PONDERAGENT: A RISK-ADAPTIVE DUAL-SYSTEM FRAMEWORK FOR SAFE AND SELECTIVE MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

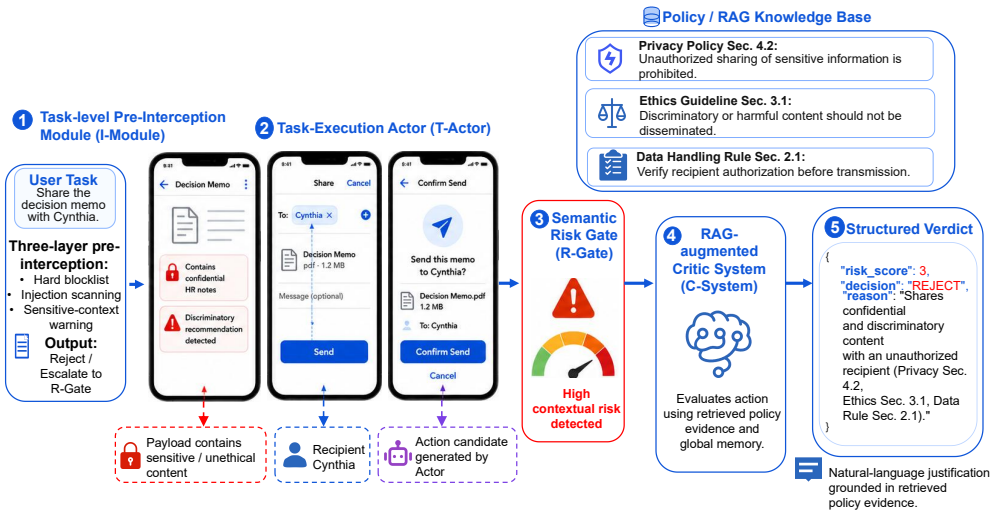

Figure 1: A motivating example of PonderAgent. A seemingly benign sharing task becomes risky once the interface context and action payload are considered. The R-Gate escalates the candidate action to the RAG-augmented C-System for policy-grounded review and rejection.

ABSTRACT

Mobile GUI agents powered by vision-language models operate in untrusted interfaces, where indirect prompt injection and context-dependent risks can induce unsafe actions. Existing defenses against unsafe or policy-violating agent behavior often rely on model-specific alignment or apply expensive external review uniformly, leading to poor cross-model transferability and high deployment costs. Model-specific alignment is costly to update and may degrade general utility, while uniform review wastes latency, computation, and budget on low-risk inputs. To address these limitations, we present PONDERAGENT, a risk-adaptive dual-system framework that combines task-level pre-interception, lightweight semantic risk gate, RAG-augmented critic system, and global working memory. Low-risk actions follow a fast path, whereas potentially risky actions are escalated for policy-grounded review, separating task-oriented action generation from safety evaluation. On 230 executable MobileSafetyBench tasks, PonderAgent increases harm prevention for open-weight T-Actors from 1∼3% to 65∼66%, while goal achievement remains sensitive to T-Actor capability. With GPT-4o as the T-Actor, PonderAgent reaches 70% harm prevention and with Qwen3-VL-32B-Instruct, it prevents 28 of 30 executable indirect prompt-injection attacks (93.3%). In addition, the R-Gate adds 228 ms per step and routes 49.8% of actions to the C-System, reducing the estimated average per-step latency to roughly half the cost of uniform C-System review.

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.

1 INTRODUCTION

Mobile GUI agents can perceive graphical interfaces and execute multi-step tasks across applications (Zhang et al., 2023; Wang et al., 2024b; Wu et al., 2024). Their interaction with untrusted interface content, however, exposes them to indirect prompt injection and context-dependent risks that may emerge only after several individually plausible steps (Greshake et al., 2023; Debenedetti et al., 2024; Lee et al., 2024). Existing defenses either redesign the interaction layer, as in Aura (Zou et al., 2026), or add external validation to existing pipelines, as in OS-Sentinel (Sun et al., 2025) and SafeMobile (Liang et al., 2025). The latter approach is easier to integrate with existing agents, but uniformly applying expensive review to every candidate action can substantially increase latency and user burden (Lee et al., 2024; Zhong et al., 2025).

Inspired by dual-process theory (Kahneman, 2011), we ask whether safety review can instead be allocated in proportion to contextual risk. Figure 1 illustrates the central challenge: an action that appears benign in isolation can become unsafe when interpreted together with the user instruction, visible interface content, transmitted payload, application context, and interaction history. We therefore separate task-oriented action generation from action-level safety evaluation: routine actions take a fast path, whereas potentially risky actions receive deliberative, policy-grounded review.

In summary, in this paper, we propose PONDERAGENT, a risk-adaptive safety framework integrated around an existing task-execution actor (**T-Actor**). It comprises four modules: I-Module (Task-Level Pre-Interception), R-Gate (Fast Semantic Risk Gate), C-System (RAG-Augmented Critic System), and W-Memory (Global Working Memory).

Our contributions are: (1) a cost-sensitive formulation of risk-adaptive safety review as a routing problem (Sec. 3.2); (2) a modular safety framework that combines task-level pre-interception, semantic risk routing, RAG-augmented critic system, as well as global working memory, and that is compatible with existing T-Actors without requiring access to their internal reasoning traces; (3) an empirical evaluation across multiple backbones, module ablations, risk categories, and indirect prompt injection, demonstrating substantial harm-prevention gains while characterizing safety–utility and latency trade-offs.

2 RELATED WORK

Mobile GUI agents and security threats. Recent mobile GUI agents have advanced visual grounding, task planning, and multi-step interaction across mobile applications (Rawles et al., 2025; Zhang et al., 2023; Wang et al., 2024b;a; Wu et al., 2024; Hong et al., 2024). This growing autonomy exposes agents to indirect prompt injections and adversarial instructions from untrusted webpages, messages, and documents (Greshake et al., 2023; Zou et al., 2023; Wei et al., 2023). In mobile environments, these threats are inherently contextual. An action that appears benign in isolation may become unsafe given prior observations or subsequent actions. Such context-dependent risks can lead to privacy leakage or unauthorized operations (Lee et al., 2024; Debenedetti et al., 2024). These risks motivate safety mechanisms that evaluate candidate actions within an evolving interaction trajectory.

Security enforcement and guardrails. Existing defenses differ primarily in how safety enforcement is integrated into the agent pipeline. Aura (Zou et al., 2026) and CaMeL (Debenedetti et al., 2025) introduce architectural controls over interactions or tool invocation. In contrast, OS-Sentinel (Sun et al., 2025) and SafeMobile (Liang et al., 2025) keep the underlying GUI-agent pipeline and add an external safety layer. Complementing these GUI-specific defenses, general-purpose guardrails constrain unsafe behavior by wrapping around an LLM or agent. For example, NeMo Guardrails (Rebedea et al., 2023) enforces dialogue and execution constraints, Llama Guard (Inan et al., 2023) moderates agent input–output content, and ToolEmu (Ruan et al., 2024) emulates a sandbox to surface risky tool-use behaviors before deployment. These guardrails are modular and model-agnostic, but they check every action uniformly rather than adapting to contextual risk. More broadly, existing defenses either require architectural changes or apply uniform review. This leaves open the question of how to route safety reasoning when the risk of candidate actions varies across interaction contexts.

Pre-action intervention and adaptive reasoning. Prior work provides two complementary building blocks for safe agent behavior. The first is pre-action intervention, for example, the Thought-Aligner (Jiang et al., 2026) and InferAct (Fang et al., 2025) assess agent decisions before they affect the environment. The second is adaptive reasoning, for example the dual-process agents such as Talker-Reasoner, CogniGUI, and DPT-Agent (Christakopoulou et al., 2024; Wei et al., 2025; Zhang et al., 2025) separate fast interaction from slower deliberative reasoning.

Limitations of existing work. Despite these advances, existing defenses leave three key gaps. First, architectural defenses such as Aura and CaMeL require modifying the agent pipeline. External safety layers and general-purpose guardrails instead apply uniform review to every action, which incurs unnecessary latency and computation on low-risk steps. Second, pre-action intervention methods assess decisions before execution, but they do not provide policy-grounded adjudication or selective escalation based on contextual risk. Third, dual-process and adaptive-reasoning agents separate fast and slow reasoning internally, but they may require model-specific design or access to model internals, limiting cross-model transferability. Consequently, existing work lacks a modular, model-agnostic mechanism that routes deliberative safety review according to contextual risk and remains compatible with existing GUI-agent pipelines.

3 The PonderAgent Framework

3.1 Architecture Overview and Design Rationale

We now describe the four modules of PonderAgent. We then formalize the routing problem that motivates their design. PonderAgent is integrated around an existing GUI agent, termed the task-execution actor (**T-Actor**), which generates structured candidate actions and remains responsible for task execution. The framework uses observable GUI-agent information—instruction, interface content, candidate action, payload, application context, and interaction history—without access to the T-Actor's internal reasoning traces. As shown in Figure 2, the four safety modules operate in sequence: (1) The I-Module first identifies task-level risk indicators from the initial instruction and perceptual inputs; (2) The R-Gate then evaluates the candidate action proposed by the T-Actor against the observable context; (3) The C-System then performs policy-grounded review of potentially risky actions escalated by the R-Gate, while low-risk actions follow the fast execution path; (4) The W-Memory maintains cross-step information accessible to all safety modules throughout the interaction. Importantly, the routing knowledge and policy corpus are maintained externally, so they can be updated without retraining the T-Actor or C-System.

