0	16.6	0.0	35.0	0.0	0.0
	Oracle-VCoT	50.0	3.3	0.0	50.0	6.6	0.0	60.0	0.0	0.0	80.0	40.0	18.0	75.0	23.3	0.0	35.0	6.6	0.0
	Oracle-GtA	**100.0**	**100.0**	**98.0**	**100.0**	**86.6**	**58.0**	**95.0**	**63.3**	**10.0**	**100.0**	93.3	**58.0**	**100.0**	**90.0**	18.0	**70.0**	3.3	0.0
	Oracle-Aug-VCoT	**100.0**	**100.0**	92.0	**100.0**	26.6	12.0	**95.0**	56.6	2.0	**100.0**	**96.6**	26.0	90.0	16.6	0.0	55.0	**10.0**	0.0
Doubao-Seed-2.0-pro	Text-CoT	80.0	43.3	28.0	55.0	10.0	4.0	**100.0**	66.6	30.0	20.0	30.0	16.0	35.0	10.0	0.0	45.0	10.0	0.0
	Oracle-VCoT	**100.0**	63.3	20.0	80.0	23.3	0.0	85.0	0.0	0.0	20.0	23.3	4.0	95.0	23.3	0.0	10.0	0.0	0.0
	Oracle-GtA	**100.0**	**93.3**	**92.0**	**100.0**	**60.0**	**28.0**	**100.0**	**86.6**	**68.0**	**100.0**	80.0	30.0	**100.0**	**80.0**	**18.0**	**90.0**	**36.6**	0.0
	Oracle-Aug-VCoT	**100.0**	86.6	26.0	**100.0**	33.3	8.0	**100.0**	76.6	38.0	**100.0**	**100.0**	**36.0**	95.0	23.3	0.0	70.0	10.0	**2.0**
Gemini-3.1-Pro-Preview	Text-CoT	75.0	20.0	24.0	60.0	13.3	0.0	60.0	36.6	8.0	90.0	0.0	0.0	30.0	6.6	0.0	55.0	3.3	0.0
	Oracle-VCoT	90.0	23.3	0.0	90.0	13.3	0.0	**100.0**	43.3	2.0	95.0	70.0	8.0	65.0	3.3	0.0	80.0	**13.3**	0.0
	Oracle-GtA	**100.0**	**93.3**	**82.0**	85.0	**60.0**	**32.0**	85.0	**96.6**	**38.0**	**100.0**	10.0	0.0	**85.0**	**26.6**	0.0	90.0	**13.3**	0.0
	Oracle-Aug-VCoT	**100.0**	60.0	22.0	**95.0**	23.3	4.0	80.0	36.6	0.0	**100.0**	**80.0**	4.0	80.0	6.6	0.0	**95.0**	**13.3**	**2.0**
GPT-5.4	Text-CoT	**100.0**	83.3	**74.0**	40.0	3.3	2.0	80.0	46.6	16.0	**100.0**	83.3	32.0	85.0	**90.0**	20.0	**100.0**	46.6	**2.0**
	Oracle-VCoT	**100.0**	26.6	0.0	55.0	3.3	0.0	85.0	6.6	0.0	95.0	76.6	8.0	**95.0**	36.6	0.0	75.0	6.6	0.0
	Oracle-GtA	**100.0**	76.6	72.0	90.0	**60.0**	**58.0**	90.0	**70.0**	**42.0**	**100.0**	**90.0**	**36.0**	**95.0**	76.6	**26.0**	95.0	**60.0**	**2.0**
	Oracle-Aug-VCoT	**100.0**	**96.6**	50.0	**95.0**	50.0	4.0	**100.0**	66.6	14.0	**100.0**	83.3	4.0	**95.0**	36.6	0.0	70.0	6.6	0.0
Unified Models																			
MammothModa2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**15.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
STAR-7B	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	**10.0**	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**10.0**	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	**10.0**	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	**10.0**	0.0	0.0	**15.0**	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	**10.0**	0.0	0.0	0.0	0.0	0.0
OmniGen2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	0.0	0.0	0.0	5.0	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	0.0	0.0	0.0	**10.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Janus-Pro	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	0.0	0.0	0.0	**5.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**5.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**25.0**	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	0.0	0.0	0.0	**5.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LLaDA2.0-Uni	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	**10.0**	0.0	0.0	**10.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	35.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	5.0	0.0	0.0	**10.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	**60.0**	0.0	0.0	0.0	0.0	0.0
Show-o2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	**15.0**	0.0	0.0	**10.0**	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	**70.0**	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	11.6	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	10.0	0.0	0.0	**10.0**	0.0	0.0	1.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BAGEL	Direct	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	10.0	0.0	0.0	20.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**15.0**	0.0	0.0	0.0	0.0	0.0	**20.0**	0.0	0.0	40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	0.0	0.0	0.0	**40.0**	0.0	0.0	5.0	0.0	0.0	**56.6**	0.0	0.0	**30.0**	0.0	0.0	0.0	0.0	0.0
LongCat-Next	Direct	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	10.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-VCoT	13.3	0.0	0.0	6.6	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	**60.0**	**13.3**	0.0	**15.0**	0.0	0.0	**30.0**	0.0	0.0	**60.0**	**26.6**	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-Aug-VCoT	5.0	0.0	0.0	8.3	0.0	0.0	15.0	0.0	0.0	**6.6**	0.0	0.0	**61.6**	0.0	0.0	0.0	0.0	0.0
Nano-Banana	Direct	5.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	15.0	0.0	0.0	20.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	5.0	0.0	0.0
	Oracle-VCoT	15.0	0.0	0.0	**25.0**	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	35.0	0.0	0.0	**25.0**	0.0	0.0	**55.0**	0.0	0.0	**85.0**	**33.3**	0.0	**15.0**	0.0	0.0	**15.0**	0.0	0.0
	Oracle-Aug-VCoT	**55.0**	0.0	0.0	**25.0**	0.0	0.0	40.0	0.0	0.0	55.0	0.0	0.0	5.0	0.0	0.0	5.0	0.0	0.0
Nano Banana 2	Direct	20.0	0.0	0.0	10.0	0.0	0.0	35.0	23.3	8.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	35.0	0.0	0.0	20.0	0.0	0.0	25.0	10.0	2.0	35.0	10.0	0.0	10.0	3.3	0.0	10.0	0.0	0.0
	Oracle-VCoT	20.0	0.0	0.0	30.0	0.0	0.0	30.0	0.0	0.0	25.0	0.0	0.0	0.0	0.0	0.0	25.0	0.0	0.0
	Oracle-GtA	**85.0**	**40.0**	**38.0**	**30.0**	0.0	0.0	**90.0**	**66.6**	**32.0**	**100.0**	**40.0**	**6.0**	**70.0**	**10.0**	0.0	**60.0**	**13.3**	0.0
	Oracle-Aug-VCoT	45.0	13.3	0.0	25.0	0.0	0.0	40.0	13.3	0.0	70.0	20.0	0.0	35.0	0.0	0.0	30.0	0.0	0.0

Table 1: Accuracy (%) on Maze, Sokoban, Path, Onet, Parking, and Klotski under distinct evaluation paradigms.

attributed to poor visual generation. Instead, the oracle results suggest that current UMMs often lack the visual understanding ability needed to effectively leverage the provided visual aids. This conclusion is further supported by the comparison between Self-GtA and Oracle-GtA: among 48 task-level comparisons from 8 open-source UMMs across 6 tasks, only 12 cases fall below their Oracle-GtA upper reference. In other words, for most task-level comparisons, replacing self-generated visual aids with oracle visual aids brings little improvement. Overall, the primary bottleneck in current generation-assisted understanding lies in visual understanding side rather than visual generation side.

Model	Setting	Maze	Sokoban	Path	Onet	Parking	Klotski
MammothModa2	Self-GtA	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VCoT	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VR	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
STAR-7B	Self-GtA	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VCoT	**1.6** (+1.6)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VR	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
OmniGen2	Self-GtA	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VCoT	0.0 (+0.0)	**5.0** (+5.0)	**5.0** (+5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VR	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	**1.6** (+1.6)	0.0 (+0.0)
Janus-Pro	Self-GtA	**5.0** (+5.0)	**5.0** (+5.0)	0.0 (+0.0)	0.0 (+0.0)	**25.0** (+10.0)	0.0 (+0.0)
	Self-VCoT	0.0 (+0.0)	**5.0** (+5.0)	**5.0** (+5.0)	0.0 (+0.0)	0.0 (-15.0)	0.0 (+0.0)
	Self-VR	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	20.0 (+5.0)	0.0 (+0.0)
LLaDA2.0-Uni	Self-GtA	0.0 (-5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VCoT	**10.0** (+5.0)	**10.0** (+10.0)	**5.0** (+5.0)	0.0 (+0.0)	**40.0** (+35.0)	0.0 (+0.0)
	Self-VR	0.0 (-5.0)	5.0 (+5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
Show-o2	Self-GtA	10.0 (+5.0)	0.0 (+0.0)	5.0 (+5.0)	0.0 (+0.0)	10.0 (+10.0)	0.0 (+0.0)
	Self-VCoT	**11.6** (+6.6)	**10.0** (+10.0)	**6.6** (+6.6)	0.0 (+0.0)	**56.6** (+56.6)	0.0 (+0.0)
	Self-VR	0.0 (-5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	10.0 (+10.0)	0.0 (+0.0)
BAGEL	Self-GtA	0.0 (-5.0)	0.0 (+0.0)	**20.0** (+15.0)	0.0 (+0.0)	**5.0** (+0.0)	0.0 (+0.0)
	Self-VCoT	**6.6** (+1.6)	**13.3** (+13.3)	5.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VR	0.0 (-5.0)	0.0 (+0.0)	5.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
LongCat-Next	Self-GtA	**20.0** (+10.0)	**5.0** (+5.0)	**15.0** (+5.0)	**5.0** (+5.0)	**10.0** (-5.0)	0.0 (+0.0)
	Self-VCoT	10.0 (+0.0)	**5.0** (+5.0)	1.6 (-8.4)	0.0 (+0.0)	5.0 (-10.0)	0.0 (+0.0)
	Self-VR	5.0 (-5.0)	0.0 (+0.0)	0.0 (-10.0)	0.0 (+0.0)	5.0 (-10.0)	0.0 (+0.0)
Nano-Banana	Self-GtA	**15.0** (+10.0)	20.0 (+5.0)	**25.0** (+5.0)	0.0 (+0.0)	**15.0** (+10.0)	**10.0** (+5.0)
	Self-VCoT	0.0 (-5.0)	**25.0** (+10.0)	20.0 (+0.0)	0.0 (+0.0)	10.0 (+5.0)	0.0 (-5.0)
	Self-VR	5.0 (+0.0)	10.0 (-5.0)	20.0 (+0.0)	0.0 (+0.0)	10.0 (+5.0)	0.0 (-5.0)
Nano Banana 2	Self-GtA	**55.0** (+20.0)	**25.0** (+5.0)	**50.0** (+25.0)	**70.0** (+35.0)	**40.0** (+30.0)	**30.0** (+20.0)
	Self-VCoT	10.0 (-25.0)	20.0 (+0.0)	30.0 (+5.0)	15.0 (-20.0)	10.0 (+0.0)	5.0 (-5.0)
	Self-VR	15.0 (-20.0)	0.0 (-20.0)	45.0 (+20.0)	25.0 (-10.0)	10.0 (+0.0)	10.0 (+0.0)

Table 2: Accuracy (%) of unified models on Maze, Sokoban, Path, Onet, Parking, and Klotski on easy cases.

4.3 Further Analysis of Self-Generation Paradigms for UMMs

A remaining question is **why UMMs do not exhibit the same paradigm trend as specialist understanding models, either under oracle-assisted evaluation or under self-generation**. This discrepancy is especially pronounced in the comparison between GtA and VCoT on Maze, Sokoban, Path, and Parking, where the paradigm-dependent instability is most evident in our results. While GtA shows relatively stable benefits in stronger understanding models, VCoT exhibits much larger fluctuations in UMMs: it can bring substantial gains on some tasks, but often fails to improve over the Text-CoT or even degrades performance on others.

To investigate the source of this instability, we construct a controlled diagnostic setup that separates visual understanding capability from the fidelity of generated visual aids. For visual understanding, we use the Qwen model series with different parameter scales as visual understanding models, providing a proxy for progressively stronger understanding capability. For visual generation, we vary the fidelity of auxiliary visual aids along three levels. **The first level** uses task-irrelevant visuals, where intermediate images are replaced with Gaussian noise to simulate complete generation failure. **The second level** uses real but incorrect visual aids, including mismatched images from other instances of the same task and outputs from a base generation model, to simulate plausible but inaccurate generation. **The third level** uses high-fidelity task-aligned visuals produced by stronger editing and generation models, such as Nano Banana 2, to approximate reliable visual generation.

