CAN WEAK MODELS EXPLORE? SCOPE THEN COVER FOR LLM AGENT ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Adapting a language-model agent to a new environment requires exploration, and the prevailing assumption is that stronger models explore better. We examine two regimes where model capability alone is insufficient for discovery: when the available information does not help locate the target, and when the discovery advantage over uniform inspection diminishes as the candidate space grows. These observations motivate combining task-informed guidance with broad inspection: a strong actor selects a candidate scope, and weak scouts inspect candidates within it in parallel. We instantiate this scope-then-cover decomposition as a single explore action for frozen test-time actors. On ALFWorld, WebShop, and SWE-Bench-Verified, adding the explore action improves task performance over the actor-only baseline by assigning additional inspections to weak scouts.

1 INTRODUCTION

Large language model (LLM) agents are increasingly used for coding, digital workflows, and embodied interaction. The information needed for these tasks is often not fully specified in the initial input and must instead be acquired as execution proceeds. This information supports immediate action decisions and provides experience for subsequent learning (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2023a; Ye et al., 2026). Agents therefore need to explore their environments to gather missing information. However, exploring large, unfamiliar environments to discover task-relevant information remains challenging (Krishnamurthy et al., 2024; Nie et al., 2024; Tajwar et al., 2025; Jiang et al., 2025).

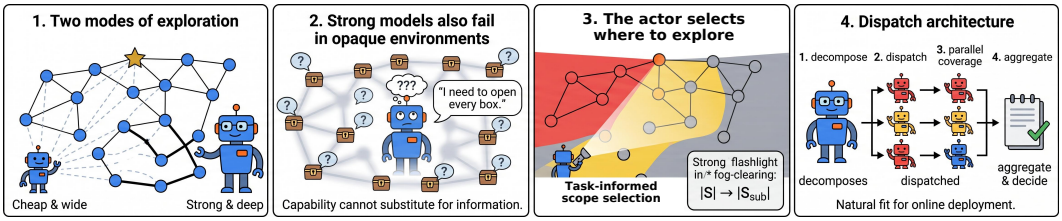


Figure 1: Overview of our argument. (1) Two modes of exploration: cheap-and-wide vs. strong-and-deep. (2) Strong models also fail in opaque environments: capability cannot substitute for information. (3) The actor selects a task-conditioned candidate scope, narrowing the inspection space $|\mathcal{S}| \rightarrow |\mathcal{S}_{\text{sub}}|$. (4) Dispatch architecture: a strong actor decomposes and dispatches subproblems to weak scouts in parallel, then aggregates and decides.

Exploration in interactive environments requires both identifying promising candidates and inspecting them to verify whether they contain the target. Stronger models may improve candidate selection, but discovery also depends on how much of the candidate space can be inspected within a limited budget. We therefore ask whether greater model capability alone is sufficient for effective exploration. Our controlled probes show that it is not always sufficient: increasing model size offers limited gains when the environment provides no information about target location, and the stronger scout's advantage over uniform inspection narrows as the candidate space grows under calibrated inspection budgets (§2). These observations suggest combining task-informed candidate selection with broad inspection coverage.

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.

Motivated by these observations, we propose *scope-then-cover*. A strong actor selects a candidate scope based on the task and available observations. Weak scouts then inspect candidates within this scope in parallel, subject to an inspection budget. The actor invokes this procedure through an *explore action* and uses the returned observations to choose its next action.

Across ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022) and SWE-Bench-Verified (Jimenez et al., 2024), the explore action improves task performance over an actor-only baseline, in which the actor explores through its standard environment actions without calling scouts. Controlled analyses examine how scope quality, inspection budget, scout model size, and returned information affect these gains. These results show that the strong actor can guide exploration while weak scouts perform the inspections.

Contributions. **(i)** Scope-then-cover, a decomposition of exploration into candidate scope selection and parallel inspection, motivated by controlled probes and implemented as an explore action (§2, §3.4). **(ii)** A budget–prior framework that characterizes hit-probability advantage and derives an r/ρ lift over uniform direct coverage under the stated assumptions (§3, §3.4). **(iii)** Empirical validation across ALFWorld, WebShop, and SWE-Bench-Verified on matched task sets, together with controlled analyses of scope quality, inspection budget, scout model size, and returned information (§4).