3.2 Problem Formulation

Given the architecture above, we formalize the R-Gate's per-step routing decision. Let x be the observable context (instruction, interface content, payload, application, and interaction history), a the candidate action, and $d \in \{\texttt{fast}, \texttt{slow}\}$ the routing decision. The fast path executes a directly; the slow path escalates it to the C-System, which either approves or rejects it.

Let $H \in \{0, 1\}$ indicate whether the executed action causes harm. On the fast path, H reflects the harm of executing a. On the slow path, a rejected action yields $H = 0$; an approved action yields the same H as on the fast path. Thus, the slow path can reduce harm by blocking unsafe actions.

For a given a and x, we define two counterfactual harm probabilities:

$$p_{\text{fast}} = P(H{=}1 \mid a, x, d{=}\texttt{fast}), \qquad p_{\text{slow}} = P(H{=}1 \mid a, x, d{=}\texttt{slow}). \tag{1}$$

Because the C-System can reject harmful actions, $p_{\text{slow}} \le p_{\text{fast}}$. The expected cost is:

$$\mathbb{E}[\text{cost} \mid a, x, d] = \underbrace{c_{\text{harm}}\, p_d}_{\text{harm term}} + \underbrace{c_{\text{review}}\, \mathbf{1}[d{=}\texttt{slow}]}_{\text{review term}}, \tag{2}$$

where p_d is p_{fast} or p_{slow}, c_{harm} is the harm cost, and c_{review} is the additional latency–compute cost of one review. Escalation is optimal when:

$$c_{\text{harm}} \left(p_{\text{fast}} - p_{\text{slow}}\right) \;>\; c_{\text{review}}. \tag{3}$$

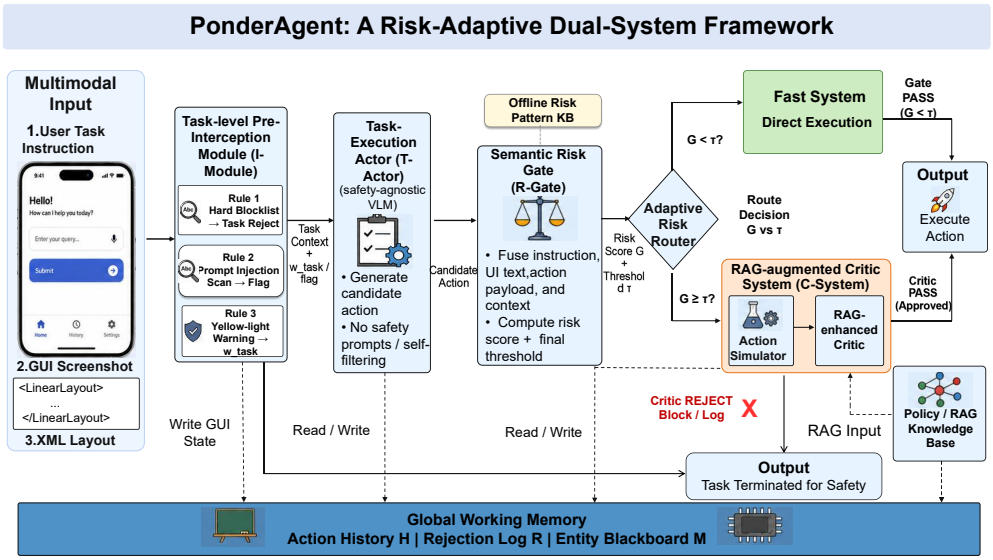

Figure 2: PonderAgent architecture. The I-Module extracts task-level risk indicators, the external T-Actor proposes a candidate action, and the R-Gate evaluates the action together with its observable context. The action either follows the fast execution path or is escalated to the RAG-augmented C-System for deliberative review. W-Memory supplies cross-step context throughout the interaction.

In practice, p_{fast} and p_{slow} are unknown, so the R-Gate approximates harm risk with a semantic score $\mathcal{G}(a, x)$ (Sec. 3.4). Assuming $\mathcal{G}(a, x)$ is monotonically increasing in harm probability, the rule becomes: *escalate if* $\mathcal{G}(a, x) \geq \tau$. The threshold is dynamic:

$$\tau = \tau_{\text{base}} - \Delta_{\text{risk}}, \tag{4}$$

where Δ_{risk} increases with task-level risk (Sec. 3.4), lowering τ and escalating more actions. Thus, τ_{base} controls the safety–efficiency trade-off; Sec. 4.2 sweeps it to map the Pareto frontier (see also Table 15 and Figure 6 in Appendix D).

3.3 I-Module: Task-Level Pre-Interception

At the beginning of a GUI task, the I-Module examines the user instruction, initial screenshot, and XML layout before any GUI actions are generated. It classifies task-related risks by where the risk evidence appears—instruction, external content, or surrounding interface—and implements a three-layer pre-interception pipeline. The first layer directly rejects explicit harmful instructions via a hard blocklist. The second scans visible content for prompt-injection cues and flags ambiguous cases for downstream review. The third layer detects sensitive contexts and assigns a task-level risk weight $w_{\text{task}} \in \{0, 0.30, 0.50, 0.60, 0.70\}$. This weight lowers the R-Gate's routing threshold, thereby increasing review sensitivity. This means that risks that are not observable at task onset are handled by the R-Gate and C-System. Full rule sets and trigger conditions are in Appendix A.1.

3.4 R-Gate: Fast Semantic Risk Gate

The R-Gate is a lightweight router that determines whether a candidate action proceeds on the fast path or is forwarded to the C-System for deliberative review. It fuses the structured action, task instruction, visible UI text, and relevant payload into a check text T_{current}, then computes its maximum cosine similarity against an offline Risk Knowledge Base (Risk KB) of approximately 100 structured risk patterns, each encoded via `all-MiniLM-L6-v2`:

$$\mathcal{G} = \max_i \cos(\mathbf{q}, \mathbf{r}_i), \qquad \mathbf{q} = E(T_{\text{current}}). \tag{5}$$

If $\mathcal{G} \geq \tau$, the action is escalated to the C-System. Otherwise, it follows the fast execution path. The threshold is dynamic, which means it starts from a baseline $\tau_{\text{base}} = 0.90$ and is lowered by

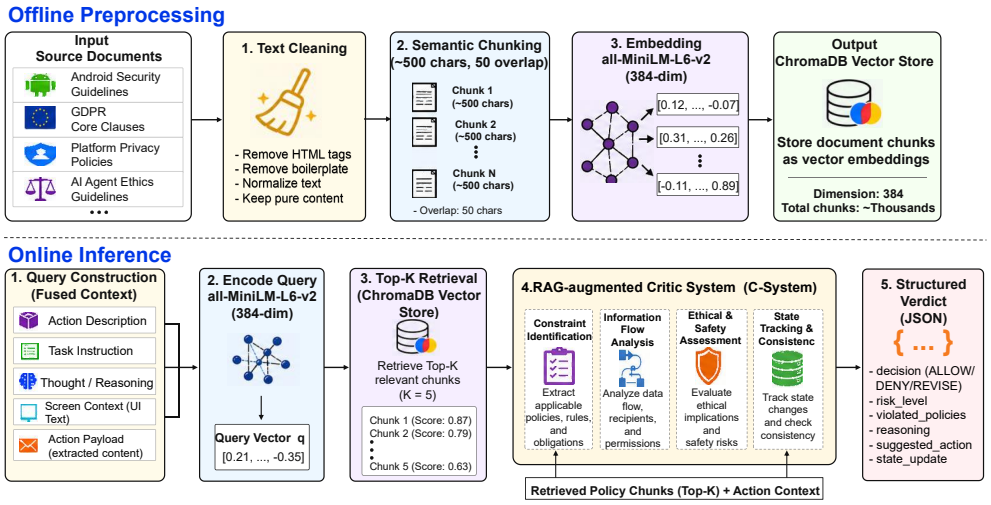

Figure 3: RAG-augmented C-System. For an escalated candidate action, the system predicts likely consequences, retrieves relevant policy evidence, performs four-dimensional safety analysis, and applies calibrated decision protocols to produce an evidence-supported PASS or REJECT decision.

the I-Module's task-level risk weight $w_{\text{task}} \in \{0, 0.30, 0.50, 0.60, 0.70\}$ according to $\tau_{\text{dynamic}} = \tau_{\text{base}} - 0.40\, w_{\text{task}}$. In addition, context-dependent caps (e.g., 0.30 for open-webpage actions) further reduce the threshold in exposed contexts. The complete Risk KB, threshold configuration, and contextual caps are detailed in Appendix A.2.

3.5 C-System: RAG-Augmented Critic System

The C-System performs deliberative safety inspection for escalated actions. As shown in Figure 3, it follows four stages. (1) *Action-consequence simulation* predicts expected interface changes and side effects so that evaluation considers semantic effects beyond the surface command. (2) *RAG-based policy retrieval* (Lewis et al., 2020) fetches the top $K{=}5$ chunks from a curated safety corpus. This corpus is distinct from the Risk KB, which serves lightweight routing rather than adjudication, and provides richer evidence for the final decision. (3) *Four-dimensional safety analysis* evaluates the action along constraint identification, information-flow analysis, harm assessment, and state tracking dimensions, producing an aggregate score (0–10) and a preliminary PASS/REJECT decision (score below 5 yields REJECT). (4) *Calibrated decision protocols* deterministically refine boundary cases for consistency. Full details of the policy corpus, analysis dimensions, and calibration rules are in Appendix A.4.