Based on Table 3, we observe two key patterns. First, when generation quality is fixed, performance generally improves as visual understanding capability becomes stronger. Second, when visual understanding capability is fixed, improving generation fidelity does not lead to monotonic gains. Instead, performance often drops when moving from task-irrelevant noise to real but incorrect visual aids, and improves substantially only when high-fidelity task-aligned visual aids are provided.

Method	Qwen3.5-0.8B	Qwen3.5-2B	Qwen3.5-4B	Qwen3.5-9B	Qwen3.5-27B	Qwen3.6-plus
Text-CoT	5.0	5.0	20.0	25.0	60.0	70.0
Vision: Gaussian noise						
Self-GtA	0.0 (-5.0)	1.6 (-3.4)	8.3 (-11.7)	8.3 (-16.7)	20.0 (-40.0)	60.0 (-10.0)
Self-VCoT	**6.6** (+1.6)	**6.6** (+1.6)	**11.6** (-8.4)	**13.3** (-11.7)	**35.0** (-25.0)	**61.6** (-8.4)
Self-VR	0.0 (-5.0)	0.0 (-5.0)	6.6 (-13.4)	6.6 (-18.4)	18.3 (-41.7)	53.3 (-16.7)
Vision: Mismatch Image						
Self-GtA	3.3 (-1.7)	1.6 (-3.4)	6.6 (-13.4)	5.0 (-20.0)	10.0 (-50.0)	15.0 (-55.0)
Self-VCoT	**6.6** (+1.6)	**6.6** (+1.6)	**8.3** (-11.7)	**15.0** (-10.0)	**30.0** (-30.0)	**35.0** (-35.0)
Self-VR	0.0 (-5.0)	0.0 (-5.0)	1.6 (-18.4)	3.3 (-21.7)	1.6 (-58.4)	21.6 (-48.4)
Vision generator: Qwen-Image (Wu et al., 2025a)						
Self-GtA	0.0 (-5.0)	0.0 (-5.0)	0.0 (-20.0)	0.0 (-25.0)	1.6 (-58.4)	38.3 (-31.7)
Self-VCoT	**5.0** (+0.0)	**5.0** (+0.0)	**11.6 (-8.4)**	**13.3 (-11.7)**	**35.0 (-25.0)**	**73.3 (+3.3)**
Self-VR	0.0 (-5.0)	0.0 (-5.0)	0.0 (-20.0)	0.0 (-25.0)	1.6 (-58.4)	23.3 (-46.7)
Vision generator: Qwen-Image-Edit (Wu et al., 2025a)						
Self-GtA	0.0 (-5.0)	3.3 (-1.7)	**18.3** (-1.7)	**26.6** (+1.6)	**31.6** (-28.4)	**58.3** (-11.7)
Self-VCoT	**3.3** (-1.7)	**5.0** (+0.0)	10.0 (-10.0)	13.3 (-11.7)	33.3 (-26.7)	33.3 (-36.7)
Self-VR	0.0 (-5.0)	0.0 (-5.0)	5.0 (-15.0)	5.0 (-20.0)	11.6 (-48.4)	25.0 (-45.0)
Vision generator: Nano Banana 2 (Raisinghani, 2026)						
Self-GtA	5.0 (+0.0)	**8.3** (+3.3)	**35.0** (+15.0)	**45.0** (+20.0)	**58.3** (-1.7)	**83.3** (+13.3)
Self-VCoT	**6.6** (+1.6)	3.3 (-1.7)	11.6 (-8.4)	25.0 (+0.0)	33.3 (-26.7)	45.0 (-25.0)
Self-VR	1.6 (-3.4)	0.0 (-5.0)	8.3 (-11.7)	8.3 (-16.7)	13.3 (-46.7)	25.0 (-45.0)
Oracle generation						
Oracle-GtA	0.0 (-5.0)	**40.0** (+35.0)	**85.0** (+65.0)	**90.0** (+65.0)	**100.0** (+40.0)	**100.0** (+30.0)
Oracle-VCoT	**5.0** (+0.0)	10.0 (+5.0)	20.0 (+0.0)	25.0 (+0.0)	45.0 (-15.0)	50.0 (-20.0)

Table 3: Diagnosis of visual understanding scale and generation fidelity across Qwen models on easy cases.

Insight 3: *Generation-assisted understanding follows a three-stage transition from interference, to misleading guidance, and finally to assistance.*

In the **interference stage**, task-irrelevant visuals such as noise contain no task-useful information and act as distractions. In the **misleading stage**, incorrect visual aids become more harmful: although they appear plausible, they encode an incorrect task state, causing the model to rely on false visual evidence. In the **assistance stage**, high-fidelity task-aligned visual aids provide reliable task-relevant information, reduce the burden of visual understanding, and support stable performance gains.

4.4 Fixing Self-VCoT Prior Leakage and Comparing Self-Generation Paradigms

As shown in Tables 3, 10, and 11, Self-VCoT exhibits relatively unstable behavior across different visual conditions. In particular, Self-VCoT can still **obtain unexpectedly high accuracy even when the intermediate visual states are replaced with task-irrelevant Gaussian noise**.

This behavior suggests that the measured gain of Self-VCoT does not necessarily come from generated visual states. In the implementation of Self-VCoT, the optimal number of steps is first obtained from the answer, and the model is then asked to perform interleaved generation for exactly this number of rounds, producing one action and one corresponding image at each round. This prior access to the optimal step count forces the generated answer to have the same length as the reference solution, thereby making the task easier to solve.

To fix this protocol-induced bias, we remove the prior access to the optimal step count from the Self-VCoT protocol. The model still performs interleaved generation one step at a round. After each round is completed, we ask the model to estimate the number of remaining steps. The interaction stops when the model predicts zero remaining steps. The results under this fixed protocol are reported in Table 4. We further apply the same fix to the disentangled diagnostic setting used in Table 3, and report the corrected results in Figure 3.

Model	Setting	Maze	Sokoban	Path	Onet	Parking	Klotski
MammothModa2	Self-VCoT	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
STAR-7B	Self-VCoT	1.6 (+1.6)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
OmniGen2	Self-VCoT	0.0 (+0.0)	5.0 (+5.0)	5.0 (+5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
Janus-Pro	Self-VCoT	0.0 (+0.0)	5.0 (+5.0)	5.0 (+5.0)	0.0 (+0.0)	0.0 (-15.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-15.0)	0.0 (+0.0)
LLaDA2.0-Uni	Self-VCoT	10.0 (+5.0)	10.0 (+10.0)	5.0 (+5.0)	0.0 (+0.0)	40.0 (+35.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (-5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
Show-o2	Self-VCoT	11.6 (+6.6)	10.0 (+10.0)	6.6 (+6.6)	0.0 (+0.0)	56.6 (+56.6)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (-5.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)	0.0 (+0.0)
BAGEL	Self-VCoT	6.6 (+1.6)	13.3 (+13.3)	5.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
	Self-VCoT-fixed	5.0 (+0.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)	0.0 (-5.0)	0.0 (+0.0)
LongCat-Next	Self-VCoT	10.0 (+0.0)	5.0 (+5.0)	1.6 (-8.4)	0.0 (+0.0)	5.0 (-10.0)	0.0 (+0.0)
	Self-VCoT-fixed	0.0 (-10.0)	5.0 (+5.0)	0.0 (-10.0)	0.0 (+0.0)	0.0 (-15.0)	0.0 (+0.0)

Table 4: Accuracy (%) of unified models on Maze, Sokoban, Path, Onet, Parking, and Klotski under Self-VCoT and fixed Self-VCoT evaluation settings on easy cases.

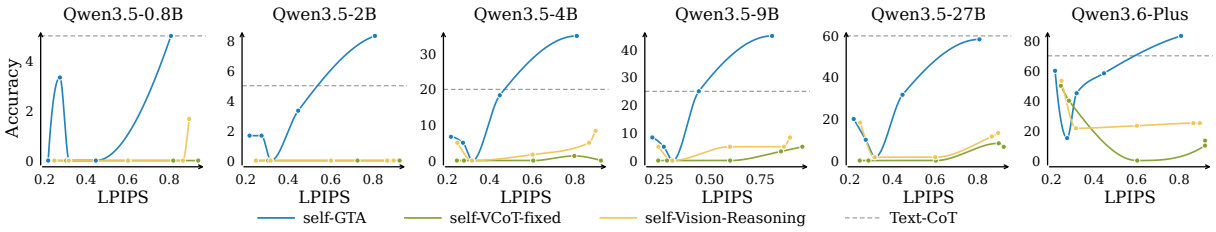

Figure 3: Effect of visual generation fidelity under fixed self-generation protocols across Qwen models. Accuracy is plotted against LPIPS for Self-GtA, Self-VCoT-fixed, Self-Vision-Reasoning, and Text-CoT.

Insight 4: *Effective visual generation should target the bottleneck of visual understanding rather than simply provide more visual reasoning steps.*

As shown in Table 4, performance under Self-VCoT-fixed declines substantially after removing the optimal-step prior. For example, Show-o2's Self-VCoT accuracy on Parking drops from 56.6% to 0.0% under the fixed protocol. Similarly, LLaDA2.0-Uni's Self-VCoT accuracy on Parking drops from 40.0% to 0.0% after the optimal-step prior is removed. Without this optimal-step prior, each intermediate visual state must be sufficiently accurate and useful for the next decision, making the iterative process more susceptible to accumulated errors. In contrast, Self-GtA remains more reliable because it directly targets the visual understanding bottleneck with a single auxiliary visual aid. These results suggest that current UMMs benefit more from targeted auxiliary visual generation than from repeatedly generating intermediate visual states. Overall, effective visual generation should address the visual understanding bottleneck rather than simply add more visual reasoning steps.

5 Conclusion

In this work, we studied whether visual generation can improve visual understanding in UMMs. To this end, we introduce *VGAU-Diag*, a fine-grained evaluation framework that stratifies samples by difficulty, unifies the evaluation of multiple vision generation-assisted reasoning paradigms, and uses oracle-assisted protocols. Our results show that generated visual aids help more reliably on easier instances but become unstable as reasoning complexity increases. Oracle-assisted diagnosis further reveals that the main bottleneck often lies in visual understanding rather than generation, as current UMMs struggle to use even faithful visual aids. We also show that effective visual generation should target visual-understanding bottlenecks rather than add more reasoning steps and identify a three-stage transition from task-irrelevant interference, to misleading plausible guidance, and finally to effective assistance. Together, these findings would be useful to guide the development of better UMMs.

6 Limitations

Although our framework provides a controlled setting for evaluating generation-assisted visual understanding, it is limited to static two-dimensional planning environments. These tasks are well suited for analyzing spatial state parsing, planning horizons, and intermediate visual reasoning, but do not fully capture temporal dynamics, embodied interaction, or continuous three-dimensional scenes. In addition, the evaluation relies on procedurally generated tasks and structured visual states to ensure controllability, unique solutions, and reliable automatic evaluation. This design may reduce visual diversity and underrepresent the noise, ambiguity, and appearance variation commonly observed in natural images. Future work can extend to video-based planning, interactive environments, embodied decision-making, and more realistic visual inputs while preserving verifiable solutions.

7 Acknowledgements

We sincerely thank all the anonymous reviewers for their constructive feedback. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants No. 62372226 and 62272215.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025a. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025b. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

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

Zihui Cheng, Qiguang Chen, Jin Zhang, Hao Fei, Xiaocheng Feng, Wanxiang Che, Min Li, and Libo Qin. 2025. Comt: A novel benchmark for chain of multi-modal thought on large vision-language models. *Preprint*, arXiv:2412.12932.

Ziyun Dai, Xiaoqiang Li, Shaohua Zhang, Yuanchen Wu, and Jide Li. 2025. See different, think better: Visual variations mitigating hallucinations in lvlms. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 3310–3319.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, Guang Shi, and Haoqi Fan. 2025. Emerging properties in unified multimodal pretraining. *Preprint*, arXiv:2505.14683.

Alisa Fortin, Guillaume Vernade, Kat Kampf, and Ammaar Reshi. 2025. Introducing gemini 2.5 flash image, our state-of-the-art image model. https://developers.googleblog.com/en/introducing-gemini-2-5-flash-image/.

Xingyu Fu, Minqian Liu, Zhengyuan Yang, John Richard Corring, Yijuan Lu, Jianwei Yang, Dan Roth, Dinei Florencio, and Cha Zhang. 2025a. Refocus: Visual editing as a chain of thought for structured image understanding. In *Forty-second International Conference on Machine Learning*.

