TEST-TIME HINTING FOR BLACK-BOX VISION-LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling (TTS) methods have proven highly effective for LLMs, yet their application to vision-language models (VLMs) remains relatively underexplored. Existing VLM TTS methods largely require open-weight model access or expensive repeated sampling, and are evaluated primarily on multimodal mathematical and scientific reasoning benchmarks rather than natural-image understanding. In this paper, we propose *Test-Time Hinting*, a method that improves VLM performance with a single call to the target VLM and requires only black-box API access, which makes it broadly applicable to frontier closed-weight models. Our method is motivated by the observation that VLM errors tend to cluster around recurring failure patterns that are shared across models and predictable from the input alone. We therefore train a lightweight hint generator model to predict, for a given test input, which "hint" should be prepended to the prompt, providing targeted contextual or procedural guidance that steers the VLM away from its characteristic failure modes without revealing the answer. We show that Test-Time Hinting significantly improves the accuracy of multiple closed-weight VLMs on natural-image VQA benchmarks and that these gains generalize to unseen benchmarks and VLMs without retraining the hint generator.

1 INTRODUCTION

Test-time scaling (TTS) offers an alternative to the conventional strategy of scaling model size or training data. It instead directs additional compute resources to improve the accuracy of a frozen model at inference time. A wide range of TTS methods, from chain-of-thought prompting (Wei et al., 2022) and self-consistency (Wang et al., 2023) to process reward models (Lightman et al., 2024) and optimal compute allocation (Snell et al., 2025), have shown that performance continues to grow (often significantly) with inference-time compute. However, TTS research has concentrated primarily on LLMs. In this paper, we develop TTS methods tailored to vision-language models (VLMs).

Existing TTS approaches for VLMs have two limitations that restrict their practical application. First, the most effective TTS strategies, such as reward-model-guided search, iterative sampling, and process supervision, require open access to model weights or internal representations, which excludes frontier closed-weight VLMs accessible only through APIs (Zhang et al., 2024b; Xu et al., 2025). Second, even methods that operate through the model interface rely on *repeated* sampling or multi-pass inference through the same target model, which increases the per-query cost at inference time (Madaan et al., 2023; Wang et al., 2023). Additionally, the evaluation of these methods has focused primarily on mathematical and scientific reasoning benchmarks, leaving more conventional tasks such as natural-image understanding comparatively unexplored.

In this paper, we develop a TTS method for VLMs that requires only black-box API access and a single call to the target model, and is applicable to natural-image understanding tasks. The method applies to any VLM, but is especially valuable in deployment settings where a frontier model is used in a domain-specific application and further improvement using proprietary data or evaluations is needed. Our method is motivated by an empirical finding that VLM errors cluster around a small set of recurring failure modes that depend largely on the input, and that are correlated across distinct VLMs. These regularities imply that the structure of VLM errors is learnable and can be anticipated from the image and text inputs alone, before the model responds.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

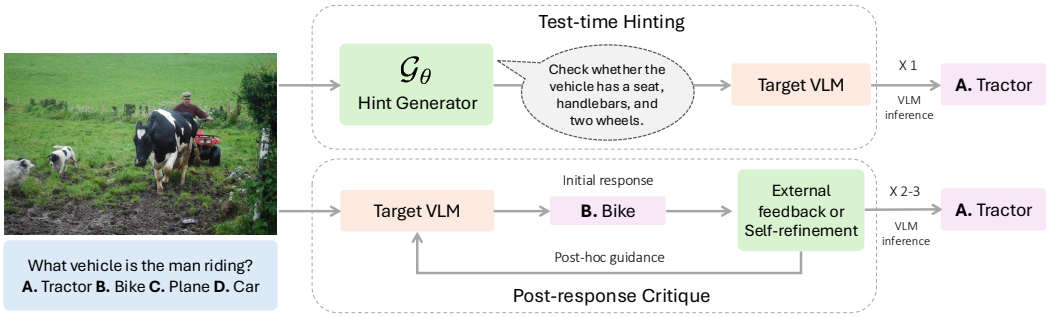


Figure 1: **Comparison between Test-Time Hinting and post-response critique.** TTH generates a hint at inference time *before* the target VLM answers, enabling single-pass inference with closed-weight models.

We call our method **Test-Time Hinting (TTH)**. TTH trains a (lightweight) hint generator to produce, for a given image–text pair, a short natural-language hint that is prepended to the target VLM's prompt before it responds. The hint provides contextual guidance that steers the model away from the failure mode it is most likely to exhibit on that input. To construct training data, we search for hints that, when prepended to the prompt, cause the target model to correct its previously incorrect response. Because failure modes recur systematically, the generator learns to anticipate them from the input alone. At inference time, the hint generator processes a new image–text pair, produces a hint, and passes it along with the original input to the black-box VLM in a single call (Figure 1).

To our knowledge, TTH is the first learned, single-call method for improving the performance of general-purpose, black-box VLMs at inference time. Specifically, our contributions are:

1. We propose TTH, a preemptive single-call TTS method for black-box VLMs, in which a lightweight hint generator prepends input-conditioned, failure-mode-anticipatory guidance to the target VLM's prompt before it answers.
2. We develop a two-stage training recipe for the hint generator that combines agentic-search distillation with multi-target reinforcement learning, and show that it effectively improves the accuracy of multiple frontier target VLMs.
3. We demonstrate that the efficacy of the hint generator transfers zero-shot, with no retraining, to five unseen target VLMs and three unseen natural-image VQA benchmarks.

We begin with an analysis of VLM failure patterns in Section 2, which motivates our TTH approach.

2 Dissecting Failure Modes in Vision-Language Models

One might expect that generating input-specific guidance toward the correct answer should be as hard as answering the question itself. We show that this is not the case: frontier VLMs exhibit a small set of recurring "failure modes" that are largely shared across models and predictable from the input alone, a structure that a simpler VLM can learn to anticipate for new multimodal inputs.

To establish this, we evaluate three proprietary models from different families: GPT-5 Nano (OpenAI), Gemini 2.5 Flash Lite (Google), and Claude 4.5 Haiku (Anthropic), on the A-OKVQA visual question answering benchmark (17,056 training questions) (Schwenk et al., 2022). Each model is queried three times per question with API-default temperature and top-p, and a question is classified as incorrect for a given model only if all three trials produce an incorrect answer. We ask two questions: do failure modes overlap across models, and is the failure mode predictable from the input alone?

Cross-Model Failure Mode Overlap. We define a taxonomy of 12 failure modes in VLMs: recognition, attribute binding, counting, spatial relation, optical character recognition, charts and tables, math/quantitative, knowledge, logic/negation, hallucination, instruction format, and "other" (full definitions in Section B.1). We use this taxonomy to categorize all error instances from various VLMs in the A-OKVQA dataset. Each incorrect response is labeled by an independent GPT-5 model with access to the image, question, ground-truth answer with rationale, and the model's incorrect response along with its reasoning.

Conditioned on shared errors among pairs of VLMs (i.e., questions both models answer incorrectly, with a pairwise Jaccard overlap of 0.32–0.37 between their incorrect question sets), we find that failure mode agreement is as high as 71–76%: when a pair of VLMs both fail, they tend to fail for the same reason (Figure 3 in Section B.2). This observation suggests that a hint targeting the anticipated failure mode for a given input could steer multiple models away from this common error.

Failure Modes are Predictable from Inputs Alone. We also test whether these failure modes can be predicted from the inputs alone. To do so, we train lightweight MLP classifiers on the final hidden state of Qwen3-VL-4B-Instruct for each image–question pair, supplemented with 22 scalar uncertainty features derived from its own output distribution (e.g., self-reported confidence, log-probability statistics, and entropy). No output of the target VLMs is used as input (Section B.3). A binary classifier achieves AUROC of 0.79–0.81 for predicting whether each target VLM will answer a given question incorrectly, and a shared-backbone variant with VLM-specific heads performs very similarly (Table 5). A 12-class classifier with the same architecture achieves top-2 accuracy of 73–78%, meaning the true failure mode can be narrowed to two candidates with high accuracy (Table 6). Together, these results suggest that a lightweight model can learn to predict not only *when* a VLM will fail but *how*, and that this capacity generalizes across architecturally distinct target VLMs.

3 Test-time Hinting for Black-box VLMs

Motivated by the observations in the previous section, we propose a method that identifies the likely failure mode of the VLM on a given input and prepends a targeted natural language hint to avert it.

Notation. We use the following notation throughout the paper:

- x, q: input image and question/prompt (the latter includes answer options for multiple-choice).
- $\hat{a}$, $\hat{r}$: a VLM's answer and the corresponding reasoning trace.
- a^*, r^*: the ground-truth answer and the corresponding rationale.
- $\mathcal{M}$: a *target VLM* whose errors we aim to fix using our method. We write $\hat{a} = \mathcal{M}(x, q)$ for its baseline response on input (x, q).
- m: a natural-language *model identifier* for $\mathcal{M}$ (e.g., "`GPT-5 Nano`"), passed as an input token to the hint generator when multiple targets are considered.
- $h = \{\texttt{"hint"} : [s_1, \ldots, s_k]\}$ with $k \in \{1, 2, 3\}$: a *hint* formatted as a short JSON list of natural-language items prepended to $\mathcal{M}$'s prompt. At least one item must be *contrastive*, prompting the model to discriminate between two plausible alternatives without revealing the answer. This contrastive form is central to the design, since it forces hints to surface perceptual or procedural cues rather than to encode the answer itself. h must not reveal a^*.
- $\mathcal{G}_\theta$: the *hint generator*, a small VLM with parameters θ that produces h from (x, q, m).

Our Test-Time Hinting approach (TTH) involves three steps (Figure 2): (1) sampling base responses from each target VLM, (2) eliciting per-question hints through an agentic loop, and (3) distilling these hints into $\mathcal{G}_\theta$ via supervised fine-tuning followed by reinforcement-learning (RL) refinement.

Step 1: Base-response sampling. We run each target VLM on every training question with three independent passes at API-default temperature and top-p. Each question-model pair is classified as *base-correct* (correct on all three trials), *base-incorrect* (incorrect on all three trials with a valid parse), or *mixed* (otherwise). Only base-correct and base-incorrect pairs are used in the following steps.