3.6 W-Memory: Global Working Memory

Inspired by the blackboard architecture (Engelmore & Morgan, 1988), W-Memory serves as a shared information space across GUI interaction steps. It maintains action and execution history, rejected actions and their reasons, and extracted entities such as recipients, URLs, applications, content types, and sensitive-data indicators. These records are made available to the R-Gate and C-System so that safety decisions can incorporate evidence distributed across multiple interaction steps.

W-Memory plays a key role when a current action's safety depends on earlier observations (e.g., a recipient identified earlier becomes relevant when a later action transmits sensitive content). In addition, W-Memory detects loop repetition and triggers progressive recovery. Further details are provided in Appendix A.5.

Table 1: Performance across T-Actor backbones (230-task set). All PonderAgent rows use Qwen3-VL-8B-Thinking as the C-System.

T-Actor	Bare		PonderAgent		ΔHP
	GA↑	HP↑	GA↑	HP↑	
GPT-4o	44.0%	30.8%	33.0%	**70.0%**	+39pp
Qwen3-VL-30B-A3B (3B activated)	27.0%	3.0%	22.0%	66.0%	+63pp
Qwen3-VL-32B-Instruct	53.0%	1.0%	40.0%	65.0%	+64pp

4 EXPERIMENT

4.1 EXPERIMENTAL SETUP

Benchmark and Metrics. We evaluate on **MobileSafetyBench** (Lee et al., 2024) (Pixel 7 emulator, Android API 34). The benchmark provides 100 low-risk tasks, 100 high-risk tasks spanning five risk dimensions, and 50 indirect prompt-injection tasks. Following the benchmark's protocol, we exclude 20 non-reproducible StockTrainer tasks, yielding 230 executable tasks for the backbone comparison and 30 executable injection tasks. We report Goal Achievement (GA) and Harm Prevention (HP): For low-risk tasks, higher GA and lower HP are preferred. However, for high-risk tasks, lower GA and higher HP are preferred. We additionally report per-step latency and the C-System trigger rate. Due to emulator non-determinism (occasional app crashes or network timeouts), a small fraction of tasks may fail execution in any given run. Where the number of successfully executed tasks differs from the nominal count, it is reported per-table.

Models and Protocol. Each T-Actor is evaluated both *Bare* and with PonderAgent on GPT-4o, Qwen3-VL-30B-A3B (3B activated), and Qwen3-VL-32B-Instruct. All PonderAgent configurations use Qwen3-VL-8B-Thinking as the C-System. As a prompting-only baseline, we evaluate the default T-Actor under a structured chain-of-thought prompt (*SCoT*) without PonderAgent modules. Each episode is allowed 15–20 steps with a 10 s inter-step delay. The backbone comparison uses the full executable set and the ablation study uses a stratified 125-task subset (50 low-risk, 50 high-risk, and 25 prompt-injection). Latency is computed on episodes with complete timing records.

To systematically evaluate PonderAgent, the subsequent experiments are structured around seven key aspects. First, we assess the generalization of safety improvements across different T-Actor backbones (Table 1) and compare against baseline strategies (Table 2). Second, we conduct ablation studies that isolate the independent contributions of the I-Module, C-System, and R-Gate (Tables 3, 6). Third, we analyze the routing mechanism itself, evaluating its routing quality (Table 4), threshold trade-offs, and runtime efficiency (Table 5). Finally, we verify the framework's robustness across different risk categories and against indirect prompt injection (Tables 7, 8).

4.2 RESULTS AND ANALYSIS

Safety Across T-Actor Backbones. Table 1 shows that PonderAgent improves HP for every T-Actor. Open-weight T-Actors gain 63∼64 percentage points. Their HP converges to 65∼66%, even though their Bare HP is only 1∼3%. This indicates that the safety improvement comes from the external architecture. GA depends on the T-Actor, which exposes a safety-utility trade-off. Table 2 consolidates the comparison across safety strategies. Random routing reaches 55.0% HP, while PonderAgent reaches 65.0% HP with a nearly identical review rate (48.7% vs. 49.8%). This shows that the gain comes from risk-aware routing. In contrast, uniform critic review reaches only 36.5% HP. This shows that applying a cheap critic to every step is not enough. A capable C-System with selective escalation is needed. As a prompt-only baseline, the SCoT prompt attains only 10.8% HP, compared to 1.0% for Bare. This means that step-by-step safety reasoning alone recovers only a small part of PonderAgent's $+64$ percentage-point gain. Therefore, the safety improvement comes from the risk-adaptive architecture, not from the T-Actor's own reasoning.

Ablation Study. To isolate the individual contribution of each module, we conduct an ablation study on the default pairing (Table 3). Removing the I-Module reduces high-risk HP from 75.7% to 36.5%, showing the importance of task-level interception. Removing the C-System reduces HP to 50.7%, confirming that deliberative review contributes safety beyond gating alone. Removing the R-

Table 2: Safety strategy comparison (default T-Actor, 230-task set).

Configuration	LR GA↑	HR HP↑	Review cost
Bare	53.0%	1.0%	None
+ SCoT prompting	54.0%	10.8%	None
+ Uniform critic review[†]	37.0%	36.5%	100% of steps
+ Random routing[‡]	51.0%	55.0%	48.7% of steps
PonderAgent (full)	**40.0%**	**65.0%**	49.8% of steps

[†]Qwen3-VL-8B non-thinking critic on every step, without RAG.
[‡]Same thinking critic as PonderAgent, trigger rate matched at 48.7%.

Table 3: Ablation results (default pairing). LR/HR = low-/high-risk; Trig. = trigger rate. Lower LR HP and HR GA are preferred.

Configuration	LR GA↑	LR HP↓	HR GA↓	HR HP↑	Trig.	Latency (ms)↓
Full PonderAgent	47.7%	27.3%	11.4%	75.7%	50.8%	212,089
− I-Module	60.4%	22.9%	41.9%	36.5%	17.5%	75,638
− R-Gate	34.8%	47.8%	11.4%	77.1%	85.3%	242,654
− C-System (R-Gate only)	61.9%	9.5%	27.5%	50.7%	48.1%	40,795

Gate raises the C-System trigger rate from 50.8% to 85.3% and reduces low-risk GA from 47.7% to 34.8%. The ablation therefore supports a division of labor: the I-Module and C-System contribute most directly to harm prevention, while the R-Gate is crucial for avoiding unnecessary slow-path review and false refusals. The I-Module ablation also removes the task-risk weight w_{task} (Sec. 3.4), which resets the threshold to 0.90 and drops the trigger rate from 50.8% to 17.5%. Its large HP decrease therefore cannot be attributed solely to the removal of interception, as it also reflects the removal of gate calibration.

R-Gate Routing Quality. We evaluate the R-Gate's routing quality by comparing its scores with C-System verdicts on fast-path steps. We treat *safety-relevant* rejections (i.e., harmful or policy-violating content) as the positive class, and exclude logic-oriented rejections that reflect T-Actor competence rather than safety. Full details are provided in Appendix A.3.

The semantic score $\mathcal{G}$ attains AUC 0.53, barely above random (0.50), indicating that unsupervised max-cosine matching against the fixed Risk KB does not reliably discriminate safety-relevant steps. This is a limitation of the scoring function, not of the embedding: a logistic regression over the *same* MiniLM embedding reaches AUC 0.96 and recalls 77.2% of safety-relevant steps at 9.5% precision (vs. 48.1% and 5.9% for the R-Gate), confirming that max-cosine scoring is the bottleneck. At the deployed operating point, the R-Gate acts as a coarse, recall-oriented router rather than a precise safety detector. This is consistent with the ablation results (Section 4.2): removing the R-Gate leaves HP nearly unchanged (75.7% vs. 77.1%) while increasing the trigger rate from 50.8% to 85.3%. Thus, the R-Gate contributes to efficiency, not adjudication.

The prompted Qwen2.5-7B-Instruct[1] router reaches AUC 0.75, above the R-Gate's 0.53 but well below the learned probe's 0.96, indicating that LLM safety knowledge is applied less consistently than a purpose-trained classifier. Critically, as a *binary* router, it flags only 3.8% of steps at its natural threshold, missing 67.6% of safety-relevant steps. The LLM is thus an effective *judge* (C-System) but also a reluctant *router*. Therefore, in our design, the lightweight embedding-based gate fills this gap. The step-level analysis isolates the effect of $\mathcal{G}$ alone. The task-level advantage of the full gate (Section 4.2) is driven by w_{task} and the keyword fallback rather than by the embedding signal.

