Can Weak Models Explore? Scope Then Cover for LLM Agent Adaptation

Anonymous ACL submission

Abstract

Adapting a language-model agent to a new environment requires exploration, and the prevailing assumption is that stronger models explore better. We show this assumption fails in two regimes: when the environment offers no signal about target location, scaling model size yields no gain; when the candidate space exceeds the inspection budget, all scales collapse to random search. In both cases, capable-model compute is wasted on a step where capability does not contribute. We identify the root cause as a structural asymmetry within exploration: locating a promising candidate subspace is capability-bound, while verifying whether the target lies inside it is not. Dispatching a capable actor for scoping and frozen cheap workers for parallel verification exploits this asymmetry directly. We instantiate this as a single explore action compatible with frozen test-time actors and online experiential learning hosts. On ALFWorld, WebShop, and SWE-Bench-Verified, the explore action improves over a no-explore strong-actor baseline across all three benchmarks, with extra cost absorbed by cheap workers.

1 Introduction

Language models deployed as interactive agents accumulate rich experience through environment interaction, yet the dominant post-training paradigm treats this experience as a byproduct rather than a resource. A growing line of work on online experiential learning addresses this gap, enabling agents to continuously improve from their own deployment trajectories (Shinn et al., 2023; Zhao et al., 2023; Wang et al., 2023a; Zuo et al., 2025; Ye et al., 2026). The ceiling of this improvement, however, is set by exploration: an agent can only learn from what it finds, and finding the right things in a large, unfamiliar environment is itself a hard problem. The natural assumption, implicit in most agent work, is that stronger models explore better (Krishnamurthy et al., 2024; Nie et al., 2024; Tajwar et al., 2025; Jiang et al., 2025). We show this assumption has two systematic failure modes, and that the way it fails suggests a different decomposition of the exploration problem.

Exploration in interactive environments involves two distinct sub-problems: reasoning about *where* the target is likely to be, and *verifying* whether it is actually there. Prior work treats these as a single capability and evaluates them together through cumulative task reward. Through two controlled probes (Figure 1), we find that scaling model size fails in two different regimes: when the environment provides no information about target location, all scales reduce to counting inspection touches regardless of capability; when the candidate space outgrows any practical inspection budget, cost-matched scouts across a wide scale range converge onto the random baseline regardless of belief quality. Between these boundaries, capability helps the reasoning sub-problem but contributes little to verification, a distinction that single-model exploration cannot exploit (§2).

This motivates a decomposition we call *scope-then-cover*: a capable actor identifies a compact, high-recall candidate subspace, then frozen cheap workers verify candidates in parallel inside it. We formalize the gain as a scope-lift factor r/ρ (recall over subspace fraction) and show in §3 that scope-then-cover strictly dominates direct exploration whenever $r > \rho$, with extra token cost falling entirely on cheap workers. We expose this decomposition as a single *explore action* added to the actor's action space, compatible with both frozen test-time actors and online weight-updating hosts (Ye et al., 2026) (§4).

The two failure modes are not just theoretical—they dominate real no-explore baselines. Auditing every failed trajectory of strong-actor baselines and tagging each as scope-bound (F1: agent never locates the right subspace—no `take`/`put` in ALFWorld, score $= 0$ in WebShop, empty patch

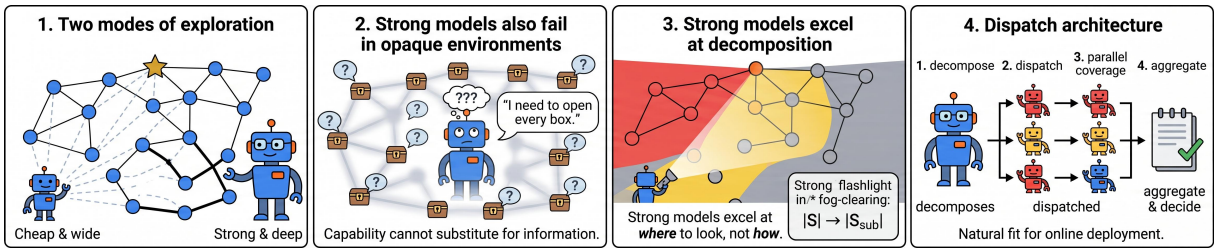


Figure 1: Overview of our argument. (1) Two modes of exploration: cheap-and-wide vs. strong-and-deep. (2) Strong models also fail in opaque environments: capability cannot substitute for information. (3) Strong models excel at decomposition, shrinking the effective state space $|\mathcal{S}| \to |\mathcal{S}_{\text{sub}}|$. (4) Dispatch architecture: a strong actor decomposes and dispatches subproblems to weak explorers in parallel, then aggregates and decides.

on SWE-Bench) or cover-bound (F2: right subspace, but execution or budget fails), we find that on SWE-Bench-mini F1 covers 26% of all tasks and F2 a further 60%; on ALFWorld F2 alone explains 69% of failures; on WebShop F1 explains 56%. Adding the explore action as an on-demand tool the actor invokes itself shrinks *both* modes simultaneously. On SWE-Bench-mini, F1 collapses from 26.0% to 0.0% (−100% relative) and F2 from 60.0% to 42.0% (−30%); on ALFWorld F2 drops from 68.6% to 55.5% (−19%); on WebShop F1 drops from 56.0% to 48.5% (−13%). The corresponding pass-rate gains (Table 1) are **+13.0** on ALFWorld (29.6→42.6), **+4.3** on WebShop (32.2 → 36.5), and **+10.0** on SWE-Bench-Verified-mini-50 (48.0 → 58.0), with extra cost absorbed by cheap workers.

Contributions. **(i)** A structural decomposition of exploration into two capability-asymmetric subproblems, validated by controlled probes showing that scaling fails in orthogonal regimes (§2). **(ii)** A budget–prior framework deriving a scope-lift factor r/ρ that predicts when and by how much the decomposition dominates direct exploration (§3, §4). **(iii)** Empirical validation across three environments and two host paradigms, with a content finding on SWE-Bench: workers help most when returning environment-grounded facts rather than judgments (§5).

2 Two Boundaries Where Capability Stops Paying Off

We probe two regimes where scaling the explorer should help and find it does not: when the prior is uninformative, and when the candidate space outgrows the cost budget.

Probe 1: when the prior is uninformative, capability stops mattering. ALFWorld pick-and-place (Shridhar et al., 2020) with randomized receptacle assignment makes the target receptacle independent of the task description. Each Qwen3 scout (0.6B / 4B / 8B) inspects up to $K \in \{1, \ldots, 8\}$ receptacles; we record whether any inspected receptacle holds the target. Coverage hit rate is set entirely by K (Fig. 2a, left): a 13× scale-up does not separate the three curves at any K, and the marginal return per inspection (right) shows no consistent ordering across scales. **Capability and inspection budget are decoupled axes here, and the capability axis is approximately flat.**

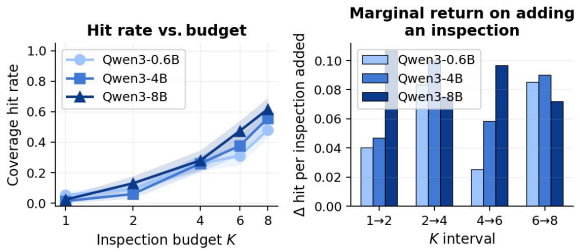
Probe 2: when $|\mathcal{S}|$ outgrows the budget, all scales collapse to random. WebShop (Yao et al., 2022a) sweeps candidate-pool size $|\mathcal{S}|$ from 10 to 1000 under wall-clock-matched budgets (0.6B at K=30, 4B at K=12, 8B at K=8). **Target-Inclusion Rate (TIR)** is the fraction of tasks where the gold product is in the scout's K proposals (Coverage TIR) or is the final pick (Decision TIR). Up to $|\mathcal{S}|$=30 the cheap scout's larger K saturates Coverage TIR while cost-matched stronger scouts already trail (Fig. 2b, left); from $|\mathcal{S}|\geq$100 the three scales converge onto the random baseline, and Decision TIR (right) follows the same collapse. **Past $|\mathcal{S}| \sim 100$ in our setup, larger-model scouts at matched cost do not improve coverage TIR over the cheap-scout reference.**

The two probes are jointly negative: capability has nothing to convert when the prior is uninformative, and what it can convert does not scale with $|\mathcal{S}|$ at fixed cost. §3 reads both off the same equation.