Step 2: Agentic hint generation and optimization. Given an input (x, q) for which the VLM $\mathcal{M}$ produces an incorrect answer $\hat{a} = \mathcal{M}(x, q) \neq a^*$, we construct a hint h such that prepending it to the same image-question pair yields the correct answer, i.e., $\mathcal{M}(x, [h; q]) = a^*$. To generate such hints, we design an iterative agentic pipeline comprising three roles, a Proposer, an Editor, and a Verifier, detailed below (pseudocode provided in Section A). Our agentic process handles two hint types: *repair* hints for base-incorrect questions, where it seeks a hint that flips the target to the correct answer, and *reinforcement* hints for base-correct questions, where it seeks a hint that retains the correct answer while keeping attention on the relevant visual evidence. A **proposer** $\mathcal{P}$ (GPT-5) with full privileged context $(x, q, a^*, r^*, \hat{a}, \hat{r})$ proposes a candidate hint. An **editor** $\mathcal{E}$ (GPT-5) modifies and revises the hint to enforce certain criteria (no direct leakage of a^*, at least one contrastive item, no certainty language, avoiding extraneous reasoning steps). The target VLM $\mathcal{M}$ then acts as the **verifier** by checking that the answer $\hat{a}$ corresponding to $(x, [h; q])$ matches a^*. The loop runs for up to

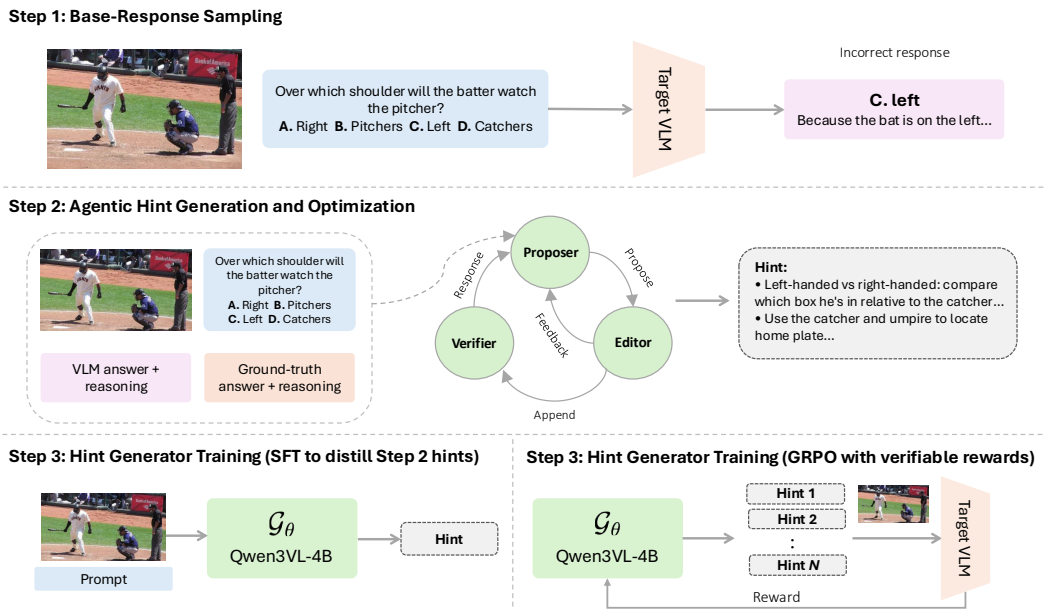


Figure 2: **Development pipeline of TTH.**

$R_{\max}$=3 rounds. For repair questions it terminates on the first correct re-answer and otherwise retains the last editor-approved hint as an *unsuccessful repair* target, which is excluded from downstream phases. For reinforcement hints, the system likewise terminates on the first correct re-answer, and trials in which every round disrupts the originally correct answer are discarded entirely.

Step 3: Hint generator training. We train a lightweight model $\mathcal{G}_\theta$ to generate the hints h elicited in Step 2 conditioned on the triplet (x, q, m) (i.e., the only information available at inference time). We initialize $\mathcal{G}_\theta$ from Qwen3-VL-4B-Instruct (Bai et al., 2025) and attach LoRA adapters (Hu et al., 2022) to the language-model backbone, the vision-language merger, and the last two blocks of the vision encoder. The full set of modules and training hyperparameters for both stages, including LoRA rank, learning rates, and optimization schedule, is provided in Section C. Training proceeds in two stages: supervised fine-tuning using the hints generated in Step 2, followed by a reinforcement learning step that directly optimizes the actual downstream behavior of the hint on the target VLMs (i.e., whether it flips incorrect answers). We show in Section 5.5 how both stages empirically contribute to $\mathcal{G}_\theta$.

Stage 1: Supervised fine-tuning on agentic hints. Each training instance is a tuple (x, q, m, h^*), where m is the natural-language target identifier and h^* is a hint produced in Step 2. We retain only Step 2 hint generations that lead to correct answers: the hint either flipped a base error to a correct answer (a successful repair) or preserved an already-correct answer (a successful reinforcement). Trials in which the verifier never reached a correct answer are discarded, since their hints were never verified end-to-end. Stage 2 picks up these harder cases by directly observing downstream behavior. The remaining pool of hints is balanced by target identifier via median-count resampling so that no single target dominates updates, and we minimize next-token cross-entropy on the hint JSON.

Stage 2: Reinforcement learning with verifiable rewards. Stage 1 supervises the generator to *imitate* hints that worked in Step 2, but it does not directly optimize the generator's own samples for downstream effect on the target VLM. Stage 2 closes this gap. For each training prompt (x, q, m) the policy samples a small group of candidate hints. Each candidate is sent to all three target VLMs, and we compare each target's hinted answer to its base (unhinted) answer, classifying the outcome as one of: *repair* (a previous error becomes correct), *no-op* (a correct answer is preserved), *harm* (a correct answer becomes wrong), or *unrepaired* (a previous error remains wrong). These outcomes are mapped to a per-target scalar score, averaged across the three targets, and used as the reward for Group Relative Policy Optimization (GRPO) (Shao et al., 2024b; Guo et al., 2025), KL-anchored to the Stage-1-trained reference. Details of the reward design choices are provided in Section C.4.

4 Related Work

Test-time scaling. Test-time scaling methods treat inference compute as a resource to optimize rather than a fixed cost. For LLMs, gains have been achieved through chain-of-thought prompting (Wei et al., 2022), self-consistency (Wang et al., 2023), process reward models (Lightman et al., 2024), and best-of-N sampling with learned verifiers (Snell et al., 2025). Analogous efforts for VLMs include multi-stage chain-of-thought reasoning (Zhang et al., 2024b; Xu et al., 2025) and reward-model-guided search (Wang et al., 2026b). Most VLM TTS methods, however, either require gradient or weight access to the model being improved, which excludes frontier closed-weight models, or rely on repeated sampling that multiplies per-query cost. Evaluation of these methods has focused on mathematical and scientific reasoning, and less on natural-image understanding. Our method addresses these limitations as it requires only black-box prompt access with a single call to the target, and is evaluated on natural-image VQA benchmarks spanning diverse image domains and question styles.

Post-hoc correction and critique. A prominent family of TTS methods improves accuracy through repeated inference, i.e., generating a response, critiquing it, and optionally revising. The critique signal varies across approaches, ranging from the target model itself (Madaan et al., 2023; Shinn et al., 2023) to multi-agent debate (Du et al., 2024; Liang et al., 2024) or a separate LLM acting as a judge (Zheng et al., 2023), and to dedicated critique models trained for the task (Zhang et al., 2025; Xie et al., 2025). Across all post-response methods, at least two inference passes through the target are required, with external-judge approaches adding a third call, incurring meaningful cost at frontier API pricing. Test-time hinting replaces these additional calls with a single pass through a compact hint generator, thereby achieving comparable or superior accuracy improvement with a single call to the target model.

Prompt optimization and adaptation. Another line of work improves a frozen model by optimizing its prompt. APE (Zhou et al., 2023), OPRO (Yang et al., 2024), and DSPy (Khattab et al., 2024) search offline for instructions or demonstrations and apply them to all inputs of a task, which helps when the instruction is the bottleneck. In natural-image VQA, however, the target already understands the task, and its likely error depends on the specific image and question. Consistent with this, a fixed task-level checklist of failure modes does not reliably help in our ablation (Section 5.5). Other methods tune prompts using gradients, repeated queries, or feedback: test-time prompt tuning adapts the prompts of CLIP-style VLMs to test inputs via gradients (Shu et al., 2022) or derivative-free black-box queries (Meng et al., 2025), online black-box prompt optimization learns from noisy performance feedback (Fang et al., 2026), and visual prompt tuning learns prompt tokens inside a frozen vision backbone (Wang et al., 2026a). TTH instead amortizes this search into a generator trained offline, which at test time needs no gradients, feedback, or per-query search, and produces an input-specific natural-language hint before a single target call (Section K).

Other prompting strategies. A complementary class of methods augments the prompt before the model answers. Chain-of-thought prompting (Wei et al., 2022) with its visual extensions (Shao et al., 2024a; Lu et al., 2022) and Tree of Thoughts (Yao et al., 2023) are task-generic and do not condition on the failure mode a given input is likely to trigger. Prophet (Yu et al., 2025) moves toward input-specific prompting by feeding candidate answers and confidence scores from a small VQA model to an LLM, but its heuristics are not anticipatory of the target's likely errors and provide candidate answers regardless of whether the target is likely to fail. LEAP (Zhang et al., 2024a) mines mistakes on few-shot examples to derive benchmark-level principles, but the resulting summaries are difficult to apply to individual questions when benchmark identity is unknown. TTH differs in that its guidance is conditioned on the specific image and question of each instance, with anticipation of the likely failure mode emerging implicitly from the training signal. Importantly, CoT, Prophet, and TTH are complementary: hints can be prepended to any prompting strategy without architectural changes.

5 Experiments

5.1 Experimental Setup

Datasets. Our primary development and evaluation benchmark is **A-OKVQA** (Schwenk et al., 2022), a knowledge-based VQA dataset with ground-truth rationales. Because its official test labels are not public, we use the validation split of 1,145 questions as our held-out *test set*. All model

Table 1: **Results on A-OKVQA. Bold**: best per row and metric; underline: second best; †: significantly more accurate than Base and every baseline.

Target Model	Overall Accuracy (%, ↑)					Repair Rate (%, ↑)				Harm Rate (%, ↓)			
	Base	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)
Claude 4.5 Haiku	78.25	80.26	77.90	80.87	**84.28**†	26.10	30.52	44.58	**49.40**	**4.69**	8.93	9.04	6.03
Gemini 2.5 Flash Lite	82.88	84.10	87.69	86.81	**90.57**†	21.43	35.71	43.37	**61.73**	2.95	**1.57**	4.22	3.48
GPT-5 Nano	84.37	82.79	85.24	**89.43**	88.12	18.44	17.88	44.69	**45.81**	5.29	**2.28**	**2.28**	4.04

development is performed on the 17,056-question training split, with 855 questions reserved as an internal *validation set* for hyperparameter selection and early stopping. The test set is used only for final model evaluation. To test whether $\mathcal{G}_\theta$ learns general failure-anticipation strategies rather than dataset-specific patterns, we additionally evaluate zero-shot transfer (no retraining or domain adaptation) on three additional natural-image benchmarks: **VCR** (Zellers et al., 2019) (5,000 unique image-question pairs sampled from the validation set), **Visual7W** (Zhu et al., 2016) (5,000 sampled test pairs), and **RealWorldQA** (xAI, 2024a;b) (628 four-way MCQ / yes-no / left-right / color questions retained from the original 765 for consistent MCQ-style evaluation).