Xingyu Fu, Siyi Liu, Yinuo Xu, Pan Lu, Guangqiuse Hu, Tianbo Yang, Taran Anantasagar, Christopher Shen, Yikai Mao, Yuanzhe Liu, Keyush Shah, Chung Un Lee, Yejin Choi, James Zou, Dan Roth, and Chris Callison-Burch. 2025b. Learning human-perceived fakeness in ai-generated videos via multimodal llms. *Preprint*, arXiv:2509.22646.

Inclusion AI, Tiwei Bie, Haoxing Chen, Tieyuan Chen, Zhenglin Cheng, Long Cui, Kai Gan, Zhicheng Huang, Zhenzhong Lan, Haoquan Li, Jianguo Li, Tao Lin, Qi Qin, Hongjun Wang, Xiaomei Wang, Haoyuan Wu, Yi Xin, and Junbo Zhao. 2026. Llada2.0-uni: Unifying multimodal understanding and generation with diffusion large language model. *Preprint*, arXiv:2604.20796.

Zhehan Kan, Xinghua Jiang, Yanlin Liu, Xiaochen Yang, Zhixiang Wei, Shifeng Liu, Yubo Zhu, Qingmin Liao, Wenming Yang, Xin Li, Yinsong Liu, Deqiang Jiang, and Xing Sun. 2026. Uvu: Improving multimodal understanding via vision-language unified autoregressive paradigm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26230–26239.

Jixuan Leng, Chengsong Huang, Langlin Huang, Bill Yuchen Lin, William W. Cohen, Haohan Wang, and Jiaxin Huang. 2025. Crosswordbench: Evaluating the reasoning capabilities of LLMs and LVLMs with controllable puzzle generation. In *Second Conference on Language Modeling*.

Boyi Li, Yifan Shen, Yuanzhe Liu, Yifan Xu, Jiateng Liu, Xinzhuo Li, Zhengyuan Li, Jingyuan Zhu, Yunhan Zhong, Fangzhou Lan, Jianguo Cao, James M. Rehg, Heng Ji, Ismini Lourentzou, and Xu Cao. 2026. Toward cognitive supersensing in multimodal large language model. *Preprint*, arXiv:2602.01541.

Chengzu Li, Wenshan Wu, Huanyu Zhang, Yan Xia, Shaoguang Mao, Li Dong, Ivan Vulić, and Furu Wei. 2025. Imagine while reasoning in space: Multimodal visualization-of-thought. In *Forty-second International Conference on Machine Learning*.

Yongyuan Liang, Wei Chow, Feng Li, Ziqiao Ma, Xiyao Wang, Jiageng Mao, Jiuhai Chen, Jiatao Gu, Yue Wang, and Furong Huang. 2026. ROVER: Benchmarking reciprocal cross-modal reasoning for omnimodal generation. In *The Fourteenth International Conference on Learning Representations*.

Meituan LongCat Team, Bin Xiao, Chao Wang, Chengjiang Li, Chi Zhang, Chong Peng, Hang Yu, Hao Yang, Haonan Yan, Haoze Sun, Haozhe Zhao, Hong Liu, Hui Su, Jiaqi Zhang, Jiawei Wang, Jing Li, Kefeng Zhang, Manyuan Zhang, Minhao Jing, and 70 others. 2026. Longcat-next: Lexicalizing modalities as discrete tokens. *Preprint*, arXiv:2603.27538.

NVIDIA, :, Amala Sanjay Deshmukh, Kateryna Chumachenko, Tuomas Rintamaki, Matthieu Le, Tyler Poon, Danial Mohseni Taheri, Ilia Karmanov, Guilin Liu, Jarno Seppanen, Guo Chen, Karan Sapra, Zhiding Yu, Adi Renduchintala, Charles Wang, Peter Jin, Arushi Goel, Mike Ranzinger, and 105 others. 2025. Nvidia nemotron nano v2 vl. *Preprint*, arXiv:2511.03929.

OpenAI. 2026. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/.

Jie Qin, Jiancheng Huang, Limeng Qiao, and Lin Ma. 2025. Star: Stacked autoregressive scheme for unified multimodal learning. *Preprint*, arXiv:2512.13752.

QwenTeam. 2026a. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5.

QwenTeam. 2026b. Qwen3.6-plus: Towards real world agents. https://qwen.ai/blog?id=qwen3.6.

Naina Raisinghani. 2026. Nano Banana 2: Combining Pro capabilities with lightning-fast speed. https://blog.google/innovation-and-ai/technology/ai/nano-banana-2/. Google Blog. Accessed: 2026-05-09.

Yawen Shao, Jie Xiao, Kai Zhu, Yu Liu, Hongchen Luo, Xueyang Fu, Yang Cao, Wei Zhai, and Zheng-Jun Zha. 2026. Back on track: Aligning rewards and states for reasoning in diffusion large language models. *Preprint*, arXiv:2606.08501.

Yawen Shao, Jie Xiao, Kai Zhu, Yu Liu, Wei Zhai, Yang Cao, and Zheng-Jun Zha. 2025. Anchoring values in temporal and group dimensions for flow matching model alignment. *arXiv preprint arXiv:2512.12387*.

Tao Shen, Xin Wan, Taicai Chen, Rui Zhang, Junwen Pan, Dawei Lu, Fanding Lei, Zhilin Lu, Yunfei Yang, Chen Cheng, Qi She, Chang Liu, and Zhenbang Sun. 2025. Mammothmoda2: A unified ar-diffusion framework for multimodal understanding and generation. *Preprint*, arXiv:2511.18262.

Yifan Shen, Yuanzhe Liu, Jingyuan Zhu, Xu Cao, Xiaofeng Zhang, Yixiao He, Wenming Ye, James Matthew Rehg, and Ismini Lourentzou. 2026. Fine-grained preference optimization improves spatial reasoning in vlms. *Preprint*, arXiv:2506.21656.

Yang Shi, Yuhao Dong, Yue Ding, Yuran Wang, Xuanyu Zhu, Sheng Zhou, Wenting Liu, Haochen Tian, Rundong Wang, Huanqian Wang, Zuyan Liu, Bohan Zeng, Ruizhe Chen, Qixun Wang, Zhuoran Zhang, Xinlong Chen, Chengzhuo Tong, Bozhou Li, Qiang Liu, and 6 others. 2026. Realunify: Do unified models truly benefit from unification? a comprehensive benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22488–22497.

The Gemini Team. 2026. Gemini 3.1 pro: A smarter model for your most complex tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/.

The Seed Team. 2026. Seed 2.0 official launch. https://seed.bytedance.com/en/blog/seed-2-0-official-launch. ByteDance Seed Blog.

Dianyi Wang, Wei Song, Yikun Wang, Siyuan Wang, Kaicheng Yu, Zhongyu Wei, and Jiaqi Wang. 2025a. Autoregressive semantic visual reconstruction helps vlms understand better. *arXiv preprint arXiv:2506.09040*.

Haochen Wang, Anlin Zheng, Yucheng Zhao, Tiancai Wang, Zheng Ge, Xiangyu Zhang, and Zhaoxiang Zhang. 2025b. Reconstructive visual instruction tuning. In *The Thirteenth International Conference on Learning Representations*.

Zhixiang Wei, Yi Li, Zhehan Kan, Xinghua Jiang, Zuwei Long, Shifeng Liu, Hongze Shen, Wei Liu, Xiaoyu Tan, Haojia Lin, Yubo Zhu, Qianyu Li, Di Yin, Haoyu Cao, Weibo Gu, Xin Li, Yinsong Liu, Deqiang Jiang, Xing Sun, and 22 others. 2026. Youtu-vl: Unleashing visual potential via unified vision-language supervision. *Preprint*, arXiv:2601.19798.

Zimo Wen, Boxiu Li, Wanbo Zhang, Junxiang Lei, Xiaoyu Chen, Yijia Fan, Qi Zhang, Yujiang Wang, Lili Qiu, Bo Li, Ziwei Liu, Caihua Shan, Yifan Yang, and Yifei Shen. 2026. Unig2u-bench: Do unified models advance multimodal understanding? *Preprint*, arXiv:2603.03241.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, and 20 others. 2025a. Qwen-image technical report. *Preprint*, arXiv:2508.02324.

Chenyuan Wu, Jiahao Wang, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, Ziyi Xia, Ze Liu, Chaofan Li, Haoge Deng, Kun Luo, Bo Zhang, Jiajun Zhang, Dong Liu, and 5 others. 2026. Omnigen2: Towards instruction-aligned multimodal generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21964–21975.

Xuecheng Wu, Jiaxing Liu, Danlei Huang, Yifan Wang, Yunyun Shi, Kedi Chen, Junxiao Xue, Yang Liu, Chunlin Chen, Hairong Dong, and Dingkang Yang. 2025b. Vic-bench: Benchmarking visual-interleaved chain-of-thought capability in mllms with free-style intermediate state representations. *Preprint*, arXiv:2505.14404.

Jinheng Xie, Zhenheng Yang, and Mike Zheng Shou. 2025. Show-o2: Improved native unified multimodal models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Wulin Xie, YiFan Zhang, Chaoyou Fu, Yang Shi, Jianshu Zeng, Bingyan Nie, Hongkai Chen, Zhang Zhang, and Liang Wang. 2026. MME-unify: A comprehensive benchmark for unified multimodal understanding and generation models. In *The Fourteenth International Conference on Learning Representations*.

Tianjiao Yu, Xinzhuo Li, Yifan Shen, Yuanzhe Liu, and Ismini Lourentzou. 2025. Core3d: Collaborative reasoning as a foundation for 3d intelligence. *arXiv preprint arXiv:2512.12768*.

Tianjiao Yu, Xinzhuo Li, Yifan Shen, Onkar Susladkar, Yuanzhe Liu, Xiaona Zhou, and Ismini Lourentzou. 2026. Elsa3d: Elastic semantic anchoring for unified 3d understanding and generation. *Preprint*, arXiv:2607.06565.

Jana Ricarda Zeller, Thaddäus Wiedemer, Fanfei Li, Thomas Klein, Prasanna Mayilvahanan, Matthias Bethge, Felix A. Wichmann, Ryan Cotterell, and Wieland Brendel. 2026. Mentisoculi: Revealing the limits of reasoning with mental imagery. In *Forty-third International Conference on Machine Learning*.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. 2025. LMMs-eval: Reality check on the evaluation of large multimodal models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 881–916, Albuquerque, New Mexico. Association for Computational Linguistics.

Yubo Zhu, Dongrui Liu, Zecheng Lin, Wei Tong, Sheng Zhong, and Jing Shao. 2025. The llm already knows: Estimating llm-perceived question difficulty via hidden representations. *Preprint*, arXiv:2509.12886.

Kai Zou, Ziqi Huang, Yuhao Dong, Shulin Tian, Dian Zheng, Hongbo Liu, Jingwen He, Bin Liu, Yu Qiao, and Ziwei Liu. 2026. Uni-MMMU: A massive multi-discipline multimodal unified benchmark. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 908–924, San Diego, California, United States. Association for Computational Linguistics.

A Appendix

A.1 Tasks Details

A.1.1 Maze

Task Definition. Given an image of a 10×10 maze, the model must move the agent from the blue start marker to the green goal marker. Black cells are walls, off-white cells are walkable, and the green goal cell is also passable. The only legal actions are `up`, `down`, `left`, and `right`, each moving the agent by exactly one grid cell. The model outputs a final action sequence as a JSON array wrapped in `<ANSWER_JSON>...</ANSWER_JSON>`. In the step-by-step setting, the model additionally generates one state image after each move, while preserving the original maze layout and moving only the blue agent marker.

Construction. We procedurally generate 100 mazes with controlled difficulty. Each maze is rendered on a 10×10 map with a fixed color palette: dark cells for walls, off-white cells for paths, a blue square for the agent, and a green hollow square for the goal. To create an instance, we first sample a simple path of a target length using randomized depth-first search. We then grow the set of walkable cells by repeatedly adding leaf cells, i.e., cells adjacent to exactly one existing walkable cell. This keeps the walkable-cell graph acyclic, guaranteeing a unique path between any two connected cells. We verify each sample by BFS and by explicitly counting simple paths from start to goal, keeping only mazes whose unique shortest path length falls into the desired bucket. The dataset contains easy cases with path lengths 1–6, medium cases with path lengths 7–17, and hard cases with path lengths 18–30.