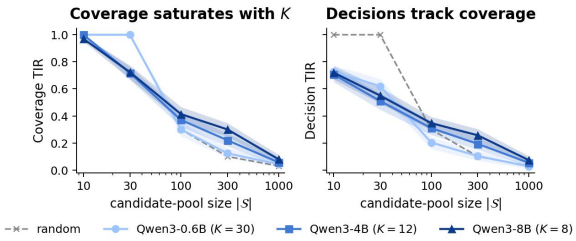
3 A Budget–Prior Framework for Exploration

3.1 Setup

We model the localization sub-problem within a multi-step agent loop. At any decision point, the



(a) **Probe 1: capability is a flat axis.** Coverage hit vs. inspection budget K (left) and marginal Δ hit per added inspection (right) for three Qwen3 scales on randomized ALFWorld. 100 tasks $\times$ 3 seeds, bootstrap-over-tasks 95% CI.



(b) **Probe 2: scales collapse to random past $|\mathcal{S}|\sim 100$.** Coverage TIR (left) and Decision TIR (right) vs. candidate-pool size $|\mathcal{S}|$ for cost-matched Qwen3 scouts on WebShop. 100 tasks $\times$ 3 seeds, bootstrap-over-tasks 95% CI.

agent faces a candidate space $\mathcal{S}$ (receptacles in a room, products in a catalog, files in a repository) and an unknown target $s^* \in \mathcal{S}$ drawn from the task-corpus marginal P^*. The agent holds a belief $b \in \Delta(\mathcal{S})$ initialized from pretraining knowledge and the task description, and updated as observations return. Within a single exploration decision it has a budget of T candidate inspections; let c_a and c_w denote the per-token cost of the capable actor and the cheap worker, with $c_w \ll c_a$.

The natural policy is greedy on b: inspect the T candidates the agent finds most probable. Its hit probability is

$$P(\text{hit}\,;\, b,\, T) \;=\; \sum_{s \,\in\, \text{top}_T(b)} P^*(s^* = s). \qquad (1)$$

Capability enters Eq. 1 through exactly one channel: a more capable model produces a belief b that places more mass on the true target. It cannot substitute for budget T, nor compensate for a candidate space too large for T to cover.

3.2 Two Boundaries

Both failure modes of §2 are readings of Eq. 1 under different parameter regimes.

Boundary 1: uninformative prior. When the task description carries no information about s^*, b is uniform regardless of which model produced it. Eq. 1 reduces to $P(\text{hit}) = T/|\mathcal{S}|$, a quantity that depends only on budget. Capability has nothing to sharpen, and the flat curves across a 13$\times$ scale-up follow directly.

Boundary 2: space outgrows budget. When b is non-uniform but $|\mathcal{S}|$ is large, the mass captured by the top-T slice shrinks proportionally, converging to the random baseline $T/|\mathcal{S}|$ even under a well-calibrated belief. Past $|\mathcal{S}| \sim 100$ in our setup, the inspection budget becomes the binding constraint, not belief quality.

The two failures share the same root: in both cases, $\text{top}_T(b)$ captures mass indistinguishable from a random draw. The difference is only in why: uninformative b in Boundary 1, insufficient T relative to $|\mathcal{S}|$ in Boundary 2.

3.3 Scope-then-Cover

Boundary 2 identifies the lever: reduce the effective candidate space before covering it. A *scope* σ proposes a subspace $\mathcal{S}_{\text{sub}} \subseteq \mathcal{S}$, characterized by its fractional size $\rho = |\mathcal{S}_{\text{sub}}|/|\mathcal{S}|$ and its recall $r = P^*(s^* \in \mathcal{S}_{\text{sub}})$. Covering after scoping has hit probability

$$P(\text{hit}\,;\, b,\, T,\, \sigma) \;=\; r \cdot P(\text{hit in } \mathcal{S}_{\text{sub}}\,;\, b',\, T), \qquad (2)$$

where $b' = b(\cdot \mid s^* \in \mathcal{S}_{\text{sub}})$ is the conditional belief inside the subspace. The effective candidate space is now $\rho \cdot |\mathcal{S}|$, weighted by recall r: the same budget T that failed to localize in $\mathcal{S}$ may now succeed in $\mathcal{S}_{\text{sub}}$.

Proposition 1 (Scope Lift)**.** *In the regime $T < \rho|\mathcal{S}|$ and under approximate uniformity of b' on $\mathcal{S}_{\text{sub}}$:*

$$\frac{P(\text{hit}\,;\, b,\, T,\, \sigma)}{P(\text{hit}_{\text{direct}})} \;=\; \frac{r}{\rho}. \qquad (3)$$

Scope-then-cover strictly dominates direct cover whenever $r > \rho$.

The proof follows from Eqs. 1–2: in the large-$|\mathcal{S}|$ limit, direct cover degrades to $T/|\mathcal{S}|$, while scope-then-cover yields $r \cdot T/(\rho|\mathcal{S}|)$; their ratio is r/ρ. The gain depends entirely on scoping quality and is independent of $|\mathcal{S}|$ and T.

Three consequences fix the wiring.

Scoping must come from the capable tier. The gain r/ρ is set by the actor's ability to produce

a small subspace with high recall, precisely the capability-bound property of b in Eq. 1. No amount of cheap parallel coverage can compensate for a poorly scoped subspace.

Covering need not. A tight scope ($\rho \ll 1$) extracts most of the signal in the actor's prior, leaving a residual belief b' that is approximately uniform on $\mathcal{S}_{\text{sub}}$. This mirrors Boundary 1: capability cannot help when the local prior is uninformative. Per-touch yield inside $\mathcal{S}_{\text{sub}}$ is therefore $1/|\mathcal{S}_{\text{sub}}|$ regardless of worker capability. When ρ is close to 1 the scope has extracted little signal and b' may retain structure (cf. Table 2, WebShop row), but the gain $r/\rho \approx 1$ already signals that scoping has not helped.

Cover cost belongs to the cheap tier. Since per-touch yield is independent of worker capability, assigning cover to a worker at cost c_w rather than c_a saves a factor c_a/c_w with no loss in hit rate:

$$\begin{aligned} T_{\text{scope+cover}} &= \tau_a \cdot c_a + T \cdot \ell \cdot c_w \\ &\ll \tau_a \cdot c_a + T \cdot \ell \cdot c_a, \end{aligned} \tag{4}$$

where τ_a is the actor's per-task token spend and ℓ the per-touch length. The saving $1 - c_w/c_a$ is independent of ρ and $|\mathcal{S}|$, and Table 1 confirms the direction empirically: the actor's step count is essentially unchanged across wirings, while added work is absorbed by cheap worker touches.

4 Method: The Explore Action

Motivation. We borrow the high-level/low-level split of the *options* framework (Sutton et al., 1999), but re-purpose it across capability tiers rather than time horizons: the strong actor plays the high-level controller and decides *where* to look, while frozen weak workers act as inner sub-policies that actually *look*. Orchestrator/worker LLM patterns (Shen et al., 2023; Wu et al., 2023) route by task type; we route by the (ρ, r) split of §3—scope stays on the actor (capability-bound), cover offloads to parallel cheap workers (uniformity-bound).

The explore action. We expose this split as a single action `explore(scope, query)` added to the actor's action space. When invoked, the actor emits a subspace $\mathcal{S}_{\text{sub}} \subseteq \mathcal{S}$ and a natural-language query; the harness dispatches K frozen weak workers in parallel inside $\mathcal{S}_{\text{sub}}$, and an aggregated observation returns to the actor. What counts as a subspace is environment-specific (a receptacle cluster in ALFWorld, a product category in WebShop, a directory subtree in SWE-Bench); the call signature is invariant. Three properties are load-bearing: (i) the actor names the subspace (delegating scope collapses to uninformed coverage); (ii) workers are frozen and parallel (extra capability inside the subspace is wasted, serial reasoning is what we are avoiding); (iii) only an aggregate returns (otherwise the $O(K)$ wall-clock saving is eaten by an $O(K)$ context blow-up). Pseudocode in Algorithm 1 (Appendix C.7).