Target models. Our primary target models are three proprietary frontier VLMs from different families: Claude 4.5 Haiku (Anthropic), Gemini 2.5 Flash Lite (Google), and GPT-5 Nano (OpenAI). The hint generator is trained on these three targets (Section 5.2) and additionally evaluated on five unseen targets without retraining (Section 5.3). For *same-family transfer*, we apply hints generated for each training model to a larger model from the same family: GPT-5 Nano hints are reused for GPT-5.2, Gemini 2.5 Flash Lite hints for Gemini 2.5 Flash, and Claude 4.5 Haiku hints for Claude 4.5 Opus. All target models are evaluated using their default API hyperparameters. Exact prompts for all methods and target models are reproduced in the supplementary material.

Baselines. We compare TTH (1 target call) with: (i) **Base** (1 call): direct prompting with the question and image; (ii) **CoT** (Wei et al., 2022; Kojima et al., 2022) (1 call): the base prompt augmented with "Think step-by-step. Then output ONLY the JSON object"; (iii) **Self-refine** (Madaan et al., 2023) (2 calls): the target answers, then is shown its own response and asked to critique and optionally correct it; (iv) **External judge** (Du et al., 2024; Zheng et al., 2023) (2 target calls + 1 judge call): a more capable model from the same family critiques the target's answer, and the target re-answers accordingly. Judges are GPT-5-Mini for GPT-5 Nano, Gemini 2.5 Flash for Gemini 2.5 Flash Lite, and Claude 4.5 Sonnet for Claude 4.5 Haiku (judges of the unseen targets are listed in Section D). Section 5.5 also compares static hints and an untrained generator.

Metrics. We report *three* metrics. *Overall accuracy* is the fraction of test questions answered correctly. The *repair rate*, $\Pr[\hat{a}_h = a^* \mid \hat{a} \neq a^*]$, restricts to the questions the base model answered incorrectly and reports what fraction each method corrects. The *harm rate*, $\Pr[\hat{a}_h \neq a^* \mid \hat{a} = a^*]$, restricts to base-correct questions and reports the fraction that flip to incorrect after applying each method. Repair rate and harm rate are the two complementary axes along which a hinting method's deployment value is most directly measured: a useful method must both correct previous errors *and* avoid disrupting answers the target already had right. The two rates determine the change in overall accuracy, $\Delta\mathrm{Acc} = (1-B)R - BH$ for base accuracy B, repair rate R, and harm rate H, so overall accuracy is our primary criterion and the two rates are its diagnostic components. We assess statistical significance with a joint paired nonparametric bootstrap over test questions (2,000 replicates), where a difference is significant if its 95% confidence interval excludes zero. Section D reports 95% confidence intervals for all results.

5.2 Main Results on A-OKVQA

Table 1 reports overall accuracy, repair rate, and harm rate on the A-OKVQA test set for the three primary target models. TTH achieves the highest repair rate on *every* target and the highest overall accuracy on two of three. The gains are most pronounced on the two weaker targets. On Claude 4.5 Haiku, TTH improves overall accuracy by 6.03 pp over base and surpasses the next-best method (external judge) by 3.41 points, while repairing 49.4% of base errors. On Gemini 2.5 Flash Lite, TTH reaches 90.57% overall accuracy and the strongest repair rate of any method-target pair (61.7%,

Table 2: **Zero-shot transfer to unseen target models on A-OKVQA.**

Transfer type	Target Model	Overall Accuracy (%, ↑)					Repair Rate (%, ↑)				Harm Rate (%, ↓)			
		Base	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)
Same family	Claude 4.5 Opus	91.00	90.92	91.09	91.27	**91.62**	13.59	5.83	12.62	**37.86**	1.43	**0.48**	0.95	3.07
	Gemini 2.5 Flash	91.88	91.53	91.88	91.44	**92.05**	12.90	7.53	6.45	**27.96**	1.52	**0.67**	1.05	2.28
	GPT-5.2	90.22	89.17	90.83	90.48	**91.09**	10.71	20.54	2.68	**30.36**	2.32	1.55	**0.00**	2.32
Different family	Grok 4 Fast Reasoning	85.33	84.89	86.90	86.38	**87.60**	7.14	11.31	14.29	**43.45**	1.74	**0.10**	1.23	4.81
	Nova Pro	83.76	83.58	84.37	83.32	**84.98**	13.44	22.04	30.11	**43.55**	**2.82**	3.59	6.36	6.99

Table 3: **Zero-shot transfer across benchmark datasets.** Markers as in Table 1.

Dataset	Target Model	Overall Accuracy (%, ↑)					Repair Rate (%, ↑)				Harm Rate (%, ↓)			
		Base	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)	CoT	Self refine	External judge	TTH (Ours)
VCR	Claude 4.5 Haiku	61.64	62.06	62.20	62.60	**64.98**†	15.10	11.46	21.35	**34.41**	8.72	**6.22**	11.73	16.00
	Gemini 2.5 Flash Lite	61.68	63.02	65.40	63.66	**67.04**†	14.58	14.58	19.27	**23.18**	6.89	**3.03**	8.76	5.71
	GPT-5 Nano	63.00	64.04	63.20	68.64	**68.68**	18.82	11.89	21.62	**24.22**	9.40	6.67	**3.75**	5.21
RealWorldQA	Claude 4.5 Haiku	62.74	60.67	63.85	60.83	**69.11**†	10.26	27.35	**49.15**	31.20	9.39	14.47	32.23	**8.38**
	Gemini 2.5 Flash Lite	65.13	66.72	68.15	71.34	**76.59**†	13.24	14.16	34.25	**34.70**	4.65	2.94	8.80	**0.98**
	GPT-5 Nano	70.06	71.50	71.97	**80.25**	75.64	27.13	26.60	**49.47**	26.60	9.54	8.64	6.60	**3.41**
Visual7W	Claude 4.5 Haiku	71.60	74.68	73.86	78.20	**81.26**†	21.13	25.35	**47.89**	37.04	4.08	6.90	9.78	**1.20**
	Gemini 2.5 Flash Lite	75.44	78.48	81.84	82.66	**85.22**†	20.33	31.71	**45.53**	45.30	2.59	1.84	5.25	**1.78**
	GPT-5 Nano	80.20	81.86	83.82	**85.08**	84.44	25.20	26.26	**41.41**	38.84	4.15	**1.97**	4.14	4.30

roughly 18 points above the next-best baseline). On both targets, TTH is significantly more accurate than Base and every baseline. For GPT-5 Nano the external judge (GPT-5-Mini) edges out TTH on overall accuracy (89.43% vs. 88.12%), but TTH still attains the best repair rate (45.81% vs. 44.69%). This accuracy difference is not significant, and TTH significantly outperforms Base, CoT, and self-refine. Notably, the external judge is the most costly and time-consuming baseline, relying on a second target call and a much more capable judge model. We discuss this case in Section 6.

5.3 Zero-Shot Transfer to Unseen Target Models

A core design goal of TTH is that hints optimized for a set of training models should generalize to unseen targets. Table 2 evaluates this on the A-OKVQA test set using the same checkpoint, without any retraining or adaptation. TTH achieves both the best overall accuracy and the best repair rate on every one of the five unseen targets. The gains are preserved in the different-family setting, where the hint generator has never observed the family's failure patterns and yet repairs base errors at 1.4–3.0× the rate of the next-best baseline. The same-family setting shows only modest absolute accuracy gains because the larger family members already operate near ceiling at base, but the relative repair-rate advantage over baselines is in fact *larger* than on the smaller targets seen during training. The contrast suggests that what the hint generator has internalized are failure-anticipation strategies tied to the visual-linguistic structure of inputs, rather than model-family-specific surface patterns. On these targets, the higher repair rate of TTH also comes with a higher harm rate than most baselines. Across all target–benchmark pairs, gains shrink with base accuracy because fewer errors remain, not because correction weakens (Section H).

5.4 Zero-Shot Transfer Across Benchmarks

Table 3 evaluates zero-shot generalization to three unseen natural-image VQA benchmarks (VCR, RealWorldQA, and Visual7W) using the same A-OKVQA-trained checkpoint. The benchmarks differ from A-OKVQA in question style and answer distribution, and VCR and RealWorldQA also in image domain (movie stills and mostly driving scenes), providing a stringent test of out-of-distribution generalization. TTH wins overall accuracy in seven of nine target-benchmark cells, confirming that the gains are not specific to A-OKVQA, and the largest single lift (+11.5 points for Gemini 2.5 Flash Lite) appears on RealWorldQA, the benchmark furthest in domain from A-OKVQA. Under the paired bootstrap, TTH is significantly more accurate than Base in all nine cases and, for Claude 4.5 Haiku and Gemini 2.5 Flash Lite, than every baseline on all three benchmarks. The two cases where the external judge, the most costly baseline, wins overall accuracy concern GPT-5 Nano paired with the

Table 4: **Ablation on the A-OKVQA test set. Bold**: best per metric per target; underline: second best.

Method	Claude 4.5 Haiku			Gemini 2.5 Flash Lite			GPT-5 Nano		
	Acc. (↑)	Repair (↑)	Harm (↓)	Acc. (↑)	Repair (↑)	Harm (↓)	Acc. (↑)	Repair (↑)	Harm (↓)
Base	78.25	–	–	82.88	–	–	84.37	–	–
Categorical failure-mode hint + CoT	76.16	7.23	**4.69**	85.24	33.16	4.00	84.89	31.84	5.29
Universal taxonomy hint	75.63	17.27	8.15	85.15	32.65	4.00	84.63	28.49	4.97
Untrained Qwen3	79.48	25.70	5.58	84.19	29.08	4.43	84.19	26.87	5.18
TTH (SFT only)	82.88	**50.20**	8.04	89.78	60.20	4.11	87.07	40.78	4.35
TTH (SFT + GRPO)	**84.28**	49.40	6.03	**90.57**	**61.73**	**3.48**	**88.12**	**45.81**	**4.04**

much stronger GPT-5-Mini judge and reproduce the pattern observed in Table 1. For this target, TTH and the external judge are statistically indistinguishable on VCR and Visual7W, and we revisit this case in Section 6. Where the external judge wins repair rate but TTH still wins overall accuracy, the judge harms more originally correct answers: as the ablation in Section 5.5 also shows, aggressive repair coverage and overall accuracy are not the same objective.

5.5 Ablations and Analysis of the Sources of Gain

To isolate where the gains in TTH come from, we ablate two design dimensions on the A-OKVQA test set: (i) what the hint actually contains, and (ii) how the generator is trained to produce it. We compare five conditions and report results in Table 4:

1. **Categorical failure-mode hint + CoT.** For base-incorrect questions we prepend the GPT-5-labeled failure mode (e.g., "`spatial_relation`") as a hint string, and for base-correct questions we fall back to plain CoT.
2. **Universal taxonomy hint.** We prepend the entire 12-mode failure taxonomy as a fixed checklist for every question, regardless of input.
3. **Untrained Qwen3.** The off-the-shelf Qwen3-VL-4B-Instruct (the initialization of $\mathcal{G}_\theta$), prompted to write a contrastive hint for each test question without ground truth or training.
4. **TTH (SFT only).** The Stage 1 hint generator without the reinforcement-learning refinement.
5. **TTH (SFT + GRPO).** The full TTH method as described in Section 3 (Stage 1 + Stage 2).