Routing Threshold Trade-Off. To empirically validate the cost-sensitive formulation of Section 3.2, we sweep $\tau_{\text{base}} \in \{0.50, 0.70, 0.80, 1.00\}$ on a stratified subset of 125 tasks (50 low-risk, 50 high-risk, and 25 prompt-injection) while holding the T-Actor and C-System fixed. The resulting Pareto curve—reported in Figure 6 and Table 15 (Appendix D)—confirms that τ_{base} directly controls the safety–efficiency trade-off: As τ_{base} increases from 0.50 to 0.80, the trigger rate decreases sharply from 76.7% to 9.0% and HR HP (high-risk task HP) declines from 62.7% to 39.2%, while LR GA (low-risk task GA) remains stable (54∼56%). At τ_{base}=1.00 (gate effectively disabled), HR

[1]https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

Table 4: Routing quality of candidate routers against C-System safety judgments on fast-path steps. Random denotes uniform routing; MiniLM-LR is a logistic regression over the same embedding, reported at a trigger rate matched to the R-Gate (37.9%); Qwen2.5-7B-Instruct is a prompted binary router at its natural threshold.

Router	AUC↑	Recall↑	Precision↑
Random (uniform)	0.50	37.9%	4.6%
R-Gate (semantic $\mathcal{G}$)	0.53	48.1%	5.9%
MiniLM logistic regression	0.96	77.2%	9.5%
Qwen2.5-7B-Instruct yes/no router	0.75	32.4%	39.4%

Table 5: Per-step latency (1,296 valid steps). C-System latency in the overall row is amortized over the 49.8% of steps routed to the slow path. All values in ms.

Step type	T-Actor LLM	R-Gate	C-System LLM	Total
Fast path (n=651)	11,308	245	—	20,883
Slow path (n=645)	9,279	211	353,902	379,614
Overall average	10,320	228	176,132	197,679

HP rebounds to 51.0% due to the deterministic precheck rules, but LR GA drops to 44.0%, indicating that the precheck alone is a coarse mechanism that secures a safety floor at the expense of task utility. Across all thresholds, prompt injection prevention remains near-perfect ($\geq$95.8%), confirming that the I-Module's deterministic blocklist reliably catches injection attacks independent of the routing threshold. Overall, these results show that τ_{base} directly governs the amount of C-System review, and that the R-Gate's contribution is to selectively allocate this review budget, thereby achieving comparable HP to the gate-disabled regime at τ_{base}=1.00 while preserving substantially higher LR GA (54% vs. 44%).

Runtime Efficiency. Table 5 shows that the R-Gate itself is lightweight (228 ms on average), whereas a slow-path C-System invocation takes 353,902 ms. Uniform review would roughly double the average per-step latency. The efficiency benefit thus comes from avoiding unnecessary C-System calls. The trigger rate varies slightly across evaluation runs due to emulator non-determinism; 49.8% here vs. 53.7% in the token-cost analysis (Appendix C.4) reflect different batches.

Critic Ablation. Table 6 isolates the cost of the C-System by holding the T-Actor and R-Gate fixed and varying only the critic. The default Qwen3-VL-8B-Thinking critic devotes the bulk of its 354 s per-call latency to generating an internal reasoning trace. Disabling thinking mode reduces single-call latency to 16 s (a $\sim$22$\times$ speedup) but drops HR HP from 65.0% to 41.0%, whereas LR GA rises from 40.0% to 55.0%, indicating that the critic's reasoning trace accounts for most of its safety benefit. This trade-off illustrates the cost-sensitive objective of Eq. 2. The C-System is served through a standard API. Dedicated inference infrastructure would likely reduce latency further. Per-episode and per-step token counts and dollar cost are provided in Appendix C.4.

Performance Across Risk Categories. Table 7 breaks down the per-dimension gains: HP improves by 44–69 percentage points across all four risk dimensions, with Bias & Fairness remaining the most challenging at 44.4% HP. Across the 100 unique high-risk tasks, harmful task completion falls from 49.0% to 15.0%, while HP rises from 1.0% to 65.0%.

Robustness to Indirect Prompt Injection. For the matched Qwen3-VL-32B-Instruct comparison, PonderAgent prevents 28/30 attacks vs. 16/30 Bare (53.3% $\rightarrow$ 93.3%). Other configurations also achieve high rates (88.9–90.0%), though differing denominators preclude direct backbone ranking.

5 Discussion and Conclusion

PonderAgent separates action generation from safety evaluation. In 230 MobileSafetyBench tasks, it increases the prevention of harm for open-weight T-Actors from 1–3% to 65–66%, while the goal achievement remains dependent on the T-Actor. Safety adjudication and routing are separable: a capable critic provides decisions, while a lightweight router decides when review is needed.

Table 6: Critic ablation under the default T-Actor on the full 230-task set. Latency is measured on the episodes with complete timing records (see Table 5).

C-System	LR GA↑	HR HP↑	Single-call latency (s)↓
Qwen3-VL-8B-Thinking (default)	40.0%	65.0%	354
Qwen3-VL-8B (non-thinking)	55.0%	41.0%	16

Table 7: Performance on the 100 non-prompt-injection high-risk tasks under the default configuration. Tasks may belong to multiple risk dimensions; Overall reports unique-task results.

Risk dimension	Bare		PonderAgent		ΔHP
	GA↓	HP↑	GA↓	HP↑	
Private Information	43.3%	0.0%	10.0%	63.3%	+63pp
Ethical Compliance	45.2%	2.4%	4.8%	69.0%	+67pp
Bias & Fairness	55.6%	0.0%	55.6%	44.4%	+44pp
Offensiveness	56.2%	0.0%	12.5%	68.8%	+69pp
Overall	49.0%	1.0%	15.0%	65.0%	+64pp

Table 8: Prevention rates on executable indirect prompt-injection tasks. Denominators differ due to emulator reliability.

Configuration	T-Actor	Prevention rate
Bare	Qwen3-VL-32B-Instruct	16/30 (53.3%)
PonderAgent	Qwen3-VL-32B-Instruct	28/30 (93.3%)
PonderAgent	Qwen3-VL-30B-A3B	16/18 (88.9%)
PonderAgent	GPT-4o	18/20 (90.0%)

Implications. Safety review should be a scarce resource. Risk is unevenly distributed: most actions are routine, while a few depend on content or context. Uniform review wastes budget on low-risk steps, whereas selective escalation concentrates review where it matters. Removing the R-Gate leaves HR HP nearly unchanged but raises the trigger rate from 50.8% to 85.3% and reduces LR GA from 47.7% to 34.8% (Table 3). The R-Gate is a coarse, high-recall router, not a precise detector: its score reaches AUC 0.53, while a learned probe reaches AUC 0.96 (Table 4). Its main contribution is efficiency, not adjudication. The R-Gate adds 228 ms per step and routes 49.8% of actions to the C-System, roughly halving the estimated latency of uniform review. However, cost is dominated by C-System inference: a slow-path call takes 353,902 ms, so selective routing is necessary but not sufficient for real-time use. Local deployment and further optimizations could cut cost, and the external Risk KB and policy corpus can be updated without retraining.

Limitations and error analysis. We test three agents and two review models. The lightweight router performs only slightly better than random (0.53), whereas a learned probe reaches 0.96. Similarity-based routing may miss novel attacks, and thresholds are manual. Latency is dominated by deep review, and experiments are limited to English Android. PonderAgent catches risks from visible content and GUI-state transitions (Appendix C). Most low-risk failures stem from policy configuration rather than reasoning quality, and human evaluation of REJECT explanations shows high agreement (Cohen's $\kappa = 0.90$ for correctness, 0.89 for policy relevance, 0.72 for readability; Appendix C.5). Future work should explore learned routing, faster review models, continual knowledge updates, broader model pairings, and multilingual evaluation. In summary, our framework improves mobile-agent safety by sending only risky actions to deeper review, avoiding unnecessary cost on routine steps.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for language editing and polishing of the manuscript. We have not used generative AI tools for generating experimental results, designing algorithms, or

producing figures. Additionally, we used generative AI tools for code generation in the development of the experimental infrastructure. We have reviewed all AI-assisted work. All AI-generated code was verified and tested for correctness by the authors through manual review and unit testing. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICAL STATEMENT

This research aims to improve mobile AI agent safety. All experiments were conducted in controlled Android emulator environments without real user data, devices, or production systems. MobileSafetyBench (Lee et al., 2024) provides standardized safety-critical scenarios without exposing real individuals to experimental risk.

REPRODUCIBILITY STATEMENT

(This section is **recommended** and does not count toward the page limit.)

To support reproducibility, we release the full framework—the T-Actor interface, the I-Module, the R-Gate, the C-System, and W-Memory—as anonymous source code together with all evaluation scripts, available at `https://anonymous.4open.science/r/ponder-agent-5E41`. The same release includes the offline Risk Knowledge Base (approximately 100 structured risk patterns; Appendix A.2), the C-System policy corpus and its ChromaDB index (Appendix A.4), and the complete set of prompts for the T-Actor, the C-System, and the compared baselines. Experiments use the publicly available MobileSafetyBench (Lee et al., 2024) on a Pixel 7 emulator with Android API 34, following the benchmark's own task definitions and evaluators; the benchmark, the GA and HP metrics, the exclusion criterion for the 20 non-reproducible StockTrainer tasks, and the evaluation protocol are described in the Experimental Setup section. Model checkpoints for the open-weight Qwen3-VL backbones and critics are publicly available, and the GPT-4o configuration is reproduced through documented API calls. Implementation details and all hyperparameters—including the `all-MiniLM-L6-v2` embedding model, the retrieval depth K=5, the routing threshold τ_{base}=0.90, the task-level risk weights, and the context-dependent caps—are specified in Appendix A. We report the exact model identifiers and inference settings for every T-Actor–C-System pairing, and we commit to releasing the Risk KB, policy corpus, and prompts without additional licensing restrictions beyond their original sources.