Evaluation. We parse the model output by extracting the final JSON action list from `<ANSWER_JSON>...<ANSWER_JSON>` and normalizing each action token. Starting from the annotated start cell, we replay the predicted moves on the ground-truth walkable-cell set. A prediction is invalid if it contains an unknown action, moves outside the grid, or enters a wall. We report `reach_any` when the replayed trajectory is legal and ends at the goal, regardless of length. We report `reach_optimal` when the trajectory reaches the goal and its length equals the BFS shortest-path length.

A.1.2 Sokoban

Task Definition. Given an image of a 10×10 single-box Sokoban puzzle, the model must control the player to push the wooden box onto the green target cell. Brick cells are walls, sand cells are walkable, the small person marks the player start, and the wooden crate marks the box. The only legal actions are `up`, `down`, `left`, and `right`, each moving the player by exactly one grid cell. If the player moves into the box, the box is pushed one cell in the same direction, and the push is legal only when the destination cell is not a wall. The model outputs a final action sequence as a JSON array wrapped in `<ANSWER_JSON>...</ANSWER_JSON>`. In the step-by-step setting, the model additionally generates one state image after each move, while preserving the original layout and updating only the player and box positions.

Construction. We procedurally generate 100 Sokoban puzzles with controlled difficulty. Each puzzle is rendered on a 10×10 map with a fixed visual style for walls, floor, player, box, and target. Each instance contains exactly one player, one movable box, and one target. We sample candidate wall layouts and object placements, then solve each candidate with a programmatic Sokoban solver to obtain the shortest valid action sequence. Unsolvable puzzles and puzzles whose optimal solution length falls outside the desired range are discarded. The dataset contains easy cases with optimal lengths 1–8, medium cases with optimal lengths 10–16, and hard cases with optimal lengths 20–39.

Evaluation. We parse the model output by extracting the final JSON action list from `<ANSWER_JSON>...</ANSWER_JSON>` and normalizing each action token. Starting from the annotated player and box positions, we replay the predicted moves on the ground-truth grid. A prediction is invalid if it contains an unknown action, moves the player into a wall, or attempts to push the box into a wall. We report `reach_any` when the replayed trajectory is legal and the box ends on the target cell, regardless of length. We report `reach_optimal` when the puzzle is solved and the predicted action sequence length equals the annotated optimal solution length.

A.1.3 Path

Task Definition. Given an image of a 10×10 map, the model must identify which labeled points are passed along the valid simple path from the orange triangle to the blue triangle. Black cells

are walls, light-gray cells are walkable, the orange triangle marks the start, and the blue triangle marks the destination. Five walkable cells are labeled with letters A, B, C, D, and E. Legal movement is restricted to up, down, left, and right, one cell per step, without entering black cells. The path is required to be simple, with no revisits or backtracking. The model outputs only the passed labels in path order, wrapped in <answer>...</answer>. In the step-by-step setting, the model additionally generates one state image after each move, while preserving the original layout and moving only the current-position marker.

Construction. We procedurally generate 100 path-following instances with controlled difficulty. Each instance is rendered on a 10×10 map with a fixed visual style for walls, walkable cells, start, destination, and letter labels. We sample a valid simple path between the start and destination, place five labeled points on walkable cells, and record the subset of labels that lie on the path in traversal order as the ground-truth answer. Candidate instances are filtered to ensure that the path does not cross walls, uses only orthogonal moves, and satisfies the target path-length range. The dataset contains easy cases with path lengths 2–5, medium cases with path lengths 8–14, and hard cases with path lengths 18–25.

Evaluation. We parse the model output by extracting the label sequence from <answer>...</answer> and normalizing all labels to uppercase letters among A–E. A prediction is considered correct only if the extracted label sequence exactly matches the ground-truth labels in both content and order. We report accuracy separately over easy, medium, and hard subsets, as well as over the full dataset.

A.1.4 Parking

Task Definition. Given an image of an 8×8 parking-exit puzzle, the model must slide vehicles to move the red car R to the exit on the right boundary. Each vehicle has a fixed orientation, either horizontal or vertical, and can only move along that orientation. In one move, exactly one vehicle slides by one grid cell. Vehicles cannot overlap or leave the board. Success is achieved when the rightmost cell of the red car reaches the right boundary on the exit row. The model outputs a final move sequence as a JSON array of objects wrapped in <ANSWER_JSON>...</ANSWER_JSON>, where each object specifies the vehicle id and direction, e.g., {"car":"A","direction":"up"}. In the step-by-step setting, the model additionally generates one state image after each move, while preserving the original layout and updating only the moved vehicle.

Construction. We procedurally generate 100 parking-exit puzzles with controlled difficulty. Each puzzle is rendered on an 8×8 board with a fixed visual style for the board, exit, and labeled vehicles. The red car R is always horizontal and aligned with the exit row. We sample vehicle sets with different numbers of cars, orientations, and lengths, initialize solvable states, and then scramble them through legal vehicle moves. For each candidate puzzle, we run BFS to compute the shortest solution and discard instances that are unsolvable or outside the target step range. Additional filtering enforces increasing structural complexity, such as more blocking vehicles in the red car's exit corridor and more non-red vehicles that must be moved. The dataset contains easy cases with optimal lengths 1–5, medium cases with optimal lengths 8–16, and hard cases with optimal lengths 18–34.

Evaluation. We parse the model output by extracting the JSON move list from <ANSWER_JSON>...</ANSWER_JSON> and normalizing vehicle ids and directions. Starting from the annotated vehicle positions, we replay the predicted moves on the ground-truth board. A prediction is invalid if it names an unknown vehicle, moves a vehicle against its orientation, causes overlap, or moves any vehicle outside the board. We report reach_solved when the replayed trajectory is legal and the red car reaches the exit. We report reach_optimal when the puzzle is solved and the predicted move sequence length equals the annotated shortest solution length.

A.1.5 Onet

Task Definition. Given an image of a 6×6 Onet board, the model must decide a sequence of pair removals that clears the entire board. Each occupied cell contains a fruit icon, and empty cells can be used as connection space. In each step, the model may remove two identical fruit icons if they can be connected by a path composed of horizontal and vertical segments with at most two turns. The connection path must not pass through any other fruit. The model outputs the full removal sequence as a JSON array of coordinate pairs wrapped in <answer_json>...</answer_json>, where each step has the form [[r1,c1],[r2,c2]]. In the step-

by-step setting, the model additionally generates one board image after each pair removal, while preserving the original layout and clearing only the removed fruit cells.

Construction. We construct 100 solvable Onet puzzles with controlled difficulty. Each puzzle is rendered on a 6×6 board with a fixed image-only style and fruit icons as tile types. Each fruit type appears exactly twice, forming removable pairs, while remaining cells may be empty. For each instance, we store the initial board, the fruit legend, and one valid clearing sequence. Candidate boards are filtered so that all pairs can be removed under the Onet rule: each removal must connect identical fruits through empty cells using at most two turns. The dataset contains easy cases with 4–6 removal steps, medium cases with 7–10 removal steps, and hard cases with 12–16 removal steps.

Evaluation. We parse the model output by extracting the JSON removal sequence from `<answer_json>...</answer_json>`. Starting from the annotated board, we replay each predicted pair removal. A prediction is invalid if either coordinate is out of bounds, the two cells are empty, the two fruits are not identical, or the pair cannot be connected by a valid path with at most two turns through empty cells. After each valid step, the two fruit cells are cleared. We report accuracy when the entire predicted sequence is legal and the board is fully cleared.

A.1.6 Klotski

Task Definition. Given an image of a 6×6 Klotski board, the model must move the 2×2 target block `T` to the goal position. The board contains one target block, multiple 1×2 vertical blocks, multiple 2×1 horizontal blocks, several 1×1 single blocks, and exactly two empty cells. In each step, the model slides exactly one block by one cell in one of `UP`, `DOWN`, `LEFT`, or `RIGHT`. Blocks cannot rotate, jump, move diagonally, leave the board, or overlap other blocks. A block is identified in the output by its current top-left coordinate at the time of the move. The goal is achieved when the top-left coordinate of the target block reaches $(4, 2)$. The model outputs the full move sequence as a JSON object with a `solution` field, where each step has the form `{"from":[r,c],"direction":"UP"}`. In the step-by-step setting, the model additionally generates one board image after each move, while preserving the original layout and updating only the moved block.

Construction. We construct 100 Klotski puzzles with controlled difficulty. Each puzzle is rendered on a 6×6 board with a fixed color palette for empty cells, the target block, vertical blocks, horizontal blocks, and single blocks. All instances use the same block inventory: one 2×2 target block, eight vertical blocks, three horizontal blocks, and eight single blocks, leaving exactly two empty cells. We sample solvable states by starting from goal configurations and walking backward through legal one-cell block slides, then reverse the sampled trajectory to obtain a valid solution from the initial state. Duplicate states and already-solved states are discarded. The dataset contains easy cases with optimal lengths 1–4, medium cases with optimal lengths 6–10, and hard cases with optimal lengths 15–20.

Evaluation. We parse the model output by extracting the JSON object containing the `solution` list. Starting from the annotated initial state, we replay each predicted move. A prediction is invalid if the referenced top-left coordinate does not correspond to a movable block in the current state, the direction is invalid, the move leaves the board, or any destination cell is occupied by another block. We report `solved` when the replayed trajectory is legal and the target block reaches $(4, 2)$. We report `optimal` when the puzzle is solved and the predicted move sequence length equals the annotated optimal solution length.

A.1.7 Difficulty validation.

Optimal solution length serves as a controllable proxy for difficulty stratification rather than the sole definition of difficulty. To illustrate that this split captures broader structural complexity, we take Maze as an example. We compute instance-level and ground-truth path-level statistics across difficulty levels. Walkable cells measure the size of the reachable state space, branch nodes are walkable cells with at least three walkable neighbors, and corridor nodes are walkable cells with exactly two walkable neighbors. As shown in Tables 5 and 6, all metrics increase consistently from easy to hard cases, showing that the length-based split reflects structural complexity rather than merely longer output sequences.

A.2 Protocol Details

A.2.1 Text-CoT

Maze. In this setting, the prompt asks the model to first reason in text about the maze and planned

Average metric	Easy	Medium	Hard
Walkable cells	20.30	30.10	39.12
Branch nodes	4.25	5.83	6.98
Corridor nodes	9.35	16.07	22.76

Table 5: Structural statistics of Maze instances.

Average path metric	Easy	Medium	Hard
Path nodes	4.10	13.27	24.64
Branch nodes on path	1.55	3.70	6.00
Corridor nodes on path	2.00	8.97	18.00

Table 6: Ground-truth solution-path statistics.

move sequence. The prompt is detailed in Table 12.

Sokoban. In this setting, the prompt asks the model to first reason in text about the player position, box position, target cell, and planned move sequence. The prompt is detailed in Table 13.

Path. In this setting, the prompt asks the model to first reason in text about the start point, destination point, valid simple path, and labeled points passed along the route. The prompt is detailed in Table 14.

Parking. In this setting, the prompt asks the model to first reason in text about the vehicle layout, blocking vehicles, exit row, and planned slide sequence for moving the red car to the exit. The prompt is detailed in Table 15.

Onet. In this setting, the prompt asks the model to first reason in text about matching fruit pairs, valid connection paths, and the planned pair-removal sequence for clearing the board. The prompt is detailed in Table 16.

Klotski. In this setting, the prompt asks the model to first reason in text about the block layout, empty cells, target block position, and planned sequence of legal block slides. The prompt is detailed in Table 17.

A.2.2 Self-GtA

Maze. In this setting, the model is first prompted to generate an auxiliary annotated maze image by adding red grid-boundary lines and coordinate labels while preserving the original maze layout, walls, start marker, and goal marker. The model then uses the generated auxiliary image to reason about the maze and output the final move sequence. The prompt is detailed in Table 18.

Sokoban. In this setting, the model is first prompted to generate an auxiliary annotated Sokoban image by adding red grid-boundary lines and coordinate labels while preserving the original walls, floor, player, box, and target. The model then uses the generated auxiliary image to reason about box-pushing dynamics and output the final move sequence. The prompt is detailed in Table 19.

Path. In this setting, the model is first prompted to generate an auxiliary annotated path image by adding red grid-boundary lines and coordinate labels while preserving the original walls, start marker, destination marker, and labeled points. The model then uses the generated auxiliary image to reason about the valid simple path and output the labels passed along the route. The prompt is detailed in Table 21.

Parking. In this setting, the model is first prompted to generate an auxiliary annotated parking image by adding red grid-boundary lines and coordinate labels while preserving the original board, exit, vehicle positions, labels, and colors. The model then uses the generated auxiliary image to reason about the blocking vehicles and output the final vehicle-slide sequence. The prompt is detailed in Table 20.