Static hints are not enough. The two static-hint baselines trail the learned generators by a wide margin on both overall accuracy and repair rate, while exhibiting comparable or slightly worse harm rates. With an oracle failure category supplied directly, the target's accuracy fails to improve reliably. This suggests that what TTH learns is qualitatively richer than failure-mode classification, i.e., making the hint input-grounded and content-specific is useful, and it is not sufficient to know which category of mistake the VLM is likely to make. An untrained hint writer is not enough either: Untrained Qwen3 gains at most 1.3 points of accuracy, whereas TTH is more accurate on all three targets and roughly doubles its repair rate at comparable harm (Section E), suggesting that the gains come from the training pipeline rather than from merely adding a hint.

Contribution of the two training stages. Compared to SFT-only, adding a GRPO stage produces a small but consistent improvement in overall accuracy on every target, with mixed effects on repair rate, and the largest and most reliable effect appears on the harm side, which it reduces uniformly across targets. This could be because imitating Step 2's repair-successful hints (i.e., the bulk of the SFT signal) teaches the generator *what to do when there is something to fix*, whereas *when not to interfere* with an already-correct answer is hard to learn from an SFT pool that only contains hints that happened to be safe. Stage 2's reward provides this discrimination directly, and the resulting overall-accuracy gain comes largely from a raised floor on previously-correct answers rather than from expanded repair coverage. The same mechanism explains why TTH wins overall accuracy in most cross-benchmark cases in Table 3 despite the external judge often winning repair rate.

Answer-leakage audit. Because the Step 2 proposer sees the ground-truth answer, we audit whether test-time hints leak it (Section F). During data construction, the editor required revisions for 8.4% of first-round proposals, and hints that still violated its constraints after the loop (2.6%) were discarded. A manual inspection of 100 random A-OKVQA test hints (with their questions and options) found

none that revealed or entailed the answer. Finally, text-only probes of increasing capacity (frozen MiniLM with a linear or MLP head, or fine-tuned end to end) trained to recover the correct option never do significantly better from the hint and options than from the options alone or than chance (25%). These audits find no evidence of explicit or statistically recoverable answer leakage.

6 Discussion

Deployment use case and inference cost. TTH is designed for settings where a downstream consumer of frontier VLMs observes systematic in-domain failure patterns but cannot modify the model, retrain it, or afford the latency and cost of multi-pass correction at scale. TTH needs one call to the black-box target, with no iterative feedback, and one pass through a compact 4B hint generator (costs in Section G).

When post-hoc critique remains competitive and the role of judge capability. The external judge is most competitive when paired with a much stronger judge, as for GPT-5 Nano with GPT-5-Mini (A-OKVQA, RealWorldQA, and Visual7W). Since self-refine provides no comparable advantage on the same target, the gap reflects the capability asymmetry between GPT-5 Nano and the much stronger judge, not the preemptive vs. post-hoc distinction. This advantage also comes at the highest inference cost of any method we consider: the external-judge pipeline requires three API calls per query (two target plus one judge), compared with one target call plus one small hint generator call for TTH. The pattern reverses for GPT-5.2 (Table 2), whose judge (GPT-5.2-Pro) is closer in capability: TTH outperforms all baselines including the external judge. TTH's relative advantage over post-hoc critique therefore grows as the target-judge capability gap narrows, a regime that becomes increasingly common as frontier models converge.

TTH in compound AI systems. In the increasingly common compound AI systems and agentic workflows (Zaharia et al., 2024; Khattab et al., 2024), self-refinement (Madaan et al., 2023; Shinn et al., 2023) and external judges or critics (Zheng et al., 2023; Xie et al., 2025) are routinely used to check and improve the outputs of large models, at the cost of additional calls to large models or longer reasoning. Whereas cost-aware systems route queries between small and large models (Chen et al., 2024; Ong et al., 2025), TTH uses a small model to steer a large one before it responds. Because the hint generator is much smaller and faster than the target and no revision rounds are needed, TTH avoids the sequential target and judge calls of post-response critique, reducing the cost and latency of each query. Its plain-text hints can also precede any black-box call in a workflow, alone or alongside reflection or judge modules.

Limitations and future directions. We evaluate TTH on multiple-choice VQA, where correctness is unambiguous and repair, harm, and answer leakage can be measured exactly. Within this setting, TTH is tested on eight VLMs from five providers and four benchmarks. The attainable gain is bounded by the errors that remain, so absolute gains are smaller for near-ceiling targets (Section H). TTH is therefore most useful where frontier models are not yet saturated, such as in domain-specific applications. Our leakage audits are empirical rather than formal guarantees. Two natural extensions follow. The error predictors of Section 2 suggest gating hint generation by predicted failure risk, so that the hint pipeline only fires on inputs the target is likely to get wrong without help, further lowering the average inference cost. A second direction is adapting the framework beyond multiple choice to open-ended generative VQA, which requires replacing exact-match correctness in the reward with a task-appropriate outcome signal and extending the no-leakage check to free-form answers. Both extensions preserve the central design of TTH, namely a small generator that produces input-conditioned hints in a single call, and we leave them to future work.

7 Conclusion

We introduced Test-Time Hinting, a preemptive single-call test-time scaling method for black-box VLMs. The method exploits an empirical regularity, namely that failure modes are recurring, cross-model, and predictable from the input alone, to train a compact generator that prepends failure-mode-aware guidance to the target's prompt. A two-stage recipe combining distillation from agentic search with reinforcement learning against the actual downstream behavior of each candidate hint on multiple target VLMs yields gains on overall accuracy across our primary targets, and these gains transfer zero-shot to unseen targets and unseen benchmarks.

AI USE STATEMENT

Large language models are components of the proposed method and its evaluation: GPT-5 serves as the proposer and editor that construct the synthetic training hints (Section 3) and as the failure-mode annotator (Section 2). Target VLMs and external judges are described in Section 5. We additionally used generative AI tools to aid and polish the writing and to help organize and clean up the codebase. The authors reviewed all AI-assisted code and text and take full responsibility for the content of this work.

ETHICS STATEMENT

This work uses publicly available datasets under their respective licenses: A-OKVQA (Apache License 2.0), VCR (AI2 custom research-only license), Visual7W (MIT License), and RealWorldQA (CC BY-ND 4.0). The hint generator is initialized from Qwen3-VL-4B-Instruct (Apache License 2.0). All proprietary models were accessed through their commercial APIs in accordance with the providers' terms of use. The study involves no human subjects, and the manual leakage audit (Section 5.5) was conducted by the authors. Because hints are plain text, they can be inspected before use. As with any method that modifies model inputs, a hint can also mislead the target (Section I), which is why we report harm rates alongside accuracy.

REPRODUCIBILITY STATEMENT

Section 3 and Section A describe the three-step pipeline, including the agentic hint-construction loop. Section C lists the LoRA target modules, all training hyperparameters, the reward design, and the training compute. Section 5.1 and Section D specify the datasets and splits, target models, external judges, baselines, and the bootstrap procedure. Section F details the leakage audits, and Section G reports the API cost of constructing the training data. Anonymized code and the prompts for all agents, baselines, and target models are provided in the supplementary material.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024.

Jinjie Fang, Runwen You, Wanli Shi, Wenkang Wang, Ganyu Wang, Haozhen Zhang, Yi Chang, and Bin Gu. Online black-box prompt optimization with regret guarantees under noisy feedback. In *International Conference on Learning Representations*, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations*, pp. 54928–54958, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17889–17904. Association for Computational Linguistics, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, volume 35, pp. 2507–2521, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023.

Fan'an Meng, Chaoran Cui, Hongjun Dai, and Shuai Gong. Black-box test-time prompt tuning for vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 6099–6107, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. In *International Conference on Learning Representations*, 2025.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-OKVQA: A benchmark for visual question answering using world knowledge. In *European Conference on Computer Vision (ECCV)*, volume 13668 of *Lecture Notes in Computer Science*, pp. 146–162. Springer, 2022.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual CoT: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 8612–8642, 2024a.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024b.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Manli Shu, Weili Nie, De-An Huang, Zhiding Yu, Tom Goldstein, Anima Anandkumar, and Chaowei Xiao. Test-time prompt tuning for zero-shot generalization in vision-language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 14274–14289, 2022.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Junze Wang, Lei Fan, Dezheng Zhang, Weipeng Jing, Donglin Di, Yang Song, Sidong Liu, and Cong Cong. Visual prompt-agnostic evolution. In *International Conference on Learning Representations*, 2026a.

Weiyun Wang, Zhangwei Gao, Lianjie Chen, Zhe Chen, Jinguo Zhu, Xiangyu Zhao, Yangzhou Liu, Yue Cao, Shenglong Ye, Xizhou Zhu, Lewei Lu, Haodong Duan, Yu Qiao, Jifeng Dai, and Wenhai Wang. VisualPRM400K: An effective dataset for training multimodal process reward models. In *International Conference on Learning Representations*, 2026b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

xAI. RealWorldQA (dataset). https://huggingface.co/datasets/xai-org/RealworldQA, 2024a. Official xAI release on Hugging Face. Accessed: 2026-03-05.

xAI. Grok-1.5 Vision Preview. https://x.ai/news/grok-1.5v, April 2024b. Introduces the RealWorldQA benchmark and provides the official dataset download link. Accessed: 2026-03-05.

Zhihui Xie, Jie Chen, Liyu Chen, Weichao Mao, Jingjing Xu, and Lingpeng Kong. Teaching language models to critique via reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68559–68577. PMLR, 2025.

Guowei Xu, Peng Jin, Ziang Wu, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. LLaVA-CoT: Let vision language models reason step-by-step. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2087–2098, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

Zhou Yu, Xuecheng Ouyang, Zhenwei Shao, Meng Wang, and Jun Yu. Prophet: Prompting large language models with complementary answer heuristics for knowledge-based visual question answering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(8):6797–6808, 2025.

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound AI systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/, 2024. Berkeley Artificial Intelligence Research (BAIR) Blog.

Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6720–6731, 2019.

Di Zhang, Jingdi Lei, Junxian Li, Xunzhi Wang, Yujie Liu, Zonglin Yang, Jiatong Li, Weida Wang, Suorong Yang, Jianbo Wu, Peng Ye, Wanli Ouyang, and Dongzhan Zhou. Critic-V: VLM critics help catch VLM errors in multimodal reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9050–9061, 2025.

Tianjun Zhang, Aman Madaan, Luyu Gao, Steven Zheng, Swaroop Mishra, Yiming Yang, Niket Tandon, and Uri Alon. In-context principle learning from mistakes. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 59520–59558. PMLR, 2024a.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *Transactions on Machine Learning Research*, 2024b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pp. 46595–46623, 2023.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7W: Grounded question answering in images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4995–5004, 2016.