Two hosts share the same action. Under a frozen test-time actor (Host A, §5.1), the action extends the action space at deployment; no weights move, the actor adapts through returned observations. Under an online experiential learning host (Ye et al., 2026) (Host B, §5.2), the same action replaces the trajectory-collection step and the resulting trajectories drive a context-distillation update, $\pi_a \leftarrow \arg\min_\pi \mathbb{E}[\mathrm{KL}(\pi \,\|\, \pi_a(\cdot \mid \text{context}))]$. Action, workers, and their frozen status are identical in both hosts; only the surrounding loop differs.

5 Experiments

The experiments work in three passes. §5.1 (Host A) and §5.2 (Host B) test whether the explore action lifts a frozen test-time actor and an OEL-trained actor over a no-explore baseline. §5.3 measures the actor's actual (ρ, r) on each environment, and §5.4 sweeps the cover budget T at fixed scope, to see whether the lift ordering tracks Prop. 1. We then open the wiring: §5.5 varies worker capability to ask where scope-sharpness comes from, and §5.6 varies what the worker sends back on SWE-Bench.

Setup. Three environments span navigation, retrieval, and code-repair: ALFWorld (Shridhar et al., 2020), WebShop (Yao et al., 2022a), and SWE-Bench-Verified-mini-50 (Jimenez et al., 2023). The strong actor is sized to the environment (Qwen3-8B / 30B-A3B / Coder-Next-80B), the weak worker is a frozen Qwen3-4B (1.7B in OEL), and both share the same `explore(scope, query)` interface. Comparisons are strictly per-instance (matched task ID) so decoding noise is absorbed; full model list, decoding settings, and per-environment setup are in Appendix C.

5.1 Host A: frozen test-time actor

With the actor frozen, the only knob is *who decides when* explore fires. We compare a no-explore baseline against four wirings: *forced* (*forced-pre*,

Table 1: **Test-time main result: the explore action under four control policies.** Score is per-instance success rate (ALFWorld, SWE-Bench) or attribute-match $\times 100$ (WebShop). Parentheses show mean decision steps per task. SWE-Bench: *none* = structural (fast) scan.

Wiring	ALFWorld	WebShop	SWE-Bench
none (no explore)	29.6 (39.2)	32.2 (14.1)	48.0 (87.9)
Forced (harness-pinned timing)			
forced-pre	36.6 (35.6)	7.9 (18.2)	52.0 (86.5)
forced-mid	37.3 (36.9)	31.2 (14.2)	50.0 (86.6)
Self-triggered (actor decides when)			
self-signal	**42.6** (34.5)	25.2 (13.8)	50.0 (89.8)
self-prompt	39.2 (35.4)	**36.5** (14.0)	**58.0** (84.8)

forced-mid) pin the call to a fixed trajectory point; *self-triggered* let the actor trigger itself, via a first-decision-token entropy threshold (*self-signal*) or by issuing `explore(scope, query)` directly (*self-prompt*). All wirings share the same worker budget (K=6, $\leq$ 3 calls/task); SWE-Bench is deferred to §5.6, where *content*, not control, dominates.

Self-triggered wirings are the most consistent across environments. Every wiring beats *none* on ALFWorld and SWE-Bench, but on WebShop *forced-pre* collapses by 24.3 because a scopeless brief over noisy products feeds the actor confident-but-wrong facts; both self-triggered wirings hold, and *self-prompt* adds +4.3. *Self-prompt* wins WebShop and SWE by keeping scoping on the high-capability tier, while *self-signal* (entropy threshold) wins ALFWorld and is cheaper to deploy: complementary readings of the prediction that the trigger should come from the actor's own belief. SWE-Bench compresses the spread because the structural *none* baseline already exposes the repo skeleton, so the remaining gain comes from *what* a call returns, dissected in §5.6.

5.2 Host B: OEL-trained actor

We instantiate Host B on Sokoban and Frozen Lake with Qwen3-4B-Instruct-2507 as the actor under consolidation. The *base* configuration runs the standard OEL loop. In the +*explore* configuration, trajectory collection is delegated: the same Qwen3-4B actor reads the task and emits a high-level plan (which subgoals to pursue, which subspace to focus on), and a frozen Qwen3-1.7B worker carries out the entire rollout in the environment under that guidance. The 4B actor never touches the environment itself; the 1.7B worker produces every action. All other components are held fixed.

Strong-guided weak-rollout collection does not degrade consolidation. Figure 3 reports held-out score on Sokoban across 40 OEL training steps. +*explore* matches *base* during round 1 (steps 12–20) and pulls ahead from step 24 onwards (peak gap +0.12 at step 32), with the collection budget paid by the cheap 1.7B worker rather than the 4B actor.

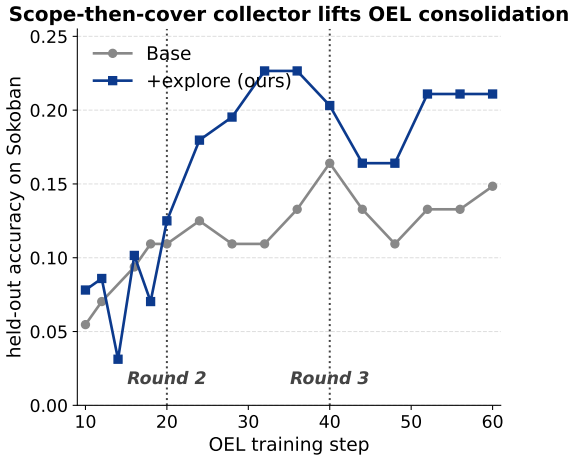


Figure 3: Held-out accuracy on Sokoban across OEL training steps.

5.3 Framework variables on real runs

With both hosts established, we now ask why this wiring works. The framework of §3 predicts the lift through scope quality (ρ, r) and a cover-stage assumption (C2). We anchor (ρ, r) in the actual runs here, sweep cover yield in §5.4, and probe the capability axis in §5.5; none of the three on its own validates C2.

For each task we measure $\rho_i = |\mathcal{S}_{\text{sub},i}|/|\mathcal{S}_i|$ on the actor's first nominated subspace and recall $r_i = \mathbf{1}[s^* \in \mathcal{S}_{\text{sub},i}]$ (for SWE: $r_i^{\text{prox}} = \mathbf{1}[G^* \cap \mathcal{S}_{\text{sub},i} \neq \emptyset]$, since the gold patch may touch multiple files).

The three environments span the framework's full range, and the r/ρ ordering in Table 2 (SWE $\gg$ ALFWorld $\gg$ WebShop) tracks the absolute scope-lift in Table 1. On **SWE-Bench** the actor's first scope is tiny but very high-recall and most in-scope cases convert to a passing patch, so both stages of scope-then-cover work as predicted. On **ALFWorld** scope is sharp and high-recall but only a fraction of in-scope cases convert at K=6; we hold K=6 across all environments for cross-env comparability, and on ALFWorld this falls below the saturation budget identified by the T-sweep in §5.4. On **WebShop** scope is loose and rarely contains the goal ASIN, so $r/\rho \ll 1$ and no scope-

Table 2: **Framework variables on the first explore call per task.** $\tilde{\rho}$: median [IQR]; $\bar{r}$: bootstrap mean [95% CI]. SWE uses r^{prox} (multi-file proxy).

Environment	$\tilde{\rho}$	$\bar{r}$	r/ρ	$P(\text{hit} \mid \text{in-scope})$
ALFWorld	$0.20\ [0.15, 0.21]$	$0.92\ [0.83, 0.98]$	4.6	0.42
WebShop	$0.60\ [0.40, 0.90]$	$0.04\ [0.02, 0.07]$	0.07	0.04
SWE-Bench†	$0.03\ [0.03, 0.03]$	$0.72\ [0.60, 0.84]$	24	0.75

lift is predicted, matching forced-wiring collapse and self-prompt's modest recovery in Table 1.

5.4 Cover scaling under fixed scope

We hold scope fixed at an actor-nominated $\mathcal{S}_{\text{sub}}$ and sweep the cover budget $T \in \{1, 2, 4, 6, 8\}$ on randomized ALFWorld pick-and-place, with the worker frozen. The point is to check whether the operating $K{=}6$ used in Table 1 sits in the regime where added touches still pay; the answer determines how much of the 0.42 in-scope cover rate of §5.3 is budget-limited rather than structural.

