LEASH: UNCERTAINTY-AWARE WEAK-TO-STRONG ESCALATION AND OVERSIGHT FOR COMPUTER USE

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents increasingly tackle long-horizon tasks, where running a frontier model at every step is expensive and a small open-weight model alone often fails. *Weak-to-strong escalation* offers a middle ground: a weak agent acts by default and hands control to a strong one only when needed. The core challenge is deciding when to escalate, which requires an uncertainty signal cheap enough to check at every step. After handoff the risk remains: the strong agent can still go astray, and a black-box API exposes no internals to monitor. To this end, we propose *LEASH*, a training-free framework that unleashes the strong agent only when the weak agent's latent uncertainty signals it is needed, and keeps it on a leash afterward. *LEASH* has three layers: initial task uncertainty for early routing, step-level uncertainty for runtime escalation, and, after handoff, a shadow observer for continued oversight of the strong black-box agent. At the core of all three layers is *latent uncertainty*: we compare the weak model's hidden states with those from successful and failed trajectories, and a state that looks more like the failures signals higher uncertainty. During weak-agent execution, runtime uncertainty reuses activations already produced during action generation, so it needs no extra forward pass or dedicated monitor training. Experiments on OSWorld across four contemporary weak-agent backbones show that *LEASH* consistently improves over standalone weak agents and outperforms random escalation by 6.55–11.94 points at matched escalation rates. With Qwen3.5-35B-A3B, *LEASH* achieves 73.66% task success at $0.4117 per task, matching always-on Claude Opus 5 (72.90%) while reducing inference cost by 19.2%.

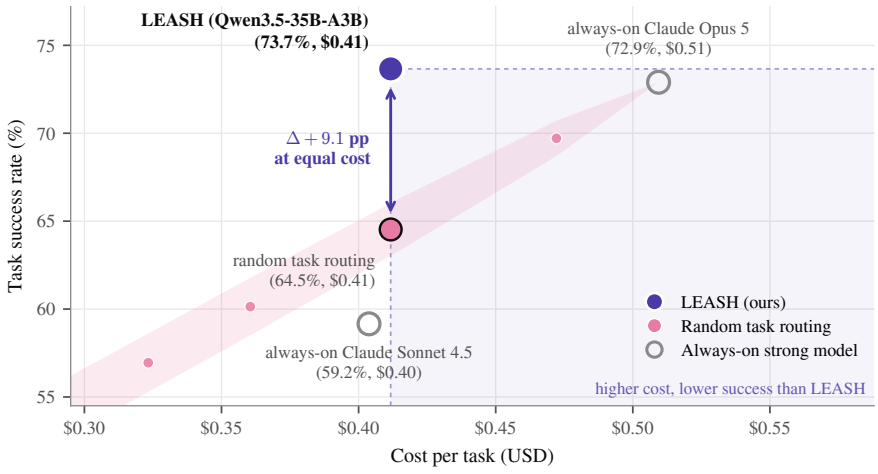

Figure 1: Accuracy–cost comparison between *LEASH* and random task-level escalation on Qwen3.5-35B-A3B. The random frontier is obtained from repeated random task allocations across different escalation budgets; the band shows ±1 s.d. over 10,000 allocations. At matched inference cost, *LEASH* achieves 9.1 percentage points higher task success.

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

Computer-use agents (CUAs) automate graphical interfaces by translating natural-language instructions and visual observations into executable actions (Xie et al., 2024; Cheng et al., 2024), and are increasingly deployed for complex, long-horizon automation tasks that can take more than an hour on average (Abhyankar et al., 2026). While frontier proprietary models have been shown to achieve impressive performance leaps (OpenAI, 2026; Anthropic, 2026b), running every single task spanning hundreds of steps with these models raises cost concerns, whereas a light open-weight model alone performs poorly on even much simpler tasks (Xie et al., 2024). This creates a fundamental trade-off between accuracy and cost efficiency, motivating *weak-to-strong escalation* (W2SE), in which a weak agent serves as the primary executor and hands control to a strong agent only when needed (Wei et al., 2026; Lei & Huang, 2026). W2SE raises two important questions: (1) When should the weak agent escalate, and (2) who oversees the strong agent after handing over?

Existing work on W2SE primarily focuses on the first question, estimating the weak agent's failure risk to decide when escalation is needed. Some of them adopt another monitoring agent to verify each step (Wang et al., 2024; Yi et al., 2026), others train an extra monitoring module on the collection of annotated trajectories (Wei et al., 2026; Lei & Huang, 2026). However, the former inevitably adds nontrivial cost to run another heavy agent step-by-step, whereas the latter creates another phase of data annotation and training, hindering direct adoption in real-world applications under varying deployment constraints. While a training-free uncertainty estimation-based approach avoids the aforementioned bottlenecks (Oh et al., 2026), prior works still rely on an external knowledge base and embedding models (Liu et al., 2026) or multiple sampling for each step (Wang et al., 2026) quickly becomes infeasible or impractical in long-horizon tasks in the wild.

Importantly, escalation alone does not eliminate execution uncertainty, resulting in the second important question in W2SE which is commonly overlooked. That is, even after a stronger agent takes control, subsequent actions may deviate from the intended task or fail to achieve the desired outcome. However, monitoring a black-box strong agent is challenging because its internal states are inaccessible and verbalized confidence can be unreliable (Han et al., 2024).

To answer both questions, we formulate W2SE as a budget-constrained sequential stopping problem, which identifies estimating the weak agent's failure risk as its central subproblem. We then propose **L**atent uncertainty for **E**scalation **A**nd **SH**adow oversight (*LEASH*), a training-free framework driven by the weak agent's hidden states. First, we record its hidden states of prompt and action tokens on a small set of held-out tasks and label them with task-level and subgoal-level outcomes to construct a representation bank. This representation bank then serves as a semantic anchor to score the newly faced prompts and actions during test-time by computing relative Mahalanobis distance in latent space, that is, by how much closer it lies to past failures than to past successes. *LEASH* puts this signal on two jobs (Figure 2): For *escalation*, an initial gate scores uncertainty from the initial task state, while a runtime gate monitors subsequent execution. For *oversight*, the weak model keeps reading the strong agent's trajectory after the one-way handoff and is probed every three steps; when its uncertainty is high, the strong agent must reflect on its progress with a higher reasoning effort before acting. Our uncertainty-driven W2SE formulation draws on an easily accessible resource, the weak agent's own past experience (Silver & Sutton, 2025), removing reliance on an external dataset or additional training without hosting another expensive agent during runtime.

We evaluate *LEASH* on OSWorld (Xie et al., 2024) with four open-weight weak agents and Claude Opus 5 as the strong agent. At matched escalation rates, *LEASH* outperforms random escalation by 6.55–11.94 points. We also report the *escalation gap recovered*, which, like the performance gap recovered of Burns et al. (2023), measures the fraction of the gap between the weak agent alone and always-on Opus that a policy closes. *LEASH* recovers 75–102% of this gap, against 37–69% for random escalation, while costing 69–81% as much as always-on Opus. With Qwen3.5-35B-A3B (Team, 2026), *LEASH* matches always-on Opus (73.7% vs. 72.9% success) at 19% lower cost, whereas random escalation at equal cost scores about 9 points lower (Figure 1).

Our contributions are summarized as follows:

- We formalize weak-to-strong escalation as a budget-constrained sequential stopping problem and show that one latent uncertainty signal can drive both escalation and oversight.