A AGENTIC HINT OPTIMIZATION: PSEUDOCODE

Algorithm 1 formalizes the three-role agentic loop summarized in Section 3. In the first proposal round, the editor required revisions for 8.4% of proposals. Hints that still violated its constraints after the loop (2.6% of all proposed hints) were discarded before Step 3, so no rejected hint was used to train $\mathcal{G}_\theta$ (Section F).

Algorithm 1 Agentic Hint Optimization

Require: Image x, question q, ground truth (a^*, r^*), target $\mathcal{M}$ with base response $(\hat{a}, \hat{r})$, proposer $\mathcal{P}$, editor $\mathcal{E}$, max rounds $R_{\max}$=3, type $\tau \in \{\texttt{repair}, \texttt{reinforcement}\}$
Ensure: Selected hint h^* and outcome flag, or DISCARD

feedback $\leftarrow \varnothing$; $\hat{a}_{prev} \leftarrow \varnothing$; $\hat{r}_{prev} \leftarrow \varnothing$; $h_{last} \leftarrow \varnothing$
for $t = 1, \ldots, R_{\max}$ **do**
  $h \leftarrow \mathcal{P}(x, q, a^*, r^*, \hat{a}, \hat{r}, \textit{feedback}, \hat{a}_{prev}, \hat{r}_{prev})$ ▷ Propose
  $(\textit{verdict}, h', \textit{feedback}) \leftarrow \mathcal{E}(x, q, a^*, r^*, \hat{a}, \hat{r}, h)$ ▷ Edit / revise
  if *verdict* $= \texttt{revise}$ **then**
    $h \leftarrow h'$
  end if
  $h_{last} \leftarrow h$
  $(\hat{a}_h, \hat{r}_h) \leftarrow \mathcal{M}(x, q, h)$ ▷ Verifier re-answers
  if $\hat{a}_h = a^*$ **then**
    $h^* \leftarrow h$
    return $(h^*, \text{SUCCESS})$
  end if
  $\hat{a}_{prev} \leftarrow \hat{a}_h$; $\hat{r}_{prev} \leftarrow \hat{r}_h$
end for
if $\tau = \texttt{repair}$ **then**
  return $(h_{last}, \text{UNSUCCESSFULREPAIR})$
else
  return DISCARD
end if

B CROSS-MODEL FAILURE STRUCTURE: FULL RESULTS

This appendix expands the motivational analysis summarized in Section 2. We provide the failure-mode taxonomy in full, the cross-model overlap and agreement results, the predictor architecture, and a short rationale for using a single model-conditioned hint generator instead of one specialist generator per target.

B.1 FAILURE-MODE TAXONOMY

The taxonomy $\mathcal{F}$ used throughout the paper consists of twelve failure modes, hand-picked by the authors during a manual review of base-incorrect responses across the three primary target models. The list is intended to cover the empirically frequent ways in which the targets fail on natural-image VQA, and is used both for the cross-model analysis below and for the categorical-failure-mode-hint baseline in Section 5.5.

- **recognition.** The target misidentifies the primary object, scene, or activity depicted in the image (e.g., confusing a cat for a dog, a kitchen for a dining room).
- **attribute binding.** The target identifies the right entities but binds attributes (color, material, state, role) to the wrong one.
- **counting.** The target's numeric answer for the count of an object, person, or instance is incorrect.
- **spatial relation.** The target misjudges the spatial layout of entities (left/right, behind/in front, above/below, near/far).
- **OCR.** The target misreads or fails to read text rendered inside the image (signs, captions, packaging).
- **chart/table.** The target misinterprets the data structure of a chart, table, or schematic.

- **math/quantitative.** The target reaches an incorrect quantitative conclusion via arithmetic or comparison error after correct perception.
- **knowledge.** The target's perception is correct but it lacks or misapplies the world knowledge required to answer.
- **logic/negation.** The target mishandles negation, conjunction, exclusion, or other propositional structure in the question.
- **hallucination.** The target asserts the presence of entities, attributes, or events that are not in the image.
- **instruction format.** The target's reasoning is correct but its final answer fails to comply with the requested output format.
- **other.** A residual category for failures that do not fit any of the above.

B.2 Cross-Model Error Overlap and Failure-Mode Agreement

Figure 3 visualizes two complementary aspects of the error landscape on the A-OKVQA training split: pairwise Jaccard overlap on base-incorrect question sets (left), and failure-mode agreement on shared errors (right). Overlap values of 0.32 to 0.37 across all three frontier-model pairs, combined with 71 to 76% failure-mode agreement on shared errors, support the design choice of training a single model-conditioned hint generator rather than separate per-model specialists. We elaborate on this design choice in Section B.4.

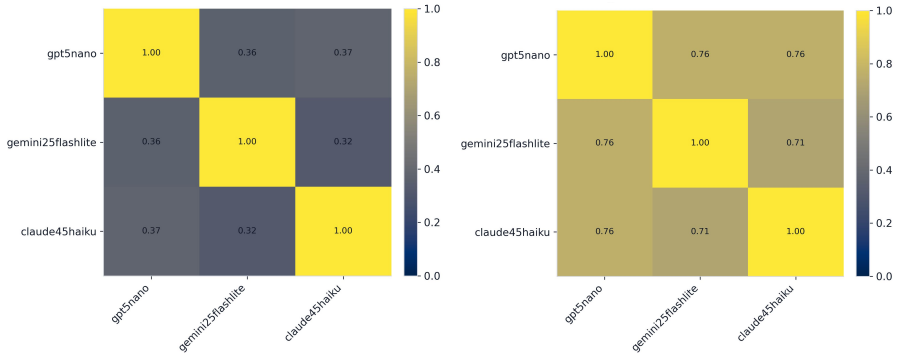


Figure 3: **Cross-model error structure on A-OKVQA (training split).** *Left:* Pairwise Jaccard overlap of base-incorrect question sets across the three frontier VLMs. *Right:* Failure-mode agreement rate among shared errors, i.e., the fraction of questions both models answer incorrectly that are assigned the same failure-mode label by the GPT-5 annotator.

B.3 Predictive Architecture

For each VQA sample we extract the final hidden state of Qwen3-VL-4B-Instruct (2560-dim) from its last transformer layer and supplement it with 22 scalar uncertainty features derived from Qwen3-VL-4B-Instruct's own output distribution (self-reported confidence, log-probability statistics, entropy, calibration ratio, overconfidence indicator, and so on). No output of the target VLMs is used as input. A hidden encoder projects the 2560-dim representation through two fully connected layers ($2560 \to 256 \to 64$), while a scalar encoder maps the 22 uncertainty features to a 32-dim embedding, yielding a 96-dim concatenated representation passed to a task-specific classification head.

Risk prediction (when models fail). For binary risk prediction, the 96-dim representation feeds a two-layer head ($96 \to 64 \to 1$) trained with BCEWithLogitsLoss. We train one such classifier per target model, plus a shared-backbone variant. Table 5 reports AUROC, max-F1, precision, and recall. AUROC of 0.79 to 0.81 confirms that model-specific failure risk is substantially predictable from the image-question representation alone, and the shared-backbone variant performs comparably to per-model heads, indicating that the predictive structure is shared across targets.

Table 5: **Binary risk prediction.** *Indiv. MLP*: one predictor per target model. *Shared MLP*: single predictor for all target models.

Target Model	AUROC (↑)		Max-F1 (↑)		Precision (↑)		Recall (↑)	
	Indiv. MLP	Shared MLP	Indiv. MLP	Shared MLP	Indiv. MLP	Shared MLP	Indiv. MLP	Shared MLP
Claude 4.5 Haiku	**0.79**	0.78	0.54	**0.54**	0.41	**0.43**	**0.76**	0.73
Gemini 2.5 Flash Lite	**0.81**	0.80	**0.46**	0.44	**0.36**	0.33	0.61	**0.68**
GPT-5 Nano	0.79	**0.80**	0.36	**0.36**	0.25	**0.28**	**0.67**	0.52

Failure-mode prediction (how models fail). Table 6 extends the analysis to 12-class failure-mode classification. The architecture is identical to the risk predictor except that the binary head is replaced with a multi-class output layer trained with CrossEntropyLoss and inverse-frequency class weighting. Top-2 accuracy of 73 to 78% indicates that the true failure mode can be narrowed to two candidates with high reliability, confirming that the signal needed for anticipatory hint generation is present in the shared visual-linguistic representation.

Table 6: **Failure-mode prediction** (12-class classification).

Target Model	Top-1 (↑)	Top-2 (↑)	Macro-F1 (↑)	Wt-F1 (↑)	BalAcc (↑)
Claude 4.5 Haiku	0.54	0.73	0.40	0.52	0.484
Gemini 2.5 Flash Lite	0.50	0.78	0.30	0.47	0.434
GPT-5 Nano	0.66	0.78	0.47	0.66	0.311

B.4 Single Model-Conditioned Generator vs. Per-Model Specialists

The cross-model overlap and failure-mode agreement reported above motivate a specific architectural choice for the hint generator: rather than training one specialist generator per target VLM, we train a single generator that takes the target identifier m as an input token. The motivation is twofold. First, a substantial fraction of failures are shared across targets, both at the question level (Jaccard overlap of 0.32 to 0.37) and at the failure-mode level (agreement of 71 to 76% on shared errors), so a single model can learn the common failure-anticipation skill from data pooled across targets, with target-specific adaptation handled through the conditioning token. Second, a shared-backbone risk predictor (Table 5) performs comparably to per-target specialists, providing direct evidence that the predictive structure underlying failure anticipation is largely shared. A single model-conditioned generator also enables zero-shot transfer to unseen target identifiers at inference time, which is the property exercised in Section 5.3.

C Hint Generator Training: Hyperparameters and Reward Design

This appendix details both stages of the Step 3 hint-generator training summarized in Section 3. Section C.1 specifies the LoRA target modules and optimization conventions shared by both stages. Section C.2 and Section C.3 give the stage-specific hyperparameters for supervised fine-tuning and GRPO respectively, and Section C.4 through Section C.7 detail the reward design and data composition for Stage 2. Training was done on 1 × H200 GPU, where Stage 1 took approximately 2 hours to finish and Stage 2 took approximately 9 hours.

C.1 Shared LoRA Scope and Optimization Conventions

Both stages adapt the same set of LoRA target modules (the language-model backbone, the vision-language merger, and the last two blocks of the vision encoder), totaling 23 LoRA target modules. The exact module patterns are:

- **Language-model backbone:** `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`.
- **Vision-language merger:** `model.visual.merger.linear_fc1`, `model.visual.merger.linear_fc2`, and the corresponding `linear_fc1`/`linear_fc2` pairs in each of the three `model.visual.deepstack_merger_list` blocks (eight modules total).
- **Vision encoder (last two blocks):** `attn.qkv`, `attn.proj`, `mlp.fc1`, `mlp.fc2`.