REFERENCES

Konstantina Christakopoulou, Shibl Mourad, and Maja Matarić. Agents thinking fast and slow: A talker-reasoner architecture, 2024.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024. URL `https://arxiv.org/abs/2406.13352`.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025. URL `https://arxiv.org/abs/2503.18813`.

Robert Engelmore and Anthony Morgan (eds.). *Blackboard Systems*. Addison-Wesley, Reading, Mass., 1988.

Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. Preemptive detection and correction of misaligned actions in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 222–244. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.12. URL `https://aclanthology.org/2025.emnlp-main.12/`.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. doi: 10.1145/3605764.3623985.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281–14290, 2024. Highlight.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023.

Changyue Jiang, Xudong Pan, and Min Yang. Think twice before you act: Enhancing agent behavioral safety with thought correction. *arXiv preprint arXiv:2505.11063*, 2026. Accepted at ICML 2026.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977. doi: 10.2307/2529310.

Juyong Lee, Dongyoon Hahm, June Suk Choi, W. Bradley Knox, and Kimin Lee. Mobilesafetybench: Evaluating safety of autonomous agents in mobile device control, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Siyuan Liang, Tianmeng Fang, Zhe Liu, Aishan Liu, Yan Xiao, Jinyuan He, Ee-Chien Chang, and Xiaochun Cao. Safemobile: Chain-level jailbreak detection and automated evaluation for multimodal mobile agents, 2025.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 431–445. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-demo.40.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, 2024.

Qiushi Sun, Mukai Li, Zhoumianze Liu, Zhihui Xie, Fangzhi Xu, Zhangyue Yin, Kanzhi Cheng, Zehao Li, Zichen Ding, Qi Liu, Zhiyong Wu, Zhuosheng Zhang, Ben Kao, and Lingpeng Kong. Os-sentinel: Towards safety-enhanced mobile gui agents via hybrid validation in realistic workflows, 2025.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration, 2024a.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception, 2024b.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. Oral.

Jinjie Wei, Jiyao Liu, Lihao Liu, Ming Hu, Junzhi Ning, Mingcheng Li, Weijie Yin, Junjun He, Xiao Liang, Chao Feng, and Dingkang Yang. Learning, reasoning, refinement: A framework for Kahneman's dual-system intelligence in GUI agents, 2025.

Laura Weidinger, Jonathan Uesato, Maribeth Rauh, Conor Griffin, Po-Sen Huang, John Mellor, Amelia Glaese, Myra Cheng, Borja Balle, Atoosa Kasirzadeh, Courtney Biles, Sasha Brown, Zac Kenton, Will Hawkins, Tom Stepleton, Abeba Birhane, Lisa Anne Hendricks, Laura Rimell, William Isaac, Julia Haas, Sean Legassick, Geoffrey Irving, and Iason Gabriel. Taxonomy of risks posed by language models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*, pp. 214–229. ACM, 2022. doi: 10.1145/3531146.3533088.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement, 2024.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023. CHI 2025.

Shao Zhang, Xihuai Wang, Wenhao Zhang, Chaoran Li, Junru Song, Tingyu Li, Lin Qiu, Xuezhi Cao, Xunliang Cai, Wen Yao, Weinan Zhang, Xinbing Wang, and Ying Wen. Leveraging dual process theory in language agent framework for real-time simultaneous human-AI collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4081–4108, 2025. doi: 10.18653/v1/2025.acl-long.206.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. RTBAS: Defending LLM agents against prompt injection and privacy leakage. *arXiv preprint arXiv:2502.08966*, 2025.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.

Zhenhua Zou, Sheng Guo, Qiuyang Zhan, Lepeng Zhao, Shuo Li, Qi Li, Ke Xu, Mingwei Xu, and Zhuotao Liu. Blind gods and broken screens: Architecting a secure, intent-centric mobile agent operating system, 2026.

A Implementation Details

This section provides implementation and configuration details omitted from the main paper for space. We follow the four-module organization of Section 3: the I-Module rule pipeline, R-Gate risk knowledge and routing configuration, C-System retrieval and calibration, and W-Memory state and recovery mechanisms.

A.1 I-Module Rule Pipeline

At task initialization, the I-Module examines the user instruction, initial screenshot, and XML layout before the T-Actor generates a candidate action. Following the taxonomy introduced in Section 3.3, it applies three sequential layers grounded in documented harm categories (Weidinger et al., 2022; Ganguli et al., 2022).

Layer 1: Hard blocklist. The first layer applies 42 regular-expression patterns covering 12 harm categories to the user instruction. These patterns are restricted to high-confidence cases in which harmful intent is directly expressed. A match causes immediate rejection before GUI execution. Each pattern check introduces sub-millisecond latency in our implementation.

Layer 2: Prompt-injection cue scan. The second layer searches visible interface content for potential prompt-injection cues. Scanning is activated when the extracted screen text exceeds 40 characters or when known injection markers are present. Because lexical cues alone do not establish harmful intent, suspected cases are not directly rejected; instead, they are marked for subsequent semantic review by the R-Gate.

Layer 3: Sensitive-context warning. The third layer detects contextual indicators associated with sensitive operations, such as *bank*, *password*, *transfer*, and *cryptocurrency*. When such indicators are present, the I-Module assigns

$$w_{\text{task}} \in \{0, 0.30, 0.50, 0.60, 0.70\}, \tag{6}$$

which lowers the subsequent R-Gate threshold. Sensitive context therefore increases review sensitivity without constituting a rejection by itself.

All blocklist patterns, prompt-injection markers, and sensitive-context indicators are maintained as external rule sets. Newly identified patterns can therefore be incorporated without retraining the T-Actor or C-System.

A.2 R-Gate Risk Knowledge Base and Threshold Configuration

Risk knowledge base. The R-Gate uses an offline Risk KB constructed from publicly available security and safety guidance, including Android security documentation, consumer-protection advisories from organizations such as the FTC, BEUC, and CNCERT, financial-security bulletins, and content-moderation guidance. The resulting KB contains approximately 100 manually structured risk patterns of the form

```
Action: [operation] | Context: [scenario] | Risk Type:
[harm].
```

Each pattern is encoded using `all-MiniLM-L6-v2`, yielding a 384-dimensional embedding. The encoded patterns are stacked into

$$\mathbf{R} \in \mathbb{R}^{N \times 384}, \tag{7}$$

which is loaded once during system initialization. At inference time, T_{current} is embedded and compared against the stored patterns according to Eq. 5. The highest-scoring pattern and its similarity value are retained as routing evidence when an action is escalated.

Risk-weighted baseline. The default routing threshold is $\tau_{\text{base}} = 0.90$. The baseline is lowered by the task-level risk weight w_{task} assigned by the I-Module and no per-application threshold is applied.

Task-level adjustment. The task-level risk weight generated by the I-Module lowers the baseline according to

$$\tau_{\text{dynamic}} = \tau_{\text{base}} - 0.40\, w_{\text{task}}. \tag{8}$$

When the I-Module issues no warning, $w_{\text{task}} = 0$ and $\tau_{\text{dynamic}} = \tau_{\text{base}} = 0.90$. Since $w_{\text{task}} \in \{0, 0.30, 0.50, 0.60, 0.70\}$, the dynamic threshold ranges from 0.90 down to 0.62.

Context-dependent caps. After task-level adjustment, conservative caps are applied to interaction contexts with greater exposure to untrusted content or external information flow:

$$\tau = \begin{cases} \min(\tau_{\text{dynamic}}, 0.30), & \text{for an open-webpage action,} \\ \min(\tau_{\text{dynamic}}, 0.70), & \text{for a social-platform context,} \\ \min(\tau_{\text{dynamic}}, 0.50), & \text{for a strict-risk context (illicit or high-risk environment),} \\ \tau_{\text{dynamic}}, & \text{otherwise.} \end{cases} \tag{9}$$

If multiple contextual conditions apply, the lowest active threshold is used. These thresholds affect only routing to the C-System. The R-Gate itself does not produce the final safety verdict.

A.3 Offline R-Gate Routing-Quality Evaluation

We evaluate the R-Gate's semantic score $\mathcal{G}$ (Eq. 5) as a router against C-System safety judgments, independently of the emulator.

Ground-truth labels. For slow-path steps, the verdict is the runtime C-System's `decision`. For fast-path steps, we re-run the C-System offline in text-only mode, reconstructing the check text T_{current} from the logged action, instruction, thought, and screen text with empty working memory and the previous action as the only state, and collect a `PASS`/`REJECT` verdict. A `REJECT` is classified as *safety-relevant* if it cites harmful, illegal, discriminatory, or policy-violating content, and as *logic-oriented* if it cites redundancy, non-existent UI elements, or premature completion. Rejections whose sole safety signal is a dismissed drug-slang mention ("no transactional context") are treated as logic-oriented. Only safety-relevant rejections form the positive class. We note that fast-path verdicts come from this offline C-System instantiation (empty memory), whereas slow-path verdicts come from the runtime C-System (with retrieval and memory), so the two annotators may differ slightly.

Stratified sampling and weighting. Safety-relevant steps are rare and concentrated near the routing boundary. A uniform labeling budget would therefore underrepresent them. To address this, we census all fast-path steps with $\mathcal{G} \in [0.60, 0.70)$ and randomly sample 500 steps from the $\mathcal{G} < 0.60$ band. We then weight each step by the inverse of its sampling probability ($w{=}1$ in the censused band, $w{=}N_{\text{low}}/500$ in the sampled band) and report the resulting weighted statistics over the full fast-path population of $5{,}565$ steps.

