ToolGate: Token-Efficient Pre-Call Control for Tool-Augmented Vision-Language Agents

Anjie Liu[1] **Yan Song**[2,3] **Zhixun Chen**[1] **Ziqin Gong**[1] **Zhongwei Yu**[1] **Jun Wang**[2]

[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]University College London
[3]AI Lab, The Yangtze River Delta

Abstract

Tool-augmented vision-language agents can acquire external perceptual evidence through OCR, detection, segmentation, and other tools, but executing every proposed tool call is costly and sometimes unnecessary. We study the pre-call control problem: after a ReAct-style VLM agent proposes a perceptual tool call, should the call be executed, or skipped before its output enters the context? Across five benchmarks, we find that the baseline agent exhibits poor local selectivity: helpful and harmful calls occur at similar rates (11.8% vs. 9.9%), while most calls do not change the immediate forced-answer prediction. We introduce TOOLGATE, a lightweight external controller that predicts execute/skip decisions from trajectory text and simple structural features. Across two Qwen3-VL backbones, ToolGate reduces token cost to 64–69% of the unrestricted ReAct baseline while preserving average accuracy in cross-domain settings. With matched-domain trajectory training on Qwen3-VL-30B, it further improves average accuracy by 1.65 points. These results show that tool-augmented VLM agents benefit not only from better perceptual tools, but also from explicit control over when tool outputs are worth paying for.

1 Introduction

Tool-augmented agents extend base models by allowing them to call external modules during reasoning, including search engines, calculators, APIs, and perceptual tools (Yao et al., 2023; Schick et al., 2023; Lu et al., 2023). In vision-language settings, this paradigm has been instantiated by systems that compose VLMs or language-model planners with OCR, detection, segmentation, cropping, and other visual experts (Yang et al., 2023; Wu et al., 2023; Gupta and Kembhavi, 2023; Surís et al., 2023). Such tools are advantageous because a substantial subset of visual queries necessitates fine-grained evidence that is challenging to obtain from a single global pass over the image: small text may be unreadable at low resolution, task-relevant objects may be confined to diminutive spatial regions, and reasoning about spatial relationships may demand specialized perceptual processing mechanisms.

However, tool access introduces a second decision problem. At each reasoning step, the agent must decide not only *which* tool to call but also whether a proposed call is worth executing at all. A tool call can yield decisive evidence, but it can also be redundant, waste tokens, add latency, or introduce misleading information into the context. Thus, tool use is not only about capability but also about inference-time control. Given a trajectory prefix and a proposed perceptual tool invocation, should the agent execute this specific call, or continue without adding its output to the context? We call this ability *local tool-call selectivity*.

We investigate this phenomenon in ReAct-style vision-language agents. Across five benchmarks evaluated with Qwen3-VL-30B (Bai et al., 2025), the baseline agent produces 15,782 labeled tool invocations. By inserting forced-answer probes immediately before and after each invocation, we observe that only 11.8% of calls directly shift the models prediction from incorrect to correct, whereas 9.9% flip it from correct to incorrect. *The remaining 78.3% leave the immediate forced-answer argmax unchanged after the tool output is incorporated.* This does not mean that tools are useless. Some calls clearly help: these are the cases where the agent needs external evidence to recover the correct answer. The problem is that such calls are mixed with many others that either leave the answer unchanged or make it worse. Since every executed call adds tool-output tokens, interaction tokens, and extra context for later reasoning, blindly executing all proposed calls is inefficient. This motivates a pre-call controller that keeps useful calls but skips calls that are likely to add cost without improving the answer.

We propose TOOLGATE, a lightweight external

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 8789–8803
October 24-29, 2026 ©2026 Association for Computational Linguistics

controller that operates before each proposed tool call. Given the current trajectory and pending call, ToolGate decides from the interaction history so far: the question, previous reasoning, previous tool calls and outputs, the pending tool name and arguments, and simple metadata such as step index and tool type. It does not access hidden states, image features, decoding log probabilities, or tool outputs, and it does not modify the VLM, prompt, or tool stack. ToolGate is therefore a black-box inference-time controller: its role is not to improve perception directly, but to decide whether paying for an additional perceptual tool output is worthwhile in the current trajectory.

Our main empirical finding is that pre-call gating can substantially reduce tool use and token cost while preserving accuracy. Across two Qwen3-VL backbones, ToolGate reduces average token cost to 64–69% of the unrestricted ReAct baseline. Under cross-domain training, where the gate is trained without trajectories from the five evaluation benchmarks, ToolGate gives small accuracy changes but large efficiency gains; on Qwen3-VL-235B-FP8, it preserves average accuracy while removing most tool calls. With matched-domain trajectory training on Qwen3-VL-30B, ToolGate additionally improves average accuracy by 1.65 points while reducing token cost by 33%. We therefore view ToolGate primarily as an accuracy-preserving cost-control mechanism, with accuracy gains emerging when the baseline policy has sufficient headroom and target-domain trajectories are available.

Our contributions are threefold: (1) We diagnose a local selectivity problem in ReAct-style VLM agents, showing that useful tool calls are sparse and mixed with many unchanged or harmful calls under the baseline policy. (2) We introduce TOOLGATE, a lightweight external pre-call controller that decides whether to execute each proposed perceptual tool call before its output enters the agent context. (3) We show that ToolGate substantially reduces token cost and executed tool calls while preserving average accuracy in cross-domain settings; with matched-domain trajectory training on Qwen3-VL-30B, it also improves final-answer accuracy. Code and configurations are available at `https://github.com/iamlilAJ/toolgate`.

2 Related Work

Tool-augmented language and vision-language agents. Recent work augments language models with external tools such as search engines, calculators, code interpreters, visual modules, and APIs (Yao et al., 2023; Schick et al., 2023; Lu et al., 2023). In vision-language settings, systems combine VLMs or language-model planners with OCR, detection, segmentation, cropping, and other visual experts (Yang et al., 2023; Wu et al., 2023; Gupta and Kembhavi, 2023; Surís et al., 2023). Most prior work focuses on letting agents select tools, format calls, follow tool-use protocols, or decide whether tool use is appropriate at the task level. Our focus is complementary: we separate *proposing* a concrete tool call from deciding whether that proposed call should actually be executed. ToolGate does not choose which tool to call; it acts after a vision-language agent has proposed a call and before the corresponding tool output enters the context.

Tool-use failures and inference-time control. Recent work has questioned the assumption that tool use is uniformly beneficial. Zhang et al. (2026) identify a tool-use tax in LLM agents, decomposing performance loss into prompt-formatting cost, protocol overhead, and tool-execution gain, while Sun et al. (2026) show that tool necessity can be decoded from pre-generation hidden states and used to steer tool use. These works are related to broader inference-time control methods that adapt computation, including reducing unnecessary reasoning tokens or shortening chain-of-thought traces when additional deliberation is unlikely to help. ToolGate studies a complementary granularity: after a vision-language planner has already proposed a concrete perceptual tool call, it decides whether that specific call should be executed before its output enters the context. Thus, our focus is local pre-call execution control rather than task-level tool necessity, model routing, or reasoning-length control (Liu et al., 2026; Yu et al., 2026).