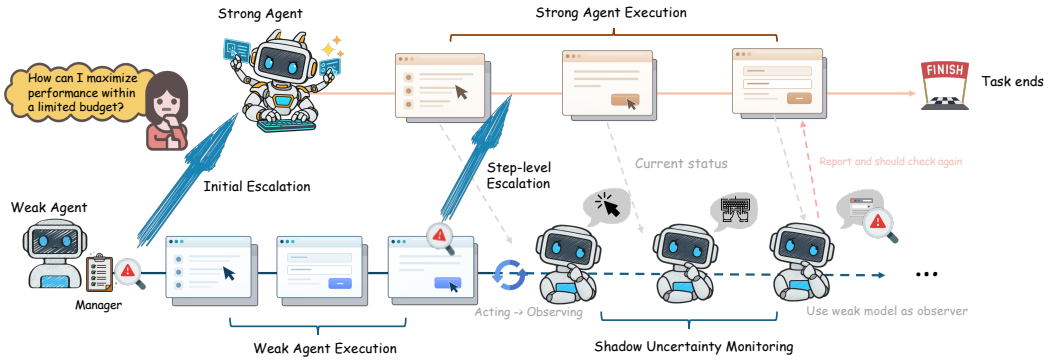

Figure 2: Overview of uncertainty-aware weak-to-strong escalation and post-handoff oversight. A manager guides the weak agent through subgoals. Hidden-state uncertainty scores trigger initial or step-level escalation; otherwise, the weak agent continues. After one-way handoff, the strong agent retains control while the weak model acts as a shadow observer, generating unexecuted actions for continued monitoring. The feedback path enables uncertainty-triggered reassessment and intervention. In short, *LEASH* unleashes the strong agent only when the weak agent's latent uncertainty signals it is needed, and keeps it on a *LEASH* afterward.

- We propose a training-free latent uncertainty score, a relative Mahalanobis distance to the weak agent's own past successes and failures, whose threshold rule robustly represents the weak agent's failure signal.
- We extend this signal to weak-to-strong oversight, in which the white-box weak agent monitors the black-box strong agent through its own hidden states.
- On OSWorld with four open-weight backbones, we show that *LEASH* outperforms random escalation at matched escalation rates and recovers 75–102% of the escalation gap at 69–81% of the cost of always-on Opus.

2 RELATED WORK

Efficient Computer-Use Agents Model routing and cascading offer a promising way to balance inference cost and performance, as demonstrated by FrugalGPT and RouteLLM (Chen et al., 2023; Ong et al., 2025). Extending this idea to GUI automation, Adaptive VLM Routing selects models based on action difficulty and confidence (Liu et al., 2026), while StepWise trains lightweight monitors to detect execution stalls and identify milestones for escalation (Wei et al., 2026). Our work instead leverages the backbone agent's internal representations to guide both task-level routing and step-level escalation, without training a dedicated monitoring network.

Weak-to-Strong Paradigms Weak-to-strong generalization studies how stronger models learn from weaker supervision (Burns et al., 2023; Somerstep et al., 2025; Medvedev et al., 2025; Xue et al., 2025). These works primarily concern training-time transfer of supervision or capabilities from a weaker model to a stronger one. Here we study the complementary inference-time setting of *weak-to-strong escalation* (W2SE), where a weak model executes by default and selectively invokes a stronger model when additional capability is needed. This shifts the weak–strong relationship from supervision transfer to sequential resource allocation. We derive escalation signals from the agent's own past experience without training an additional monitor.

Uncertainty Estimation and Agent Monitoring Uncertainty estimation of agentic systems has emerged as one of the most important problems to monitor and guide agent decisions (Oh et al., 2026; Han et al., 2024), while SafeGround and Argus investigate uncertainty estimation for GUI grounding, including calibrated predictions and representation-based methods (Wang et al., 2026; Kumar et al., 2026). At the trajectory level, beyond individual predictions, existing work explores weak-to-strong oversight of agent misbehavior and failure monitoring through observable trajectories (Kale et al., 2026; Pan et al., 2026). Our work integrates representation-based uncertainty

estimation into the execution workflow, reusing the backbone model's hidden states for escalation and extending the same signal to post-handoff oversight of a closed-source strong agent.

3 METHOD

We introduce *LEASH*, a training-free uncertainty-driven agentic inference framework for W2SE and post-handoff oversight (Figure 2). By comparing the weak model's hidden states with successful and failed reference executions, we estimate uncertainty at task initialization and during interaction, which guides both escalation to the strong agent and continued oversight after handoff. We first formulate the problem setup in Section 3.1 and then introduce each component in subsequent sections.

3.1 PROBLEM FORMULATION

Given a task query, a computer-use agent generates an action conditioned on the query, the screen, and history. We consider a lightweight, weak agent, e.g., a relatively small open-weight model with accessible hidden states, and a large, strong agent, e.g., a proprietary black-box frontier model, sharing a budget of T actions. In addition to these two actor agents, we also host a Manager agent (with a strong model) who fires at the beginning to decompose the task and subsequently at subgoal boundaries to verify progress, constructing a minimal version of multi-agent workflow (Agashe et al., 2025a;b; Koh et al., 2026). It decomposes the task query into subgoals with screenshot-verifiable completion criteria and decides whether to advance, keep, or revise each subgoal (Appendix C). This decomposition naturally provides finer-grained supervision, allowing execution steps to inherit subgoal-level outcomes rather than relying solely on the final task-level label.

We adopt the weak model as a starting agent for all tasks until control is transferred to the strong model by hitting some flags. Then, the strong model runs over the remaining action budget until the end of the task. Let $x \sim \mathcal{D}$ denote a task drawn from the deployment distribution, and let π denote an escalation policy. For each task x, policy π induces a stopping time $t^{\pi}_{\text{esc}}(x) \in \{0, 1, \ldots, T\}$, where $t^{\pi}_{\text{esc}}(x) = 0$ denotes the earliest escalation opportunity and $t^{\pi}_{\text{esc}}(x) = T$ denotes no escalation. Since the strong agent dominates per-step cost, we use the total escalation rate, p_{esc}, as a budget proxy and want to maximize task success under limited budget:

$$\max_{\pi} \ \mathbb{E}_{x\sim\mathcal{D}}[S^{\pi}(x)] \quad \text{s.t.} \quad \Pr_{x\sim\mathcal{D}}\left(t^{\pi}_{\text{esc}}(x) < T\right) \leq p_{\text{esc}}, \tag{1}$$

where $S^{\pi}(x) \in \{0, 1\}$ indicates task success under policy π. That is, we want to run tasks and each step within them through the weak agent as many times as possible unless it fails, to minimize the cost while achieving a high task success rate, becoming a problem **when-to-escalate**. Unlike existing works that adopt other monitoring agents (Wang et al., 2024; Gao et al., 2026) incurring additional inference costs or training-based monitors (Wei et al., 2026) requiring annotation costs and an additional training phase, we propose to use simple uncertainty estimates as a signal to determine when-to-escalate, without inducing any training cost.

3.2 REFERENCE-BASED LATENT UNCERTAINTY ESTIMATION

Motivated by well-established observations revealing that LLMs internalize the likelihood of task success or failure in their representations (Kadavath et al., 2022; Azaria & Mitchell, 2023; Kossen et al., 2024; Orgad et al., 2025) even before generating any tokens (Ji et al., 2024; Cencerrado et al., 2025), we propose a training-free, representation-distance-based uncertainty estimation technique for agent systems.