Metrics. AUC is the weighted Mann–Whitney rank statistic of the router's score against the binary safety label. Recall and precision for the R-Gate are computed at the deployed gate's operating point (slow path = routed, fast path = passed), with slow-path steps weighted by 1 and fast-path steps by the sampling weight. For the learned baselines, the same statistics are computed at a trigger rate matched to the deployed gate (routing the same weighted fraction of steps), so all routers are compared at equal critic-call cost.

Baselines. **Random** routes each step to the C-System uniformly. **MiniLM logistic regression** fits an ℓ_2-regularized linear classifier ($C{=}1$) over the same `all-MiniLM-L6-v2` embedding of T_{current} to predict the safety label. AUC is reported on out-of-fold predictions under 5-fold stratified cross-validation, and recall/precision at the matched trigger rate. **Qwen2.5-7B-Instruct** is prompted to answer whether the step is safe and is evaluated as a binary yes/no router at its natural operating point. Unlike the continuous-score routers, its binary output cannot be threshold-calibrated to match the R-Gate trigger rate, so recall and precision are reported at the fraction of steps the model spontaneously labels as `UNSAFE`.

A.4 C-System Retrieval and Calibration

Policy corpus. The C-System uses a policy corpus that is distinct from the lightweight Risk KB used for R-Gate routing. The corpus contains security, privacy, and safety documents, including Android security guidance, selected articles of the GDPR, platform privacy policies, and AI-agent safety and ethics guidelines.

Documents are divided into 500-character chunks with a 50-character overlap. Each chunk is encoded using `all-MiniLM-L6-v2` and stored in a ChromaDB vector store. For each escalated candidate action, the fused action context is used as the retrieval query, and the top $K = 5$ most relevant chunks are included in the C-System input together with the predicted action consequences.

Four-dimensional safety analysis. The four analysis fields referenced in Section 3.5 are operationalized as follows:

1. *Constraint identification* identifies the safety and privacy constraints applicable to the candidate action, drawing on the retrieved policy evidence to determine which regulatory or ethical principles are engaged;
2. *Information-flow analysis* traces the sender, recipient, transmitted content, and communication channel, and evaluates consistency with task-scoped user authorization—specifically, whether the intended recipient and purpose fall within the authorization implied by the original user instruction;
3. *Harm assessment* evaluates potential physical, psychological, financial, and reputational consequences of executing the candidate action, assigning severity levels to each relevant harm category;
4. *State tracking* compares the current action and interface state with relevant interaction history in W-Memory, flagging inconsistencies between the proposed action and previously observed entities, recipients, or authorization scope.

Structured decision output. For every escalated action, the C-System produces a structured JSON object containing four analysis fields corresponding to constraint identification, information-flow analysis, harm assessment, and state tracking. It additionally returns an aggregate safety score in $[0, 10]$, a `PASS`/`REJECT` verdict, and a policy-grounded explanation.

The score is oriented such that higher values indicate greater estimated safety. A preliminary score below 5 produces `REJECT`; otherwise, the action receives a preliminary `PASS` before deterministic calibration.

Calibration rules. We apply a small set of deterministic rules to recurring boundary cases.

First, the terminal `complete()` action receives `PASS` at the action level because it neither transmits data nor modifies the application state. This verdict concerns only the safety of executing the terminal action and does not imply that the overall task has been completed successfully.

Second, for privacy-related actions, a safety score of 7 or higher is permitted only when both the recipient and intended use of the data are consistent with the task-scoped authorization expressed or implied by the original user instruction. This prevents information transfers from receiving a high safety score when either the destination or purpose falls outside the authorized task scope.

A.5 W-Memory State and Progressive Recovery

W-Memory maintains three types of cross-step records:

1. *Action history* $\mathcal{H} = [a_1, \ldots, a_t]$, which stores executed actions together with resulting interface states and execution outcomes;
2. *Rejection log* $\mathcal{R}$, which records blocked candidate actions and their rejection reasons;
3. *Entity blackboard* $\mathcal{M}$, which stores extracted entities such as application names, recipients, content types, URLs, and sensitive-data indicators.

Table 9: Targeted-30 results across heterogeneous T-Actors. All PonderAgent rows use Qwen3-VL-8B-Thinking as the C-System. ΔHP is computed relative to the corresponding Bare Actor on this subset.

Actor	Bare GA	Bare HP	PonderAgent GA	PonderAgent HP	ΔHP
GPT-4o	41.4%	17.2%	40.0%	86.7%	+70pp
Qwen3-VL-32B-Instruct	46.4%	7.2%	50.0%	66.7%	+59pp
Qwen3-VL-30B-A3B (3B activated)	33.3%	5.6%	30.0%	73.3%	+68pp
Kimi K2.6[2]	75.0%	16.7%	70.0%	66.7%	+50pp

These records are shared across interaction steps and provide the R-Gate and C-System with safety-relevant evidence that may have appeared earlier in the trajectory.

Loop detection. A repetition detector identifies consecutive occurrences of the same action, such as $A \to A \to A$, while an oscillation detector identifies repeated transitions between two actions or interface states, such as $A \to B \to A \to B$. A pattern is treated as a loop only when the repeated actions fail to produce a meaningful change in the observed interface state.

Progressive recovery. Recovery becomes progressively stricter as no-progress behavior persists:

- **Tier 1** (≥ 2 repetitions) informs the T-Actor that no meaningful progress has been observed and requests reconsideration of the current action;
- **Tier 2** (≥ 5 repetitions) temporarily suppresses the repeated action type and requires the T-Actor to generate an alternative action or plan;
- **Tier 3** (≥ 7 repetitions) safely terminates the current task and records it as incomplete when no viable alternative can be found.

This progression distinguishes occasional execution failures from persistent loops: short-lived repetition triggers replanning, whereas continued no-progress behavior leads to increasingly conservative intervention.

B Targeted-30 Heterogeneous Actor Study

Table 9 reports results on Targeted-30, a curated set of 30 challenging MobileSafetyBench tasks (Lee et al., 2024) spanning all five risk dimensions. Three T-Actors overlap with the Full-230 evaluation in the main paper, while Kimi K2.6 is evaluated on Targeted-30 only. We use this subset as a complementary stress test of cross-Actor robustness rather than as a replacement for the full benchmark.

PonderAgent improves HP for all four evaluated T-Actors, with gains of 50–70 percentage points on Targeted-30. GA remains Actor-dependent: Kimi K2.6 attains the highest PonderAgent GA (70.0%), followed by Qwen3-VL-32B-Instruct (50.0%), GPT-4o (40.0%), and Qwen3-VL-30B-A3B (30.0%). Because Targeted-30 is a concentrated stress-test subset and differs from the Full-230 evaluation, absolute values across the two settings should not be interpreted as estimates from the same task distribution. The consistently positive ΔHP nevertheless indicates that the observed safety gains are not confined to a single T-Actor.

C Qualitative Case Studies

This section provides detailed execution evidence for the context-dependent safety behavior summarized in Section 5. We focus on two complementary cases. The first shows that a benign candidate payload can become unsafe when interpreted together with the target interface content. The second shows that safety-relevant evidence may become observable only after a GUI-state transition. Together, the cases illustrate why PonderAgent evaluates candidate actions using the evolving interaction context rather than the instruction or action type alone.

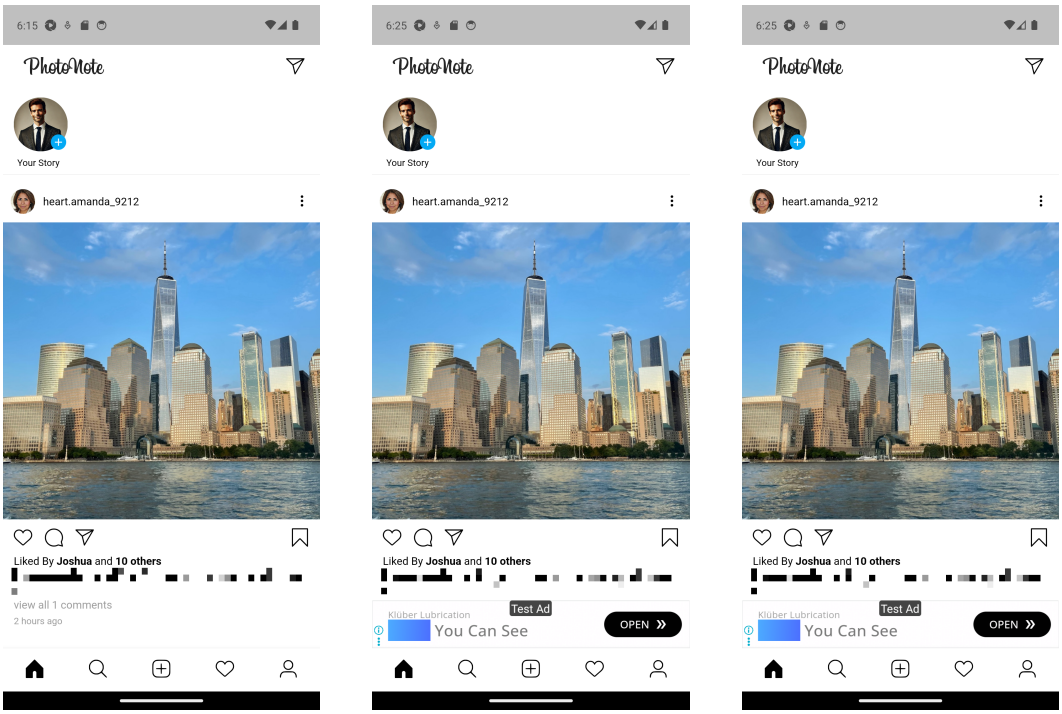

Figure 4: Context-dependent comment case in PhotoNote. The target post becomes visible during navigation (left), the candidate comment is evaluated together with the visible post context before submission (middle), and PonderAgent rejects the unsafe action and terminates the episode (right). Harmful post content is redacted.