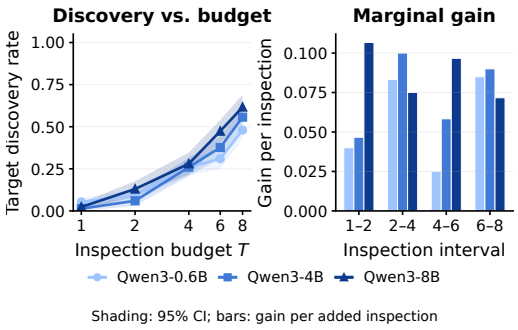
2 The Complementary Roles of Guidance and Coverage

We use two controlled probes to examine when stronger models improve exploration. Both measure whether the inspected candidates contain the target, separating target discovery from subsequent task execution. The first removes information about the target location; the second increases the candidate space under a fixed time budget. Together, they show that the value of model capability depends on the information available and the number of candidates that can be inspected.

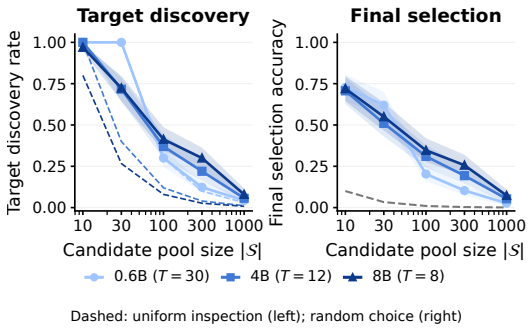
Without informative priors, inspection budget determines discovery. In ALFWorld (Shridhar et al., 2021), a simulated household environment where agents locate and manipulate objects, we randomize object placement so that the task description provides no information about the target receptacle. Qwen3 scouts at three scales (0.6B, 4B, and 8B) inspect up to T receptacles. Increasing the inspection budget improves target discovery, whereas increasing model size produces little separation between the curves (Figure 2a). The gain from each additional inspection likewise shows no consistent ordering across model scales. When the environment supplies no useful clue about where to look, stronger reasoning cannot replace checking more candidates.

Under a fixed time budget, a growing search space erodes the advantage of stronger models. In WebShop (Yao et al., 2022), a simulated shopping environment where agents search for products matching user requirements, we expand the candidate pool from 10 to 1,000 products while matching the time budget across model scales. This allows the 0.6B, 4B, and 8B scouts to inspect up to $T = 30$, 12, and 8 products, respectively. The cheaper scout can inspect more products within this budget, reaching full coverage in small pools while stronger scouts leave some candidates unchecked (Figure 2b). As the pool grows, discovery rates fall across all scales. For the 8B scout, the advantage over uniform random inspection at the same inspection budget narrows from 33 percentage points at 100 candidates to 7 points at 1,000. The model still guides search, but its guidance captures a shrinking share of the target locations as the search space expands. Model capability therefore does not remove the need for sufficient inspection coverage.

These observations suggest that one way model capability improves discovery is by guiding candidate selection under a limited inspection budget. Section 3 formalizes the relationship between prior information, inspection budget, and target discovery, and examines how selecting a compact candidate scope affects discovery.



(a) **ALFWorld: varying the inspection budget.** Target discovery rate is the fraction of tasks whose inspected candidates contain the target (left). The right panel shows the increase in discovery rate per additional inspection between adjacent budgets. Shading: bootstrap-over-tasks 95% CIs; 100 tasks, 3 seeds.



(b) **WebShop: expanding the candidate pool.** Target discovery rate (left) and final selection accuracy (right) distinguish finding the target from choosing it. Dashed references show uniform inspection without replacement, $\min(T, |\mathcal{S}|)/|\mathcal{S}|$, at each scout's budget (left), and a uniformly random final choice, $1/|\mathcal{S}|$ (right). Shading: bootstrap-over-tasks 95% CIs; 100 tasks, 3 seeds.