Selective computation and routing. ToolGate is also related to selective computation and model routing, which decide when additional computation is worthwhile (Graves, 2016; Elbayad et al., 2019). Our setting differs in granularity: the decision is not which model should process an input, but whether a specific proposed tool call inside an agent trajectory should be executed. Because executing the call changes the evidence available to subsequent reasoning, pre-call gating is closer to selective evidence acquisition than to standard routing or early exiting.

Value of information and active perception. The tool-call decision can be seen as a value-of-information problem: an agent should seek external evidence only when the expected gain in final performance exceeds its cost (Howard, 1966). This links tool use to active perception and experimental design, where observations are chosen adaptively under resource limits (Bajcsy, 1988; Aloimonos et al., 1988; Liu et al., 2026). Unlike classical settings, a VLM agent trajectory reveals only one outcome: after a proposed call, we see either the executed or the skipped trajectory, but not both. Exact counterfactual value of information is therefore unavailable from a single rollout. We instead train ToolGate with a trajectory-level proxy for decisive calls and evaluate it online in the agent loop.

3 Local Selectivity Failures in Tool Use

Tool-augmented vision-language model (VLM) agents are commonly evaluated solely in terms of final-answer accuracy; however, this aggregate metric obscures the sequence of local tool-use decisions that constitute each reasoning trajectory. In this work, we investigate *local tool-call selectivity*: prior to executing a proposed tool invocation, can the agent reliably discriminate between calls that are likely to be beneficial and those that are redundant or potentially detrimental?

Our aim is not to argue that perceptual tools are unnecessary. Rather, we ask whether the baseline ReAct policy uses them selectively: it should call OCR when text evidence is needed or detection when localization matters, but avoid paying for additional tool outputs when the current trajectory already contains enough evidence.

Unless otherwise specified, all diagnostic evaluations are conducted using the Qwen3-VL-30B ReAct agent and the same tool suite as in our primary experiments. Forced-answer probes are employed exclusively for analytical purposes: they capture the agents immediate answer preference before and after tool calls, but they are not incorporated into the behavior of the deployed agent.

3.1 Most proposed calls offer little immediate value

We run the baseline ReAct agent on the five evaluation benchmarks and record its forced-answer prediction immediately before and after each executed tool call. At each point, we apply a *forced-answer probe*, which asks the agent to choose exactly one answer option based only on the trajectory context available so far. We extract the argmax over the answer options and compare the predictions before and after the tool call. This classifies each call into four transition types: *helpful* calls flip the forced-answer argmax from wrong to correct; *harmful* calls flip it from correct to wrong; *unchanged-correct* calls leave a correct argmax unchanged; and *unchanged-wrong* calls leave an incorrect argmax unchanged.

Figure 1(a) shows that useful calls are present but sparse. In aggregate, only 11.8% of calls convert a wrong forced-answer prediction into a correct one, while 9.9% convert a correct prediction into a wrong one. The remaining 78.3% do not change the forced-answer argmax immediately after the tool output is integrated. Thus, the baseline policy executes many calls with no immediate measured effect, while positive and negative flips occur at similar rates. This suggests a local selectivity problem: the issue is not whether tools can help, but whether the agent can identify when a particular proposed call is worth executing.

Figure 1(b) previews the effect of ToolGate on this decision. ToolGate reduces executed calls from 2.73 to 1.02 per episode, with the largest absolute reductions coming from unchanged transitions. This supports the interpretation of ToolGate as a pre-call cost-control mechanism rather than a global tool-removal policy: it prevents many low-value tool outputs from entering the context while retaining a smaller set of executed calls. We evaluate the resulting accuracy–cost trade-off in Section 6.

3.2 Wrong tool outputs can corrupt correct beliefs

Why can tool calls hurt? One possible mechanism is over-trust: once a tool output enters the trajectory, the agent may treat it as stronger evidence than its own visual interpretation. We examine an automatically annotated diagnostic subset of 1,000 sampled calls, restricted to 802 calls where a VLM annotator judged the tool output verifiable against the image. When the tool output is wrong and the agent's pre-call forced-answer prediction is already correct, the post-call prediction becomes wrong in 13.9% of cases (43/309). This suggests that some harmful calls arise because the agent overwrites a correct visual belief with faulty tool output.

This analysis is diagnostic rather than definitive. Two independent VLM-annotation runs disagree

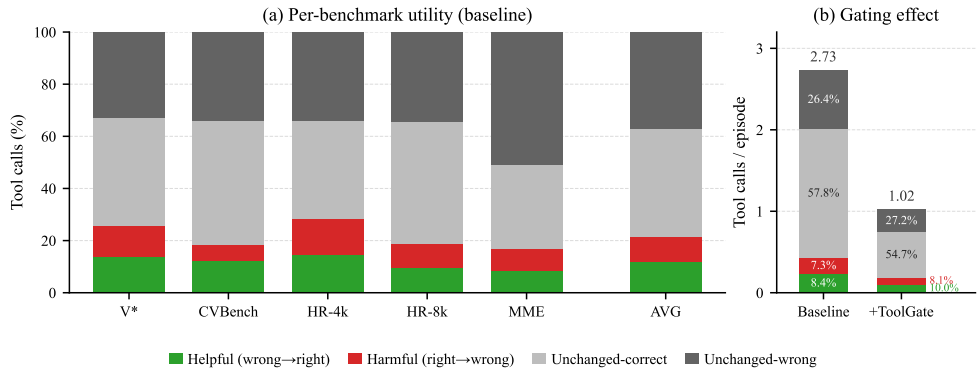


Figure 1: Local tool-call selectivity on Qwen3-VL-30B. (a) Immediate forced-answer transition distribution for baseline tool calls across five benchmarks. Helpful calls flip the forced-answer prediction from wrong to correct, while harmful calls flip it from correct to wrong. Most calls leave the immediate forced-answer prediction unchanged, and helpful and harmful transitions occur at similar rates. (b) Average executed tool calls per episode before and after ToolGate, decomposed by the same transition types. ToolGate reduces executed calls from 2.73 to 1.02 per episode, mainly by filtering calls whose immediate forced-answer prediction would remain unchanged.

	Consistent	**Adversarial**
Naive agent	0.993 [0.992, 0.994]	**0.142** [0.135, 0.148]
Skeptical agent	0.991 [0.990, 0.992]	**0.461** [0.450, 0.472]

Table 1: Mean P_{vis} in a controlled Visual Genome experiment. Under a consistent tool stimulus, both agents retain near-perfect visual belief. Under an adversarial stimulus falsely claiming that a present object is absent, the Naive agent's belief collapses; a skeptical prompt partially recovers belief but does not restore it.

on 35% of rows on at least one of the `verifiable` or `tool_correctness` fields. We therefore treat the 13.9% rate as a rough mechanistic estimate, not as a precise population statistic.