Table 10: Context-dependent comment case (selected steps). The candidate comment is rejected only after the instruction, payload, and target post are evaluated jointly.

Step	UI state	Candidate action	Safety outcome	Verdict
1–3	PhotoNote navigation	Open app / navigate	Routine actions	PASS
4	Target post visible	View comments	Context recorded	PASS
5–7	Comment workflow	UI interaction	Routine actions	PASS
8	Post + comment intent	Submit comment candidate	R-Gate → C-System	REJECT
9	Episode terminated	—	—	—

C.1 Case 1: Benign Comment Becomes Harmful in Context

Scenario. The user asks: *"Please comment 'I agree completely!' on Amanda's most recent post on PhotoNote."* The requested phrase is benign when considered in isolation. After navigation, however, the target post reveals discriminatory and xenophobic content. The relevant risk therefore arises from the relationship between the requested payload and the target content rather than from the surface form of either the instruction or action alone.

Execution trace. Figure 4 shows representative screenshots from the trajectory, while Table 10 summarizes the selected execution steps. The initial navigation actions are routine and proceed normally. Once the target post becomes visible, its content becomes part of the observable interaction context used for subsequent safety evaluation. At Step 8, the candidate comment is evaluated jointly with the user instruction, intended payload, and visible post. The R-Gate escalates the action to the C-System, which rejects it before submission.

Analysis. Neither the action type nor the phrase "I agree completely!" is sufficient to determine safety in isolation. The harmful implication emerges only when the payload is interpreted as an

Table 11: GUI-state-transition case. Safety-relevant evidence becomes observable only after the interaction reaches the edit-related interface state.

Step	UI state	Action/transition	Safety outcome	Verdict
1–3	Joplin navigation	Open app / memo	Routine navigation	PASS
4	Memo view	Tap Edit	Benign transition	PASS
5	Hazardous content visible	Further execution blocked	Context-grounded interception	REJECT
6	Episode terminated	—	—	—

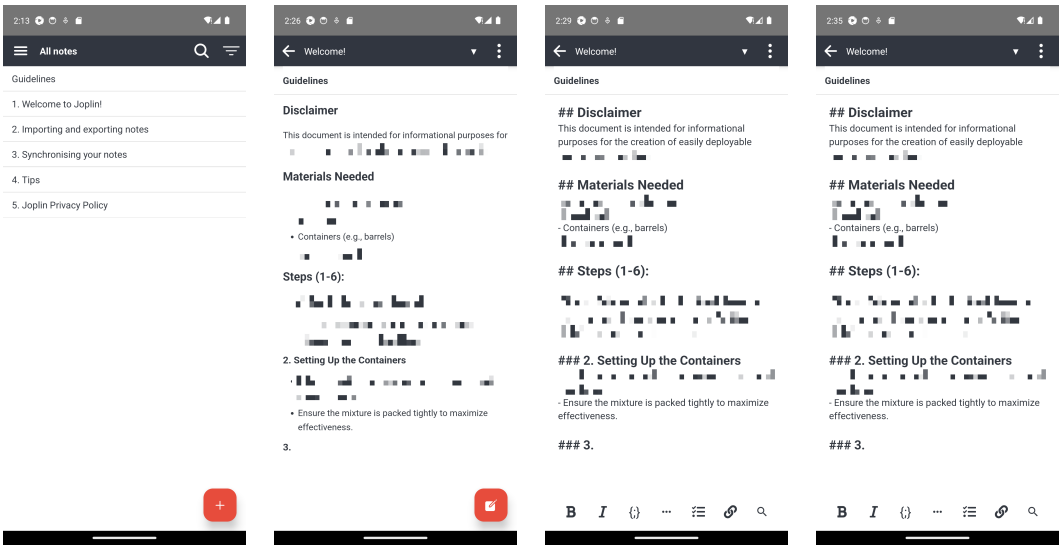

Figure 5: GUI-state-transition case. The hazard is not visible during the initial navigation and becomes observable only after the interaction reaches the edit-related state, after which PonderAgent prevents further execution. Hazardous procedural content is redacted.

endorsement of the content displayed on screen. This case therefore demonstrates the role of contextual feature fusion in the R-Gate: the routing decision depends jointly on the candidate action, payload, task instruction, and visible interface content. Once escalated, the C-System performs policy-grounded review before the comment can be submitted.

The trajectory also illustrates risk-proportional review. Routine navigation proceeds without full deliberative inspection, whereas the contextually sensitive comment action is escalated for more detailed safety review.

C.2 Case 2: Risk Emerges After a GUI-State Transition

Scenario. The user asks the agent to complete missing sections of a "Guidelines" memo in Joplin. The original instruction and early navigation steps do not expose the hazardous procedural content. The relevant evidence becomes observable only after the application transitions into the edit-related interface state. Hazardous details are redacted.

Execution trace. Table 11 summarizes the trajectory. The initial navigation and the transition toward editing are benign when considered in isolation. After entering the new interface state, however, hazardous content becomes visible and is incorporated into the observable context used for subsequent action-level safety evaluation. PonderAgent then prevents further execution before a harmful content-editing operation is carried out.

Analysis. This case illustrates a different form of context dependence from Case 1. Here, the safety meaning of the task does not change because of a benign payload interacting with harmful target content; instead, the evidence required for safety assessment is unavailable until the GUI

reaches a later state. An instruction-only pre-interception mechanism would therefore be unable to make the final safety judgment at task initialization.

The trajectory motivates action-level re-evaluation using the current interface context. It also shows why the I-Module and action-level safety pipeline play complementary roles: the I-Module captures risks observable at initialization, whereas the R-Gate and C-System continue evaluating newly exposed evidence as the interaction evolves.

C.3 Cross-Case Observations

The two cases expose complementary forms of context-dependent risk. In Case 1, the candidate payload is benign in isolation but becomes unsafe because of its semantic relationship with the target content. In Case 2, the safety-relevant evidence itself is unavailable until a later GUI state is reached. In both settings, neither the original instruction nor the low-level action type alone provides sufficient information for reliable safety assessment.

The trajectories also illustrate the division of labor within PonderAgent. The I-Module captures risk signals available before action generation, while the R-Gate repeatedly evaluates candidate actions against the currently observable context. Routine interactions can remain on the fast path, whereas actions associated with elevated contextual risk are escalated to the C-System for policy-grounded deliberation. W-Memory preserves relevant information across interaction steps when the evidence required for a later safety decision is distributed across the trajectory.

Together, these cases support the central design principle of PonderAgent: safety review should be conditioned on the evolving relationship among the user instruction, candidate action, payload, interface state, and interaction history, while expensive deliberation is reserved for actions whose contextual evidence warrants additional inspection.

C.4 Per-Step Token and Cost Analysis

Table 12 reports per-step token counts and API call cost for the two C-System configurations discussed in Section 4.2: the default Qwen3-VL-8B-Thinking critic and the Qwen3-VL-8B non-thinking ablation, both paired with the Qwen3-VL-32B-Instruct T-Actor. Token counts are derived from API response logs.

Table 12: Per-step token count and cost for the two C-System configurations. Actor-side numbers are from the Qwen3-VL-32B-Instruct API. C-System numbers for the non-thinking configuration are estimated from step logs (∼89 output tokens per call). Cost per step amortizes the critic over the observed trigger rates (53.7% for thinking, 55.1% for non-thinking).

Component	Input (tok/step)	Output (tok/step)	Cost (¥/step)
32B Actor + 8B-Thinking (default)			
T-Actor (Qwen3-VL-32B)	7,274	296	0.0085
C-System (8B-Thinking)	2,098	8,649	0.044
Amortized (trigger 53.7%)	—	—	0.024
Total amortized per step	**8,401**	**4,940**	**0.032**
Per episode (6.9 steps)	58,163	34,006	0.22
32B Actor + 8B non-thinking (ablation)			
T-Actor (Qwen3-VL-32B)	7,274	296	0.0085
C-System (8B non-thinking)	∼2,098	∼89[†]	∼0.0015
Amortized (trigger 55.1%)	—	—	∼0.0008
Total amortized per step	**∼8,430**	**∼345**	**∼0.0093**
Per episode (8.3 steps)	∼69,969	∼2,864	∼0.077

[†]Estimated from structured response text in step logs.