Figure 4a shows the curve is monotone and roughly linear through $T{=}6$ ($0.30 \rightarrow 0.41$), then bends to 0.45 at $T{=}8$. The operating $K{=}6$ sits just before the bend, so on ALFWorld $K{=}6$ is below the saturation budget, and raising K would close part of the in-scope cover gap reported in Table 2. We chose to hold $K{=}6$ across all environments rather than tune per-env so the cost-asymmetry comparison in Table 1 stays clean; this trades a few points of ALFWorld absolute score for cross-env comparability.

5.5 Capability sharpens scout draws: a dual cut on the wiring

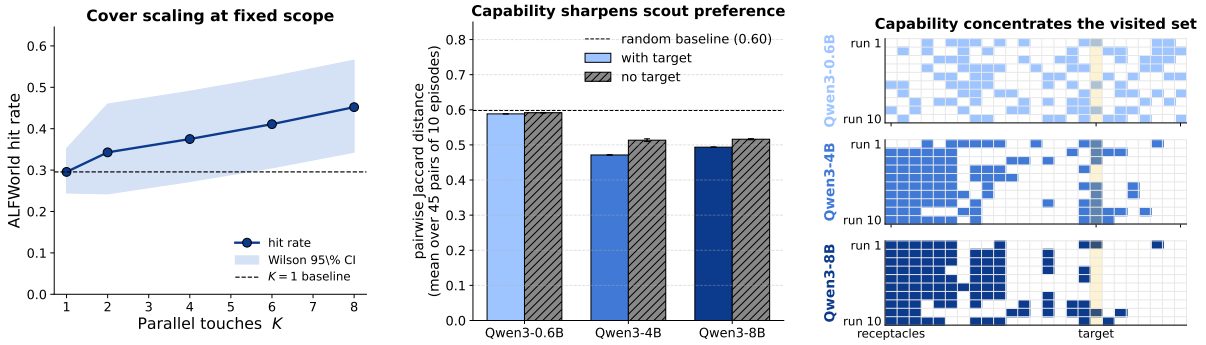
This section tests C1: whether scope sharpness improves with actor capability. We run frozen Qwen3-{0.6B, 4B, 8B} scouts on ALFWorld (10 runs each at $K{=}10$, under *with target*/*no target* prompts) and measure run-to-run diversity by mean pairwise Jaccard distance, where lower Jaccard means the scout commits to a sharper subset (full grid in App. C.8; a direct C2 test is in App. F.2).

Figure 4b shows a sharp Jaccard drop from 0.6B to 4B/8B: 0.6B sits at the random-draw expectation, larger scouts commit to a stable subset. Removing the target widens 4B/8B Jaccard but barely moves 0.6B, so the cue only sharpens the draw when capability can act on it, supporting the default of scoping on the high-capability tier and covering with a smaller worker.

Why strong models scope and weak models cover. The diversity result lets us cash out a question that has been implicit since §2: why does the strong actor fail on its own, and why is a small worker enough to fix it? Two pieces of evidence join here. First, on the cover side the strong model has nothing to spend capability on: once scope $\mathcal{S}_{\text{sub}}$ is set, the residual prior b' is approximately uniform (assumption C2), so per-touch yield is $1/|\mathcal{S}_{\text{sub}}|$ regardless of which model issues the touch (Prop. 1); empirically, $P(\text{hit} \mid \text{in-scope})$ in Table 2 is controlled by the cover budget K, not by worker scale; this is exactly the failure mode of Boundary 2 (§2), where cost-matched larger scouts on WebShop trail the cheap scout once $|\mathcal{S}| \geq 100$. Worse, capable models then *double down*: Fig. 4b shows the 4B/8B scout collapses to a near-deterministic subset (Jaccard well below random), so spending more of its budget at cover time mostly re-touches the same receptacles instead of widening coverage. Diversity is what cover needs, and a sharper prior actively destroys it.

Scope is the opposite story. The lift r/ρ in Eq. 3 is set by the actor's ability to nominate a small high-recall subspace, and Fig. 4b–4c show that this is exactly where capability is irreplaceable: 0.6B's draws are indistinguishable from random, while 8B locks onto the same cabinet column across all 10 runs even before the target is revealed. Putting the two halves together gives the wiring of §4 not as a preference but as a direct consequence: the property that hurts the strong model at cover (low draw diversity) is the same property that makes it succeed at scope, and the property that helps the weak model at cover (uniform sampling inside $\mathcal{S}_{\text{sub}}$) is the same property that disqualifies it from scope.

Direct test of C2: strong cover changes little. To separate residual within-scope structure from weak-worker capability, we rerun the same 100 ALFWorld tasks at $K{=}6$ with Qwen3-8B cover workers in place of Qwen3-0.6B, keeping scope, prompts, and task list fixed.



(a) Hit rate vs. parallel touches $T \in \{1, 2, 4, 6, 8\}$ at fixed actor-nominated scope on randomized ALFWorld pick-and-place (T=1 from a separate no-explore baseline); shaded band is Wilson 95%.

(b) Pairwise Jaccard across 10 ALFWorld runs (K=10 each), mean over $C(10, 2)$=45 pairs and 100 tasks; error bars are std over 3 outer seeds. The dashed line is the random-draw expectation $1-K/(2N-K)$.

(c) One representative task (find a saltshaker; ground-truth shelf 2, highlighted): rows are 10 explorations, columns are receptacles. 0.6B touches every cell; 8B reuses the same cabinet block in all 10 runs.

Figure 4: **Cover scaling and capability sharpening.** (a) Cover yield vs. touches T at fixed scope, K=6 marked. (b) Pairwise Jaccard across 100 rooms: 0.6B sits at the random baseline; 4B/8B with target commit to a sharper subset and the gap widens when the target is revealed. (c) One task: 0.6B touches every receptacle, 8B reuses the same cabinet column across all 10 runs.

Table 3: **Strong-cover ablation** on ALFWorld (n=100, K=6). Same scope; only the cover worker changes.

Cover worker	P(hit)	Δ	tokens/call
6× Qwen3-0.6B (default)	0.35	—	∼ 1.6k
6× Qwen3-8B (strong)	0.41	+0.07	∼ 1.4k

The lift in Table 3 is small relative to the parameter jump, so the remaining miss rate is not mainly a weak-worker bottleneck; it is dominated by residual within-scope structure and harness losses. This supports the C2 reading used in the framework analysis above.

5.6 What goes inside the explore call: scoping the message

We hold the wiring fixed and vary *what* the worker sends back, on SWE-Bench, where the actor's failures are content-level (wrong file, patch misaligned with the test's intent).

We hold dispatch (single worker, one call per task) constant and vary the return content across a structural baseline (Tier 0) plus six worker-call variants split into two tiers: Tier 1 returns a worker summary that includes a judgment (a fix direction or a skip-view hint); Tier 2 returns only retrieved artifacts (a factual brief, a source slice, a test slice, or all three bundled). Full variant definitions are in App. C.5. As shown in Table 4, Tier 2 sits at 52–58% pass; Tier 1 sits at 48%, tying the Tier 0 baseline. The 4–10-point across-tier gap matches what §3 would predict if the same logic that puts scoping on the high-capability tier extended to the return channel: judgments are decisions the actor cannot cheaply re-derive, while facts cost the worker little to forward.

Actor scale alone does not recover the lift. Rerunning the same ALFWorld wirings with a larger strong actor (Table 5) preserves the same pattern: the baseline is essentially unchanged, and the gains from *+explore* remain larger than the gains from simply scaling the actor.

Scaling the actor 4× does not, by itself, recover the same lift the worker provides; the bottleneck is information access, not capability.

6 Related Work

Exploration in LLM agents. Existing work studies exploration as a single-model loop, in bandit/MDP settings (Krishnamurthy et al., 2024; Arumugam and Griffiths, 2025; Nie et al., 2024; Jiang et al., 2025; Tajwar et al., 2025) and in agent scaffolds with in-rollout reasoning or tree search (Yao et al., 2022b; Shinn et al., 2023; Wang et al., 2023a; Yao et al., 2023; Zhou et al., 2023; Hao et al., 2023). We instead ask where capability stops paying inside the loop and route prior-bound and coverage-bound subproblems across capability tiers.

Test-time compute, training, and experiential learning. A complementary line scales inference compute on a frozen model (Wang et al., 2022; Brown et al., 2024; Snell et al., 2024) or converts

Table 4: **SWE-Bench content variants.** All rows start from the same structural scan baseline (Tier 0); each + row adds one worker call returning the indicated payload. *Pass* is per-instance pass rate; Δ columns are paired differences against the baseline on the matched 50-task set. Worker-on rows differ in token cost by $\leq 3\%$, so within-family comparisons are essentially token-neutral.