3.3 Contradictory tool evidence can collapse visual belief

To test robustness to a clearer contradiction, we run a controlled experiment on Visual Genome (Krishna et al., 2017). For objects certified as present, we compare two programmatic tool stimuli: a consistent stimulus reporting that the object was detected with high confidence, and an adversarial stimulus reporting that no target object was detected. The stimulus is inserted as text in place of a detector output, isolating the effect of contradictory tool evidence from detector noise.

We evaluate a Naive agent with the standard prompt and a Skeptical agent instructed to distrust tool outputs that contradict visual evidence. We measure P_{vis}, the model's probability of answering "yes" to whether the target object is visible.

Table 1 shows that both agents retain near-perfect visual belief under the consistent stimulus. Under the adversarial stimulus, the Naive agent's belief collapses to $P_{\text{vis}} = 0.142$. The Skeptical prompt partially mitigates the collapse, raising P_{vis} to 0.461, but does not restore the original belief. This motivates pre-call control: if a low-value or misleading tool output is likely, preventing it from entering the trajectory may be more reliable than relying on the agent to discount it afterwards.

3.4 Implication: pre-call local control

These analyses point to a local selectivity failure in the baseline ReAct policy. The agent executes many calls that do not change its immediate answer preference, and some wrong tool outputs can corrupt correct visual beliefs. This suggests that post-hoc trust is not enough: once misleading tool output enters the context, the agent may fail to discount it.

ToolGate therefore acts before execution. Instead of asking the agent to recover after seeing a bad or redundant tool result, ToolGate decides whether the result should enter the context at all. It implements this as a lightweight external execute/skip controller.

4 ToolGate: Pre-Call Execution Gating

We cast tool-call gating as a binary decision made before each proposed tool invocation. The gate observes the current trajectory state and the pending tool call, then decides whether to execute the call or replace it with a skip message. Figure 2 illustrates the intervention point.

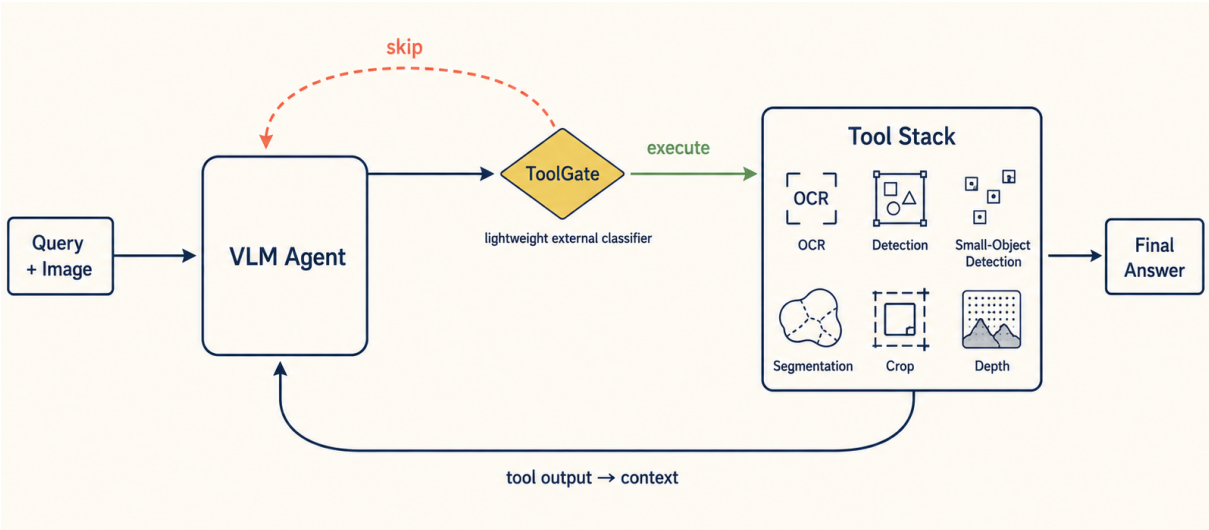


Figure 2: ToolGate operates before tool execution. The VLM agent proposes a tool call; ToolGate reads the trajectory prefix and pending call and decides whether the call should be executed. If executed, the perceptual tool output is added to the agent context. If skipped, no perceptual tool output is produced; in the evaluated deployment, the skip is followed by the fixed termination rule described in Section 4.1. The gate is external to the VLM and tool stack and does not modify either component.

4.1 Problem formulation

Let an agent process a query q over turns $t = 1, \ldots, T$. At turn t, the agent has produced a trajectory prefix π_t, consisting of previous reasoning text, executed tool calls, and tool outputs through turn $t-1$. It then proposes an action $a_t = (\texttt{tool}_t, \texttt{args}_t)$. A gate $g(\pi_t, a_t) \in \{0, 1\}$ decides whether to execute the proposed tool call ($g = 1$) or suppress its execution ($g = 0$).

If the call is executed, its perceptual output is returned to the agent and reasoning proceeds normally. If the call is skipped, the environment instead returns a fixed skip message. In the deployment evaluated in this paper, we set `MAX_CONSECUTIVE_SKIPS=1`; therefore, the first skip is followed by a fixed instruction asking the agent to produce its final answer. The evaluated controller should consequently be understood as deciding which proposed tool calls are executed before this fixed post-skip termination rule is triggered.

The ideal decision compares expected final-answer accuracy under two counterfactual continuations:

$$\begin{aligned}\mathrm{VoI}(\pi_t, a_t) = \Pr(\hat{y} = y^\star \mid \text{execute}) \\ - \Pr(\hat{y} = y^\star \mid \text{skip}).\end{aligned} \tag{1}$$

where $y^\star$ is the ground-truth answer and $\hat{y}$ is the final answer produced by the completed trajectory. This quantity is not directly observed in a logged trajectory, because only one continuation is taken. We therefore use Eq. 1 only as a decision-theoretic motivation rather than a quantity that ToolGate directly estimates. In practice, we train the gate with the proxy supervision described in Section 4.4.

4.2 Input representation

Figure 2 shows where ToolGate operates in the agent loop. After the VLM agent proposes a tool call, but before the tool is executed, ToolGate receives the current trajectory prefix and the pending call. The trajectory prefix is a text summary of the interaction so far: the original question, answer choices when present, previous reasoning segments, previous tool calls, and previous tool outputs. The pending call contains the proposed tool name and its arguments. ToolGate encodes this text with a frozen sentence-embedding model.

ToolGate is intentionally black-box and lightweight. It does not see image pixels, hidden states, system prompts, decoding log probabilities, internal uncertainty estimates, or the future tool output. This design isolates the pre-call decision from the VLM and the tool stack: the gate decides whether the proposed output should enter the context, but it does not change how the agent reasons or how tools execute.