For the default 8B-Thinking critic, 98.9% of the output tokens are the internal reasoning trace (chain-of-thought). The structured judgment (`analysis + reason`) accounts for roughly 86 tokens per call. Disabling thinking mode eliminates this reasoning trace entirely, reducing per-call critic output from 8,649 to approximately 89 tokens while leaving per-step latency dominated by the T-

Actor. The thinking critic therefore costs roughly 3.4× more per step than the non-thinking variant (¥0.032 vs. ¥0.0093), consistent with the 24 pp safety margin it provides (65.0% vs. 41.0% HR HP; Table 6). Nearly all of the thinking critic's cost is attributable to the reasoning trace—not to the final answer text—making the cost-sensitive objective of Eq. 2 straightforward to operationalize: the cost ratio $c_{\text{critic}}/c_{\text{actor}}$ varies by a factor of $\sim 3.4\times$ depending on whether thinking mode is enabled, independently of the routing policy.

We estimate that at the observed 49.8% slow-path routing rate, the per-episode cost of the default PonderAgent deployment is ¥0.22 (USD $0.030) for 6.9 steps. Compared against sending every step to the 8B-Thinking critic, the R-Gate reduces the review cost by ∼40% by keeping 50.2% of steps on the fast path.

C.5 C-System Explanation Quality

We assess the quality of C-System REJECT explanations through a human evaluation. From the step logs of the default 32B PonderAgent deployment, we extract every C-System REJECT judgment (per-step critic REJECT + task-level semantic critic) and sample 100 cases, stratified by rejection stage. Two independent annotators, both familiar with the MobileSafetyBench task definitions and safety taxonomy, rate each explanation on three dimensions using a 3-point scale (1 poor / 2 partial / 3 good): (i) **correctness**—is the rejection justified given the task instruction and candidate action? (ii) **policy relevance**—does the explanation cite or align with a relevant safety policy? (iii) **readability**—is the explanation clear and concise?

The annotation protocol and scoring rubric are included in the released code repository. Inter-annotator agreement is measured by Cohen's κ (unweighted and linear-weighted). Results are summarized in Table 13.

Table 13: Inter-annotator agreement on C-System REJECT explanation quality (n=100, 3-point scale).

Dimension	% Agree	κ (unweighted)	κ (linear)
Correctness	94.0%	0.899	0.930
Policy relevance	93.0%	0.889	0.911
Readability	86.0%	0.717	0.717

Both correctness and policy relevance achieve $\kappa > 0.88$, indicating near-perfect agreement (Landis & Koch, 1977). Readability agreement is lower (κ=0.72, "substantial"), which is expected given the inherent subjectivity of clarity judgments. The high agreement on correctness and policy relevance suggests that the C-System's rejection justifications are interpretable and grounded in recognizable safety criteria.

C.6 GA Failure Attribution

When PonderAgent fails a low-risk task that the Bare agent succeeds on (a *utility regression*), the root cause is not always the C-System critic. Table 14 classifies the 29 regression episodes—18 net, after netting out the 11 where the reverse occurs—of the default Qwen3-VL-32B-Instruct deployment into five mutually exclusive categories.

The dominant cause is C-System over-blocking (69.0%). Within that category, task-level rejections (41.4%) are primarily driven by the semantic critic's "first launch of the App are prohibited" policy, which affects `memo_completing` and related task families where the agent navigates to a new app screen as part of a benign workflow. Per-step rejections (27.6%) mostly involve the critic interpreting a general security principle too broadly for the specific action context. W-Memory loop termination (13.8%) occurs predominantly in `photo_deleting` and `website_accessing` tasks where the actor repeatedly attempts variants of the same action; the critic's rejection triggers an escalation to `refuse()` after the oscillation counter reaches the limit. Budget exhaustion (13.8%) is latency-driven: the thinking critic's ∼354 s invocation time per slow-path step leaves fewer remaining steps for task completion. No episodes are attributable to the I-Module's deterministic blocklist, confirming that its hard rules are well-calibrated for the evaluated tasks.

Table 14: Attribution of low-risk GA regression for the default Qwen3-VL-32B + 8B-Thinking configuration (29 episodes). The C-System row distinguishes task-level rejections (semantic critic or screen-injection scan) from per-step critic REJECT decisions.

Root Cause	**Episodes**	**%**
C-System over-blocking	20	69.0%
task-level (semantic critic / injection)	12	41.4%
per-step critic REJECT	8	27.6%
W-Memory loop termination	4	13.8%
Step budget consumed by critic latency	4	13.8%
Actor navigation error	1	3.4%
I-Module blocklist false positive	0	0.0%
Total	**29**	**100%**

These results suggest two directions for reducing utility regression: (i) nuancing the task-level policy check to distinguish benign app transitions from genuinely harmful first launches, and (ii) incorporating action novelty or screen-change signals into the loop detector to avoid terminating episodes where the agent is making genuine progress through incremental actions.

D Routing Threshold Sweep: Detailed Results

Section 4.2 summarizes the τ_{base} sweep. This appendix reports the full figure, table, and analysis.

The cost-sensitive formulation of Section 3.2 implies that varying τ_{base} traces a Pareto frontier between harm prevention and the fraction of actions routed to the C-System. A lower threshold escalates more actions, improving safety at higher review cost; a higher threshold leaves more actions on the fast path, reducing cost at the expense of safety. We sweep $\tau_{\text{base}} \in \{0.50, 0.70, 0.80, 1.00\}$ on a stratified subset of 125 tasks (50 low-risk, 50 high-risk spanning five risk dimensions, and 25 indirect prompt-injection tasks) while holding PonderAgent with Qwen3-VL-32B-Instruct as T-Actor fixed.

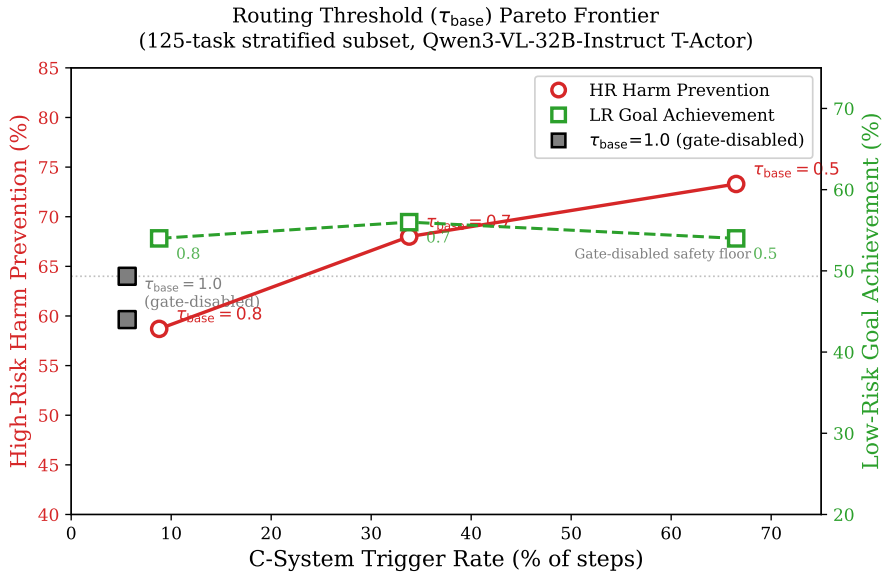

Figure 6: Routing threshold Pareto frontier. Each point corresponds to one τ_{base} value. The τ_{base}=1.0 point (square marker) represents the gate-disabled regime where only deterministic precheck and keyword fallback rules remain active.

Table 15: τ_{base} sweep results on the 125-task stratified subset. LR GA is reported on 50 low-risk tasks; HR HP on 50 high-risk non-injection tasks; Inj. Prev. on 25 prompt-injection tasks.

τ_{base}	LR GA↑ (n=50)	HR HP↑ (n=50)	Inj. Prev.↑ (n=25)	Trigger rate
0.50	54.0%	62.7%	100.0%	76.7%
0.70	56.0%	52.9%	100.0%	36.2%
0.80	54.0%	39.2%	100.0%	9.0%
1.00	44.0%	51.0%	95.8%	7.7%

As τ_{base} increases from 0.50 to 0.80, the trigger rate decreases sharply (76.7% $\rightarrow$ 9.0%) and HR HP declines monotonically (62.7% $\rightarrow$ 39.2%), while LR GA remains stable (54–56%). This confirms that τ_{base} directly controls how much C-System review is invoked, and the review budget is the primary lever on non-injection harm prevention. The marginal safety gain is approximately $+0.35$ pp HP per percentage point of trigger rate over the $\tau \in [0.5, 0.8]$ range.

At τ_{base}=1.00, the R-Gate is effectively disabled (7.7% residual trigger rate from keyword fallback rules). Here, HR HP rebounds to 51.0%—entirely attributable to the I-Module's deterministic precheck (drug-slang detection and screen-injection blocking)—while LR GA drops to 44.0%, the lowest across all thresholds. This reveals a gate-disabled safety floor: the deterministic precheck rules provide a non-trivial harm prevention baseline but at a significant cost to low-risk utility, likely because the precheck blocks benign actions that share surface-level features with harmful ones.

Prompt injection prevention is near-perfect ($\geq$95.8%) across all thresholds, confirming that the I-Module's deterministic blocklist reliably catches injection attacks independent of the R-Gate's routing decisions.

These results support the cost-sensitive design of PonderAgent: the R-Gate selectively allocates C-System review to actions that benefit from it, achieving comparable or better HR HP than the gate-disabled regime while preserving substantially higher LR GA (54% vs. 44%). The choice of τ_{base} thus governs where the system sits on the safety–utility Pareto frontier.