Representations and reference banks. We first initialize an agent called Manager (operated by a strong LLM), who decomposes the task into multiple subgoal segments given the task prompt. Then, for each task from a small held-out set, we run a weak agent to collect some preliminary inference trajectories without escalating anything to a strong agent. During this inference over the held-out set, we extract two final-layer representations, $h_0 \in \mathbb{R}^d$ at the last task query prompt token, and $h_t \in \mathbb{R}^d$ at the last generated action token of each execution step t. The former captures information over the instruction and initial screen before decoding, whereas the latter also incorporates the action executed by the agent. These hidden states build representation banks pairing each hidden state with

progress outcome: The query bank $\mathcal{B}_q = \{(h_{i,0}, y_i)\}$ adopts task outcomes $y_i \in \{\text{PASS}, \text{FAIL}\}$, whereas the action bank $\mathcal{B}_a = \{(h_{i,t}, y_{i,t})\}$ assigns each step its subgoal segment's outcome, $y_{i,t}$, PASS if Manager confirms the subgoal achievement at that step, FAIL if it revises it, and the task-level outcome if the segment remains open at termination.

Uncertainty score. For each channel of representation banks $c \in \{q, a\}$, we fit a shared-covariance Gaussian discriminant model on $\mathcal{B}_c$. Let $z_c(h)$ denote the representation standardized by the bank statistics, $\mu_{c,y}$ the channel centroid, and $\Sigma_{c,\lambda} = \widehat{\Sigma}_c + \lambda_c I$ the pooled within-channel covariance regularized by $\lambda_c > 0$. Using the squared Mahalanobis distance,

$$d^2_{c,y}(h) = \big(z_c(h) - \mu_{c,y}\big)^\top \Sigma^{-1}_{c,\lambda}\big(z_c(h) - \mu_{c,y}\big), \tag{2}$$

we define uncertainty score through the relative distance to successful and failed references:

$$u_c(h) = d^2_{c,\text{PASS}}(h) - d^2_{c,\text{FAIL}}(h). \tag{3}$$

A larger score indicates greater relative proximity to failed references. We use u_q for initialization and u_a for runtime action monitoring, selecting λ_c for each model backbone and channel by task-grouped cross-validated AUROC (detailed in Appendix A.1).

Interpretation. The design of our relative Mahalanobis distance score is highly inspired by distance-based approaches in out-of-distribution detection literature (Lee et al., 2018; Ren et al., 2021; Sun et al., 2022). Here, subtracting the two distances cancels their shared quadratic term, making u_c affine in h and sensitive to displacement toward failed references. Under the shared-covariance Gaussian model, the corresponding population score equals twice the failure-to-success log-likelihood ratio. This motivates using its ranking for escalation. Note that this Gaussian fitting is simply done by computing per-outcome means and a singular value decomposition without gradient-based training. It can also be easily repeated as the bank grows, making it friendly for continual updates in self-improving systems (Shinn et al., 2023; Ouyang et al., 2026).

3.3 Uncertainty-Aware Weak-to-Strong Escalation

Escalation gates. With the measure of uncertainty, we now define the routing gates for each uncertainty channel. Two channels address different sources of uncertainty: ambiguity in task specification and failure likelihood across stacked interactions (Oh et al., 2026). Here, the initial query-level routing can identify tasks beyond the weak agent's capability in advance, but cannot anticipate every difficulty encountered during runtime execution. Therefore, we combine an initial escalation gate, $g_0 = \mathbb{I}\{u_q(h_0) > \tau_q\}$ which escalates when $g_0 = 1$, with a runtime escalation gate, $g_t = \mathbb{I}\{u_a(h_t) > \tau_a\}$ for $t \geq 1$ that monitors each action of tasks retained by the initial gate given calibrated thresholds τ_q and τ_a. Runtime escalation occurs immediately after the first action with $g_t = 1$; The strong agent then receives the instruction, current screenshot, and recent steps ran by the weak agent, and keeps control until the task termination (See Appendix C for details).

Calibrating gates to an escalation budget. We set each threshold to a quantile of uncertainty scores on the reference tasks, so that each gate escalates a target fraction of tasks used to construct the representation bank. Let p_q be the target fraction of tasks escalated at initialization and p_{esc} the target overall escalation rate. Since the runtime gate only monitors the tasks that the initial gate leaves to the weak agent, it should escalate a fraction $p_a = (p_{\text{esc}} - p_q)/(1 - p_q)$ of them, so that $p_q + (1 - p_q)p_a = p_{\text{esc}}$. The thresholds are then expressed as follow:

$$\tau_q = Q_{1-p_q}\big(\{u_q(h_{i,0})\}_i\big), \qquad \tau_a = Q_{1-p_a}\big(\{U_i : u_q(h_{i,0}) \leq \tau_q\}\big), \tag{4}$$

where Q_α is the empirical α-quantile over reference tasks and $U_i = \max_{1 \leq t \leq T_i} u_a(h_{i,t})$ is the maximum runtime score of the i-th reference trajectory of length T_i. Intuitively, a fraction p_q of the reference tasks have initial scores above τ_q, and among the tasks kept by the initial gate, a fraction p_a have a maximum runtime score above τ_a.

Note that we target escalation rates rather than failure probabilities because the gates only use the ranking of the scores. We compute τ_a from each task's maximum score because a single step above τ_a is enough for handoff; a quantile over individual steps would escalate more tasks than targeted. All scores in Eq. (4) are computed on held-out reference tasks (Appendix A.2).

3.4 POST-HANDOFF OVERSIGHT AND INTERVENTION

We note that the strong agent can also get stuck in repeated action loops or get lost in multi-turn interactions (Laban et al., 2026). However, its hidden states are inaccessible, while verbalized confidence is often highly unreliable as an uncertainty signal (Han et al., 2024). We thus reuse the weak agent as a shadow observer to provide continued post-handoff oversight of the strong agent.

Shadow oversight. After handoff, the weak model is reused as a *Shadow observer* for continued post-handoff oversight of the strong agent. We append each executed strong-agent step to the weak model's history in its own action format and query with the current observation every $k = 3$ steps. Given that context, the weak model generates a candidate action, allowing us to extract h_t^{sh} at the same token position. Assuming that a higher uncertainty from the weak model also indicates a greater difficulty for the strong agent on average (Pan et al., 2026; Kale et al., 2026), we can use that representation for the runtime action uncertainty estimation again $u_{\mathrm{sh}} = u_a\big(h_t^{\mathrm{sh}}\big)$, to define the Shadow oversight gate $g_{\mathrm{sh}} = \mathbb{I}\big\{u_{\mathrm{sh}} > \tau_{\mathrm{sh}}\big\}$ given a threshold τ_{sh}.

Uncertainty-triggered intervention. When $g_{\mathrm{sh}} = 1$, we discard the strong agent's pending action and trigger a reflection-based intervention, which reassesses progress toward the task goal and chooses a new action using higher reasoning effort. We set the threshold τ_{sh} to the 90th percentile of per-step reference scores of the action representation bank. We allow at most six checkpoint interventions per episode, while suppressing further interventions after two checkpoints have changed the strong agent's action (Appendix A.3).

4 EXPERIMENTS

We evaluate *LEASH* on OSWorld (Xie et al., 2024) across four backbones, examining task success and inference cost under uncertainty-guided escalation and post-handoff oversight. We further analyze the contributions of individual components and assess the discriminative quality of the underlying uncertainty estimates.

4.1 EXPERIMENTAL SETUP

Benchmark and agents. We evaluate our framework on OSWorld (Xie et al., 2024). We use four backbones: EvoCUA-8B (Xue et al., 2026), Qwen3.5-35B-A3B (Team, 2026), Nemotron-3-Nano-Omni (30B-A3B) (Deshmukh et al., 2026), and Qwen3.8-27B (Qwen Team, 2026). Claude Opus (Anthropic, 2026a) serves as the strong executor, while a Claude-based Manager decomposes tasks into subgoals and verifies their completion. We compare against always-on Claude Opus 5 and Claude Sonnet 4.5 (Anthropic, 2025). We evaluate four cumulative configurations: *Alone*, *Manager*, *Escalation*, and *LEASH*. The definitions, execution settings, and model identifiers are provided in Appendix B.1.