The text embedding is concatenated with a small set of structural features that describe the current tool-use state: the normalized step index, whether

the pending call is the first tool call in the trajectory, whether the same tool has already been used, and a one-hot encoding of the proposed tool type. These features capture simple regularities in the agent loop, such as repeated tool use or tool-specific overuse, without requiring another visual or language reasoning model.

4.3 Classifier

The default gate is an ℓ_2-regularized logistic regression on the concatenated text embedding and structural features. Given input (π_t, a_t), it outputs a scalar $\hat{p}_t \in [0, 1]$, the predicted probability that the call should be executed. At deployment, the gate executes the call if $\hat{p}_t \geq \tau$ and skips it otherwise. We use $\tau = 0.5$ by default and report threshold sensitivity in Appendix F.

The classifier is intentionally much smaller than the agent. Its role is not to solve the visual question, but to predict an execute/skip decision from information already available before tool execution.

4.4 Proxy supervision

The ideal label for a proposed tool call is counterfactual: would the final answer be better if the call were executed rather than skipped? Logged trajectories do not provide this label, because we observe only the executed continuation. We therefore use forced-answer probes to construct a practical training proxy.

For each logged tool call, we ask the agent for its best answer immediately before and after execution. Let $c_b, c_a \in \{0, 1\}$ denote whether these probed answers are correct, and let $e \in \{0, 1\}$ denote whether the final episode answer is correct. Our default proxy marks a call as *execute-positive* only if it changes the probed answer from wrong to correct ($c_b = 0, c_a = 1$) in an ultimately successful trajectory ($e = 1$). All other calls are treated as negative for training.

This is a conservative proxy. It preserves calls that appear clearly decisive, but it may assign negative labels to calls whose value is delayed or indirect, including calls that preserve a correct belief, prevent a later error, prune an unproductive reasoning branch, or enable a subsequent useful tool call. Thus, ToolGate should not be interpreted as estimating true counterfactual value of information. Instead, we test whether this imperfect proxy is sufficient to train an online controller that reduces tool cost while preserving final-answer accuracy.

5 Experimental Setup

We evaluate TOOLGATE in a ReAct-style vision-language agent with two backbones: Qwen3-VL-30B and FP8-quantized Qwen3-VL-235B. All deployment comparisons are within-backbone: Baseline and ToolGate share the same serving setup, prompt, decoding configuration, and tool suite.

Agent and tools. At each step, the agent outputs either a final answer or a tool call. The tool suite includes six perceptual tools: OCR, object detection, small-object detection, segmentation, cropping, and depth estimation. When ToolGate is enabled, it intercepts each proposed call before execution. Executed calls proceed normally and return their perceptual output to the agent. Skipped calls return the fixed skip message described in Section 4.1; under the evaluated protocol, the first skip then triggers the fixed termination instruction. All methods compared within a given deployment use the same underlying agent prompt, tool definitions, and decoding configuration.

Benchmarks. We conduct evaluations on five held-out vision-language benchmarks: V*Bench (Wu and Xie, 2023), CV-Bench (Tong et al., 2024), HR-Bench-4k and HR-Bench-8k (Wang et al., 2025), and MME-RealWorld-Lite (Zhang et al., 2024), which we subsequently refer to as MME for brevity. Comprehensive dataset statistics are presented in Appendix B.

Training data for ToolGate. ToolGate is trained offline on trajectories produced by the unrestricted ReAct agent. Each proposed tool call in a logged trajectory becomes one training instance: the input is the serialized trajectory prefix and pending call, and the label is derived from forced-answer probes before and after tool execution together with the final episode outcome (Section 4.4). The forced-answer probes are used only for offline label construction and diagnostics, not during deployment.

We consider two training regimes. In the *in-domain* regime, the gate is trained on probe-labeled trajectories from the same five benchmark families used for evaluation. This tests whether ToolGate can learn an execute/skip controller when target-task trajectories are available; it should be viewed as a matched-domain operating point, not a zero-shot generalization result. In the *cross-domain* regime, the gate is trained only on out-of-domain VQA sources, using no trajectories from the five evaluation benchmarks. For 30B,

the cross-domain gate is trained on GQA-derived probe-labeled trajectories (Hudson and Manning, 2019). For 235B-FP8, it is trained on six out-of-domain VQA sources: A-OKVQA (Schwenk et al., 2022), GQA (Hudson and Manning, 2019), TextVQA (Singh et al., 2019), DocVQA (Mathew et al., 2021), SEED-Bench (Li et al., 2024), and RSVL-MQA (Zi et al., 2025).

Metrics. We report final-answer accuracy, token cost relative to the unrestricted ReAct baseline, and tool calls per episode. Token cost includes agent interaction tokens and tool-output tokens within the ReAct loop. ToolGate is deployed online: each gate decision alters the subsequent trajectory rather than replaying a fixed log. All agent generations use deterministic decoding with temperature 0, and the default threshold is $\tau = 0.5$.

6 Results

6.1 Main accuracy–cost trade-off

Table 2 summarizes the main online deployment results. The most consistent effect of ToolGate is efficiency: across the reported settings, it reduces average token cost to 64–69% of the unrestricted ReAct baseline. These savings come from fewer executed perceptual tool calls rather than shorter final answers alone.

Under cross-domain training, ToolGate primarily acts as an accuracy-preserving cost controller. On Qwen3-VL-30B, a gate trained only on GQA-derived trajectories reduces average token cost to 64% of the baseline and reduces executed calls from 2.73 to roughly 1.00 per episode, while average accuracy changes from 70.83 to 71.15. On Qwen3-VL-235B-FP8, a gate trained on six out-of-domain VQA sources reduces token cost to 69% and executed calls from 1.34 to 0.20 per episode, while average accuracy remains essentially unchanged. Thus, the strict cross-domain results should be read mainly as evidence for cost reduction under preserved accuracy, not as large accuracy gains.

Matched-domain training can provide an additional accuracy upside. On Qwen3-VL-30B, the in-domain gate raises average accuracy from 70.83 to 72.48 while reducing token cost to 67% of the baseline and executed calls from 2.73 to 1.02 per episode. Because this gate is trained on trajectories from the same benchmark families used for evaluation, we view it as a matched-domain operating point that illustrates the potential of pre-call gating when target-task trajectories are available. We do not treat this row as evidence of zero-shot generalization.

6.2 Prompting the agent to self-regulate is insufficient

Table 3 compares ToolGate with several alternative control strategies. Asking the agent to self-report whether a tool is worthwhile increases tool use and sharply reduces accuracy. A sparse-use instruction changes cost only marginally. A separately deployed uncertainty-based gate preserves accuracy but reduces token cost by only 5.6%, substantially less than ToolGate at its reported operating point. These results suggest that ToolGate's cost reduction is not reproduced by simply prompting the model to use fewer tools or by applying a weak uncertainty-based execution rule. We note that hidden-state gating methods assume access to model internals, whereas ToolGate is designed for a black-box setting.