The LoRA configuration is identical across both stages: rank r=16, scaling α=32, dropout 0.05. Both stages use the AdamW optimizer with bf16 mixed precision and gradient checkpointing enabled.

C.2 Stage 1: Supervised Fine-Tuning Hyperparameters

Stage 1 minimizes next-token cross-entropy on the hint JSON over the Step 2 trials retained as described in Section 3. We optimize with AdamW at learning rate 1.6×10^{-4}, weight decay 0, gradient clipping at max norm 1.0, and warmup ratio 0.03. The LoRA scope and mixed-precision setup follow Section C.1.

C.3 Stage 2: GRPO Optimization

Stage 2 is initialized from the Stage 1 LoRA adapter and KL-anchored (with coefficient β=0.04) to that reference, retaining the LoRA scope and configuration described in Section C.1. We use GRPO (Shao et al., 2024b) with group size G=8, sampling temperature 0.9, PPO clip ε=0.2, and AdamW learning rate 1×10^{-5}, training for 300 optimization steps.

C.4 Reward Truth Table

Table 7 gives the per-target reward as a function of the target model's correctness with vs. without the hint. The reward used to update the policy on a given prompt is the average of these per-target scores across the three target VLMs.

Table 7: **Per-target reward for Stage 2 reinforcement learning.**

Outcome name	Description	Reward
Repair	Base wrong, hinted correct	+1
No-op	Base correct, hinted correct	0
Harm	Base correct, hinted wrong	−1
Unrepaired	Base wrong, hinted wrong	−0.5

C.5 Why Average the Reward Across Three Targets

A natural alternative to averaging the per-candidate reward across all three targets would be to score each candidate against a single (e.g.,, the matching) target. We chose the multi-target averaged reward for two reasons.

First, it acts as a variance-reduction mechanism for the advantage estimate. Within a GRPO group, the advantage of each candidate is computed relative to the within-group mean reward. Per-target rewards are noisy (a single target's response to a hint depends on stochastic decoding artifacts and on whether the hint happens to align with that target's idiosyncratic style), and averaging across three targets reduces this noise before the advantage is computed, yielding tighter group-relative advantages and a more stable policy gradient.

Second, it acts as a regularizer that discourages the policy from chasing model-specific shortcuts. A reward signal scored against a single target will, in the limit, encourage the policy to exploit any prompt patterns that happen to flip that specific target without producing perceptual or procedural guidance that would generalize. Averaging the reward across three architecturally distinct targets penalizes such shortcuts directly: a candidate hint can only achieve a high reward if it shifts behavior in the same direction across all three targets, which empirically requires guidance grounded in the

input itself rather than in target-specific quirks. This regularization is also what enables the zero-shot transfer to unseen target VLMs reported in Section 5.3.

C.6 Why the Unrepaired Outcome Is Downweighted

The reward in Table 7 is asymmetric: the harm outcome (base correct, hinted wrong) receives -1, while the unrepaired outcome (base wrong, hinted wrong) receives only -0.5. The natural-looking choice would have been to keep the two penalties symmetric at -1, since both correspond to a wrong hinted answer. We did not do this for the following reason.

A symmetric -1 penalty creates an over-commit pressure on the policy. On hard inputs where every candidate hint is likely to leave the target wrong, a symmetric scheme treats all candidates in a group as equally bad, and the policy receives a strong undifferentiated negative signal that, propagated through the model, is most easily reduced by increasing the assertiveness of the hint, since assertive hints are the ones that occasionally get a hard input correct. Empirically, this assertiveness-inducing style does not stay localized to hard inputs: the same prompts and decoding patterns spill over into nearby base-correct prompts and produce harm.

Downweighting the unrepaired outcome to -0.5 keeps the actual harm signal intact (the policy is still penalized strongly when it disrupts a correct answer) while permitting cautious or no-op-style hints when the input is genuinely ambiguous. We selected the -0.5 value via a small sweep on the internal validation set, and harm-rate behavior was consistent across nearby choices.

C.7 Mixing Base-Correct Prompts into the Stage 2 Pool

The Step 1 outputs are heavily skewed: roughly 71% of training prompts are answered correctly by all three targets at base. Strictly excluding these would deprive the policy of exposure to the no-op condition, which is exactly what the harm metric measures at evaluation. We therefore retain a configurable share of all-base-correct prompts in the Stage 2 training pool. A small sweep over shares of $\{30, 40, 50, 70\}\%$ identified 50% as the most balanced operating point: enough exposure to teach the policy to leave already-correct answers alone, without diluting the harm-bearing prompts to the point of weakening repair coverage. We use 50% for all reported TTH results.

D Evaluation Protocol and Results with Confidence Intervals

Bootstrap and significance. We quantify uncertainty with a joint paired nonparametric bootstrap over test questions. Each of the 2,000 replicates resamples the test questions with replacement, and the identical resampled question indices are shared across all methods and target models. Within each replicate, overall accuracy, repair rate, and harm rate are recomputed, with repair and harm rates computed on the resampled questions that the target answers incorrectly and correctly at base, respectively. Brackets in Tables 8 to 10 give 95% confidence intervals. To compare two methods, we compute the difference in their accuracy within each replicate and deem the difference significant if the 95% confidence interval of this paired difference excludes zero.

External judges. The external judge is a more capable model from the same family as the target: GPT-5-Mini for GPT-5 Nano, Gemini 2.5 Flash for Gemini 2.5 Flash Lite, and Claude 4.5 Sonnet for Claude 4.5 Haiku. For the unseen targets, the judges are GPT-5.2-Pro for GPT-5.2, Gemini 3 Flash for Gemini 2.5 Flash, Claude 4.6 Opus for Claude 4.5 Opus, Grok 4.1 Fast Reasoning for Grok 4 Fast Reasoning, and Nova Premier for Nova Pro.

Paired-test conclusions. On A-OKVQA (Table 8), TTH is significantly more accurate than Base and every baseline for Claude 4.5 Haiku and Gemini 2.5 Flash Lite. For GPT-5 Nano, it is significantly more accurate than Base, CoT, and self-refine, and its difference from the external judge is not significant. On the unseen benchmarks (Table 10), TTH is significantly more accurate than Base for every target on every benchmark, and than every baseline for Claude 4.5 Haiku and Gemini 2.5 Flash Lite. For GPT-5 Nano, TTH and the external judge are statistically indistinguishable on VCR and Visual7W, while on RealWorldQA the external judge, which is the most costly baseline and consults the much more capable GPT-5-Mini, is significantly more accurate. Table 9 reports the five unseen targets on A-OKVQA.

Table 8: **Results on A-OKVQA with 95% confidence intervals** (Table 1). Markers as in Table 1.

Target Model	Method	Overall Accuracy (%, ↑)		Repair Rate (%, ↑)		Harm Rate (%, ↓)	
Claude 4.5 Haiku	Base	78.25	[75.89, 80.61]		–		–
	CoT	80.26	[77.90, 82.53]	26.10	[20.71, 31.53]	**4.69**	[3.35, 6.11]
	Self-refine	77.90	[75.45, 80.26]	30.52	[24.81, 36.36]	8.93	[7.13, 10.83]
	External judge	80.87	[78.60, 83.14]	44.58	[38.41, 50.75]	9.04	[7.10, 11.11]
	TTH (Ours)	**84.28**†	[82.09, 86.45]	**49.40**	[43.09, 55.56]	6.03	[4.33, 7.83]
Gemini 2.5 Flash Lite	Base	82.88	[80.70, 85.06]		–		–
	CoT	84.10	[81.92, 86.20]	21.43	[15.84, 27.45]	2.95	[1.92, 4.05]
	Self-refine	87.69	[85.77, 89.61]	35.71	[28.96, 42.52]	**1.57**	[0.83, 2.40]
	External judge	86.81	[84.80, 88.73]	43.37	[36.37, 50.51]	4.22	[3.01, 5.55]
	TTH (Ours)	**90.57**†	[88.73, 92.31]	**61.73**	[54.83, 68.56]	3.48	[2.29, 4.80]
GPT-5 Nano	Base	84.37	[82.27, 86.47]		–		–
	CoT	82.79	[80.52, 84.97]	18.44	[12.91, 24.20]	5.29	[3.92, 6.77]
	Self-refine	85.24	[83.14, 87.25]	17.88	[12.43, 23.70]	**2.28**	[1.36, 3.28]
	External judge	**89.43**	[87.60, 91.18]	44.69	[37.35, 52.02]	**2.28**	[1.42, 3.28]
	TTH (Ours)	88.12	[86.11, 90.04]	**45.81**	[38.46, 53.05]	4.04	[2.73, 5.42]

Table 9: **Zero-shot transfer to unseen target models on A-OKVQA, with 95% confidence intervals** (Table 2). **Bold**: best per target and metric; underline: second best.

Target Model	Method	Overall Accuracy (%, ↑)		Repair Rate (%, ↑)		Harm Rate (%, ↓)	
Same family							
Claude 4.5 Opus	Base	91.00	[89.34, 92.66]		–		–
	CoT	90.92	[89.17, 92.58]	13.59	[7.34, 20.56]	1.43	[0.76, 2.21]
	Self-refine	91.09	[89.43, 92.66]	5.83	[1.86, 10.82]	**0.48**	[0.10, 0.95]
	External judge	91.27	[89.61, 92.84]	12.62	[6.59, 19.54]	0.95	[0.38, 1.60]
	TTH (Ours)	**91.62**	[90.05, 93.19]	**37.86**	[28.09, 47.37]	3.07	[1.98, 4.15]
Gemini 2.5 Flash	Base	91.88	[90.22, 93.45]		–		–
	CoT	91.53	[89.87, 93.10]	12.90	[6.45, 20.00]	1.52	[0.85, 2.28]
	Self-refine	91.88	[90.22, 93.45]	7.53	[2.50, 13.34]	**0.67**	[0.20, 1.22]
	External judge	91.44	[89.78, 93.01]	6.45	[2.04, 11.93]	1.05	[0.48, 1.71]
	TTH (Ours)	**92.05**	[90.73, 93.34]	**27.96**	[19.61, 36.28]	2.28	[1.16, 3.35]
GPT-5.2	Base	90.22	[88.47, 91.88]		–		–
	CoT	89.17	[87.34, 90.92]	10.71	[5.26, 16.82]	2.32	[1.44, 3.29]
	Self-refine	90.83	[89.08, 92.49]	20.54	[13.34, 28.34]	1.55	[0.87, 2.34]
	External judge	90.48	[88.73, 92.14]	2.68	[0.00, 6.09]	**0.00**	[0.00, 0.00]
	TTH (Ours)	**91.09**	[89.43, 92.66]	**30.36**	[21.70, 39.58]	2.32	[1.47, 3.29]
Different family							
Grok 4 Fast Reasoning	Base	85.33	[83.41, 87.25]		–		–
	CoT	84.89	[82.97, 86.72]	7.14	[3.49, 10.82]	1.74	[0.85, 2.70]
	Self-refine	86.90	[84.98, 88.73]	11.31	[5.50, 17.53]	**0.10**	[0.00, 0.31]
	External judge	86.38	[84.46, 88.22]	14.29	[8.58, 20.40]	1.23	[0.42, 2.14]
	TTH (Ours)	**87.60**	[85.68, 89.43]	**43.45**	[35.93, 51.13]	4.81	[3.57, 6.15]
Nova Pro	Base	83.76	[81.58, 85.86]		–		–
	CoT	83.58	[81.40, 85.68]	13.44	[8.76, 18.50]	**2.82**	[1.84, 3.90]
	Self-refine	84.37	[82.23, 86.34]	22.04	[16.18, 28.12]	3.59	[2.67, 4.61]
	External judge	83.32	[81.14, 85.50]	30.11	[22.97, 37.38]	6.36	[4.58, 8.19]
	TTH (Ours)	**84.98**	[82.88, 87.08]	**43.55**	[37.00, 50.31]	6.99	[5.74, 8.32]

Table 10: **Zero-shot transfer across benchmark datasets, with 95% confidence intervals** (Table 3). Markers as in Table 1.