Uncertainty estimation. We compare token-probability uncertainty with three representation-based alternatives: Euclidean distance to class centroids, kNN, and pooled LDA with covariance-aware Mahalanobis distance (Lee et al., 2018; Sun et al., 2022). We evaluate their discrimination using AUROC, where initial uncertainty is assessed against final task outcomes and runtime uncertainty against Manager-assigned subgoal outcomes. Detailed evaluation and calibration procedures are provided in Appendix B.1.

Metrics. We report task success rate and mean inference cost per task. Costs are estimated from recorded token usage at fixed reference prices, including weak agent, Manager, strong-executor, and monitoring calls when enabled. These estimates measure token-priced inference cost rather than local hardware expenditure. Missing usage is not treated as zero cost. Cost coverage and accounting details are provided in Appendix B.1.

4.2 TASK SUCCESS AND INFERENCE COST

LEASH consistently improves task success over standalone weak-agent execution across all four backbones (Table 1). At closely matched escalation rates, it also outperforms random escalation

Table 1: Task success and estimated inference cost on OSWorld. *Random* escalation routes a matched fraction of tasks to Opus 5 uniformly at random; success is averaged over 10,000 random allocations. Esc.% denotes the fraction of tasks handed off through the initial or runtime gate and excludes post-handoff Shadow monitoring. Δ Success is the absolute improvement over the corresponding standalone agent. Cost is reported in USD per task.

Agent	Escalation Method	Esc. (%)	Success (%) ↑	Δ Success	Cost ($) ↓
EvoCUA-8B	-	0.0	42.37	–	–
	Random	45.4	59.98	+17.61	0.3031
	LEASH	45.4	67.18	+24.81	0.3627
Qwen3.5-35B-A3B	-	0.0	37.40	–	–
	Random	64.4	61.72	+24.32	0.3768
	LEASH	64.1	**73.66**	**+36.26**	**0.4117**
Nemotron-3 (30B-A3B)	-	0.0	45.04	–	0.0132
	Random	43.3	55.41	+10.37	0.2782
	LEASH	43.1	66.03	+20.99	0.3500
Qwen3.8-27B	-	0.0	51.91	–	–
	Random	51.3	63.30	+11.39	0.3249
	LEASH	51.1	69.85	+17.94	0.3622
Claude Opus 5	Always on	100.0	72.90	–	0.5094
Claude Sonnet 4.5	Always on	100.0	59.16	–	0.4038

Table 2: AUROC of uncertainty estimators under task-grouped 10-fold cross-validation. Initial and runtime AUROC use task outcomes and Manager-assigned subgoal outcomes, respectively. Bold indicates the highest value in each row.

Stage	Backbone	Token prob.	Euclidean	kNN	**Pooled LDA(Ours)**
Initial	Nemotron-3 (30B-A3B)	0.575	0.692	0.672	**0.749**
	Qwen3.8-27B	**0.671**	0.541	0.572	0.600
Runtime	Nemotron-3 (30B-A3B)	0.488	0.583	0.489	**0.609**
	Qwen3.8-27B	0.540	0.564	0.522	**0.575**

by 6.55–11.94 percentage points, showing that the training-free uncertainty signal provides useful information for selective weak-to-strong escalation rather than merely increasing access to the strong executor.

With Qwen3.5-35B-A3B (Team, 2026) as the backbone, *LEASH* achieves 73.66% task success while escalating 64.1% of tasks, compared with 72.90% for always-on Opus (Anthropic, 2026a). Its estimated inference cost is $0.4117 per task, 19.2% lower than the $0.5094 cost of always-on Opus. The benefit also persists at lower escalation rates: *LEASH* improves Nemotron-3 (Deshmukh et al., 2026) from 45.04% to 66.03% success while escalating 43.1% of tasks, and EvoCUA-8B (Xue et al., 2026) from 42.37% to 67.18% while escalating 45.4%. Together, these results demonstrate that *LEASH* can substantially improve weak-agent performance while limiting reliance on the strong executor, yielding a favorable accuracy–cost trade-off across diverse backbones. The gains are also consistent across weak agents with substantially different standalone capabilities. For example, Qwen3.8-27B is the strongest standalone backbone at 51.91% success, yet *LEASH* still improves it by 17.94 percentage points with only 51.1% of tasks escalated. Moreover, all four *LEASH* configurations outperform the always-on Sonnet reference, and three of them do so at lower estimated inference cost. This suggests that the benefit of selective escalation is not restricted to particularly weak starting agents.

Table 3: Component ablation of *LEASH* on Qwen3.5-35B-A3B. Checkmarks indicate enabled components. Shadow includes post-handoff oversight and checkpoint intervention.

Configuration	Manager	Escalation	Shadow	Success (%) ↑	Cost ($) ↓
Weak agent alone	×	×	×	37.40	–
+ Manager	✓	×	×	41.15	0.1372
+ Escalation	✓	✓	×	67.94	0.4050
LEASH	✓	✓	✓	**73.66**	0.4117

Table 4: Subset analysis of post-handoff Shadow monitoring with Qwen3.5-35B-A3B. *Both escalated* contains tasks handed off by both Shadow and No-Shadow policies. *Shadow triggered* contains tasks with at least one checkpoint trigger, while *Escalated, not triggered* contains escalated tasks without a trigger. *Opus only* reports always-on Opus accuracy on the same task IDs.

Subset	Tasks	Shadow (%)	No Shadow (%)	Opus only (%)
Both escalated	131	69.5	67.9	64.9
Shadow triggered	31	**64.5**	48.4	51.6
Escalated, not triggered	137	74.5	69.3	74.5

4.3 Uncertainty Estimation

We compare token-probability uncertainty with three representation-based alternatives: Euclidean distance to class centroids, kNN over reference representations (Sun et al., 2022), and pooled LDA using covariance-aware Mahalanobis distance (Lee et al., 2018). As shown in Table 2, uncertainty discrimination varies across both backbones and execution stages. At initialization, pooled LDA performs best for Nemotron, whereas token probability is stronger for Qwen3.8-27B. At runtime, pooled LDA achieves the highest AUROC for both backbones, although the overall discrimination remains modest. Notably, runtime AUROC reflects Manager-assigned subgoal outcomes rather than individual action correctness. Overall, these results suggest that representation-based uncertainty can provide a training-free signal for runtime monitoring, particularly highlighting the effectiveness of our LDA-based estimation, while its effectiveness remains dependent on the backbone and execution stage. Meanwhile, the uncertainty estimator is modular in *LEASH*, allowing alternative uncertainty signals to be substituted without changing the escalation workflow.

4.4 Ablation Study

As shown in Table 3, the largest gain comes from enabling uncertainty-guided escalation: task success increases from 41.15% with the Manager to 67.94%, at a higher inference cost. Adding Shadow monitoring and checkpoint intervention further improves success to 73.66% with only a small additional cost.

This final comparison reflects the complete policy rather than an isolated causal effect of the post-handoff intervention. Additional backbone ablations and a more detailed Shadow analysis are provided in Appendices B.3 and B.5.

What does Shadow oversight detect? We further examine the tasks on which post-handoff monitoring fires in Table 4. Shadow triggers concentrate on particularly difficult trajectories: always-on Opus succeeds on only 51.6% of triggered tasks, compared with 74.5% of escalated tasks without a trigger. On the triggered subset, the Shadow-enabled policy achieves 64.5% success, compared with 48.4% without Shadow. These comparisons suggest that the Shadow observer identifies cases where difficulty persists even after escalation. Because the subsets are defined by the Shadow policy itself, however, these results are descriptive rather than a causal estimate of the checkpoint intervention.