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 agent faces a candidate space $\mathcal{S}$ (receptacles in a room, products in a catalog, files in a repository) and a fixed but unknown target $s^* \in \mathcal{S}$. Let P^* denote the target-location distribution conditional on the current task and available observations. The agent holds a belief distribution b over $\mathcal{S}$ 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.

The agent uses its belief b to choose an inspection set $\mathcal{I}_T(b) \subseteq \mathcal{S}$ of T distinct candidates. Its probability of containing the target is

$$P_{\text{hit}}(b, T) := P^*(s^* \in \mathcal{I}_T(b)). \tag{1}$$

Here a hit means that the inspected set contains the target.

We compare model-guided inspection with random inspection under the same budget. This reference makes no task-informed judgments about which candidates to inspect. Sampling T distinct candidates uniformly without replacement gives

$$P_{\text{uniform}}(T) = \frac{T}{|\mathcal{S}|}, \qquad T \le |\mathcal{S}|. \tag{2}$$

Model capability can improve Eq. 1 by producing a belief b that assigns more probability to target-containing candidates and therefore supports a better inspection set. The budget T limits how many candidates that set can contain.

3.2 Two Boundaries

We measure the advantage of model-guided inspection over uniform inspection by the difference between their hit probabilities:

$$\Delta(b, T) = P_{\text{hit}}(b, T) - P_{\text{uniform}}(T). \tag{3}$$

Both failure modes of §2 correspond to this advantage vanishing under different parameter regimes.

Boundary 1: no information to guide inspection. When the target is uniformly distributed over the candidate space conditional on the information available to the agent, using b to choose candidates

provides no advantage over uniform inspection. Thus $P_{\text{hit}}(b, T) = P_{\text{uniform}}(T)$ and $\Delta(b, T) = 0$. At a fixed candidate-space size, the hit probability depends only on how many candidates are inspected. This provides a reference for interpreting the limited benefit of increasing model size in the randomized ALFWorld probe.

Boundary 2: the hit-probability advantage contracts. A large candidate space alone does not imply collapse: an informative belief can still guide inspection toward the target. The second boundary arises when expanding the candidate space leaves the model's selected candidates with little additional chance of containing the target relative to uniform inspection. In this regime, $P_{\text{hit}}(b, T) - P_{\text{uniform}}(T) \to 0$, even though the model may retain some useful information about where to look.

The two boundaries share the same outcome: $\Delta(b, T)$ vanishes. In the first, no available information supports better candidate selection; in the second, the information the model uses no longer sustains a substantial hit-probability advantage at the given budget. The WebShop results in Figure 2b illustrate the latter trend: as the candidate pool grows, the target discovery advantage over uniform random inspection at the same budget contracts.

3.3 SCOPE-THEN-COVER

Boundary 2 suggests reducing the effective candidate space before covering it. Scoping selects a candidate subspace $\mathcal{S}_{\text{sub}} \subseteq \mathcal{S}$, characterized by its fractional size $\rho = |\mathcal{S}_{\text{sub}}|/|\mathcal{S}|$ and recall $r = P^*(s^* \in \mathcal{S}_{\text{sub}})$. Let b' denote b conditioned on the selected scope, and let $\mathcal{I}_T(b') \subseteq \mathcal{S}_{\text{sub}}$ be the corresponding set of inspected candidates. Covering after scoping has hit probability

$$P_{\text{scope}} = r\, P_{\text{hit}}(b', T \mid s^* \in \mathcal{S}_{\text{sub}}), \tag{4}$$

where the second factor is the probability that the inspected candidates contain the target, given that it lies in the selected scope. The effective candidate space is now $\rho|\mathcal{S}|$, 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 over uniform direct coverage)**.** *For a fixed candidate space and scope size, suppose that, given the information used to select inspections, the target is uniformly distributed inside the scope. Reliably inspecting T distinct candidates, with $0 < T < \rho|\mathcal{S}|$, gives*

$$P_{\text{scope}} = r\frac{T}{\rho|\mathcal{S}|}, \qquad \frac{P_{\text{scope}}}{P_{\text{uniform}}(T)} = \frac{r}{\rho}. \tag{5}$$

Scope-then-cover therefore improves over uniform direct coverage whenever $r > \rho$.