Configuration	Pass (%)	Avg. tok / task	Avg. steps	ΔPass	Δtok
Tier 0. Structural scan	48.0	2.90M	87.0	—	—
Tier 1: + worker summary including a judgment					
+ Brief: narrative + fix advice	48.0	2.91M	87.3	+0.0	+0.7%
+ Read source + skip-view hint	48.0	2.84M	85.8	+0.0	−1.8%
Tier 2: + facts only, no judgment					
+ Brief: factual (caller graph)	52.0	2.89M	86.5	+4.0	−0.2%
+ Read source	52.0	2.85M	86.1	+4.0	−1.4%
+ Read test	**58.0**	**2.81M**	**84.8**	**+10.0**	**−2.9%**
+ Read all (source + test + callers)	54.0	2.91M	86.5	+6.0	+0.3%

Table 5: **ALFWorld actor-scale ablation.** Same harness, same wirings, same T=0.7, same 274-task set. Strong actor scaled from Qwen3-8B-Instruct to Qwen3-30B-A3B-Instruct.

Configuration	8B win %	30B win %	8B Δ	30B Δ
none (no worker)	29.6	29.2	—	—
+ prefetch	42.7	38.0	+13.1	+8.8
+ midway	36.5	35.8	+6.9	+6.6
+ explore (parallel $K \leq 6$)	42.7	40.1	+13.1	+10.9

deployment trajectories into improved policies / in-context experience (Zhao et al., 2023; Shinn et al., 2023; Zuo et al., 2025; Ye et al., 2026; Akyürek et al., 2024). These improve how trajectories are sampled or consolidated by the actor itself; we instead target collection-time exploration, scaling a *different*, weaker model in parallel inside a strong-actor-chosen scope, and drop the same operator into OEL's collection step (§5.1, §5.2).

Strong–weak collaboration, sub-agents, and swarms. Cost-based cascades and routers (Chen et al., 2023; Ong et al., 2024) dispatch each query to the cheapest sufficient model; planner–executor frameworks (Shen et al., 2023; Wu et al., 2023; Wang et al., 2023b) factor a task across roles; recent agentic systems run parallel sub-agents or swarms under a lead model (Kimi Team, 2025; Anthropic, 2025; OpenAI, 2024). These keep each spawned agent on the full task or a hand-defined role; we ground dispatch in state-space scoping, with workers running strictly inside a strong-actor-chosen $\mathcal{S}_{\text{sub}}$.

7 Conclusion

LLM-agent exploration has two distinct bottlenecks, capability-bound scoping and budget-bound covering, and scaling a single explorer addresses neither. A minimal hit-rate equation (§3) reads both off the same form and predicts an r/ρ lift whenever recall exceeds subspace fraction. Realized as a single *explore action* with strong-actor scopes and frozen weak-worker cover, it improves a no-explore baseline across ALFWorld, WebShop, and SWE-Bench-mini-50, with added cost absorbed by the cheap tier. The first lever in an agent loop is not a stronger explorer but a sharper scope.

8 Discussion

Classical HRL options (Sutton et al., 1999) decompose actions; we decompose by *subproblem kind* and route across capability tiers. Within OEL the actor's dispatched scopes form a labeled curriculum that could co-train the worker, with implications for the residual-uniformity assumption Prop. 1 relies on. The explore action is compatible with both frozen test-time actors and online weight-updating hosts, and the same syntax could extend to multi-tier worker pools or retrieval-augmented scoping.

Limitations

Two assumptions bound the result. The actor's prior must resolve at the subspace level: in genuinely novel domains it degrades to coverage-bound, with WebShop near this boundary at $r/\rho \approx 0.07$ (Table 2). Worker feedback must be ground-truth: judgment-laden feedback erodes the "facts, not judgments" discipline of §5.6. Eq. (4) is also a lower bound, since a strong-only run would pay extra for growing context. We freeze the worker here to isolate the lift from the wiring.

References

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. 2024. The surprising effectiveness of test-time training for few-shot learning. *arXiv preprint arXiv:2411.07279*.

Anthropic. 2025. Claude code sub-agents. https://docs.claude.com/en/docs/claude-code/sub-agents.

Dilip Arumugam and Thomas L. Griffiths. 2025. Toward efficient exploration by large language model agents. *arXiv preprint arXiv:2504.20997*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

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

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. 2025. Meta-rl induces exploration in language agents. *arXiv preprint arXiv:2512.16848*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Kimi Team. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. 2024. Can large language models explore in-context? In *Advances in Neural Information Processing Systems 37*. Neural Information Processing Systems Foundation, Inc. (NeurIPS).

Allen Nie, Yi Su, Bo Chang, Jonathan N. Lee, Ed H. Chi, Quoc V. Le, and Minmin Chen. 2024. Evolve: Evaluating and optimizing llms for in-context exploration. *arXiv preprint arXiv:2410.06238*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

OpenAI. 2024. Swarm: Lightweight multi-agent orchestration. https://github.com/openai/swarm.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *arXiv preprint arXiv:2303.17580*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

Fahim Tajwar, Yiding Jiang, Abitha Thankaraj, Sumaita Sadia Rahman, J Zico Kolter, Jeff Schneider, and Ruslan Salakhutdinov. 2025. Training a generally curious agent. *arXiv preprint arXiv:2502.17543*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023b. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate

problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. 2026. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. 2025. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*.

A Per-Mode Failure Reduction (Visual Breakdown)

Figure 5 visualizes the F1 / F2 reduction numbers reported in the introduction. For every failed trajectory of the no-explore baseline we tag it as F1 (scope-bound: agent never locates the right subspace—no `take`/`put` in ALFWorld, score $= 0$ in WebShop, empty patch on SWE-Bench) or F2 (cover-bound: right subspace, but execution or budget fails). Each panel pairs the baseline bar (light) with the +explore (ondemand) bar (dark) and an arrow labelled with the absolute change (Δ pp; blue $=$ down, red $=$ up). The only ups are within-noise on already near-zero F1 (ALFWorld F1 base $= 2.2\%$, WebShop F2 base $= 34.5\%$).

B Proofs for the framework of §3

We restate and prove the propositions referenced from §3. Notation as in §3.1: candidate space $\mathcal{S}$, target $s^* \sim P^*$, actor prior π, per-task touch budget T, scope $\sigma : \pi \mapsto \mathcal{S}_{\text{sub}} \subseteq \mathcal{S}$, $\rho := |\mathcal{S}_{\text{sub}}|/|\mathcal{S}|$, $r := P^*(s^* \in \mathcal{S}_{\text{sub}})$, residual prior $\pi'(s) := P^*(s \mid s^* \in \mathcal{S}_{\text{sub}})$.

Proposition 2 (Scope–cover split)**.** *Let π_w be any worker prior used to greedily select T touches inside $\mathcal{S}_{\text{sub}}$. Then*

$$P_{\sigma,\pi_w}(\text{hit}; T) = r \cdot \sum_{s \in \text{top}_T(\pi_w|\mathcal{S}_{\text{sub}})} \pi'(s) \le r \cdot \min\big(1,\ T \cdot \max_{s \in \mathcal{S}_{\text{sub}}} \pi'(s)\big).$$

Under the working assumption that π' is uniform on $\mathcal{S}_{\text{sub}}$, the bound becomes

$$P_{\sigma,\pi_w}(\text{hit}; T) \le r \cdot \min\big(1,\ T/(\rho|\mathcal{S}|)\big),$$

is achieved with equality, and is independent of π_w.

Proof. Conditioning P^* on $\{s^* \in \mathcal{S}_{\text{sub}}\}$ gives $P^*(s^* = s) = r \cdot \pi'(s)$ for $s \in \mathcal{S}_{\text{sub}}$. Greedy cover under π_w touches the top-T states of π_w restricted to $\mathcal{S}_{\text{sub}}$; the hit probability is r times the sum of π' on those touches, giving the equality. Bounding the sum of any T values of a probability distribution by T times its maximum entry gives the first inequality. Under uniformity, $\max \pi' = 1/(\rho|\mathcal{S}|)$ and any T-subset attains the same sum, so the bound is tight and π_w-independent. □

Remark. Without uniformity the bound depends on π_w: a sharper π_w that concentrates on π''s heavy region yields a higher hit. This is the precise sense in which the residual-uniformity assumption (C2 in §3.3) is a working assumption: residual non-uniformity reintroduces a (weakened) capability dependence on the cover step. §5.5 (Mechanism check) reports the empirical gap between weak- and strong-worker yield inside actor-chosen $\mathcal{S}_{\text{sub}}$ as a direct test.