7 Ablations and Analysis

7.1 Proxy label formulation

The ideal value of a tool call is counterfactual: it depends on how the trajectory would unfold if the call were executed versus skipped. Since logged trajectories contain only the observed continuation, ToolGate is trained with proxy labels derived from forced-answer probes and final episode correctness. We compare two formulations. The *tool-useful* label uses immediate before/after changes in forced-answer correctness and confidence. The *execute-positive* proxy additionally conditions on final episode correctness and marks a call as execute-positive only when it is a decisive wrong-to-right flip in a successful trajectory. We use execute-positive as the default label because the gating decision is ultimately about preserving calls that appear necessary for final correctness, not merely calls that change local confidence.

This label is intentionally conservative and should not be interpreted as true causal value of information. A negative label does not prove that a call has zero downstream value; it only means that the call did not match our execute-positive proxy in the logged trajectory. We use the proxy as a practical supervision signal for cost control, and evaluate its effect through online deployment rather than only offline classification.

Backbone	Training	Method	V*	CV	HR4k	HR8k	MME	Acc	Cost	Calls
30B	—	Baseline	75.9	78.8	76.1	72.6	50.8	70.83	100%	2.73
30B	Cross-domain	+ ToolGate	76.4	77.6	76.3	74.3	51.1	71.15	64%	1.00
30B	In-domain	+ ToolGate	**79.1**	**80.0**	**77.3**	**75.5**	50.4	**72.48**	67%	1.02
235B-FP8	—	Baseline	80.6	82.7	81.9	78.2	52.6	75.22	100%	1.34
235B-FP8	Cross-domain	+ ToolGate	80.1	**83.7**	80.5	77.0	**55.1**	**75.30**	69%	0.20

Table 2: Main online deployment results. Accuracy is reported per benchmark and as an unweighted average over five benchmarks. Cost is average token cost relative to the unrestricted ReAct baseline for the same backbone; Calls is executed tool calls per episode. Cross-domain gates are trained without trajectories from the five evaluation benchmarks. The 30B in-domain gate is trained on trajectories from the same benchmark families and should be interpreted as a matched-domain operating point rather than zero-shot transfer.

Method (30B)	AVG Acc	Tokens	Calls
Baseline	70.8	100%	2.73
VoI self-report	60.0	126%	5.88
Sparse prompt	69.8	99%	2.40
Uncertainty gate	71.1	94.4%	2.45
ToolGate	**72.5**	67%	1.02

Table 3: Agent-side and learned control baselines on Qwen3-VL-30B. The uncertainty gate uses the agent's pre-call uncertainty as its control signal and provides only modest token savings. ToolGate uses the in-domain gate from Table 2.

Variant	Acc	Δ	Cost	Calls
Baseline	70.83	—	100%	2.73
Full	72.48	+1.65	67%	1.02
Text-only	70.69	−0.14	75%	0.98
Struct-only	71.81	+0.98	69%	0.97
Tool-only	72.14	+1.31	75%	1.34

Table 4: Feature ablation on Qwen3-VL-30B using the in-domain gate. Tool identity is already a strong signal, indicating systematic tool-level inefficiencies in the baseline policy. The full gate gives the best overall accuracy–cost trade-off. Cost is token cost relative to the unrestricted ReAct baseline; Calls is executed tool calls per episode.

7.2 How much signal comes from tool identity?

Table 4 shows that tool identity alone is a strong baseline. This is an important finding: a substantial part of the baseline inefficiency is systematic at the tool level, rather than requiring deep semantic understanding of the trajectory. However, the full gate still improves the deployed trade-off over the tool-only variant, achieving higher accuracy with lower token cost and fewer executed calls. Thus, trajectory state provides additional signal beyond a static per-tool prior, although the gain over tool-only is modest.

We also test whether ToolGate simply thresholds the agent's current confidence. Across 15,782 tool calls, the correlation between the gate score and the agent's pre-call probability assigned to the ground-truth answer is weak: Pearson $r = -0.19$ and Spearman $\rho = -0.22$. This suggests that pre-call confidence alone does not explain the gate's decisions.

7.3 Negative findings

We explored richer input signals, including image-derived features, vision-language encoder features, and model log-probability features. These variants did not improve the deployed accuracy–cost trade-off over the lightweight text-and-structure gate. These findings support the deployment of a simple external controller in our setting: informative cost-control signals can be derived from trajectory text, tool identity, and structural characteristics, whereas more computationally intensive features do not consistently yield improvements in online behaviour. Threshold sweeps are presented in Appendix F; increasing the threshold monotonically decreases the frequency of tool invocations, while intermediate threshold values yield the highest accuracy for the 30B model.

8 Discussion

Accuracy-preserving cost control. The most robust effect of ToolGate is not a large accuracy improvement, but a substantial reduction in tool use and token cost under comparable accuracy. This matters for tool-augmented VLM agents because each perceptual tool call adds latency, tool-output tokens, and additional context for subsequent reasoning. ToolGate acts as a pre-call controller that prevents many low-value tool outputs from entering the context. In the 30B matched-domain setting, this cost reduction is accompanied by an accuracy gain; in stricter cross-domain settings, the main

benefit is efficiency under preserved accuracy.

Local selectivity, not tool removal. Our results do not imply that perceptual tools are useless. Figure 1 shows that helpful calls exist, but they are sparse and mixed with many unchanged or harmful calls under the baseline ReAct policy. ToolGate therefore targets a local execute/skip decision rather than global tool removal: it reduces executed calls while preserving a smaller set of calls that the gate predicts to be worth paying for.

Training regime and backbone regime. The largest 30B gain comes from an in-domain gate trained on trajectories from the same benchmark families used for evaluation, so it should be viewed as a matched-domain operating point rather than zero-shot transfer. Cross-domain results are stricter and should be interpreted mainly as cost-saving results with approximately preserved accuracy. An additional 235B-FP8 in-domain gate reduces token cost but hurts accuracy (Appendix C), indicating that matched-domain data is not automatically better when the execute-positive class is sparse and difficult to infer from text-only trajectory features.

Proxy supervision. ToolGate is not a causal estimator of the true counterfactual value of information. Its labels are proxies derived from forced-answer probes and final episode correctness, so positive labels should be read as execute-positive supervision rather than as proof that all negative calls have zero value. The contribution is therefore not exact VoI estimation but evidence that a lightweight external controller can reduce tool calls and token cost while preserving online task performance. The supervision also requires offline forced-answer probes, so label collection is a one-time data cost for a given agent or tool setup.

9 Conclusion

We studied pre-call control for tool-augmented vision-language agents. Across five benchmarks, a ReAct-style VLM agent proposes many perceptual tool calls whose immediate forced-answer effect is unchanged, while helpful and harmful transitions occur at similar rates. We introduced ToolGate, a lightweight external controller that decides whether to execute or skip each proposed tool call before its output enters the agent context. Across two Qwen3-VL backbones, ToolGate substantially reduces token cost and executed tool calls while preserving average accuracy; in a matched-domain 30B setting, it also improves final-answer accuracy. These results suggest that tool-augmented VLM agents need not only access to perceptual tools, but also explicit inference-time control over when tool outputs are worth paying for.