Onet. In this setting, the model is first prompted to generate an auxiliary annotated Onet image by adding red grid-boundary lines and coordinate labels while preserving the original board geometry and fruit icons. The model then uses the generated auxiliary image to reason about valid matching pairs and output the final pair-removal sequence. The prompt is detailed in Table 22.

Klotski. In this setting, the model is first prompted to generate an auxiliary annotated Klotski image by adding red grid-boundary lines and coordinate labels while preserving the original board geometry, block positions, labels, and colors. The model then uses the generated auxiliary image to reason about legal block slides and output the final solution sequence. The prompt is detailed in Table 23.

A.2.3 self-VCoT

Maze. In this setting, the model solves the maze iteratively by predicting one move at each step. After each predicted move, the model generates the corresponding next-state image, where only the agent marker should move while the maze layout remains unchanged. The process continues until

the model reaches the goal and outputs the final move sequence. The prompt is detailed in Table 24.

Sokoban. In this setting, the model solves the Sokoban puzzle iteratively by predicting one player move at each step. After each predicted move, the model generates the corresponding next-state image, updating the player and box positions while preserving the walls, floor, and target. The process continues until the box reaches the target and the model outputs the final move sequence. The prompt is detailed in Table 25.

Path. In this setting, the model traces the path iteratively by predicting one movement step at a time. After each predicted move, the model generates the corresponding next-state image, where only the current-position marker should move while the map and labels remain unchanged. The process continues until the destination is reached, and the model outputs the labels passed along the path. The prompt is detailed in Table 26.

Parking. In this setting, the model solves the parking-exit puzzle iteratively by predicting one vehicle slide at each step. After each predicted move, the model generates the corresponding next-state image, updating only the moved vehicle while preserving the board, exit, and other vehicles. The process continues until the red car reaches the exit and the model outputs the final vehicle-slide sequence. The prompt is detailed in Table 28.

Onet. In this setting, the model solves the Onet puzzle iteratively by predicting one removable fruit pair at each step. After each predicted removal, the model generates the corresponding next-state image, where only the selected pair is cleared and the remaining fruit icons stay unchanged. The process continues until the board is cleared and the model outputs the final pair-removal sequence. The prompt is detailed in Table 27.

Klotski. In this setting, the model solves the Klotski puzzle iteratively by predicting one block slide at each step. After each predicted move, the model generates the corresponding next-state image, updating only the moved block while preserving the board geometry and all other blocks. The process continues until the target block reaches the goal position and the model outputs the final solution sequence. The prompt is detailed in Table 29.

A.2.4 self-VR

Maze. In this setting, the model is prompted to generate a visual reasoning image that overlays a continuous red path from the start marker to the goal while preserving the original maze. The prompt is detailed in Table 30.

Sokoban. In this setting, the model is prompted to generate a visual reasoning image that highlights a feasible player trajectory for pushing the box onto the target while preserving the original puzzle state. The prompt is detailed in Table 31.

Path. In this setting, the model is prompted to generate a visual reasoning image that overlays a continuous red path from the orange start triangle to the blue destination triangle while preserving the walls and labeled points. The prompt is detailed in Table 32.

Parking. In this setting, the model is prompted to generate a visual reasoning image that adds concise visual hints for a feasible vehicle-moving strategy while preserving the original parking board and vehicle positions. The prompt is detailed in Table 33.

Onet. In this setting, the model is prompted to generate a visual reasoning image that highlights valid connection paths between removable matching fruit pairs while preserving the original Onet board. The prompt is detailed in Table 34.

Klotski. In this setting, the model is prompted to generate a visual reasoning image that adds visual hints for a feasible sequence of legal block slides while preserving the original Klotski board state. The prompt is detailed in Table 35.

A.2.5 oracle-GtA

Maze. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary maze image by overlaying red grid-boundary lines on the 10×10 maze and placing coordinate labels on each cell.

Sokoban. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary Sokoban image by overlaying red grid-boundary lines on the board and placing coordinate labels on each cell.

Path. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary path image by overlaying red grid-boundary lines on the 10×10 map and placing coordinate labels on each cell.

Parking. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary parking image by overlaying red grid-boundary lines on the board and placing coordinate labels on each cell.

Onet. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary Onet image by overlaying red grid-boundary lines on the 6×6 board and placing coordinate labels on each cell.

Klotski. Compared with *Self-GtA*, this setting does not ask the model to generate the auxiliary annotated image. Instead, we directly provide an auxiliary Klotski image by overlaying red dashed grid-boundary lines on the 6×6 board and placing coordinate labels on each cell.

A.2.6 oracle-VCoT

Maze. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted move. Instead, after the model predicts one move, we directly provide the updated maze state image by moving the agent marker according to that move while keeping the maze layout unchanged.

Sokoban. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted move. Instead, after the model predicts one player move, we directly provide the updated Sokoban state image by applying the move to the player and box positions while keeping the walls, floor, and target unchanged.

Path. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted move. Instead, after the model predicts one movement step, we directly provide the updated path state image by moving the current-position marker while keeping the map and labels unchanged.

Parking. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted move. Instead, after the model predicts one vehicle slide, we directly provide the updated parking state image by moving the selected vehicle while keeping the board, exit, and other vehicles unchanged.

Onet. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted pair removal. Instead, after the model predicts one removable pair, we directly provide the updated Onet state image by clearing the selected cells while keeping the remaining board unchanged.

Klotski. Compared with *self-VCoT*, this setting does not ask the model to generate the next-state image after each predicted move. Instead, after the model predicts one block slide, we directly provide the updated Klotski state image by moving the selected block while keeping the board and other blocks unchanged.

A.2.7 oracle-Aug-VCoT

Maze. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary maze image before the iterative solving process. The auxiliary image is created by overlaying red grid-boundary lines on the 10×10 maze and placing coordinate labels on each cell. After each predicted move, the updated state image is also provided with the same grid and coordinate annotations.

Sokoban. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary Sokoban image before the iterative solving process. The auxiliary image is created by overlaying red grid-boundary lines on the board and placing coordinate labels on each cell. After each predicted move, the updated state image is also provided with the same grid and coordinate annotations.

Path. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary path image before the iterative solving process. The auxiliary image is created by overlaying red grid-boundary lines on the 10×10 map and placing coordinate labels on each cell. After each predicted move, the updated state image is also provided with the same grid and coordinate annotations.

Parking. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary parking image before the iterative solving process. The

auxiliary image is created by overlaying red grid-boundary lines on the board and placing coordinate labels on each cell. After each predicted move, the updated state image is also provided with the same grid and coordinate annotations.

Onet. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary Onet image before the iterative solving process. The auxiliary image is created by overlaying red grid-boundary lines on the 6×6 board and placing coordinate labels on each cell. After each predicted pair removal, the updated state image is also provided with the same grid and coordinate annotations.

Klotski. Compared with *oracle-VCoT*, this setting additionally provides an auxiliary Klotski image before the iterative solving process. The auxiliary image is created by overlaying red dashed grid-boundary lines on the 6×6 board and placing coordinate labels on each cell. After each predicted move, the updated state image is also provided with the same grid and coordinate annotations.

A.3 Model Details

The evaluated models are categorized into three groups:

1) **Open-source VLMs:** We include widely adopted open-weights models such as Nemotron-nano-12b-v2-vl (NVIDIA et al., 2025), Qwen2.5-VL-7B-Instruct (Bai et al., 2025b), Qwen3-VL-8B-Instruct (Bai et al., 2025a), and Qwen3.5-9B (QwenTeam, 2026a). These models are specifically selected because their parameter scales are highly comparable to those of the majority of the evaluated open-source UMMs, serving as a scale-matched reference to investigate the behavioral differences between standard visual-language models and unified architectures.

2) **Closed-source VLMs:** We evaluate state-of-the-art (SOTA) proprietary models, including Qwen3.6-Plus (QwenTeam, 2026b), Doubao-Seed-2.0-pro (The Seed Team, 2026), Gemini-3.1-Pro-Preview (The Gemini Team, 2026), and GPT-5.4 (OpenAI, 2026). We include these models for two primary reasons: (1) to investigate whether the SOTA models can still achieve additional performance gains through generative visual augmentations; and (2) to establish an empirical upper bound for complex visual reasoning, serving as a critical reference point to guide the future evolution of UMMs.

Task	Oracle-VCoT	Oracle-VCoT-last-image
Maze	25.0	20.0
Sokoban	20.0	20.0
Path	25.0	20.0

Table 7: Oracle-VCoT accuracy with all images or only the last image on Qwen3.5-9B.

3) **Unified Multimodal Models (UMMs):** We evaluate recent UMMs, including Mammoth-Moda2 (Shen et al., 2025), STAR-7B (Qin et al., 2025), OmniGen2 (Wu et al., 2026), Janus-Pro (Chen et al., 2025), BAGEL (Deng et al., 2025), Show-o2 (Xie et al., 2025), LongCat-Next (Meituan LongCat Team et al., 2026), LLaDA2.0-Uni (Inclusion AI et al., 2026). Consistent with prior work, we treat Nano-Banana (Fortin et al., 2025) and Nano Banana 2 (Raisinghani, 2026) as a unified model for comparison.

A.4 Evaluation Details

We set `max_new_tokens` to 8192 and use greedy decoding with temperature set to 0 for all evaluated models. For UMMs, we run each experiment independently three times and report the average performance. Our evaluation implementation is based on lmms-eval (Zhang et al., 2025).

Additionally, due to the current capability limitations of UMMs, they require manual iterative calls to achieve interleaved text-image generation. For models that support only single-image input, such as Show-o2 and LLaDA2.0-Uni, we use the output image from the previous step as the visual input for the next step. This evaluation setting is consistent with previous works (Zou et al., 2026; Wen et al., 2026). To assess the impact of the single-image-input, we evaluate Qwen3.5-9B under the same setting, where only the most recently generated image is provided at each answering step. As shown in Table 7, using only the last image does not substantially change Oracle-VCoT performance on these tasks.

Model	Method	Maze			Sokoban			Parking			Klotski		
		Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Open-source VLMs													
Qwen2.5-VL-7B-Instruct	Direct	10.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
Nemotron-nano-12b-v2-vl	Direct	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	10.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	35.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen3-VL-8B-Instruct	Direct	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	5.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	55.0	0.0	0.0	0.0	0.0	0.0	25.0	0.0	0.0	0.0	0.0	0.0
Qwen3.5-9B	Direct	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	30.0	0.0	0.0	30.0	0.0	0.0	25.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	95.0	33.3	6.0	55.0	10.0	6.6	80.0	6.6	0.0	65.0	0.0	0.0
Closed-source VLMs													
Qwen3.6-Plus	Text-CoT	70.0	3.3	0.0	30.0	3.3	2.0	95.0	73.3	14.0	60.0	0.0	0.0
	Oracle-GtA	100.0	100.0	98.0	100.0	86.6	58.0	100.0	90.0	18.0	100.0	3.3	0.0
Doubao-Seed-2.0-pro	Text-CoT	80.0	43.3	28.0	55.0	20.0	6.0	35.0	10.0	0.0	45.0	10.0	0.0
	Oracle-GtA	100.0	93.3	92.0	100.0	100.0	52.0	100.0	80.0	18.0	90.0	36.6	0.0
Gemini-3.1-Pro-Preview	Text-CoT	75.0	20.0	24.0	60.0	23.3	2.0	30.0	6.6	0.0	55.0	3.3	0.0
	Oracle-GtA	100.0	93.3	82.0	85.0	70.0	42.0	85.0	26.6	0.0	90.0	13.3	0.0
GPT-5.4	Text-CoT	100.0	83.3	74.0	40.0	10.0	4.0	85.0	90.0	20.0	100.0	46.6	2.0
	Oracle-GtA	100.0	76.6	72.0	90.0	70.0	60.0	95.0	76.6	26.0	95.0	60.0	2.0
Unified Models													
MammothModa2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
STAR-7B	Direct	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	10.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
OmniGen2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Janus-Pro	Direct	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	0.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	5.0	0.0	0.0	5.0	0.0	0.0	25.0	0.0	0.0	0.0	0.0	0.0
LLaDA2.0-Uni	Direct	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	0.0	0.0	0.0	5.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
Show-o2	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	15.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
BAGEL	Direct	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	5.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	15.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
LongCat-Next	Direct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	10.0	0.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
	Oracle-GtA	60.0	13.3	0.0	15.0	0.0	0.0	10.0	0.0	0.0	30.0	0.0	0.0
Nano-Banana	Direct	20.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	40.0	3.3	0.0	20.0	0.0	0.0	15.0	3.3	0.0	10.0	0.0	0.0
	Oracle-GtA	85.0	40.0	38.0	30.0	0.0	0.0	90.0	43.3	4.0	60.0	16.6	0.0
Nano Banana 2	Direct	20.0	0.0	0.0	10.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0
	Text-CoT	35.0	0.0	0.0	20.0	0.0	0.0	15.0	3.3	0.0	10.0	0.0	0.0
	Oracle-GtA	85.0	40.0	38.0	30.0	0.0	0.0	70.0	10.0	0.0	60.0	13.3	0.0

Table 8: Feasible-solution accuracy (%) on Maze, Sokoban, Parking, and Klotski under oracle settings.