Proposition 3 (Cost asymmetry)**.** *With c_a, c_w the per-token costs of actor and worker, τ_a the actor's task-side token spend, T the worker touch count, and ℓ the per-touch step length in tokens, the scope-then-cover and strong-only token costs are*

$$T_{\text{scope+cover}} = \tau_a \cdot c_a + T \cdot \ell \cdot c_w,$$
$$T_{\text{strong-only}} = \tau_a \cdot c_a + T \cdot \ell \cdot c_a.$$

Their difference is

$$T_{\text{strong-only}} - T_{\text{scope+cover}} = T \cdot \ell \cdot (c_a - c_w),$$

and whenever $T \cdot \ell \cdot c_a \gg \tau_a \cdot c_a$ the relative saving is $1 - c_w/c_a$, independent of ρ and $|\mathcal{S}|$.

Proof. Direct subtraction; no assumption beyond linearity of token cost in step count. □

Remark (uniqueness). Eq. (2) and Prop. 3 together impose two constraints on any wiring: σ must be produced by a sharp π, and cover must be

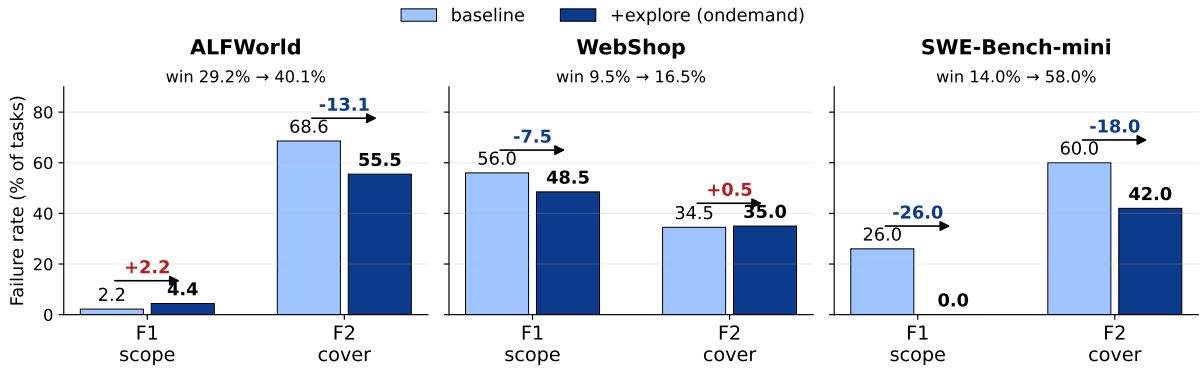


Figure 5: **Adding an explore action shrinks both failure modes across three environments.** Baseline (light blue) vs. +explore ondemand (dark blue); arrow labels are Δ pp.

charged at $c_w \ll c_a$. With one strong actor and one frozen weak worker, the only assignment satisfying both is strong-actor scope + weak-worker cover; assigning scope to the weak worker violates the first constraint, and assigning cover to the strong actor violates the second.

C Implementation Details

C.1 Artifact Licenses and Terms

We use existing artifacts only for research evaluation and do not redistribute the underlying benchmark datasets, model weights, or environment code as part of this paper. ALFWorld is released under the MIT license, with its bundled dependencies carrying their own licenses (including TextWorld under MIT and Fast Downward under GPLv3). WebShop is distributed under Princeton's research-use terms. SWE-Bench and its evaluation harness are released under the MIT license. The Qwen3 model family used for the actor and worker models is released under Apache 2.0. For OEL, we use the public research codebase and text-game harness according to the terms distributed with that artifact. All uses in this work are limited to benchmark evaluation, agent rollouts, and controlled ablations.

C.2 Strong Actor (test-time setting)

The test-time experiments (Section 5.1) use a frozen Qwen3-Coder-Next-80B-A3B (AWQ-4bit) actor served via vLLM at TP=2 on B200 GPUs. No parameters are updated. Decoding: $T = 0.7$, top-$p = 0.8$, top-$k = 20$, min-$p = 0.0$, repetition penalty 1.05, max 4096 tokens per call, max 100 steps per task, max context 65,536 tokens.

C.3 Strong Actor (online-learning setting)

The online-learning experiments (Section 5.2) use Qwen3-4B-Instruct-2507 as the trainable strong actor, updated by on-policy context distillation following the OEL protocol (Ye et al., 2026). We use the OEL codebase's default Sokoban configuration with the short-version schedule (20 consolidation steps per round, 64 game samples per step, save_freq=5). Each round runs the three OEL stages in sequence: experience extraction (10 ckpt seeds, 100 validation samples per seed), trajectory collection (20 deploy steps), and consolidation (20 training steps). Hardware: 8× NVIDIA B200.

C.4 Weak Explorer Configuration

The weak explorer is held *frozen* across all experiments. For SWE-Bench (Section 5.1) the explorer is Qwen3-8B-Instruct served via vLLM at TP=1; the operator returns a structural scan, a brief, or a pre-fetched bundle depending on the configuration. For Sokoban (Section 5.2) the explorer is Qwen3-1.7B served via vLLM at TP=1, chosen to be capability-sufficient for short-horizon planning while still cheaper than the trainable Qwen3-4B actor. The explorer's parameters are never updated by anything; it is a cheap drop-in worker.

C.5 Operator Returns (test-time variants)

Section 5.6 sweeps a structural baseline plus six worker-call variants, organized in two tiers by whether the worker volunteers a judgment.

Tier 0 (baseline). *(0) Structural scan*: repository layout and test-runner detection via shell commands, no LLM call. Every other row starts from this and adds one explore call.

Tier 1 (worker summary with a judgment). *(i) Brief: narrative + fix advice*: a $\leq$ 1,500-

character free-form explainer including a suggested fix direction. *(ii) Read source + skip-view hint*: a $\sim$ 200-line source slice plus a short hint suggesting which layers to skip.

Tier 2 (facts only, no judgment). *(iii) Brief: factual*: a $\leq$ 600-character description of what is, with recommendation phrases banned and a caller graph appended. *(iv) Read source*: a $\sim$ 200-line excerpt of the suspect source file. *(v) Read test*: the relevant test function source. *(vi) Read all*: source + test + caller list bundled.

All Tier 1 / Tier 2 returns are produced by the same Qwen3-8B explorer with $\leq$ 10 tool steps and a hard guard that rejects `done()` calls before step 4 or before the explorer has invoked at least one `grep` and one `read`.

C.6 Environment Harnesses

Sokoban and FrozenLake (Section 5.2) use the OEL codebase's textgame harness: 5 turns per episode, max 1,024 tokens per response, held-out evaluation of 128 maps averaged over 10 seeds. SWE-Bench-Verified-mini-50 (Section 5.1) uses the standard Docker-based evaluation harness, with each of the 10 parallel agents assigned a dedicated container; we evaluate on the 50-instance `verified-mini` subset balanced across Django and Sphinx.

C.7 Per-Rollout Pseudocode

Algorithm 1 Scope-then-Cover Explore Action (referenced from §4).