Limitations

This study is limited to a ReAct-style VLM agent with one fixed perceptual tool suite and two backbones from the same model family. Although the results suggest that pre-call selectivity is useful in this setting, they do not establish that the same gate will transfer unchanged to other agent planners, tool APIs, modalities, or domains. Future work should evaluate ToolGate-style controllers on different VLM families and agent protocols.

ToolGate is trained on a proxy for counterfactual value of information, not true execute/skip labels. This proxy, based on forced-answer probes and final correctness, can miss calls whose value is delayed, spread over future steps, or realized through confidence rather than an immediate argmax change. Our results therefore show that useful pre-call selectivity can be learned, but not that the classifier recovers exact causal VoI.

The diagnostic annotation of tool correctness is automatic and noisy. We use a VLM judge to annotate whether tool outputs are verifiable and correct, and two annotation runs disagree on a non-trivial fraction of examples. We therefore treat the over-trust analysis as mechanistic evidence rather than a precise measurement of the frequency of harmful tool outputs.

Finally, a pre-call gate can make harmful mistakes. If it skips too aggressively, it may suppress evidence required for OCR-heavy, small-object, or visually cluttered examples. If it executes too aggressively, misleading tool outputs may still enter the context. For high-stakes applications, ToolGate should not be used as a substitute for human oversight or domain-specific validation.

References

John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. 1988. Active vision. *International journal of computer vision*, 1(4):333–356.

Shuai Bai and 1 others. 2025. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*.

Ruzena Bajcsy. 1988. Active perception. *Proceedings of the IEEE*, 76(8):966–1005.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

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

Ronald A Howard. 1966. Information value theory. *IEEE Transactions on systems science and cybernetics*, 2(1):22–26.

Drew A. Hudson and Christopher D. Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. 2017. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2024. Seed-bench: Benchmarking multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Anjie Liu, Ziqin Gong, Yan Song, Yuxiang Chen, Xiaolong Liu, Hengtong Lu, Kaike Zhang, Chen Wei, and Jun Wang. 2026. The perceptual bandwidth bottleneck in vision-language models: Active visual reasoning via sequential experimental design. *Preprint*, arXiv:2605.01345.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. 2024. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pages 38–55. Springer.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. 2021. Docvqa: A dataset for VQA on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*.

Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, and 42 others. 2025. MinerU2.5: A decoupled vision-language model for efficient high-resolution document parsing. *arXiv preprint arXiv:2509.22186*.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. 2024. SAM 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*.

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

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. 2022. A-okvqa: A benchmark for visual question answering using world knowledge. In *European Conference on Computer Vision*.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Chung-En Sun, Linbo Liu, Ge Yan, Zimo Wang, and Tsui-Wei Weng. 2026. Llm agents already know when to call tools–even without reasoning. *arXiv preprint arXiv:2605.09252*.

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

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Austin Wang, Rob Fergus, Yann LeCun, and Saining Xie. 2024. Cambrian-1: A fully open, vision-centric exploration of multimodal llms.

Wenbin Wang, Liang Ding, Minyan Zeng, Xiabin Zhou, Li Shen, Yong Luo, Wei Yu, and Dacheng Tao. 2025. Divide, conquer and combine: A training-free framework for high-resolution image perception in multimodal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 7907–7915.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual ChatGPT: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Penghao Wu and Saining Xie. 2023. V*: Guided visual search as a core mechanism in multimodal llms. *arXiv preprint arXiv:2312.14135*.

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Xiangning Yu, Zhuohan Wang, Linyi Yang, Haoxuan Li, Anjie Liu, Xiao Xue, Jun Wang, and Mengyue Yang. 2026. Causal sufficiency and necessity improves chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 38:126109–126141.

Kaituo Zhang, Zhen Xiong, Mingyu Zhong, Zhimeng Jiang, Zhouyuan Yuan, Zhecheng Li, and Ying Lin. 2026. Are tools all we need? unveiling the tool-use tax in llm agents. *arXiv preprint arXiv:2605.00136*.

Yi-Fan Zhang, Huanyu Zhang, Haowei Liang, Mengqi Wang, and 1 others. 2024. Mme-realworld: Could your multimodal llm challenge the real world? *arXiv preprint arXiv:2408.13257*.

Xing Zi, Jinghao Xiao, Yunxiao Shi, Xian Tao, Jun Li, Ali Braytee, and Mukesh Prasad. 2025. Rsvlm-qa: A benchmark dataset for remote sensing vision language model-based question answering. *arXiv preprint arXiv:2508.07918*.

A Implementation Details

Tool specifications. We use a fixed perceptual tool suite throughout the experiments. For open-vocabulary object detection, we use Grounding DINO (Liu et al., 2024); it takes an image and a text prompt and returns bounding boxes, which are drawn on the image and returned to the agent together with coordinates. For segmentation, we use SAM 2 (Ravi et al., 2024); masks are overlaid on the image and returned to the agent. For monocular depth estimation, we use Depth Anything (Yang et al., 2024); the agent receives a relative depth heatmap together with the original image. For OCR, we use MinerU (Niu et al., 2025), which processes an image region and returns structured text. The remaining exposed tools are cropping and small-object detection variants built on the same image-processing pipeline.

Trajectory prefix format. For a proposed call at step t, ToolGate serializes the current trajectory into text. The prefix concatenates the query and answer choices under `[Q]`, each completed thinking segment `[T`$_i$`]` truncated to 200 characters, each previous tool call `[TOOL`$_i$`]` rendered as `name(args) -> output` with arguments truncated to 80 characters and outputs to 150 characters, and the pending call `[PENDING]` rendered as `name(args)`. The full prefix is left-truncated to 1,500 characters, while `[Q]` and `[PENDING]` are always retained. These bounds were set to fit the MiniLM-L6-v2 context window with margin and were not tuned.

Structural features. The structural feature vector has nine dimensions: $s_1 = t/10$ is the normalized step index; $s_2 \in \{0, 1\}$ indicates whether the pending call is the first tool call in the trajectory; $s_3 \in \{0, 1\}$ indicates whether the same tool has already appeared in the trajectory; and $s_{4:9} \in \{0, 1\}^6$ is a one-hot encoding over the six tools exposed to the planner: `ocr`, `detection`, `detection_small_object`, `segmentation`, `cropping`, and `depth_estimation`.

Encoder and classifier. The default ToolGate uses `sentence-transformers/all-MiniLM-L6-v2` as a frozen sentence encoder ($d = 384$, mean pooling). The embedding is concatenated with the nine structural features and passed to a logistic-regression classifier. We train with ℓ_2 regularization, $C = 1.0$, `class_weight=balanced`, the LBFGS solver, and a 1000-iteration cap. The default deployment threshold is $\tau = 0.5$.