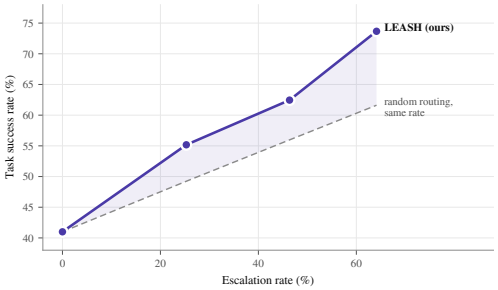

Figure 3: Sensitivity to escalation rate on Qwen3.5-35B-A3B. LEASH consistently outperforms random task-level routing at matched escalation rates.

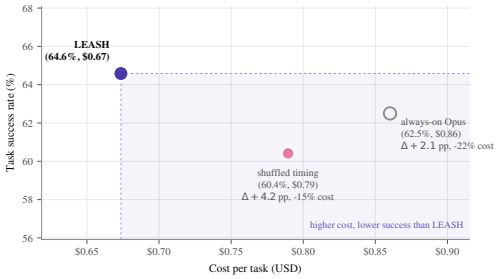

Figure 4: Effect of escalation timing on escalation-prone tasks. *LEASH* achieves higher success at lower estimated cost than shuffled runtime escalation and always-on Opus.

Table 5: Sensitivity to reference bank size. Initial-stage AUROC is reported as mean ± standard deviation across subsamples.

Bank size	Qwen3.5-35B-A3B	EvoCUA-8B
40	0.701 ± 0.100	0.633 ± 0.099
60	0.744 ± 0.060	0.654 ± 0.061
80	0.749 ± 0.036	0.681 ± 0.049
99	0.770 ± 0.021	0.696 ± 0.022

4.5 Sensitivity Analysis

Which tasks to escalate. We first examine whether uncertainty helps identify which tasks should receive strong-model assistance. Figure 3 varies the escalation budget and compares *LEASH* with random task-level routing at matched escalation rates. Across operating points, *LEASH* consistently achieves higher task success, indicating that its gains come from selectively allocating strong-model computation to more informative tasks rather than simply increasing strong-model usage.

When to escalate. We next examine whether the timing of handoff matters once a task is likely to require escalation. We focus on an escalation-prone subset of tasks that frequently receive strong-model assistance and compare *LEASH* with a shuffled-timing baseline. As shown in Figure 4, *LEASH* achieves 64.6% task success at $0.67 per task, compared with 60.4% at $0.79 under shuffled timing. This 4.2-point improvement, together with 15% lower cost, suggests that effective weak-to-strong escalation depends not only on selecting the right tasks, but also on transferring control at the right point during execution.

Reference bank size. We examine the effect of reference bank size on initial-stage uncertainty estimation. As shown in Table 5, increasing the bank size from 40 to 99 improves AUROC from 0.701 to 0.770 for Qwen3.5-35B-A3B and from 0.633 to 0.696 for EvoCUA-8B, while reducing variability.

5 Conclusion

We introduced *weak to strong escalation* as a general inference paradigm for adaptive agent systems, in which inexpensive weak models handle routine execution and stronger models are invoked selectively when additional capability is needed. Under this paradigm, we proposed *LEASH*, a training-free, uncertainty-aware framework that supports initial routing, runtime escalation, and continued post-handoff oversight. Our results suggest that a shared latent uncertainty signal can support both selective escalation and post-handoff oversight, providing a practical mechanism for dynamically allocating model capability while retaining continued control over strong-agent execution.

LIMITATION

Our current evaluation is limited to computer-use agents on OSWorld, so the generality of weak-to-strong escalation beyond this domain remains to be validated. In addition, our uncertainty score estimates weak-agent failure risk rather than the expected benefit of escalation, which may differ when both weak and strong agents struggle on the same task. Finally, post-handoff Shadow monitoring operates under a distribution shift from weak-agent trajectories, and its intervention effect is not causally isolated in our current evaluation.

AI USE STATEMENT

Generative AI tools were used solely to assist with language editing, such as improving grammar, phrasing, and readability. All scientific content, including the methodology, experimental design, analyses, interpretations, and conclusions, was developed and verified by the authors, who take full responsibility for the manuscript.

ETHICS STATEMENT

This work follows the ICLR Code of Ethics. All experiments are conducted on existing publicly available benchmarks and do not involve human participants, personally identifiable information, or newly collected user data.

REPRODUCIBILITY STATEMENT

We provide the implementation code in the supplementary materials. The paper and appendix document the experimental setup, model configurations, uncertainty estimation procedure, calibration strategy, evaluation protocol, and cost accounting used in our experiments to support reproduction of the reported results.

REFERENCES

Reyna Abhyankar, Qi Qi, and Yiying Zhang. Osworld-human: Benchmarking the efficiency of computer-use agents. *Proceedings of Machine Learning and Systems*, 8:482–494, 2026.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pp. 22924–22946, 2025a.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025b.

Anthropic. Introducing claude sonnet 4.5, September 2025. URL `https://www.anthropic.com/news/claude-sonnet-4-5`.

Anthropic. Introducing claude opus 5, July 2026a. URL `https://www.anthropic.com/news/claude-opus-5`.

Anthropic. Introducing Claude Opus 5.5, 2026b. URL `https://www.anthropic.com/claude-opus-5-5`.

Amos Azaria and Tom Mitchell. The internal state of an llm knows when its lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 967–976, 2023.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

Iván Vicente Moreno Cencerrado, Arnau Padrés Masdemont, Anton Gonzalvez Hawthorne, David Demitri Africa, and Lorenzo Pacchiardi. No answer needed: Predicting llm answer accuracy from question-only linear probes. *arXiv preprint arXiv:2509.10625*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Amala Sanjay Deshmukh, Kateryna Chumachenko, Tuomas Rintamaki, Matthieu Le, Tyler Poon, Danial Mohseni Taheri, Ilia Karmanov, Guilin Liu, Jarno Seppanen, Arushi Goel, et al. Nemotron 3 nano omni: efficient and open multimodal intelligence. *arXiv preprint arXiv:2604.24954*, 2026.

Wenbo Gao, Renxi Liu, Xian Wang, Fang Guo, Shuai Yang, Xi Chen, Hui-Ling Zhen, Hanting Chen, Weizhe Lin, Xiaosong Li, et al. Agentcollab: A self-evaluation-driven collaboration paradigm for efficient llm agents. *arXiv preprint arXiv:2603.26034*, 2026.

Jiuzhou Han, Wray Buntine, and Ehsan Shareghi. Towards uncertainty-aware language agent. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6662–6685, 2024.

Ziwei Ji, Delong Chen, Etsuko Ishii, Samuel Cahyawijaya, Yejin Bang, Bryan Wilie, and Pascale Fung. Llm internal states reveal hallucination risk faced with a query. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 88–104, 2024.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Christina Knight, and Zifan Wang. Reliable weak-to-strong monitoring of llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 78083–78138, 2026.

Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Multi-agent computer use. *arXiv preprint arXiv:2606.01533*, 2026.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in llms. *arXiv preprint arXiv:2406.15927*, 2024.

Divake Kumar, Sina Tayebati, Devashri Naik, Amanda Sofie Rios, Nilesh Ahuja, Omesh Tickoo, Ranganath Krishnan, and Amit Ranjan Trivedi. Uncertainty quantification for computer-use agents: A benchmark across vision-language models and gui grounding datasets. *arXiv preprint arXiv:2606.25760*, 2026.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=VKGTGGcwl6`.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Ji'an Lei and Jian Huang. Tacit-switch: Cost-aware model escalation for llm agents from censored supervision. *arXiv preprint arXiv:2608.27911*, 2026.

Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Adaptive vision-language model routing for computer use agents. *arXiv preprint arXiv:2603.12823*, 2026.