Target Model	Method	Overall Accuracy (%, ↑)		Repair Rate (%, ↑)		Harm Rate (%, ↓)	
VCR ($n = 5{,}000$)							
Claude 4.5 Haiku	Base	61.64	[60.31, 62.97]		–		–
	CoT	62.06	[60.73, 63.39]	15.10	[13.55, 16.74]	8.72	[7.77, 9.77]
	Self-refine	62.20	[60.81, 63.53]	11.46	[10.10, 12.94]	**6.22**	[5.42, 7.12]
	External judge	62.60	[61.27, 63.93]	21.35	[19.54, 23.21]	11.73	[10.63, 12.90]
	TTH (Ours)	**64.98**†	[63.67, 66.29]	**34.41**	[32.32, 36.63]	16.00	[14.88, 17.11]
Gemini 2.5 Flash Lite	Base	61.68	[60.29, 63.01]		–		–
	CoT	63.02	[61.69, 64.35]	14.58	[13.05, 16.20]	6.89	[6.05, 7.80]
	Self-refine	65.40	[64.07, 66.73]	14.58	[13.05, 16.22]	**3.03**	[2.50, 3.68]
	External judge	63.66	[62.33, 64.99]	19.27	[17.58, 21.08]	8.76	[7.80, 9.80]
	TTH (Ours)	**67.04**†	[65.78, 68.30]	**23.18**	[21.09, 25.40]	5.71	[4.75, 6.72]
GPT-5 Nano	Base	63.00	[61.67, 64.33]		–		–
	CoT	64.04	[62.71, 65.37]	18.82	[17.09, 20.66]	9.40	[8.41, 10.45]
	Self-refine	63.20	[61.87, 64.53]	11.89	[10.47, 13.40]	6.67	[5.84, 7.56]
	External judge	68.64	[67.31, 69.90]	21.62	[19.77, 23.56]	**3.75**	[3.14, 4.46]
	TTH (Ours)	**68.68**	[67.42, 69.94]	**24.22**	[22.19, 26.37]	5.21	[4.22, 6.26]
RealWorldQA ($n = 628$)							
Claude 4.5 Haiku	Base	62.74	[58.92, 66.56]		–		–
	CoT	60.67	[56.69, 64.49]	10.26	[6.56, 14.29]	9.39	[6.61, 12.34]
	Self-refine	63.85	[60.03, 67.51]	27.35	[21.67, 33.07]	14.47	[11.11, 18.01]
	External judge	60.83	[57.01, 64.65]	**49.15**	[42.68, 55.56]	32.23	[27.65, 36.85]
	TTH (Ours)	**69.11**†	[65.45, 72.77]	31.20	[24.64, 37.57]	**8.38**	[5.15, 11.52]
Gemini 2.5 Flash Lite	Base	65.13	[61.47, 68.79]		–		–
	CoT	66.72	[63.06, 70.38]	13.24	[8.91, 17.89]	4.65	[2.72, 6.78]
	Self-refine	68.15	[64.49, 71.65]	14.16	[9.70, 18.94]	2.94	[1.46, 4.69]
	External judge	71.34	[67.68, 74.84]	34.70	[28.16, 40.92]	8.89	[6.20, 11.77]
	TTH (Ours)	**76.59**†	[73.25, 79.93]	**34.70**	[27.31, 42.36]	**0.98**	[0.24, 2.01]
GPT-5 Nano	Base	70.06	[66.40, 73.56]		–		–
	CoT	71.50	[67.84, 75.00]	27.13	[20.88, 33.69]	9.54	[6.88, 12.35]
	Self-refine	71.97	[68.47, 75.47]	26.60	[20.35, 33.16]	8.64	[6.11, 11.32]
	External judge	**80.25**	[77.07, 83.28]	**49.47**	[42.31, 56.60]	6.60	[4.39, 9.02]
	TTH (Ours)	75.64	[72.14, 78.98]	26.60	[19.72, 33.55]	**3.41**	[2.01, 4.70]
Visual7W ($n = 5{,}000$)							
Claude 4.5 Haiku	Base	71.60	[70.34, 72.86]		–		–
	CoT	74.68	[73.48, 75.88]	21.13	[19.06, 23.31]	4.08	[3.46, 4.78]
	Self-refine	73.86	[72.66, 75.06]	25.35	[23.12, 27.65]	6.90	[6.10, 7.77]
	External judge	78.20	[77.06, 79.34]	**47.89**	[45.30, 50.47]	9.78	[8.83, 10.79]
	TTH (Ours)	**81.26**†	[80.18, 82.34]	37.04	[34.13, 39.97]	**1.20**	[0.87, 1.56]
Gemini 2.5 Flash Lite	Base	75.44	[74.24, 76.64]		–		–
	CoT	78.48	[77.34, 79.56]	20.33	[18.15, 22.67]	2.59	[2.10, 3.14]
	Self-refine	81.84	[80.76, 82.92]	31.71	[29.14, 34.38]	1.84	[1.42, 2.30]
	External judge	82.66	[81.58, 83.67]	**45.53**	[42.78, 48.34]	5.25	[4.58, 5.99]
	TTH (Ours)	**85.22**†	[84.21, 86.23]	45.30	[41.98, 48.81]	**1.78**	[1.39, 2.10]
GPT-5 Nano	Base	80.20	[79.06, 81.28]		–		–
	CoT	81.86	[80.78, 82.94]	25.20	[22.54, 27.98]	4.15	[3.56, 4.80]
	Self-refine	83.82	[82.81, 84.83]	26.26	[23.55, 29.12]	**1.97**	[1.58, 2.44]
	External judge	**85.08**	[84.07, 86.03]	**41.41**	[38.38, 44.48]	4.14	[3.56, 4.80]
	TTH (Ours)	84.44	[83.36, 85.45]	38.84	[36.08, 41.81]	4.30	[3.65, 5.01]

E Ablation Details

Untrained Qwen3. To test whether the gains of TTH come from its training pipeline rather than from the presence of any contrastive hint, we use the off-the-shelf Qwen3-VL-4B-Instruct, i.e., the initialization of $\mathcal{G}_\theta$ before Step 3, as the hint writer. It is prompted to write a contrastive hint for each test question, without access to the ground-truth answer or any training. TTH is more accurate than Untrained Qwen3 on all three targets and roughly doubles its repair rate at comparable harm (Table 4). For 72 of the 1,145 test questions (6.3%), the untrained model's hint showed a clear inclination toward one of the answer options despite instructions not to reveal an answer. This option reflects the untrained model's own guess and is not necessarily correct. We keep these questions in its reported accuracy.

Static-hint baselines. The categorical failure-mode hint is an oracle: its label is assigned by the GPT-5 annotator of Section 2, which sees the ground-truth answer and the target's incorrect response, and it is only prepended to questions that the target answers incorrectly at base, so it is not available at deployment. For base-correct questions, the target receives the CoT prompt without a hint. The universal taxonomy hint prepends the same 12-mode checklist (Section B.1) to every question.

Generator size. We did not select the 4B generator through a size sweep and do not claim that it is optimal. We chose Qwen3-VL-4B-Instruct because it can run on a personal computer and fits a modest training budget. This choice shows that anticipatory hinting can be distilled into a compact local model that may be deployed in compute-limited settings.

F Answer-Leakage Audits

Editor screening. The editor enforces no answer leakage together with the other hint constraints (contrastive form, no certainty language, and no extraneous reasoning). It required revisions for 8.4% of first-round proposals. After the proposer–editor loop, 2.6% of all proposed hints still failed its checks and were dropped, so no rejected hint was used to train $\mathcal{G}_\theta$.

Manual audit. We randomly sampled 100 hints generated by $\mathcal{G}_\theta$ for the A-OKVQA test set and inspected each alongside its question and answer options. None revealed or semantically entailed the correct answer.

Text-only answer-recovery probes. If a hint encoded the identity of the correct option, a learner that never sees the image or the question could recover the answer from the hint and the answer options more accurately than from the options alone. We therefore train probes on the official A-OKVQA training split and evaluate them on the held-out test split. The control probe receives only the four answer options, and the treatment probe additionally receives the target-conditioned hint generated by $\mathcal{G}_\theta$. We test three probe capacities. The *frozen linear* probe uses a frozen `all-MiniLM-L6-v2` sentence encoder with an L2-regularized multinomial logistic-regression head. The *frozen MLP* probe uses the same frozen encoder with a trainable $384 \to 256 \to 4$ GELU MLP. In the *end-to-end* probe, MiniLM and the MLP head are fine-tuned jointly so that the representation itself can adapt to any answer-bearing pattern. Results are means over three optimization seeds (3407, 42, and 1234), with 95% confidence intervals from 20,000 hierarchical paired-bootstrap replicates over seeds and test questions. Across all three hint sets and probe capacities, adding the hint never significantly improves recovery over the options-only control or over the 25% chance level (Table 11), and increasing probe capacity through end-to-end fine-tuning does not change this conclusion.

Table 11: **Text-only answer-recovery probes.**

Target Model	Probe	Options only accuracy (%)		Hint + options accuracy (%)	
Claude 4.5 Haiku	Frozen linear	24.56	[22.03, 27.10]	22.38	[20.02, 24.83]
	Frozen MLP	25.67	[23.25, 28.09]	23.60	[20.86, 26.34]
	End-to-end	27.36	[24.56, 30.19]	28.55	[22.09, 37.76]
Gemini 2.5 Flash Lite	Frozen linear	24.56	[22.03, 27.10]	23.69	[21.24, 26.14]
	Frozen MLP	25.67	[23.25, 28.09]	23.43	[20.89, 25.99]
	End-to-end	27.36	[24.56, 30.19]	23.86	[21.53, 26.22]
GPT-5 Nano	Frozen linear	24.56	[22.03, 27.10]	26.14	[23.60, 28.67]
	Frozen MLP	25.67	[23.25, 28.09]	23.51	[20.98, 26.08]
	End-to-end	27.36	[24.56, 30.19]	30.48	[22.90, 37.33]

G Inference and Training Cost

Inference. Per test query, TTH runs one forward pass of the 4B hint generator followed by one call to the black-box target, with no iterative target feedback. Base and CoT make one target call, self-refine makes two sequential target calls, and the external-judge baseline makes two target calls and one call to a more capable judge model, all in sequence. Because the hint generator is much smaller and faster than the frontier targets and runs before a single target call, TTH avoids the sequential rounds of post-response critique. The compute of the hint generator must still be counted, but it amounts to a single pass through a compact model that can run on local hardware.