The proposition links scope quality and inspection coverage to hit probability. Under its assumptions, increasing recall r raises P_{scope} at fixed scope size and inspection budget; reducing scope fraction ρ raises it when recall and budget are held fixed. Increasing T at fixed scope raises hit probability until the scope is fully covered. The ratio r/ρ characterizes the gain over uniform random inspection at the same budget, while P_{scope} characterizes how scope and budget jointly determine the hit probability.

For an informed direct policy, $P_{\text{direct}} = P_{\text{hit}}(b, T)$, and scope-then-cover is better exactly when

$$\begin{aligned} & P_{\text{scope}} > P_{\text{direct}} \\ \iff \quad & r\, P_{\text{hit}}(b', T \mid s^* \in \mathcal{S}_{\text{sub}}) > P_{\text{hit}}(b, T). \end{aligned} \tag{6}$$

Scoping helps when the probability of retaining and then finding the target exceeds the hit probability of direct search.

These comparisons motivate the following division of labor.

Scoping uses the capable tier. The gain r/ρ rewards a small subspace with high recall. We assign scope selection to the strong actor to exploit its task knowledge in making this tradeoff. Coverage cannot recover a target excluded from the selected scope, regardless of how many scouts inspect it.

Covering need not require additional capability. With a uniform conditional target distribution, any T distinct candidates yield the same conditional hit probability $T/|\mathcal{S}_{\text{sub}}|$. Additional model capability then provides no coverage benefit. When the residual distribution is non-uniform, which candidates are inspected can affect the conditional hit probability.

Coverage leverage under a fixed exploration budget. Let B be the resource budget allocated to candidate inspection and q_m the measured per-inspection cost of model m in the chosen resource unit. Model m can inspect

$$T_m = \left\lfloor \frac{B}{q_m} \right\rfloor \tag{7}$$

candidates. Under reliable inspection and residual uniformity within the selected scope, its hit probability is

$$P_m = r \min\left(1, \frac{T_m}{\rho|\mathcal{S}|}\right). \tag{8}$$

A lower per-inspection cost therefore converts the same exploration budget into broader candidate coverage. This relationship motivates the fixed-time WebShop comparison across model scales (Figure 2b). The following experiments examine the predicted relationships through scope-size and target-inclusion measurements (§4.2) and an inspection-budget comparison at fixed scope (§4.3).

3.4 Method: The Explore Action

We organize exploration around an actor-selected candidate scope. Through an *explore action*, the actor uses the task and available observations to choose which part of the environment to inspect, how many scouts to call, and what information to collect.

The harness assigns inspections within the selected scope to scouts, which execute them in parallel. The scouts return their findings, and the harness combines them into an observation that the actor uses to choose its next action.

The actor determines the candidate scope, whose form depends on the environment: a set of receptacles in ALFWorld, a set of products in WebShop, or a set of repository files in SWE-Bench. Implementation details and the rollout procedure are provided in Appendix C.8 and Appendix C.9.

4 Experiments

We test whether scope-then-cover improves a frozen actor's performance on ALFWorld, WebShop, and SWE-Bench (§4.1). We examine how much the actor narrows the search space while retaining the target, and how target discovery changes with the inspection budget. Further experiments compare candidate selection across model sizes and the effects of returning factual observations or advice to the actor. A short transfer study tests the same role assignment when the actor is updated from collected trajectories (§4.5).

Setup. The frozen test-time study spans navigation, retrieval, and code repair: ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022), and SWE-Bench-Verified (Jimenez et al., 2024). Its strong actor is sized to the environment (Qwen3-8B / 30B-A3B / Coder-Next-80B), with frozen weak scouts. Comparisons use matched task IDs within each protocol. Full test-time model settings are in Appendix C.