Input: actor π_a, frozen worker π_w, environment $\mathcal{E}$, worker budget K, rounds R
for round $r = 1, \ldots, R$ **do**
  for each task **do**
    repeat
      $a \leftarrow \pi_a(\text{context})$
      if a = `explore(scope, query)` **then**
        // Scope: actor nominates subspace
        $\mathcal{S}_{\text{sub}} \leftarrow$ `scope`, $q \leftarrow$ `query`
        // Cover: K weak workers in parallel
        $o \leftarrow \texttt{aggregate}\big(\{\pi_w(\mathcal{S}_{\text{sub}}, q)\}_{k=1}^{K}\big)$
        update context with o
      else
        execute a in $\mathcal{E}$
      end if
    until task done
  end for
  // Host B: context distillation
  if Host B **then**
    $\pi_a \leftarrow \arg\min_{\pi} \mathbb{E}[\mathrm{KL}(\pi(\cdot) \,\|\, \pi_a(\cdot \mid \text{context}))]$
  end if
end for

C.8 Diversity Experiment Setup (§5.5)

The capability-vs-diversity sweep of §5.5 runs 100 ALFWorld pick-and-place rooms with frozen Qwen3 scouts at three scales (0.6B, 4B, 8B). Each scout repeats every room 10 times at K=10 receptacle inspections per run, with no within-run revisits but independent runs. Two prompt conditions are crossed: *with target* reveals the goal object; *no target* replaces it with a generic "learn the layout" instruction. Sampling, notepad format, and candidate enumeration are held fixed. The full $3 \times 100 \times 10 \times 2$ grid is repeated under 3 outer seeds. We report mean pairwise Jaccard distance $1 - |V_i \cap V_j|/|V_i \cup V_j|$ over the $C(10, 2)$=45 trajectory pairs per room, averaged across rooms. Coverage is uninformative here (the no-revisit constraint pins unique-receptacle count at K/N) and entropy is squeezed into a narrow band by the same constraint, so we use Jaccard, whose feasible range $[0,\ 1{-}K/(2N{-}K)]$ spans the regime of interest.

D WebShop Controlled-Experiment Details

D.1 Candidate Sampling

For each user instruction, we retrieve the top-1000 candidates by BM25 score, force-include the ground-truth target item, sample $|\mathcal{S}|$ items uniformly without replacement, and shuffle the order. Forcing target inclusion guarantees that the target is present in every cell, isolating coverage as the exploration challenge.

D.2 Wall-Clock Cost Matching

On a single H100 with batch size 1 and sampling temperature 0.6, per-inspection wall-clock scales roughly with model size across the 0.6B / 4B / 8B columns. The inspection budgets $K \in \{30, 12, 8\}$ for 0.6B / 4B / 8B are chosen so that $K \times$ per-inspection cost is matched to within 10% across columns.

D.3 Notepad Format and Actor Decision

Each notepad entry records the inspected item's title, price, attributes, and option list in a fixed JSON-like format. After the scout exhausts K inspections, the fixed Qwen3-8B actor reads the full notepad and emits a single index as its final answer; ties are broken by lowest index.

D.4 Prompt Templates

ALFWorld actor (system prompt, no-explore baseline).

```
You are an expert AI agent solving
ALFWorld household tasks. Each turn
output ONE action.

OUTPUT FORMAT: Output ONLY a single line:
'Action: <one action>'. The action MUST
be chosen from the ADMISSIBLE COMMANDS
list shown each turn. NO planning text,
NO multi-step plans.

ALFWORLD RULES (violating these makes
the env return 'Nothing happens'):
1. To take/put/examine an object: you
MUST first 'go to' its receptacle.
2. To cool an object: take it, then
'go to fridge X', then 'cool <obj> with
fridge X'.
3. To heat an object: take it, then 'go
to microwave X', then 'heat <obj> with
microwave X'.
4. To clean an object: take it, then
'go to sinkbasin X', then 'clean <obj>
with sinkbasin X'.
5. After cool/heat/clean, 'go to
<target_recep>' then 'put <obj> in/on
<recep>'.
6. Closed containers (cabinet, drawer,
fridge, microwave) must be 'open'ed
first.
7. 'put X in Y' uses 'in' for closed
containers, 'on' for open surfaces.

Be efficient — 50-turn budget.
```

ALFWorld actor (+explore addendum, ondemand).

```
You can dispatch K small-model scouts to
investigate the environment IN PARALLEL.
Each scout independently goes to one
location, looks/opens, and reports back.

How to call (output this exact
format on a SINGLE line as your Action):
Action: ask_scouts(targets=["cabinet
1","cabinet 2","drawer 3"], goal="find
the butterknife")

Constraints: 1-6 receptacle names
exactly as they appear in admissible
commands; goal is a SHORT imperative.

The harness spawns one small-model
scout per target (parallel), each goes
to its target, opens (if closed), looks,
and reports e.g. "cabinet 1 → empty;
cabinet 2 → contains [bowl 1, plate
2]; drawer 3 → contains [butterknife
1] ✓". You may call ask_scouts AT MOST
3 TIMES per task.
```

ALFWorld scout (per-target system prompt).

```
You are a Scout. You have ONE target:
'{target}'. Your goal: {goal}.
Plan: go to target, open it (if closed),
look. Output ONE valid action per turn
(no other text).
```

WebShop actor (system prompt, no-explore baseline).

```
You are a shopping agent on WebShop.
Find a product matching the user's
instruction and click "Buy Now" on the
product detail page. Each turn output
ONE action.
Format: Thought: <one short sentence>\n
Action: <tool>(<args>)
Tools: search(query) — full-text
product search (HOME or RESULTS page
only); click(target) — click product
asin (B0XXXXXXX), an option label
(fuchsia, medium, ...), or a button
(Buy Now, Description, < Prev).

Page phases: HOME (first turn,
MUST search); RESULTS (click ASIN to
enter detail; cannot Buy Now here);
PRODUCT (detail page; click options
then click("Buy Now")).

Reward = number of required attributes
matched. 20-turn budget.
```

WebShop actor (+explore addendum).

```
After a search() returns 10+ candidates,
you can dispatch parallel scouts to
inspect K products:
Action: ask_scouts(
targets=["B07XYZ", "B08ABC", "B09DEF"],
goal="check size medium and color
fuchsia")
Each scout clicks a product page, reads
attributes/options/price, and reports
back. Constraints: 1-6 ASINs from
the search results; goal is a short
imperative; at most 3 ask_scouts calls
per task.
```

WebShop scout (per-target system prompt).

```
You are a Scout. You have ONE target
product asin: '{target}'. Goal: {goal}.
Plan: click the asin, read the page,
optionally click an option to verify.
Output ONE action per turn.
```

E Additional Results

E.1 Per-Environment Breakdown

Table 6 reports per-environment score (matched to the metric used in Table 1), average task length,

and prompt+completion token cost for the test-time host (Qwen3-8B actor, Qwen3-0.6B cover workers; ALFWorld n=274 valid-full, WebShop n=200 AgentGym slice).

The cover-token column is a small fraction of the total in every environment ($< 1\%$ on ALFWorld, $\approx 2\%$ on WebShop, $\approx 3\%$ on SWE), confirming that the inference cost added by parallel weak workers is dominated by the actor's own token usage. ALFWorld and SWE convert this cost into substantial gains (+13.0 and +10.0 pp); WebShop sees only a +4.3 partial-credit lift, consistent with the very low scope-recall reported in Table 2 ($\bar{r}$=0.04 on WebShop vs. 0.92 on ALFWorld and 0.72 on SWE).

E.2 Qualitative Examples

We pick one task per environment where the no-explore baseline failed and the +`explore` host succeeded under the same actor (Qwen3-8B), same prompts, same task instance, same seed. Trajectories are abridged; observation strings truncated to the relevant span.

ALFWorld — pick-and-place a saltshaker in a drawer. Task: "put a saltshaker in drawer." Baseline burns the entire 50-turn budget linearly opening drawers and cabinets one at a time and never finds a saltshaker; +`explore` dispatches one parallel call across 9 candidate cabinets, gets back a list of contents, and finishes in 6 turns.

The baseline is doing the same thing as the cover stage of +`explore`, but serially and inside the strong actor's expensive context. The actor's scope step (one 9-target call) runs the same exploration in parallel under cheap workers and returns a fact-only digest; the actor then commits in 5 remaining turns.

WebShop — multi-attribute product match. Task: "Find me quick drying, moisture wicking women's activewear with long sleeve, color c-navy blue, size large, under $50." Baseline searches once, then clicks 10 candidate ASINs from the result page sequentially and never selects options on any product page; +`explore` does the same search, dispatches one batched scout over the result list, then commits.

Baseline's failure mode is *premature commitment without verification*: it never selects the colour/size options on any product page, so even when it lands on the right ASIN it cannot finish the buy sequence. +`explore` uses scouts as a cheap cover stage that returns the verifiable colour/size facts, then commits to the one ASIN that survives.

SWE-Bench — pre-fetched test files re-shape the search. Task: `django__django-11790` (`AuthenticationForm` username field max length; gold patch touches `django/contrib/auth/forms.py` and an associated regression test). Baseline (fastscan; no test pre-fetch) fails (reward 0) while +`explore` (v5_tc; pre-fetched test file in the cover payload) passes. SWE trajectories are too long to render verbatim (83/87 steps); we summarise the first six file inspections to make the scoping difference visible.