Marko Medvedev, Kaifeng Lyu, Dingli Yu, Sanjeev Arora, Zhiyuan Li, and Nathan Srebro. Weak-to-strong generalization even in random feature networks, provably. *arXiv preprint arXiv:2503.02877*, 2025.

Changdae Oh, Seongheon Park, To Eun Kim, Jiatong Li, Wendi Li, Samuel Yeh, Sean Du, Hamed Hassani, Paul Bogdan, Dawn Song, et al. Uncertainty quantification in llm agents: Foundations, emerging challenges, and opportunities. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16219–16250, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026. URL https://openai.com/index/gpt-6-astra/.

Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, and Yonatan Belinkov. Llms know more than they show: On the intrinsic representation of llm hallucinations. In *International Conference on Learning Representations*, volume 2025, pp. 66880–66913, 2025.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Sitong Pan, Yipeng Shen, Yilin Lu, Caiwen Ding, Lu Cheng, and Qianwen Wang. Monitoring web agents without internal signals: Observable trajectories and key-step supervision. *arXiv preprint arXiv:2609.02057*, 2026.

Qwen Team. Qwen3.8-max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Jie Ren, Stanislav Fort, Jeremiah Liu, Abhijit Guha Roy, Shreyas Padhy, and Balaji Lakshminarayanan. A simple fix to mahalanobis distance for improving near-ood detection. *arXiv preprint arXiv:2106.09022*, 2021.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1(11):6, 2025.

Seamus Somerstep, Felipe Maia Polo, Moulinath Banerjee, Yaacov Ritov, Mikhail Yurochkin, and Yuekai Sun. A transfer learning framework for weak to strong generalization. In *International Conference on Learning Representations*, volume 2025, pp. 100714–100748, 2025.

Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *International conference on machine learning*, pp. 20827–20840. PMLR, 2022.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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. *Advances in Neural Information Processing Systems*, 37:2686–2710, 2024.

Qingni Wang, Yue Fan, and Xin Eric Wang. Safeground: Know when to trust gui grounding models via uncertainty calibration. *arXiv preprint arXiv:2602.02419*, 2026.

Jinbiao Wei, Kangqi Ni, Yilun Zhao, Guo Gan, and Arman Cohan. Step-level optimization for efficient computer-use agents. *arXiv preprint arXiv:2604.27151*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026.

Yihao Xue, Jiping Li, and Baharan Mirzasoleiman. Representations shape weak-to-strong generalization: Theoretical insights and empirical predictions. *arXiv preprint arXiv:2502.00620*, 2025.

Biao Yi, Xueyu Hu, Yurun Chen, Shengyu Zhang, Hongxia Yang, and Fan Wu. EcoAgent: An efficient device-cloud collaborative multi-agent framework for mobile automation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

A METHOD DETAILS

A.1 REPRESENTATIONS AND ESTIMATOR FITTING

Layer and token. We use the last transformer layer of each weak backbone (Table 6). For the runtime channel, h_t is the hidden state at the last generated token of the action; we apply no pooling within a step and no temporal window across steps, so every step contributes one representation. For the initial channel, we reissue the task's first request with a one-token generation limit and read the hidden state at the final prompt position, obtained during prefill.

Table 6: Hidden layer used for each weak backbone (0-indexed layer / number of layers).

Backbone	Layer
EvoCUA-8B	35 / 36
Qwen3.5-35B-A3B	39 / 40
Nemotron-3-Nano-Omni (30B-A3B)	51 / 52
Qwen3.8-27B	63 / 64

Standardization and covariance. Standardization uses the feature-wise mean and standard deviation of the channel's bank, pooled across both classes. The class centroids are computed in the standardized space, and $\widehat{\Sigma}_c$ is the pooled within-class scatter normalized by $n - 2$, where n is the number of representations in the bank. We never form $\Sigma_{c,\lambda}^{-1}$ explicitly: with the singular value decomposition of the class-centered bank, the distance in Eq. (2) is evaluated in closed form through the Woodbury identity.

Selecting the ridge parameter. We select λ_c from a grid spanning 10^{-2} to 10^{3} by task-grouped 10-fold cross-validated AUROC, separately for each backbone and channel. Grouping by task is essential: steps from one trajectory share much of their context and, at runtime, their segment label. In a preliminary check, a split that ignored task membership produced a near-perfect AUROC of 0.997, which reflected leakage across steps of the same trajectory rather than genuine discrimination.

A.2 REFERENCE BANKS AND THRESHOLD CALIBRATION

Reference banks. For each backbone, we construct a small held-out reference bank of Manager-guided trajectories collected with escalation disabled and disjoint from the evaluation tasks. Initial representations are paired with task-level outcomes, while runtime representations inherit the outcomes of their corresponding subgoal segments, providing finer-grained supervision for step-level uncertainty estimation.

Held-out scoring. Scores computed on the same trajectories used to fit the estimator separate successes from failures more cleanly than scores on new tasks, which would bias the thresholds. We therefore obtain every score in Eq. (4) by task-grouped 10-fold cross-validation: each reference task is scored by an estimator fit on the other nine folds, and all steps of a task stay in the same fold. After the thresholds are set, the deployed estimators are refit on the complete banks and the thresholds are kept unchanged. The reference trajectories can be collected with escalation disabled because the runtime gate does not change execution before its first crossing, so whether a task would be escalated depends only on its escalation-free trajectory. Given the scores, threshold selection uses no outcome labels: it controls how often each gate fires, not how often escalated tasks would have failed, and realized rates on the evaluation tasks can differ from the targets (Table 1).

Targets and thresholds. All four backbones use the same targets, $p_q = 0.25$ and $p_{\text{esc}} = 0.50$, so that $p_a = (p_{\text{esc}} - p_q)/(1 - p_q) = 1/3$. The thresholds τ_q and τ_a are exact order statistics of the held-out scores in Eq. (4). Because the reference set is finite, the achievable rates are discrete, with a resolution of one reference task.

A.3 POST-HANDOFF OVERSIGHT

Probe schedule and context. Shadow probes run at the first strong-agent step after handoff and at every third step thereafter. At every strong-agent step, including steps without a probe, the executed action is appended to the weak model's history in that model's own action format, so that each probe conditions on the actual trajectory rather than on the pre-handoff context. A probe calls the weak agent's own action-generation routine, with its usual system prompt, tool schema, and image history, and does not invoke the Manager. The candidate action is logged for analysis but never executed or shown to the strong agent.

Shadow threshold. τ_{sh} is the 90th percentile of the pooled per-step runtime scores over the 99-task reference pool, computed separately for each backbone. It is derived directly from the reference pool rather than from the escalation thresholds. For EvoCUA-8B, the in-sample value ($+1.7309$) is close to the corresponding out-of-fold value ($+1.7106$).

Checkpoint budget. At most six checkpoints can fire in an episode. In addition, once two checkpoints have changed the strong agent's action, further firings are suppressed; checkpoints that leave the action unchanged do not count toward this second limit. Because probes occur every three steps, consecutive checkpoints are at least three steps apart.

Checkpoint mechanics. When a checkpoint fires, the strong agent's conversation is truncated to its length at the start of the current step, which removes the pending action; the neutral checkpoint message in Figure 6 is appended; and the strong agent re-decides the step with a raised reasoning-effort setting. The weak model's candidate action is deliberately excluded from the checkpoint message, so the strong agent is never steered toward a specific action proposed by the weaker model.