Model	Setting	Maze	Sokoban	Parking	Klotski
MammothModa2	Self-GtA	0.0	0.0	0.0	0.0
	Self-VR	0.0	0.0	0.0	0.0
STAR-7B	Self-GtA	0.0	0.0	0.0	0.0
	Self-VR	0.0	0.0	0.0	0.0
OmniGen2	Self-GtA	0.0	0.0	0.0	0.0
	Self-VR	0.0	0.0	10.0	0.0
Janus-Pro	Self-GtA	5.0	5.0	25.0	0.0
	Self-VR	0.0	0.0	20.0	0.0
LLaDA2.0-Uni	Self-GtA	0.0	0.0	0.0	0.0
	Self-VR	0.0	5.0	0.0	0.0
Show-o2	Self-GtA	10.0	0.0	10.0	0.0
	Self-VR	0.0	0.0	10.0	0.0
BAGEL	Self-GtA	0.0	0.0	5.0	0.0
	Self-VR	0.0	0.0	0.0	0.0
LongCat-Next	Self-GtA	20.0	5.0	10.0	0.0
	Self-VR	5.0	0.0	5.0	0.0
Nano-Banana	Self-GtA	15.0	20.0	15.0	10.0
	Self-VR	5.0	10.0	10.0	0.0
Nano Banana 2	Self-GtA	55.0	25.0	40.0	30.0
	Self-VR	15.0	0.0	10.0	10.0

Table 9: Feasible-solution accuracy (%) of unified models on Maze, Sokoban, Parking, and Klotski under two self-generation evaluation settings.

Table 10: Results on Sokoban.

Method	Qwen3.5-0.8B	Qwen3.5-2B	Qwen3.5-4B	Qwen3.5-9B	Qwen3.5-27B	Qwen3.6-plus
Text-CoT	0.0	10.0	25.0	30.0	15.0	30.0
Vision: Gaussian noise						
Self-GtA	0.0 (+0.0)	**8.3** (-1.7)	11.6 (-13.4)	**21.6** (-8.4)	20.0 (+5.0)	23.3 (-6.7)
Self-VCoT	**6.6** (+6.6)	**8.3** (-1.7)	**15.0** (-10.0)	15.0 (-15.0)	**21.6** (+6.6)	**41.6** (+11.6)
Self-VR	0.0 (+0.0)	3.3 (-6.7)	11.6 (-13.4)	16.6 (-13.4)	18.3 (+3.3)	21.6 (-8.4)
Vision: Mismatch Image						
Self-GtA	0.0 (+0.0)	6.6 (-3.4)	10.0 (-15.0)	20.0 (-10.0)	31.6 (+16.6)	11.6 (-18.4)
Self-VCoT	**11.6** (+11.6)	**6.6** (-3.4)	**10.0** (-15.0)	**13.3** (-16.7)	**33.3** (+18.3)	**26.6** (-3.4)
Self-VR	0.0 (+0.0)	5.0 (-5.0)	10.0 (-15.0)	10.0 (-20.0)	15.0 (+0.0)	18.3 (-11.7)
Vision generator: Qwen-Image						
Self-GtA	0.0 (+0.0)	0.0 (-10.0)	8.3 (-16.7)	20.0 (-10.0)	23.3 (+8.3)	20.0 (-10.0)
Self-VCoT	**6.6 (+6.6)**	**6.6 (-3.4)**	**10.0 (-15.0)**	**11.6 (-18.4)**	**23.3 (+8.3)**	**45.0 (+15.0)**
Self-VR	0.0 (+0.0)	0.0 (-10.0)	6.6 (-18.4)	8.3 (-21.7)	15.0 (+0.0)	16.6 (-13.4)
Vision generator: Qwen-Image-Edit						
Self-GtA	0.0 (+0.0)	1.6 (-8.4)	**20.0** (-5.0)	**38.3** (+8.3)	**40.0** (+25.0)	**78.3** (+48.3)
Self-VCoT	**3.3** (+3.3)	**8.3** (-1.7)	8.3 (-16.7)	8.3 (-21.7)	25.0 (+10.0)	31.6 (+1.6)
Self-VR	0.0 (+0.0)	1.6 (-8.4)	6.6 (-18.4)	6.6 (-23.4)	10.0 (-5.0)	23.3 (-6.7)
Vision generator: Nano Banana 2						
Self-GtA	0.0 (+0.0)	**1.6** (-8.4)	**28.3** (+3.3)	**50.0** (+20.0)	**48.3** (+33.3)	**83.3** (+53.3)
Self-VCoT	**8.3** (+8.3)	10.0 (+0.0)	15.0 (-10.0)	18.3 (-11.7)	33.3 (+18.3)	36.6 (+6.6)
Self-VR	0.0 (+0.0)	6.6 (-3.4)	6.6 (-18.4)	16.6 (-13.4)	15.0 (+0.0)	21.6 (-8.4)
Oracle generation						
Oracle-GtA	0.0 (+0.0)	**10.0** (+0.0)	**55.0** (+30.0)	**55.0** (+25.0)	**75.0** (+60.0)	**100.0** (+70.0)
Oracle-VCoT	**10.0** (+10.0)	5.0 (-5.0)	20.0 (-5.0)	20.0 (-10.0)	15.0 (+0.0)	50.0 (+20.0)

Table 11: Results on Parking.

Method	Qwen3.5-0.8B	Qwen3.5-2B	Qwen3.5-4B	Qwen3.5-9B	Qwen3.5-27B	Qwen3.6-plus
Text-CoT	5.0	5.0	25.0	25.0	40.0	70.0
Vision: Gaussian noise						
Self-GtA	**3.3** (-1.7)	3.3 (-1.7)	**20.0** (-5.0)	**33.3** (+8.3)	6.6 (-33.4)	**88.3** (+18.3)
Self-VCoT	**3.3** (-1.7)	**26.6** (+21.6)	8.3 (-16.7)	23.3 (-1.7)	10.0 (-30.0)	75.0 (+5.0)
Self-VR	0.0 (-5.0)	3.3 (-1.7)	10.0 (-15.0)	16.6 (-8.4)	**30.0** (-10.0)	56.6 (-13.4)
Vision: Mismatch Image						
Self-GtA	0.0 (-5.0)	5.0 (+0.0)	**20.0** (-5.0)	**41.6** (+16.6)	0.0 (-40.0)	30.0 (-40.0)
Self-VCoT	**3.3** (-1.7)	**28.3** (+23.3)	8.3 (-16.7)	6.6 (-18.4)	**10.0** (-30.0)	23.3 (-46.7)
Self-VR	**3.3** (-1.7)	1.6 (-3.4)	5.0 (-20.0)	8.3 (-16.7)	3.3 (-36.7)	**51.6** (-18.4)
Vision generator: Qwen-Image						
Self-GtA	**1.6** (-3.4)	10.0 (+5.0)	**15.0** (-10.0)	21.6 (-3.4)	**16.6** (-23.4)	63.3 (-6.7)
Self-VCoT	**1.6** (-3.4)	**25.0** (+20.0)	10.0 (-15.0)	**23.3** (-1.7)	10.0 (-30.0)	**86.6** (+16.6)
Self-VR	0.0 (-5.0)	1.6 (-3.4)	3.3 (-21.7)	5.0 (-20.0)	3.3 (-36.7)	43.3 (-26.7)
Vision generator: Qwen-Image-Edit						
Self-GtA	1.6 (-3.4)	8.3 (+3.3)	**51.6** (+26.6)	**51.6** (+26.6)	**71.6** (+31.6)	**90.0** (+20.0)
Self-VCoT	0.0 (-5.0)	**35.0** (+30.0)	5.0 (-20.0)	23.3 (-1.7)	10.0 (-30.0)	73.3 (+3.3)
Self-VR	**5.0** (+0.0)	3.3 (-1.7)	5.0 (-20.0)	3.3 (-21.7)	13.3 (-26.7)	38.3 (-31.7)
Vision generator: Nano Banana 2						
Self-GtA	**8.3** (+3.3)	16.6 (+11.6)	**50.0** (+25.0)	**80.0** (+55.0)	**93.3** (+53.3)	**100.0** (+30.0)
Self-VCoT	0.0 (-5.0)	**31.6** (+26.6)	10.0 (-15.0)	15.0 (-10.0)	10.0 (-30.0)	81.6 (+11.6)
Self-VR	1.6 (-3.4)	3.3 (-1.7)	5.0 (-20.0)	16.6 (-8.4)	20.0 (-20.0)	53.3 (-16.7)
Oracle generation						
Oracle-GtA	**20.0** (+15.0)	25.0 (+20.0)	**90.0** (+65.0)	**80.0** (+55.0)	**100.0** (+60.0)	**100.0** (+30.0)
Oracle-VCoT	0.0 (-5.0)	**55.0** (+50.0)	5.0 (-20.0)	60.0 (+35.0)	5.0 (-35.0)	75.0 (+5.0)

You are a precise maze solver.

SEMANTICS
- Black squares: walls (impassable)
- White squares: path (walkable)
- Blue dot: start (the agent)
- The green square marks the goal cell. It is passable, not a wall, and success is achieved only when the agent moves onto the cell occupied by the green square.
- Legal moves: up, down, left, right only. One cell per step; no diagonals, no jumps; never cross walls.

OUTPUT FORMAT
1) Describe your reasoning briefly: identify the start (agent) and goal, explain the planned move sequence, and confirm never enters black-wall cells.
2) Output exactly one final move list as a JSON array of lowercase strings, wrapped as: `<ANSWER_JSON>["move1","move2",...]</ANSWER_JSON>`

Table 12: prompt for maze under Text-CoT setting.

You are solving a single-box Sokoban puzzle.

Legend:
- Brick wall: blocked
- Sand floor: walkable
- Small person: player start
- Wooden crate with X: box
- Green X: target

Rules:
1) Allowed moves: up, down, left, right (one cell per step).
2) If the player moves into the box, the box is pushed by one cell in the same direction.
3) A push is valid only if the box destination is not a wall.
4) No diagonal movement.

Goal:
Push the box onto the target cell.

OUTPUT FORMAT
1) Think briefly about legality of moves and box position updates.
2) Then output strictly:
`<ANSWER_JSON>["move1","move2",...]</ANSWER_JSON>`

Table 13: prompt for sokoban under Text-CoT setting.

In the image there are five labeled points: A, B, C, D, E.
From the orange triangle (start) to the blue triangle (destination), determine which labeled points lie on valid simple path without entering black grids.

SEMANTICS
- Black squares: walls (impassable)
- Light-gray cells are walkable.
- Orange triangle: path start.
- Blue triangle: destination.
- A/B/C/D/E: a label point, which is walkable.
- Legal moves: up, down, left, right only. One cell per step; no diagonals, no jumps; never cross walls.
- Follow the valid simple path from start to destination: no backtracking and no revisits.

Task:
From the orange triangle (start) to the blue triangle (destination), determine which labeled points lie on the valid simple path

OUTPUT FORMAT
1) Describe your reasoning briefly: identify the start and destination, explain the planned move sequence, and confirm never enters black-wall cells.
2) Answer with only the passed letters in path order, using this format:
`<answer>point1,point2,...</answer>`

Table 14: prompt for path under Text-CoT setting.

You are solving a parking-exit puzzle on a {grid_size}× {grid_size} board.

Rules:
1) Vehicle orientations are fixed (horizontal or vertical).
2) In one move, exactly one vehicle slides by one cell along its orientation.
3) No overlap and no leaving the board.
4) Success is achieved when the rightmost cell of the red car "R" reaches the right boundary on row index {exit_row - 1} (indices are 0-based).

OUTPUT FORMAT
1) Describe your reasoning briefly.
2) Answer using this format:
`<ANSWER_JSON>[{"car":"A","direction":"up"}, ...]</ANSWER_JSON>`

Table 15: prompt for parking under Text-CoT setting.

You are solving an Onet puzzle on a 6x6 board.

Coordinate system
- Use 0-based coordinates.
- (r, c) means row r, column c.
- (0,0) is the top-left cell.
- r increases downward, c increases to the right.