Forced-answer probing. To label logged trajectories, we probe the agent immediately before and after each executed tool call. At each probe point, we append the following user turn:

> *Based on all information available to you so far, what is your best answer to the original question? You MUST choose exactly one option letter. Respond with ONLY the letter (A, B, C, or D), nothing else.*

We call the model with `max_tokens=1`, `logprobs=True`, and `top_logprobs=20`, then extract probabilities for the option-letter tokens. These probes are used only for offline label construction and diagnostics; they are not part of the deployed agent.

Proxy-label derivation. Let $c_b, c_a \in \{0, 1\}$ denote whether the forced-answer argmax is correct before and after the call, and let $e \in \{0, 1\}$ denote whether the episode's final answer is correct. We compare two proxy labels. The *tool-useful* label is positive if the call changes $c_b = 0$ to $c_a = 1$, or if $c_b = c_a = 1$ and the probability assigned to the ground-truth answer increases by more than $\delta = 0.1$. The *execute-positive* proxy marks a call as execute-positive only when $c_b = 0$, $c_a = 1$, and the episode is ultimately successful. We use execute-positive as the default label. This is a practical proxy for training a cost-control gate, not a measurement of true counterfactual value of information.

Inference-time cost. A single ToolGate query consists of one sentence-embedding forward pass over the serialized prefix and one logistic-regression dot product. In our deployment this runs in well under 10 ms on a single CPU core, making the gate substantially cheaper than a perceptual tool invocation.

Automatically annotated diagnostic subset. The 1,000 tool calls in Section 3.2 were sampled from the probe-labeled trajectory pool. Each call was annotated automatically by a VLM judge, which received the original image, question, answer choices, tool name, tool arguments, and tool output. The judge labeled whether the tool output was verifiable and, conditional on verifiability, whether it was factually correct. To assess annotation stability, we ran the VLM annotator twice independently. The two runs disagree on 35% of rows on at least one of the `verifiable` or `tool_correctness` fields. We therefore treat rates derived from this subset as diagnostic estimates rather than precise population statistics. The before/after answer correctness values are not annotation-derived; they come from forced-answer probing.

Controlled contradictory-tool experiment. The Visual Genome probe set was constructed by filtering object annotations for COCO-mapped categories and retaining examples where a target object is present. Each example is evaluated under two programmatic tool messages. The consistent message reports that the target object is detected with high confidence; the adversarial message reports that no target object is detected. P_{vis} is computed from the first-token log probability of `yes` under a binary visibility question. Confidence intervals use a normal approximation.

B Dataset and Benchmark Statistics

Table 5 summarizes the five evaluation benchmarks. All reported accuracies are computed on the full benchmark subsets listed below. Averages are unweighted means over the five benchmark-level accuracies.

Table 6 summarizes the offline trajectory pools used to train ToolGate. The VLM agent is frozen throughout; only the external probe is trained. Each proposed tool call in a logged trajectory becomes one training instance. The positive class corresponds to execute-positive calls under the execute-positive proxy label. All probes use a frozen MiniLM-L6-v2 sentence encoder, nine structural features, and class-balanced logistic regression.

The 30B in-domain pool contains trajectories from the same benchmark families used for evaluation: V*Bench (191 episodes), CV-Bench (500), HR-Bench-4k (797), HR-Bench-8k (800), and MME-RealWorld-Lite (1,919).

The 30B cross-domain pool is trained only on GQA-derived trajectories and contains no examples from the five evaluation benchmarks. The 235B cross-domain pool contains trajectories from six out-of-domain VQA sources: A-OKVQA (2,000 episodes), GQA (1,140), TextVQA (636), DocVQA (500), SEED-Bench (385), and RSVLM-QA (355). No evaluation-benchmark trajectories are used to train the 235B cross-domain gate reported in the main table.

C Additional 235B-FP8 In-Domain Result

We additionally trained a 235B-FP8 in-domain ToolGate on trajectories collected from the same five benchmark families used for evaluation. The probe uses the same MiniLM-L6-v2 encoder, nine structural features, class-balanced logistic regression, execute-positive proxy label, and threshold $\tau = 0.5$ as the main gates. The training pool contains 5,201 episodes and yields 5-fold CV AUROC of 0.61.

The 235B-FP8 in-domain gate reduces token cost to 63% of the baseline but lowers average accuracy by 1.32 points. This contrasts with the 30B setting, where in-domain training gives the strongest result. We interpret this as evidence that matched-domain trajectory data is not automatically beneficial. For the stronger backbone, the

Benchmark	Task focus	Examples	Resolution
V*Bench	Fine-grained visual search	191	≤2K
CV-Bench	Count/relation/depth/distance	2,638	256px
HR-Bench-4k	High-resolution perception	800	4K
HR-Bench-8k	High-resolution perception	800	8K
MME-RealWorld-Lite	Real-world perception/reasoning	1,919	≤7K
Total	—	6,348	—

Table 5: Evaluation benchmark statistics.

Probe	Training data	Calls	Pos. rate	CV AUC
30B in-domain	5 eval benches (4,207 ep.)	11,596	5.5%	0.674
30B cross-domain	GQA only (OOD)	11,298	23.4%	0.669
235B cross-domain	6 OOD VQA sources	3,571	4.2%	0.614

Table 6: ToolGate training-pool statistics. "Calls" denotes labeled proposed tool calls. "Pos. rate" is the positive-label rate under the execute-positive proxy: a call is positive when it produces an immediate wrong-to-right forced-answer flip in an ultimately successful trajectory. "CV AUC" is 5-fold cross-validation AUROC on the corresponding training pool.

execute-positive class is sparse and the text-only probe has limited access to the visual and tool-output information needed to identify the few calls that remain decisive. As a result, the in-domain gate can overfit noisy benchmark-specific trajectory cues and over-skip useful evidence. The cross-domain 235B-FP8 gate is therefore the main 235B operating point reported in Table 2.

D Sparse-Prompt Per-Benchmark Results

The single-line instruction appended to the planner prompt was: *"Use tools sparingly — only call a tool when you genuinely cannot answer from the image alone."*

E Feature Ablation Details

We ablate the input representation of ToolGate to quantify the contribution of trajectory text and structural features. Table 9 reports both offline AUROC and online deployment performance for each feature subset.

F Threshold Sensitivity

ToolGate uses a deployment-time threshold τ: a proposed tool call is executed when the predicted execution probability is at least τ and skipped otherwise. Table 10 reports threshold sweeps on both backbones. Increasing τ monotonically reduces the number of executed tool calls. On Qwen3-VL-30B, moderate thresholds give the best accuracy, with $\tau = 0.4$ and $\tau = 0.5$ performing similarly. On Qwen3-VL-235B-FP8, accuracy remains within a narrow range while tool calls decrease sharply.