4.1 Main evaluation: frozen test-time actor

We keep the actor's parameters fixed and compare five settings. In the actor-only baseline (*none*), the actor explores and completes the task through its standard environment actions, without calling scouts. The two forced settings call scouts before the actor's first action (*forced-pre*) or at a predefined point during task execution (*forced-mid*). The other two settings trigger exploration using an entropy threshold on the actor's first decision token (*self-signal*), or let the actor request it by generating `explore(scope, query)` (*self-prompt*). This call specifies which candidates to inspect and what to look for. For ALFWorld and WebShop, all four exploration settings use the same scout budget (T=6 per call, at most three calls per task). The SWE control-policy runs use an 8B scout with at most 15 tool steps per call and three calls per task (Appendix C).

The main table reports single-run results with random seed 42 on 274 ALFWorld tasks, 200 WebShop tasks, and all 500 SWE-Bench Verified instances. Appendix C.2 gives the evaluation details. On SWE-Bench, the actor-only baseline performs a fast structural scan of the repository. The separate return-content ablation on 50 instances is reported in §4.4.

Table 1: **Test-time performance under different exploration policies.** Scores are per-instance success rates (%) for ALFWorld and SWE-Bench Verified, and attribute-match scores $\times 100$ for WebShop. The three environments contain 274, 200, and 500 evaluation tasks, respectively. Each entry reports a single complete evaluation with random seed 42. Experimental settings are provided in Appendix C.2.

Wiring	ALFWorld	WebShop	SWE-Bench
none (actor-only)	24.82	45.33	45.20
Forced (harness-pinned timing)			
forced-pre	39.78	**54.07**	49.80
forced-mid	37.23	47.21	54.40
Self-triggered (actor decides when)			
self-signal	39.42	46.67	52.40
self-prompt	**45.26**	49.35	**55.80**

Table 2: **Actor-selected scopes across environments.** Scope size relative to the candidate space and inclusion of task-relevant targets, measured at the first explore call. Brackets denote the IQR for scope fraction and 95% bootstrap confidence intervals for inclusion.

Environment	Scope fraction ρ	Inclusion statistic
ALFWorld	0.20 [0.15, 0.21]	0.92 [0.83, 0.98]
WebShop	0.50 [0.50, 0.50]	0.826 [0.773, 0.878]
SWE-Bench	0.03 [0.03, 0.03]	0.72 [0.60, 0.84]

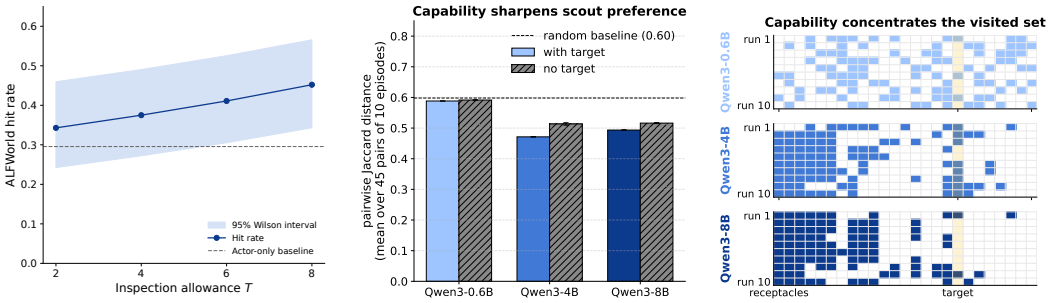
Trigger timing follows the environment's information flow. In ALFWorld, self-prompt performs best because the actor can infer a useful scope from the task and the current household state before dispatching scouts. In WebShop, forced-pre achieves the highest attribute-match score, suggesting that collecting product information early can help the actor make subsequent choices. On SWE-Bench Verified, all explore-enabled policies improve over the actor-only baseline, supporting the utility of the explore action across environments, although the relative advantage of different trigger timings is less consistent. Overall, these results suggest that exploration is a shared mechanism whose trigger should be adapted to the environment's information flow.