Goal:
- Decide whether the whole board can be fully cleared.
- In each step, you may remove a pair of identical fruits if they can be linked.
- A valid link uses only horizontal/vertical segments and can have at most 2 turns.
- The path must not pass through any other fruit.
OUTPUT FORMAT:

1) Brief reasoning.
2) Final answer on a separate line:
<answer_json>[[[r1,c1],[r2,c2]],[[r3,c3],[r4,c4]],...]</answer_json>

Table 16: prompt for onet under Text-CoT setting.

You are an expert Klotski puzzle solver.

Coordinate system
- Use 0-based coordinates.
- (r, c) means row r, column c.
- (0,0) is the top-left cell.
- r increases downward, c increases to the right.

Board
- Grid size: width=6, height=6.
- There are exactly two empty cells at any time.
- Every non-empty cell belongs to exactly one block from the listed block shapes.

Block shapes
  - One 2x2 target block.
  - Multiple 1x2 vertical blocks.
  - Multiple 2x1 horizontal blocks.
  - Multiple 1x1 single blocks.

Block reference in actions
- In the action output, a block is identified by its CURRENT TOP-LEFT coordinate at the time of that move.

Legal move definition
1. Each step slides ONE block by exactly ONE cell in UP, DOWN, LEFT, RIGHT.
2. No rotation, no diagonal move, no jump.
3. A move is legal only if EVERY destination cell of that block is currently empty.
  - Let current_cells be the cells currently occupied by the moving block.
  - Let destination_cells be the cells occupied after translating the block by one cell in the chosen direction.
  - A move is legal iff every cell in (destination_cells - current_cells) is empty before the move.
  - For 1x2 / 2x1 blocks, required empty destination cells can be 1 or 2 depending on move direction.
  - For 2x2 blocks, required empty destination cells are 2.
4. Cells occupied by OTHER blocks are never passable.

Goal
- Make the 2x2 target block's TOP-LEFT coordinate reach (4,2).

OUTPUT FORMAT
Before outputting the final sequence, briefly describe your reasoning for the solution.
Finally, output the solution in the following format:
{
  "solution": [
    "from": [<r1>, <c1>], "direction": "<DIRECTION>",
    "from": [<r2>, <c2>], "direction": "<DIRECTION>"
  ]
}

Table 17: prompt for klotski under Text-CoT setting.

[stage1: generate]

Edit the input maze image to create exactly one auxiliary annotated image.

Maze semantics:
- Black squares are walls and must remain unchanged.
- White squares are walkable cells.
- The blue dot is the start cell and must remain unchanged.
- The green square is the goal cell, is walkable, and must remain unchanged.

Editing requirements:
1. Preserve the original maze geometry, cell layout, wall positions, start marker, goal marker, and original colors.
2. Overlay thin red grid-boundary lines that divide the maze image into 10 equal parts along the horizontal axis and 10 equal parts along the vertical axis; draw 9 evenly spaced vertical red lines at 10%, 20%, ..., 90% of the image width and 9 evenly spaced horizontal red lines at 10%, 20%, ..., 90% of the image height, forming a 10×10 partition aligned with the maze cells. As a placement reference, the green square should lie exactly inside one grid cell, with red grid boundaries flush with its four sides.
3. Label every cell (black squares, white squares, blue dot and green square) at its center with its coordinate in black text, using the format (r,c). The top-left cell is (0,0). The first row is labeled left-to-right as (0,0), (0,1), (0,2), (0,3), ... until the last cell in that row. The second row is labeled left-to-right as (1,0), (1,1), (1,2), (1,3), ... In general, row index r increases by 1 when moving downward, and column index c increases by 1 when moving rightward.
4. Do not add any arrows, path lines, legend, explanation text, title, or extra decorations.

[stage2: answer]

{question}

Table 18: prompt for maze under Self-GtA setting.

[stage1: generate]

Edit the input Sokoban image to create exactly one auxiliary annotated image.

Sokoban semantics:
- Brick walls remain unchanged.
- Sand floor is walkable.
- The player marker, box marker, and green target marker must remain unchanged.

Editing requirements:
1. Preserve original colors, geometry, walls, player, box, and target.
2. Overlay thin red grid-boundary lines aligned with the puzzle cells (one line per row/column boundary).
3. Label every non-wall cell at its center with its coordinate in black text using the format (r,c). Top-left cell is (0,0); r increases downward, c increases rightward.
4. Do not add arrows, path lines, captions, titles, legend, or any other decoration.

[stage2: answer]

{question}

Table 19: prompt for sokoban under Self-GtA setting.

[stage1: generate]

Edit the input parking-exit puzzle image to create exactly one auxiliary annotated image.

Puzzle semantics:
- The board is a {grid_size}x{grid_size} grid.
- Each vehicle has fixed orientation and length.
- Legal moves slide one vehicle by one cell along its orientation.
- The red car "R" must exit to the right boundary on row index {exit_row - 1} (0-based).

Editing requirements:
1. Preserve the original board geometry, vehicle positions, labels, and colors exactly.
2. Overlay thin red grid-boundary lines aligned to the board cell boundaries.
3. Label every cell at its center with black text in the format (r,c). The top-left cell is (0,0); r increases downward and c increases rightward.
4. Do not add arrows, path lines, legends, captions, titles, or any extra decorations.
[stage2: answer]

{question}

Table 20: prompt for parking under Self-GtA setting.

[stage1: generate]

Edit the input image to create exactly one auxiliary annotated image.

Image semantics:
- Black squares are walls and must remain unchanged.
- Light-gray cells are walkable and must remain unchanged.
- The orange triangle is the start cell and must remain unchanged.
- The blue triangle is the destination cell and must remain unchanged.
- The letters A, B, C, D, E mark walkable label points and must remain unchanged.

Editing requirements:
1. Preserve the original geometry, cell layout, wall positions, start marker, destination marker, A/B/C/D/E label markers, and original colors.
2. Overlay thin red grid-boundary lines that divide the image into 10 equal parts along the horizontal axis and 10 equal parts along the vertical axis; draw 9 evenly spaced vertical red lines at 10%, 20%, ..., 90% of the image width and 9 evenly spaced horizontal red lines at 10%, 20%, ..., 90% of the image height, forming a 10x10 partition aligned with the cells.
3. Label every cell (black squares, light-gray cells, orange triangle cell, blue triangle cell, A/B/C/D/E label cells) at its center with its coordinate in black text, using the format (r,c). The top-left cell is (0,0). The first row is labeled left-to-right as (0,0), (0,1), (0,2), ..., until the last cell in that row. In general, row index r increases by 1 when moving downward, and column index c increases by 1 when moving rightward.
4. Do not add any arrows, path lines, legend, explanation text, title, or extra decorations.
[stage2: answer]

{question}

Table 21: prompt for path under Self-GtA setting.

[stage1: generate]

Edit the input Onet puzzle image to create exactly one auxiliary annotated image.

Onet puzzle semantics:
- The board is a 6x6 grid of cells.
- Each cell contains a fruit icon or is empty.
- Identical fruit icons can be removed as a pair if they can be connected by a path with at most 2 turns using only horizontal/vertical segments that do not pass through other fruits.

Editing requirements:
1. Preserve the original board geometry, cell layout, fruit icons, and original colors exactly.
2. Overlay thin red grid-boundary lines that divide the image into a 6x6 grid: draw 5 evenly spaced vertical red lines and 5 evenly spaced horizontal red lines, aligned with cell boundaries.
3. Label every cell at its center with its coordinate in black text, using the format (r,c). The top-left cell is (0,0). Row index r increases downward; column index c increases rightward.
4. Do not add any arrows, path lines, legend, explanation text, title, or extra decorations.

[stage2: answer]

{question}

Table 22: prompt for onet under Self-GtA setting.

[stage1: generate]

Edit the input Klotski image to create exactly one auxiliary annotated image.

Klotski semantics:
- Grid is 6x6.
- Blocks keep shape and orientation; every move slides one block by one cell.
- Target is to move the 2x2 block T so its top-left reaches (4,2).

Editing requirements:
1. Preserve original image dimensions, board geometry, block positions, labels, and colors exactly. The input image may omit visible cell lines; infer the implied 6x6 tiling from the layout and keep each block's cell extent unchanged (e.g. 1x2 stays 1x2; do not shrink, split, or merge blocks).
2. Overlay thin red dashed grid-boundary lines aligned to the 6x6 cell boundaries.
3. Label every cell center with coordinate "(r,c)" in black text.
4. Do not add arrows, legends, captions, titles, or extra decorations.

[stage2: answer]

{question}

Table 23: prompt for klotski under Self-GtA setting.

You are a precise maze solver.

SEMANTICS
- Black squares: walls (impassable)
- White squares: path (walkable)
- Blue dot: start (the agent)
- The green square marks the goal cell. It is passable, not a wall.
- Legal moves: up, down, left, right only. One cell per step.

OUTPUT FORMAT (STRICT)

1) MULTI-IMAGE MODE : generate a SEQUENCE OF SEPARATE IMAGES, one per move:
- Each output image must depict the maze state AFTER applying exactly one legal move.
- Do NOT include the initial (pre-move) state.
- Keep palette/layout/scale identical to the input; only the blue dot moves.
- The number of returned images MUST equal the number of moves in the final answer (see step 2). - Absolutely FORBIDDEN: any collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no arrows, captions, or overlays; no GIFs/animations/video.

2) After all step images, emit EXACTLY ONE LINE containing ONLY the final move list as a JSON array of lowercase strings, wrapped as: `<ANSWER_JSON>["right","down","left"]</ANSWER_JSON>`

NO EXTRAS
- No tools, no OCR, no explanations, and no text other than the single `<ANSWER_JSON>` line.
- Do not restate the instructions or the condition.

REMINDERS
- Decide the full path first, then emit the image sequence (one image per move), then the single `<ANSWER_JSON>` line.
- One move per image; images must be separate files/parts, not stitched together in any way.

Table 24: prompt for maze under Self-VCoT setting.

You are solving a single-box Sokoban puzzle.

Legend:
- Brick wall: blocked
- Sand floor: walkable
- Small person: player start
- Wooden crate with X: box
- Green X: target

Rules:
1) Allowed moves: up, down, left, right (one cell per step).
2) If the player moves into the box, the box is pushed by one cell in the same direction.
3) A push is valid only if the box destination is not a wall.
4) No diagonal movement.

Goal:
Push the box onto the target cell.

OUTPUT FORMAT (STRICT)

1) MULTI-IMAGE MODE: generate a SEQUENCE OF SEPARATE IMAGES, one per move:
- Each output image must depict the maze state AFTER applying exactly one legal move.
- Do NOT include the initial (pre-move) state.
- Keep palette/layout/scale identical to the input; only the blue dot moves.
- The number of returned images MUST equal the number of moves in the final answer (see step 2).
- Absolutely FORBIDDEN: any collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no arrows, captions, or overlays; no GIFs/animations/video.

2) After all step images, emit EXACTLY ONE LINE containing ONLY the final move list as a JSON array of lowercase strings, wrapped as: `<ANSWER_JSON>["right","down","left"]</ANSWER_JSON>`

NO EXTRAS
- No tools, no OCR, no explanations, and no text other than the single `<ANSWER_JSON></ANSWER_JSON>` line.
- Do not restate the instructions or the condition.

REMINDERS
- Decide the full path first, then emit the image sequence (one image per move), then the single `<ANSWER_JSON>` line.
- One move per image; images must be separate files/parts, not stitched together in any way.

Table 25: prompt for sokoban under Self-VCoT setting.

In the image there are five labeled points: A, B, C, D, E. From the orange triangle (start) to the blue triangle (destination), determine which labeled points lie on valid simple path without entering black grids.

SEMANTICS
- Black squares: walls (impassable)
- Light-gray cells are walkable.
- Orange triangle: start/current position.
- Blue triangle: destination.
- A/B/C/D/E: labeled walkable points.
- Legal moves: up, down, left, right only. One cell per step.
- Follow the valid simple path from start to destination: no backtracking and no revisits.

OUTPUT FORMAT (STRICT)
You need to start from the start point, plan the path to the destination step by step, and output the passed labeled points.
1) MULTI-IMAGE MODE: generate a SEQUENCE OF SEPARATE IMAGES, one per move:
- Each output image must depict the state AFTER applying exactly one legal move.
- Do NOT include the initial (pre-move) state.
- Keep palette/layout/scale identical to the input; only the current position marker moves.
- The number of returned images MUST equal the number of moves in the final path.
- Absolutely FORBIDDEN: any collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no arrows, captions, overlays, GIFs, animations, or video.