For SWE-Bench the cover stage delivers *file content*, not receptacle existence as in ALFWorld or attribute facts as in WebShop. The qualitative effect is the same: cheap workers replace the actor's expensive scan inside the candidate space and the actor enters its decision loop already inside the relevant slice. On this instance the action budget is similar between the two configurations but each early step now operates on the right files, so the patch lands on the regression test the first time it is exercised.

F Extra Ablations

F.1 Learned-Dispatch Proof of Concept

The hand-crafted +*explore* tool encodes a heuristic for *when* and *what* to call. A natural extension is to learn the dispatch policy directly from task reward, training the actor's dispatch decisions against task pass as the only reward while holding the worker frozen. We leave a full empirical study of this learned-dispatch variant, and its scaling across environments, to future work.

F.2 Mechanism Check: Dispatched $|\mathcal{S}_{\text{sub}}|$ vs. $\mathcal{S}^*$

The dispatched-size histogram in Fig. 4a(c) provides the per-round picture: across n=100 tasks the actor saturates the budget cap on most calls (mode at K=6), and the empirical $\mathcal{S}^*$ from Probe 1 of §2 ($\sim$30 receptacles total in a typical kitchen) sits well above any individual dispatch. Trajectories therefore operate in the small-space regime that the two-bound view of §3 predicts is where scope-then-cover pays.

Scope quality (Prop. 1) and worker scaling. The three panels of Fig. 4a (§5.4) directly mea-

Table 6: **Per-environment breakdown of the test-time host.** Score uses the same metric as Table 1 (per-instance success rate for ALFWorld and SWE-Bench, attribute-match $\times 100$ for WebShop). Token columns are mean prompt+completion tokens per task; the actor and the $K \leq 6$ cover workers are accounted separately. Steps are environment turns.

Environment	Method	Score	Steps	Actor tok.	Cover tok.	Total
ALFWorld (n=274)	Baseline (no `explore`)	29.6	38.9	73.7k	0	73.7k
	+`explore` (best wiring)	42.6	34.3	87.5k	0.7k	88.2k
WebShop (n=200)	Baseline	32.2	20.0	125.7k	0	125.7k
	+`explore` (best wiring)	36.5	19.5	119.5k	2.8k	122.3k
SWE-Bench-V-mini-50	Baseline (+ structural scan)	48.0	—	2.90M	0	2.90M
	+`explore` (v5_tc)	58.0	—	2.81M	$\sim$ 80k	$\sim$ 2.89M

ALFWorld qualitative trajectory. The no-explore baseline searches serially; +`explore` dispatches the same cover work in parallel and commits to a successful path.

```
BASELINE (turns=50, won=False)
t1: go to drawer 1 → drawer 1 closed
t2: open drawer 1 → [creditcard 1]
t3: go to drawer 2
t4: open drawer 2 → [dishsponge, spatula]
t5: go to drawer 3
t6: open drawer 3 → [fork, peppershaker]
t7: take peppershaker 2 from drawer 3
  ← wrong object
t8: go to cabinet 3 → [bowl, glassbottle, vase]
... 42 more sequential opens;
never finds saltshaker
```

```
+EXPLORE (turns=6, won=True, scout_calls=1)
t1: ask_scouts(targets=cabinets 1-6,
  drawers 1-3; goal=find saltshaker)
  SCOUT REPORT (parallel, ~3.6 turns):
  • cabinet 6 → [glassbottle 3,
    saltshaker 2, vase 2]
  • cabinet 1 → [plate 1, saltshaker 1]
  • cabinet 2 → [glassbottle 1, plate 2]
  • cabinet 3 → [bowl 1, glassbottle 2, vase 1]
  • cabinet 4 → [peppershaker 1]
  • cabinet 5 → [dishsponge 2]
t2: go to cabinet 1 → [plate 1, saltshaker 1]
t3: take saltshaker 1 from cabinet 1
t4-t6: drawer 1; open; move saltshaker ✓
```

sure the two quantities entering the Prop. 1 lift r/ρ on ALFWorld: $\rho = \mathbb{E}[|\mathcal{S}_{\text{sub}}|/|\mathcal{A}_t|]$ from the actor's first explore call, and the hit-rate sweep over $K \in \{1, 2, 4, 6, 8\}$ at fixed scope. The cover-scaling panel rules in linear-then-elbow behaviour at the operating budget; the scope-quality panel exposes the gap between coarse predicted lift and measured lift, which the cover-stage success rate $P(\text{hit} \mid \text{in-scope})$=0.42 in Table 2 attributes to residual within-scope structure (a partial violation of C2). The dispatched-size histogram confirms the K=6 cap is binding on most calls. Two improvements are deferred to a later draft: replacing the permissive r proxy (any non-empty scout report) with a target-grounded indicator from the won-trajectory receptacle, and extending the sweep to K=12 to locate the saturation point.

Direct test of C2: strong-cover ablation. The $P(\text{hit} \mid \text{in-scope})$=0.42 gap can come from either residual within-scope non-uniformity (a partial C2 violation) or from weak-worker capability. To separate the two, we re-run the same 100

Table 7: **Strong-cover ablation** on ALFWorld (n=100, K=6). Same scope; only the cover worker changes.

Cover worker	$P(\text{hit})$	Δ	tokens/call
6× Qwen3-0.6B (default)	0.35	—	$\sim$ 1.6k
6× Qwen3-8B (strong)	0.41	+0.07	$\sim$ 1.4k

ALFWorld tasks at K=6 but replace the Qwen3-0.6B cover workers with Qwen3-8B workers (same model class as the actor). Same scope, same prompts, same task list; only the cover-stage capability changes.

Swapping the cover worker from 6×Qwen3-0.6B to 6×Qwen3-8B (a $\sim$13× parameter increase) lifts $P(\text{hit} \mid \text{in-scope})$ by only +0.07 (0.35 → 0.41, n=100). The residual $1 - P \approx 0.6$ failure is therefore not mainly a worker-capability bottleneck; it comes from residual within-scope structure (partial C2 violation) and harness losses. This rules out the strongest alternative reading, that the 0.42 in Table 2 reflects weak workers failing inside good scopes. The cost-asymmetry argument is

WebShop qualitative trajectory. Scouts verify colour and size attributes before the actor commits to one product page.

BASELINE (turns=20, score=0.0)
t1: search("quick drying ... c-navy blue large under 50")
t2: click(B08JD14GJF)
t3: click(B07WMMYB6G)
t4: click(B09QCP4579)
...
t2–t11: 10 sequential ASIN clicks;
never reaches option selection
t12: reset
... *8 more wasted turns*

+EXPLORE (turns=8, score=1.0, scout_calls=1)
t1: search("quick drying ... c-navy blue large under $50")
t2: ask_scouts(targets=10 ASINs,
goal=check navy blue + large)
SCOUT REPORT:
per-product attribute summary;
one ASIN matches options
t3–t4: re-search to refresh page state
t5: click("B08JD14GJF")
t6: click("c-navy blue")
t7: click("large")
t8: click("Buy Now") ✓

SWE-Bench qualitative trajectory. Pre-fetched test content moves the actor's early inspections onto the source and regression-test files.

BASELINE first 6 actions
(django-11790, reward=0)
step 0: view /testbed
(top-level scan)
step 1: view /testbed/django/contrib
(intermediate scan)
step 2: view /testbed/django/contrib/auth/forms.py
(✓ gold src)
step 3: bash probe
step 4: view /testbed/django/forms/fields.py
(detour)
step 5: view /testbed/django/contrib/auth/forms.py
(back to gold src)
... 77 more steps; never opens
the regression test; final patch fails.

+EXPLORE first 6 actions
(v5_tc, reward=1)
step 0: view /testbed
step 1: view /testbed/django/contrib/auth/
forms.py (gold src, no detour)
step 2: view /testbed/tests/auth_tests/
test_forms.py (gold test, pre-fetched)
step 3: bash probe
step 4: view test_forms.py
(re-read as oracle)
step 5: view test_forms.py
(different range)
... 81 more steps; patch passes
the regression test.

preserved: strong cover gives a small per-touch lift while using roughly the same prompt+completion tokens at $K{=}6$, so dispatching the strong actor to cover would multiply per-call cost without proportional yield. Caveat: r remains the permissive "any non-empty scout report" proxy, identical across both rows, so only the Δ is tested.