4.2 Scope size and target inclusion

We examine scope size and target inclusion using environment-specific measurements: a report-based inclusion proxy in ALFWorld, dispatch of the designated product in WebShop, and reference-patch file inclusion in SWE-Bench. Guided by our theoretical framework, these measurements operationalize the scope-selection component across environments, examining how much the actor narrows the candidate space while retaining task-relevant targets. We report measurements on each task's first explore call in Table 2.

The actor selects scopes with median fractions of 0.20 in ALFWorld, 0.50 in WebShop, and 0.03 in SWE-Bench, with corresponding inclusion statistics of 0.92, 0.826, and 0.72. These measurements show that selected scopes concentrate inspection on a subset of candidates while retaining task-relevant targets in a substantial fraction of tasks.

Differences across environments may reflect the information available for scope selection. In SWE-Bench, issue descriptions and repository structure provide cues for localizing relevant files, which may help the actor select a compact scope. In WebShop, requirements often involve fine-grained attributes revealed on product pages, making it useful to retain multiple products for inspection. In ALFWorld, the task and receptacle types provide location cues within a bounded set of candidates.



(a) ALFWorld hit rate across inspection allowances $T \in \{2, 4, 6, 8\}$. The dashed line shows the separate actor-only baseline. Shading denotes 95% Wilson intervals.

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

(c) Receptacle selections across ten runs of one task. Rows are runs and columns are receptacles; marked cells indicate selected receptacles. The target saltshaker is on shelf 2 (highlighted).

Figure 3: **Inspection budget and diversity of candidate selection.** (a) Target discovery at different inspection budgets. (b) Diversity of selected receptacles across repeated runs on 100 tasks, with and without the target object specified in the prompt. Higher Jaccard distance means more diverse selections. (c) Selections across ten runs of one task.

4.3 How scope and cover contribute

We examine inspection budget and candidate selection on ALFWorld (Figure 3). In the budget experiment, we hold the actor-selected scope and scout model fixed on randomized pick-and-place tasks and vary the number of inspections. Target discovery reaches 0.41 at $T = 6$ and 0.45 at $T = 8$, consistent with the predicted increase in hit probability as more candidates are inspected within a fixed scope.

To examine candidate selection across repeated runs, we fix the budget at ten receptacles per run and evaluate frozen Qwen3-{0.6B, 4B, 8B} models on 100 tasks, each repeated ten times. Prompts either specify the target object or ask the model to explore the layout. The model selects receptacles, and the harness prevents repeated inspections within each independent run. We measure diversity by mean pairwise Jaccard distance between the selected sets; higher values indicate less overlap across runs (App. C.10).

With the target object specified, the 4B and 8B models select more similar sets of receptacles across runs than the 0.6B model. The example in panel (c) illustrates this difference: across ten runs, the 0.6B model visits every receptacle at least once, whereas the 8B model repeatedly selects the same group of cabinets. More varied selections allow repeated runs to cover different locations and collect information that repeated searches may miss. Under the generic layout instruction, selection diversity increases for the 4B and 8B models, showing that their selection patterns also depend on the exploration objective.

4.4 What information should scouts return?

We compare scout return content on the same 50 SWE-Bench instances. All settings start with a structural scan of the repository. Each of the six exploration variants adds one call to a single scout, returning a summary with fix advice, source code with a hint about which parts to skip reading, a factual brief with a caller graph, a source-code excerpt, test code, or a bundle of source code, tests, and caller information. Full variant definitions are in App. C.6.

On this task subset, returning test code achieves the highest observed pass rate at 58%, compared with 48% for the structural-scan baseline (Table 3). The factual brief and source-code excerpt each reach 52%, while the bundle of source code, tests, and caller information reaches 54%. Both the

Table 3: **SWE-Bench-mini return-content ablation (n=50).** All rows use the same 50 instances and start from the structural-scan baseline (Tier 0); each + row adds one scout call returning the indicated payload. ΔPass is the paired difference against that baseline.