2) STEP DECISION FORMAT
for EACH step decision, output exactly one JSON object: `<STEP_JSON>{"move":"up|down|left|right","hit_labels":["A"]}</STEP_JSON>`
- hit_labels must be labels newly passed on THIS move only. - Use uppercase letters among A/B/C/D/E only.
- If no new label is passed this step, output an empty list [].

3) FINAL LINE — after all step images/decisions, emit exactly one line: `<answer>point1,point2,...</answer>`

NO EXTRAS
- No extra explanation text.

Table 26: prompt for path under Self-VCoT setting.

You are solving an Onet puzzle on a 6x6 board.
Coordinate system
- Use 0-based coordinates.
- (r, c) means row r, column c.
- (0,0) is the top-left cell.
- r increases downward, c increases to the right.

Goal:
- Decide whether the whole board can be fully cleared.
- In each step, remove exactly one pair of identical fruits if they can be linked.
- A valid link uses only horizontal/vertical segments and can have at most 2 turns.
- The path must not pass through any other fruit.

OUTPUT FORMAT (STRICT)

1) MULTI-IMAGE MODE: generate a SEQUENCE OF SEPARATE IMAGES, one per move:
- Each output image must depict the board state AFTER applying exactly one pair removal.
- Do NOT include the initial (pre-move) state.
- Keep palette/layout/scale identical to the input; empty cells must be clearly indicated (e.g., cleared/blank).
- The number of returned images MUST equal the number of moves in the final answer (see step 2).
- Absolutely FORBIDDEN: any collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no arrows, captions, or overlays; no GIFs/animations/video.

2) After all step images, emit EXACTLY ONE LINE containing ONLY the final move list as a JSON array of coordinate pairs, wrapped as: `<answer_json>[[[r1,c1],[r2,c2]],[[r3,c3],[r4,c4]],...]</answer_json>`

NO EXTRAS
- No tools, no OCR, no explanations, and no text other than the single `<ANSWER_JSON>` line.
- Do not restate the instructions or the condition.

REMINDERS
- Determine the full sequence of moves first, then emit the image sequence (one image per move), then the single `<ANSWER_JSON>` line.
- One move per image; images must be separate files/parts, not stitched together in any way.

Table 27: prompt for onet under Self-VCoT setting.

You are solving a parking-exit puzzle on {grid_size}x{grid_size} board.

Rules:
1) Vehicle orientations are fixed (horizontal or vertical).
2) In one move, exactly one vehicle slides by one cell along its orientation.
3) No overlap and no leaving the board.
4) Success is achieved when the rightmost cell of the red car "R" reaches the right boundary on row index exit_row - 1 (indices are 0-based).

OUTPUT FORMAT (STRICT)

1) MULTI-IMAGE MODE: generate a SEQUENCE OF SEPARATE IMAGES, one per move:
- Each output image must depict the board state AFTER applying exactly one legal move.
- Do NOT include the initial (pre-move) state.
- Keep palette/layout/scale identical to the input; only the moved vehicle changes position.
- The number of returned images MUST equal the number of moves in the final answer (see step 2).
- Absolutely FORBIDDEN: any collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no arrows, captions, or overlays; no GIFs/animations/video.

2) After all step images, emit EXACTLY ONE LINE containing ONLY the final move list as a JSON array of move objects, wrapped as: <ANSWER_JSON>[{"car":"A","direction":"up"},{"car":"R","direction":"right"}]</ANSWER_JSON>

NO EXTRAS
- No tools, no OCR, no explanations, and no text other than the single <ANSWER_JSON> line.
- Do not restate the instructions or the condition.

REMINDERS
- Decide the full move sequence first, then emit the image sequence (one image per move), then the single <ANSWER_JSON> line.
- One move per image; images must be separate files/parts, not stitched together in any way.

Table 28: prompt for parking under Self-VCoT setting.

You are an expert Klotski puzzle solver.

Coordinate system
- Use 0-based coordinates.
- (r, c) means row r, column c.
- (0,0) is the top-left cell.
- r increases downward, c increases to the right.

Board
- Grid size: width=6, height=6.
- There are exactly two empty cells at any time.
- Every non-empty cell belongs to exactly one block from the listed block shapes.

Block shapes
- One 2x2 target block.
- Multiple 1x2 vertical blocks.
- Multiple 2x1 horizontal blocks.
- Multiple 1x1 single blocks.

Block reference in actions
- In the action output, a block is identified by its CURRENT TOP-LEFT coordinate at the time of that move.

Legal move definition
1. Each step slides ONE block by exactly ONE cell in UP, DOWN, LEFT, RIGHT.
2. No rotation, no diagonal move, no jump.
3. A move is legal only if EVERY destination cell of that block is currently empty.
   - Let current_cells be the cells currently occupied by the moving block.
   - Let destination_cells be the cells occupied after translating the block by one cell in the chosen direction.
   - A move is legal iff every cell in (destination_cells - current_cells) is empty before the move.
   - For 1x2 / 2x1 blocks, required empty destination cells can be 1 or 2 depending on move direction.
   - For 2x2 blocks, required empty destination cells are 2.
4. Cells occupied by OTHER blocks are never passable.

Goal
- Make the 2x2 target block's TOP-LEFT coordinate reach (4,2).

OUTPUT FORMAT (STRICT)

1) MULTI-IMAGE MODE: generate a SEQUENCE OF SEPARATE IMAGES, one per move:
   - Each output image must depict the board state AFTER applying exactly one legal move.
   - Do NOT include the initial (pre-move) state.
   - Keep board layout/scale/colors consistent with the input board.
   - The number of returned images MUST equal the number of moves in your final answer.
   - Absolutely FORBIDDEN: collage/montage/spritesheet/grid/multi-panel/side-by-side/stacked images; no GIF/animation/video.

2) For each planning turn, output exactly one move as:
`<STEP_JSON>{"from":[r,c],"direction":"UP|DOWN|LEFT|RIGHT"}</STEP_JSON>`

3) After reaching the goal, output exactly one final JSON object:
```
{
   "solution": [
      "from": [<r1>, <c1>], "direction": "<DIRECTION>",
      "from": [<r2>, <c2>], "direction": "<DIRECTION>"
   ]
}
```

Table 29: prompt for klotski under Self-VCoT setting.

Edit the input maze image to create exactly one auxiliary annotated image that highlights a feasible solution path.

Maze semantics:
- Black squares are walls and must remain unchanged.
- White squares are walkable cells.
- The blue dot is the start cell and must remain unchanged.
- The green square is the goal cell, is walkable, and must remain unchanged.

Editing requirements:

1. Preserve the original maze geometry, cell layout, wall positions, start marker, goal marker, and original colors.

2. Overlay a single continuous RED polyline that traces a legal path from the blue start cell to the green goal cell:
   - The polyline must follow walkable (white) cells only and must NEVER cross or touch any black wall cell.
   - Movement is restricted to up / down / left / right between cell centers; no diagonals, no jumps.
   - The polyline starts exactly at the center of the blue start cell and ends exactly at the center of the green goal cell.
   - Use a clean, solid bright red stroke of moderate thickness (roughly 10-15 percent of a cell width) so the path is clearly visible against the maze background.

3. Do not draw arrows, dots along the way, grid lines, coordinate labels, legend, captions, titles, or any other decoration. Only the single red polyline should be added.

4. Do not modify the colors or positions of the start marker, goal marker, walls, or walkable cells in any other way.

Table 30: prompt for maze under Self-VR setting.

Edit the input Sokoban image to create exactly one auxiliary annotated image that highlights a feasible player path that pushes the box onto the target.

Sokoban semantics:
- Brick walls remain unchanged.
- Sand floor is walkable.
- The player, box, and green target must remain unchanged.
- Pushes happen when the player steps into the box; the box moves one cell in the same direction; pushes into walls are illegal.

Editing requirements:

1. Preserve original colors, geometry, walls, player, box, and target.

2. Overlay a single continuous RED polyline that traces the player's trajectory from the player start cell, ending exactly at the cell from which the box gets pushed onto the green target. The polyline:
   - Follows non-wall cells only and never crosses a wall.
   - Moves only orthogonally (up / down / left / right) between cell centers.
   - Uses a clean solid red stroke (10-15 percent of a cell width) so it is clearly visible.

3. Do NOT draw the box's trajectory, arrows, dots, grid lines, coordinate labels, captions, titles, or any other decoration. Only the player's path polyline is added.

Table 31: prompt for sokoban under Self-VR setting.

Edit the input image to create exactly one auxiliary annotated image that highlights a feasible solution path.

Image semantics:
- Black squares are walls and must remain unchanged.
- Light-gray cells are walkable.
- The orange triangle is the start cell and must remain unchanged.
- The blue triangle is the destination cell and must remain unchanged.
- The letters A, B, C, D, E mark walkable label points and must remain unchanged.

Editing requirements:

1. Preserve the original geometry, cell layout, wall positions, start marker, destination marker, A/B/C/D/E label markers, and original colors.

2. Overlay a single continuous RED polyline that traces a legal simple path from the orange-triangle start cell to the blue-triangle destination cell: - The polyline must follow walkable (non-wall) cells only and must NEVER cross or touch any black wall cell.
- Movement is restricted to up / down / left / right between cell centers; no diagonals, no jumps.
- The polyline starts exactly at the center of the orange-triangle start cell and ends exactly at the center of the blue-triangle destination cell.
- The path must be a simple path: no revisits and no backtracking.
- Use a clean, solid bright red stroke of moderate thickness (roughly 10-15 percent of a cell width) so the path is clearly visible against the background.

3. Do not draw arrows, dots along the way, grid lines, coordinate labels, legend, captions, titles, or any other decoration. Only the single red polyline should be added.

4. Do not modify the colors or positions of the start marker, destination marker, A/B/C/D/E label markers, walls, or walkable cells in any other way.

Table 32: prompt for path under Self-VR setting.

Edit the input parking-exit puzzle image to create exactly one auxiliary annotated image that highlights a feasible move strategy for the red car.

Puzzle semantics:

- The board is a {grid_size}x{grid_size} grid.
- Each vehicle has fixed orientation and length.
- Legal moves slide one vehicle by one cell along its orientation.
- The red car "R" must exit to the right boundary on row index exit_row - 1 (0-based).

Editing requirements:
1. Preserve the original board geometry, vehicle positions, labels, and colors exactly.
2. Add visual hints that indicate a feasible strategy (for example, numbered move markers or directional cues near vehicles) while keeping all hints on valid cells only.
3. Ensure the hints correspond to legal single-step moves and do not imply illegal orientation changes or overlaps.
4. Do not alter car locations in the image; only overlay hints. Do not add unrelated decorations.

Table 33: prompt for parking under Self-VR setting.

Edit the input Onet puzzle image to create exactly one auxiliary annotated image that highlights all matching pairs with their connection paths.

Onet puzzle semantics:

- The board is a 6x6 grid of cells.
- Each cell contains a fruit icon or is empty.
- Identical fruit icons can be removed as a pair if they can be connected by a path with at most 2 turns using only horizontal/vertical segments that do not pass through other fruits.

Editing requirements:
1. Preserve the original board geometry, cell layout, fruit icons, and original colors exactly.
2. For each pair of identical fruits that can be validly connected (forming a complete clearing solution), draw a solid colored polyline from the center of one fruit cell to the center of the other, following the valid connection path (horizontal/vertical segments, at most 2 turns, not passing through other fruits). Use a distinct bright color per pair so all pairs are visually distinguishable.

3. Use a clean stroke of moderate thickness (roughly 10-15 percent of a cell width) so paths are clearly visible against the background.
4. Do not add grid lines, coordinate labels, arrows, legend, captions, titles, or any other decoration. Only the colored polylines should be added. 5. Do not modify the fruit icons, empty cells, or board layout in any other way.

Table 34: prompt for onet under Self-VR setting.

Edit the input Klotski image to create exactly one auxiliary annotated image that highlights one feasible solving strategy.

Klotski semantics:
- Grid is 6x6.
- Blocks keep shape and orientation; every move slides one block by one cell.
- Target is to move the 2x2 block T so its top-left reaches (4,2).

Editing requirements:
1. Preserve original image dimensions, board geometry, block positions, labels, and colors exactly. The input image may omit visible cell lines; infer the implied 6x6 tiling from the layout and keep each block's cell extent unchanged (e.g. 1x2 stays 1x2; do not shrink, split, or merge blocks).
2. Add concise visual hints indicating a feasible move strategy (for example, ordered markers and local directional cues near moved blocks).
3. Hints must correspond to legal one-cell slides and must not imply illegal overlaps or rotations.
4. Do not actually move blocks in the edited image; only overlay hints.
5. Do not add unrelated decorations or long text.

Table 35: prompt for klotski under Self-VR setting.