One-time training cost. Constructing the training data for the 17,056 A-OKVQA training questions cost approximately $260 in API calls (Table 12), including repeated target calls and extended revision rounds in the agentic loop. This cost is driven mainly by the choice of target model rather than by the method: at the prices used for this estimate, Claude 4.5 Haiku costs roughly 10–20× more than the other targets, and running the identical pipeline over the same questions with only Gemini 2.5 Flash Lite and GPT-5 Nano as targets brings the total to roughly $150. The rest of the budget is dominated by the Step 2 proposer and editor, which are independent of the target and could be reduced further with a cheaper proposer or fewer revision rounds. The cost is incurred once, offline, when the training data for the hint generator is constructed (about 1.5 cents per training question), and it does not recur at deployment. It also scales with the size of the training set, so adapting TTH to a smaller domain-specific dataset would cost correspondingly less. Training the hint generator itself took about 11 hours on a single H200 GPU (2 hours for Stage 1 and 9 hours for Stage 2).

Table 12: **Approximate one-time API cost (USD) of constructing the training data.**

Step	Cost	Main expense
Step 1: base-response sampling	$70	$54 in Claude 4.5 Haiku target calls
Step 2: agentic hint generation and optimization	$150	$100 in GPT-5 proposer and editor calls
Step 3: hint generator training	$40	$30 in Claude 4.5 Haiku target calls
Total	$260	

H Gains and Base Accuracy

We examine how the gains of TTH relate to base accuracy across the 17 target–benchmark combinations in Tables 1 to 3: the eight targets on A-OKVQA and the three primary targets on each of VCR, RealWorldQA, and Visual7W. For each combination, let B denote base accuracy and G the accuracy of TTH minus base accuracy, both in percentage points. We report Pearson's r for the linear association, Spearman's ρ as a rank-based check, and two-sided permutation p-values over the 17 combinations. The analysis asks whether stronger base models show smaller absolute gains and, if so, whether this reflects shrinking error headroom or weaker correction of the errors that remain.

Absolute gains shrink near ceiling. B and G are moderately negatively associated (Pearson $r = -0.60$ with $p = 0.012$, and Spearman $\rho = -0.61$ with $p = 0.012$), and a least-squares fit associates each 10-point increase in base accuracy with about 1.9 points less gain. As an intuitive summary, the four combinations with base accuracy of at least 85% gain 0.98 points on average, compared with 6.17 points for the other 13.

The reduction reflects shrinking headroom rather than weaker correction. With repair rate R and harm rate H (both in percent), the gain decomposes as $G = ((100 - B)R - BH)/100$, so the repairable portion $100 - B$ necessarily shrinks as B grows. To separate this mechanical ceiling from correction quality, we normalize the gain by the remaining error headroom, $G/(100 - B)$, whose association with base accuracy is weak and not significant ($r = -0.19$, $p = 0.46$). Direct checks agree: the repair rate does not decline with base accuracy (a nonsignificant positive correlation, $r = +0.41$, $p = 0.10$), and the harm rate shows a nonsignificant negative correlation ($r = -0.42$, $p = 0.09$).

Higher base accuracy therefore places a practical ceiling on the absolute improvement because fewer errors remain to be repaired, while the ability of TTH to correct the remaining errors does not diminish. The net benefit of TTH is accordingly largest where base performance is less saturated, such as in domain- or knowledge-specific applications.

I Analysis of Harmful Hints

To understand how hints can hurt, we manually examined 40 cases in which TTH changed a base-correct response to an incorrect one. Three patterns emerged, listed in decreasing order of frequency:

1. **Unhelpful foreground redirection.** The hint asks the target to inspect a salient foreground element that is not useful for answering the question, distracting it from the relevant evidence.
2. **Failure despite useful guidance.** The hint correctly directs attention to relevant evidence, but the target nevertheless flips to an incorrect response. These cases are consistent with target-model capability limits or response instability rather than a factual defect in the hint.
3. **Fabricated, unnecessary details.** The hint introduces unverifiable features that are not needed to answer the question, causing the target to search for nonexistent evidence or become distracted.

Harmful cases are thus most often associated with attention misallocation or target-side instability, while outright fabrication in the hint is less frequent.

J Qualitative Examples of Hint-Driven Error Repair

This appendix presents one representative repair example for each of the three primary target models, drawn from the A-OKVQA test set. Each example shows a question the base model answered incorrectly, the hint produced by $\mathcal{G}_\theta$, and the corrected response obtained after prepending that hint to the prompt. In every case the hint directs attention toward discriminative visual evidence without revealing the correct answer, which illustrates the contrastive design discussed in Section 3. The full code, prompts, and additional examples are provided in the supplementary material.

J.1 Claude 4.5 Haiku

Question. `Who are these three people? A. visitors B. customers C. firefighters D. passengers`

Ground-truth answer. `A.`

Base response (incorrect).

- **Answer:** `C.`
- **Reasoning:** `The image shows three people sitting in a red vehicle with "SCARBOROUGH FIRE DEPT" written on it. The vehicle appears`



Figure 4: Qualitative example for Claude 4.5 Haiku.

to be a fire truck, suggesting they are likely firefighters or participating in an event related to the fire department. The presence of the fire department's name on the vehicle strongly indicates their affiliation.

Generated hint.

```
{"hint": [
"Firefighters vs visitors: check whether they wear uniforms/helmets or just casual summer clothes and sunglasses.",
"Working crew vs ride-along passengers: see whether they are operating gear or simply seated for a ride/photo.",
"Focus on the people, not the truck branding; read the text but verify what their clothing and activity suggest."
]}
```

Hinted response (correct).

- **Answer:** A.
- **Reasoning:** The image shows three people sitting in a red vehicle with "SCARBOROUGH FIRE DEPT." written on it. They are dressed in casual clothing and sunglasses, suggesting they are not actively working as firefighters. Their relaxed demeanor and the context of being in a fire department vehicle imply they are likely visitors or participants in an event.

J.2 Gemini 2.5 Flash Lite



Figure 5: Qualitative example for Gemini 2.5 Flash Lite.

Question. `Why is the guy standing in the median? A. awaiting green-light B. washing windshields C. panhandling D. selling flowers`

Ground-truth answer. `A.`

Base response (incorrect).

- **Answer:** `C.`
- **Reasoning:** `The person is standing in the median of a busy street with traffic. They are not actively engaged in any activity like washing windshields or selling flowers. Their posture and location suggest they are soliciting money from passing drivers.`

Generated hint.

```
{"hint": [
"Waiting to cross vs panhandling: check whether he's at the curb by
the crosswalk under the signal, facing traffic, or positioned in the
middle of the lane with a raised hand and any sign of begging.",
"Verify the traffic light above him: is it red/stop or
green/advance?",
"Scan his hand and the ground for any flowers or items he might be
selling."
]}
```

Hinted response (correct).

- **Answer:** `A.`
- **Reasoning:** `The image shows a person standing in the median of a street at a crosswalk. The pedestrian signal displays as a red hand, indicating that it is not safe to cross. This suggests the person is waiting for the light to change to a walk signal.`

J.3 GPT-5 NANO



Figure 6: Qualitative example for GPT-5 Nano.

Question. `What vehicle is the man riding? A. tractor B. bike C. plane D. car`

Ground-truth answer. `A.`

Base response (incorrect).

- **Answer:** `B.`

- **Reasoning:** `The man is seated on a four-wheeled vehicle (quad bike/ATV) behind him, which aligns with the "bike" option.`

Generated hint.

```
{"hint": [
"Tractor vs bike: check whether the vehicle has a seat, handlebars,
and two wheels, or a large engine with a front loader and wide rear
tires.",
"Focus on the wheels and the presence of handlebars.",
"Plane/car vs ground vehicle: confirm it's on the grassy ground
rather than in the air or on a road."
]}
```

Hinted response (correct).

- **Answer:** `A.`
- **Reasoning:** `The man is sitting on a red farm vehicle behind the cow, which resembles a small tractor rather than a bicycle, car, or plane. The handlebars and farm-oriented build indicate a tractor.`

K Extended Comparison with Related Work

Prompt optimization (APE, OPRO, and DSPy). Prompt optimizers and TTH both improve behavior through prompts, but they solve different problems under different deployment constraints. APE (Zhou et al., 2023) uses demonstrations to propose and score candidate instructions, then applies one selected task-level instruction to all test inputs. This helps when a shared instruction is the bottleneck. In natural-image VQA, however, the target already knows the task, while its likely error depends on the image and question, and a shared instruction cannot identify what a particular input is likely to make the target mishandle. Our ablation supports this point: neither the universal 12-mode taxonomy nor an oracle failure category improves accuracy reliably (Section 5.5), plausibly because a category such as spatial relation is too vague without the current visual context. OPRO (Yang et al., 2024) searches for a task-level instruction by giving an optimizer LLM a history of prompts and scores. The search can run offline, but its deployed result is again one shared prompt. Using it for instance-specific repair would require evaluating candidates for each test image, which needs a correctness signal that is unavailable before the answer, as well as repeated target calls. DSPy (Khattab et al., 2024) compiles a user-defined LM program against examples and a metric. It could wrap a VQA API and optimize task-level instructions or demonstrations, but it does not itself identify the current image's likely target-specific failure. Matching TTH would require the designer to construct multimodal failure-diagnosis and conditional-repair modules. TTH instead amortizes this optimization into an input-conditioned generator that learns the mapping directly from image–question inputs and target outcomes, and it deploys with one compact-generator pass and one black-box target call, without prompt history or per-query search.

Test-time and black-box prompt adaptation. B^2TPT (Meng et al., 2025) learns low-dimensional text and vision prompts for CLIP-style classification at test time: it forms pseudo-labels from target outputs and evaluates candidate prompts iteratively with derivative-free evolutionary optimization. AOZPT (Fang et al., 2026) learns prompts sequentially from noisy performance feedback, combining zeroth-order updates with uncertainty scaling to obtain sublinear regret. Visual prompt-agnostic evolution (Wang et al., 2026a) inserts learnable visual prompt tokens into a frozen ViT and models their layerwise evolution with a shared Koopman operator and Lyapunov regularization, which requires the backbone architecture and internal token sequence that a closed API does not expose. In contrast, TTH learns its generator offline, uses no target output, confidence score, or online correctness signal at test time, and emits an input-specific natural-language hint before the target responds, with one target call per example.