B ADDITIONAL EXPERIMENTAL RESULTS

B.1 EXPERIMENTAL DETAILS

Evaluation setting. We evaluate on OSWorld with screenshot observations and a shared budget of 50 environment actions per task. The lightweight backbones are EvoCUA-8B (Xue et al., 2026), Qwen3.5-35B-A3B (Team, 2026), Nemotron-3-Nano-Omni (30B-A3B) (Deshmukh et al., 2026), and Qwen3.8-27B (Qwen Team, 2026). Claude Opus 5 (Anthropic, 2026a) serves as both the Manager and strong executor, and Claude Sonnet 4.5 (Anthropic, 2025) is an additional always-on reference. The always-on references run the strong model alone, without the Manager, so they isolate strong-model capability rather than our scaffolding.

Configurations and evaluation sets. *Alone* uses only the weak agent; *Manager* adds subgoal planning and verification; *Escalation* further enables initial and runtime handoff; and *LEASH* additionally enables post-handoff Shadow monitoring and checkpoint intervention. These configurations are cumulative. Shared results below follow Tables 1 and 3. The evaluation task pool contains 262 tasks. Most runs completed all of them; evaluator exceptions cost one task each in the EvoCUA-8B Escalation run and in the Qwen3.5-35B-A3B, Nemotron-3 and Qwen3.8-27B Manager runs, and two tasks in the Nemotron-3 Escalation run. The shortfalls do not overlap, so rather than impose one global set we restrict each comparison to the tasks completed by every run it involves; the resulting denominators are stated where they differ.

Reference banks and calibration. No reference task appears among the 262 evaluation tasks. We obtain task-grouped 10-fold out-of-fold scores, keeping all observations from a task in one fold; preprocessing and estimator fitting use only the training folds, while the ridge coefficient is selected once on the full bank by the same grouped criterion. Initial scores use final task outcomes and runtime scores use Manager-assigned subgoal outcomes. Thresholds are order statistics of the held-out scores targeting a 25% initial and 50% overall escalation frequency, with runtime calibration based on per-task maximum scores. The resulting thresholds are then used with estimators refitted on the full reference bank, which shifts the score scale; together with the shift between reference and evaluation distributions this is why realized escalation rates (43–64%) differ from the calibration targets.

Table 7: Gate configurations for Qwen3.5-35B-A3B. Each configuration retains the Manager; the first two disable post-handoff Shadow monitoring and checkpoint intervention.

Gate configuration	Success (%) ↑	Cost ($) ↓
Initial only	59.92	0.3122
Runtime only	71.65	0.4169
LEASH	**73.66**	0.4117

Post-handoff execution. After handoff, Shadow probes query the lightweight model every three strong-agent steps. Probe actions are not executed; their hidden representations provide the monitoring scores. Higher scores indicate greater uncertainty and can trigger a checkpoint that asks the strong agent to reassess its current strategy while retaining control.

Cost estimation. We estimate inference cost from recorded token usage using OpenRouter reference prices, accounting for cached tokens where applicable; cached reads dominate strong-executor input, so pricing all input at the uncached rate would substantially overstate cost. Total cost includes lightweight-agent, Manager, strong-executor, and monitoring calls. For locally served models, this is a reference inference-cost estimate rather than measured hardware expenditure, and only Qwen3.5-35B-A3B is priced from a listing of the same model; the other three use same-class proxies. Missing usage is not treated as zero cost: the standalone runners record no token usage, which is why three *Alone* costs are reported as dashes, and nine tasks of the always-on Opus run carry no token log and are therefore excluded from that row's cost mean but not from its success rate.

B.2 Uncertainty Estimation and Reference-Bank Sensitivity

Estimator comparison. We compare pooled LDA with Euclidean distance, kNN, and token-probability baselines. Performance varies across backbones and execution stages, with pooled LDA providing the strongest overall discrimination in most settings. Runtime AUROC reflects discrimination of Manager-assigned subgoal-segment outcomes rather than individual action correctness.

Reference-bank size. Table 5 reports initial-stage AUROC as the bank is subsampled. Beyond the mean trend, the dispersion across subsamples is the operationally relevant quantity: at 40 tasks the spread is roughly ± 0.10, which is wider than the gap between the estimators being compared, so a bank of that size cannot rank them. It narrows to about ± 0.02 at the full bank. Because thresholds are quantiles of this same pool, a small bank also gives coarse control of the realized escalation rate.

B.3 Additional Component Ablations

Initial and runtime escalation. Table 7 compares different gate configurations for Qwen3.5-35B-A3B. Runtime-only escalation achieves higher success than initial-only escalation, while the complete framework attains the best overall performance. These results indicate that runtime monitoring provides the stronger escalation signal, with limited additional gain from the initial gate in this setting.

B.4 Inference Cost

The total costs follow Table 1; we do not separately add monitoring charges to these values. With Qwen3.5-35B-A3B, the complete framework costs $0.4117 per task, compared with $0.5094 for always-on Opus, a 19.2% reduction. Among the other backbones, total costs are $0.3627 for EvoCUA-8B, $0.3500 for Nemotron-3, and $0.3622 for Qwen3.8-27B.

Runtime scoring reuses hidden states from action generation, so it does not require another model-generation call. Initial and Shadow probes do require lightweight-model inference, and checkpoints may incur additional strong-model computation. These costs are included in the complete workflow's reported total. The cost comparison therefore concerns the entire execution policy, rather than the uncertainty estimator in isolation.

B.5 POST-HANDOFF OVERSIGHT

Overall policy comparison. In the main-text Qwen3.5-35B-A3B ablation, Escalation achieves 67.94% success at $0.4050 per task, while the complete framework achieves 73.66% at $0.4117. This is a 5.72-percentage-point difference for $0.0067 additional mean cost. However, the policies also differ in which tasks they escalate, so the overall difference cannot be assigned solely to checkpoint intervention.

Subset analysis. We further characterize the tasks identified by Shadow oversight. Table 4 partitions escalated tasks according to whether both policies escalate and whether the Shadow policy triggers a checkpoint. The lower always-on Opus accuracy on triggered tasks suggests that Shadow tends to activate on harder trajectories. These comparisons are descriptive rather than causal because the subsets are policy-dependent and different policies may follow different intermediate trajectories.

C PROMPT TEMPLATES

Figures 5 and 6 show the core prompts used for planning, verification, handoff, and checkpoint intervention. The planning prompt is excerpted, with omissions marked by `[...]`; braces denote runtime placeholders in the handoff template. Escalation gates use statistical scores and require no prompt. Shadow candidate actions are neither executed nor shown to the strong agent.

Execution context. The inherited OSWorld system prompt mentions shell and editor tools, although this harness exposes only the `computer` tool. A system suffix corrects the sudo password for the Docker environment. The diagnostic state query is logging-only, and the alternative `think_reflect` prompt is not used by the hidden-state checkpoint.

D CASE STUDY

Runtime escalation case studies. Figure 7–9 present three representative examples of runtime escalation. Across all three cases, the weak agent remains on an unproductive GUI-based trajectory before the runtime gate triggers handoff. After takeover, the strong agent changes the execution strategy rather than simply continuing the weak agent's actions.

In the first case, the weak agent repeatedly interacts with an unresponsive macro interface while attempting to standardize text-box fonts. *LEASH* hands off at step 11, after which the strong agent switches from the macro GUI to a terminal-based `python-pptx` solution and completes the task in 27 actions; the Manager-only run instead exhausts the 50-action budget.

In the second case, the weak agent repeatedly opens, inspects, and closes candidate photos. Handoff occurs at step 8, and the strong agent replaces this loop with keyboard navigation and multi-selection, successfully creating the requested archive in 27 actions. The corresponding Manager-only run creates an incorrect archive and receives a score of zero.

In the third case, the weak agent spends 23 steps browsing and searching without installing the requested LibreOffice extension. After handoff at step 24, the strong agent switches to a terminal-based installation strategy and successfully installs LanguageTool, whereas the Manager-only run exhausts the action budget without completing the installation.