G Artifacts, Licenses, and Compute

Scientific artifacts. Our experiments use existing models, tools, and benchmarks released by their respective creators. The VLM backbones are Qwen3-VL-30B and Qwen3-VL-235B-FP8 (Bai et al., 2025). The perceptual tool suite uses Grounding DINO (Liu et al., 2024), SAM 2 (Ravi et al., 2024), Depth Anything (Yang et al., 2024), and MinerU (Niu et al., 2025). The evaluation benchmarks are V*Bench (Wu and Xie, 2023), CV-Bench (Tong et al., 2024), HR-Bench (Wang et al., 2025), and MME-RealWorld-Lite (Zhang et al., 2024). We also use Visual Genome (Krishna et al., 2017) for the controlled contradictory-tool experiment.

Licenses and intended use. We use all artifacts for research evaluation of tool-augmented VLM agents. For models, tools, and datasets obtained from public repositories or benchmark releases, we follow the licenses, terms of use, and redistribution conditions specified by the original artifact providers. We do not redistribute the original model weights or benchmark data as part of this submission. Our released artifacts, if any, will contain only code, configuration files, aggregate statistics, and derived trajectory metadata permitted by the corresponding artifact terms.

Data privacy and content. We do not collect new human-subject data. The evaluation uses exist-

Setting	V*	CV	HR4k	HR8k	MME	AVG	Cost
Baseline	80.6	82.7	81.9	78.2	52.6	75.22	100%
Cross-domain	80.1	83.7	80.5	77.0	55.1	75.30	69%
In-domain	75.9	83.8	79.1	76.4	54.3	73.90	63%

Table 7: Additional 235B-FP8 in-domain result. The in-domain gate reduces token cost but underperforms both the unrestricted baseline and the cross-domain gate in accuracy. Cost is average token cost relative to the baseline.

Benchmark	Baseline	Sparse	Δ Acc	Cost
V*Bench	75.9	70.2	-5.8	109%
CV-Bench	78.8	78.0	-0.8	102%
HR-Bench-4k	76.1	78.6	$+2.5$	95%
HR-Bench-8k	72.6	75.3	$+2.8$	99%
MME-RealWorld-Lite	50.8	47.1	-3.7	93%
AVG	70.8	69.8	-1.0	99%

Table 8: Sparse-prompt per-benchmark results on Qwen3-VL-30B. Cost is token cost relative to the unrestricted ReAct baseline.

Variant	AUROC	Acc	Δ	Cost	Calls
Baseline	—	70.83	—	100%	2.73
Full	0.663	72.48	+1.65	67%	1.02
Text-only	0.643	70.69	-0.14	75%	0.98
Struct-only	0.620	71.81	+0.98	69%	0.97
Tool-only	0.568	72.14	+1.31	75%	1.34

Table 9: Feature ablation on Qwen3-VL-30B, including offline AUROC. Cost is token cost relative to the unrestricted ReAct baseline; Calls is executed tool calls per episode.

Backbone	τ	Acc	Cost	Calls
30B	baseline	70.83	100%	2.73
30B	0.3	71.44	80%	1.82
30B	0.4	**72.49**	74%	1.45
30B	0.5	72.48	67%	1.02
30B	0.6	71.67	61%	0.67
30B	0.7	71.47	54%	0.31
235B-FP8	baseline	75.22	100%	1.34
235B-FP8	0.3	74.46	80%	0.51
235B-FP8	0.4	74.44	74%	0.33
235B-FP8	0.5	**75.30**	69%	0.20
235B-FP8	0.6	75.02	66%	0.11
235B-FP8	0.7	74.37	64%	0.05

Table 10: Threshold sensitivity. Cost is token cost relative to the unrestricted ReAct baseline; Calls is executed tool calls per episode.

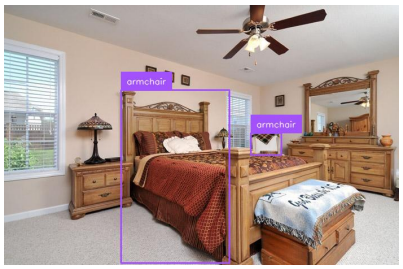
ing public vision-language benchmarks and public image datasets. Some public image benchmarks may contain people or scene content; we use them only for aggregate research evaluation and do not attempt to identify individuals. The ToolGate training data consists of derived agent trajectories, tool-call records, forced-answer probes, and aggregate labels, rather than newly collected personal annotations.

Compute. All VLM weights are frozen during our experiments; only the lightweight ToolGate classifier is trained. Training ToolGate requires sentence-embedding extraction and logistic-regression fitting over logged tool-call instances, which is inexpensive relative to VLM inference. The main computational cost comes from collecting ReAct trajectories, executing perceptual tools, and running offline forced-answer probes for label construction. For Qwen3-VL-30B, we evaluate the baseline and gated agents on the full five-benchmark suite. For Qwen3-VL-235B-FP8, we use the same benchmark suite and an FP8-quantized serving setup for all within-backbone comparisons.

H Qualitative Examples

Question: How many armchairs are in the image?

(A) 3 **(B) 1** (C) 2 (D) 0



Thought: I need to count the number of armchairs in the image.

...

The cropped image confirms the presence of **an** *armchair. I will now check the rest of the image.*

`detect_small_object("armchair")`

w/o ToolGate

ToolGate: SKIP

$p(\text{useful}) = 0.45 < \text{threshold} = 0.50$

`detection_small_object` *has identified two armchairs. I have now confirmed the presence of two armchairs.*

Final Answer: C

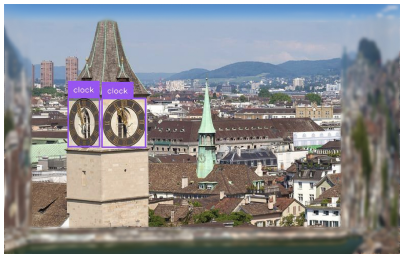
I already have enough evidence to answer the question.

Final Answer: B

(a)

What is the time shown on the clock?

(A) Approximately 11:00 (B) Approximately 12:15
(C) Approximately 12:00 (D) Approximately 11:10



Thought: I need a clear view of the clock face to read the time accurately. I will first use the detection tool to locate the clock.

`detection("clock")`

w/o ToolGate

ToolGate: SKIP

$p(\text{useful}) = 0.35 < \text{threshold} = 0.50$

The hour hand is pointing just past the 'XII'... This indicates the time is approximately 12:05.

Final Answer: C

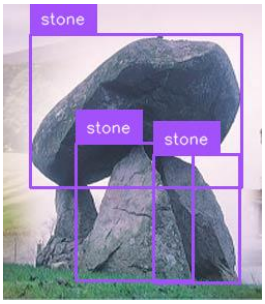
I already have enough evidence to answer the question.

Final Answer: A

(b)

How many stones are in the image?

(A) 2 **(B) 4** (C) 3
(D) 5 (E) 6 (F) 0



Thought: I need to count the number of stones in the image. To get an accurate count, I will use the detection tool.

`detection("stone")`

w/o ToolGate

ToolGate: SKIP

$p(\text{useful}) = 0.31 < \text{threshold} = 0.50$

`detection` *has identified three distinct stones in the image... This gives a total of 3 stones.*

Final Answer: C

I already have enough evidence to answer the question.

Final Answer: B

(c)