Together, these examples illustrate that runtime escalation can identify stalled or inefficient execution trajectories and enable a strong agent to adopt qualitatively different strategies rather than merely continuing the weak agent's current path.

Manager: task decomposition (system prompt excerpt)

```
You are a planning agent for a SINGLE computer-use agent operating ONE
↪  Ubuntu desktop. You are a regular LLM  you cannot click or type
↪  yourself; a downstream agent executes pyautogui actions one step
↪  at a time. Your job is to take one desktop task and decompose it
↪  into a SHORT, ORDERED list of subgoals the agent completes
↪  sequentially, each with a concrete done-criterion that a verifier
↪  can confirm from a screenshot.

[...]

## Output format
Respond with VALID JSON only (no markdown code fences, no extra
↪  prose). Schema:
{
  "task_analysis": "one sentence on the task and how you split it",
  "subgoals": [
    {"id": "subgoal_1", "description": "what to accomplish", "done":
    ↪  "observable on-screen condition that confirms completion"},
    {"id": "subgoal_2", "description": "...", "done": "..."}
  ]
}
Rules: ids are unique and sequential (subgoal_1, subgoal_2, ...); 16
↪  subgoals is typical; the subgoals array must never be empty.
```

Manager: strict verification (appended system instruction)

```
## Strict verification (IMPORTANT)
When you choose `advance`, you MUST positively confirm from the
↪  CURRENT screenshot that the focus sub-goal's completion criterion
↪  is CONCRETELY and FULLY satisfied by the actual end-state  not
↪  merely that the relevant application, window, file, menu, or
↪  element is present, open, or focused. If you cannot positively
↪  verify the required end-state from what is visibly on screen, do
↪  NOT advance  choose `no_change` or `revise` instead. Do not assume
↪  an action succeeded just because it was attempted.
```

Figure 5: Manager prompt excerpts. The strict verification instruction is appended to the replanning system prompt; it is not a standalone verifier prompt.

Strong-agent handoff (message template)

```
### User (for each retained step):
Earlier step {s} (previous agent's attempt):
[Step screenshot]

### Assistant:
Thought: {reason[:200]}
Action: {nl[:120]}

### User (handoff):
[Current screenshot]
{instruction}

The attempts above were made by a weaker agent and did not complete
↪  the task. You are taking over. Work from the CURRENT screenshot.
```

Post-handoff checkpoint (intervention message)

```
CHECKPOINT. Pause before acting.
Answer these three questions in at most 60 words total:
  1. What is the current state of the task, and what have your recent
  ↪  actions actually changed?
  2. Can the approach you are currently using complete this task, or
  ↪  has it stalled?
  3. If it can, say so and continue with it. If it cannot, say what
  ↪  you will do differently.
Then take your next action.
```

Figure 6: Handoff and checkpoint prompts. Handoff retains up to six recent weak-agent steps by default; bracketed screenshot markers indicate image inputs, not literal prompt text. The neutral checkpoint requests reassessment without asserting that execution has stalled and explicitly permits continuing the current approach.

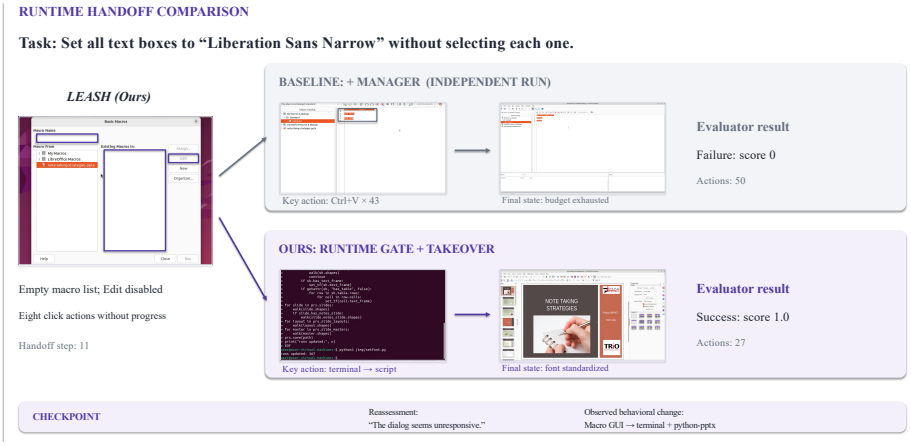

Figure 7: **Runtime escalation case study I: font standardization.** The weak agent becomes stuck in repeated macro-interface interactions. After handoff at step 11, the strong agent switches to a terminal-based `python-pptx` strategy and successfully completes the task.

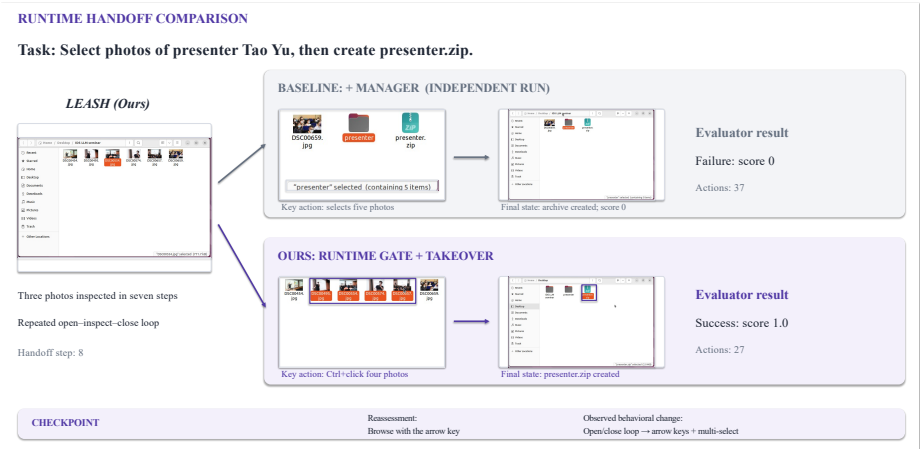

Figure 8: **Runtime escalation case study II: photo selection.** The weak agent enters a repeated open–inspect–close loop. After handoff at step 8, the strong agent changes to keyboard navigation and multi-selection and successfully creates the requested archive.

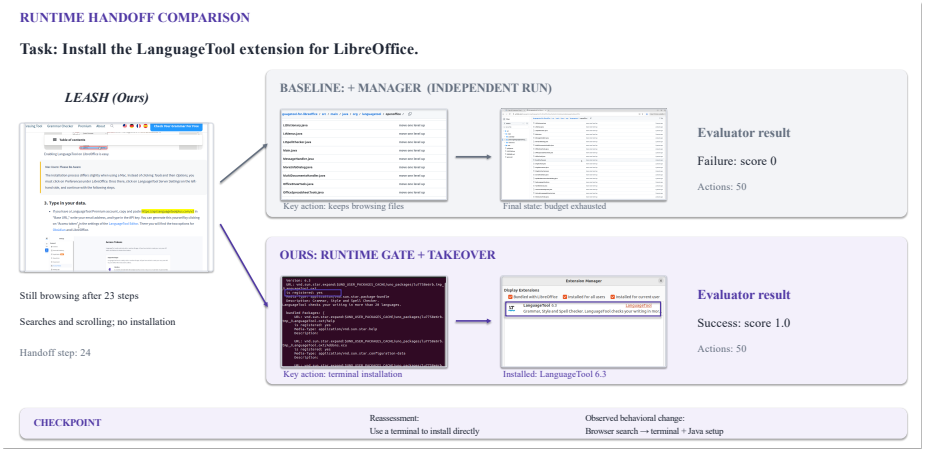

Figure 9: **Runtime escalation case study III: extension installation.** The weak agent continues browsing without completing the installation. After handoff at step 24, the strong agent switches to a terminal-based installation strategy and successfully installs the requested extension.