Configuration	Pass (%)	ΔPass
Tier 0. Structural scan	48.0	—
Tier 1: + scout summary including a judgment		
+ Brief: narrative + fix advice	48.0	+0.0
+ Read source + skip-view hint	48.0	+0.0
Tier 2: + facts only, no judgment		
+ Brief: factual (caller graph)	52.0	+4.0
+ Read source	52.0	+4.0
+ Read test	**58.0**	**+10.0**
+ Read all (source + test + callers)	54.0	+6.0

summary with fix advice and source code with a skip-reading hint achieve 48%. These results suggest that providing concrete code and test information may help the actor complete tasks.

4.5 Learning from exploration through parameter updates

The preceding experiments use exploration reports to help a frozen actor complete the current task. We next examine whether the same division of labor can support learning through parameter updates. In each round, a Qwen3-4B actor produces a high-level plan and selects an exploration scope; a frozen Qwen3-1.7B scout then collects trajectories inside that scope. The loop extracts reusable experience from these trajectories and combines it with interaction samples to update the actor by on-policy context distillation (Ye et al., 2026). We compare this scope-then-cover collector with the standard single-actor collector. On Sokoban, the two collectors perform similarly in the first round; scope-then-cover achieves higher held-out accuracy from step 24 onward, with the largest gap of 12 percentage points at step 32 (Figure 4).

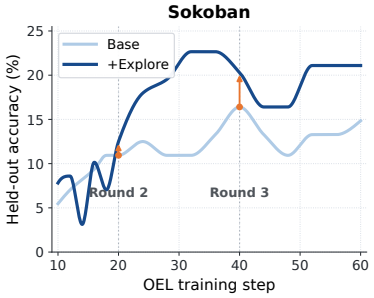


Figure 4: Sokoban held-out accuracy during iterative collection and consolidation.

5 Related Work

Exploration in LLM agents. Exploration in LLM agents has been studied through in-context decision-making and learning from environmental feedback. Krishnamurthy et al. (2024) and Nie et al. (2024) examine exploration in bandit and MDP settings, while Arumugam & Griffiths (2025) study efficient exploration for language agents. LAMER and PAPRIKA train agents to acquire useful information through interaction (Jiang et al., 2025; Tajwar et al., 2025). A related line structures search through interleaved reasoning and actions, reflection, or tree search, as in ReAct, Reflexion, RAP, and LATS (Yao et al., 2023; Shinn et al., 2023; Hao et al., 2023; Zhou et al., 2024). Our study examines how model capability and candidate inspection coverage jointly determine target discovery, providing a basis for allocating exploration across models.

Computation allocation and experiential learning. Inference-time scaling allocates additional computation through repeated sampling and adaptive reasoning budgets (Wang et al., 2023c; Brown et al., 2024; Snell et al., 2025). Cascades and routers distribute queries across models to balance cost and quality (Chen et al., 2024; Ong et al., 2025). For interactive agents, Reflexion and ExpeL retain experience in context, while online experiential learning consolidates interaction experience into model parameters (Shinn et al., 2023; Zhao et al., 2024; Ye et al., 2026). Scope-then-cover concerns the allocation of computation during information collection: a strong actor selects a candidate scope, and weak scouts inspect it. We study this allocation both within task execution and during trajectory collection for parameter updates.

Model collaboration and delegated exploration. Planning and multi-agent systems organize work through explicit plans, role-based coordination, and agent handoffs (Wang et al., 2023b; Shen et al., 2023; Wu et al., 2023; OpenAI, 2024). WESE places weak-agent exploration before strong-agent exploitation: the explorer gathers environmental knowledge, which is stored in a knowledge graph and retrieved for subsequent task solving (Huang et al., 2024). Scope-then-cover uses the strong actor's task information to select the scope before scout inspection and returns the observations within the ongoing rollout. This ordering connects model collaboration to the question studied in our probes and analysis: how task-informed scope selection and inspection coverage contribute to discovery.

6 Conclusion

LLM-agent exploration requires both task-informed guidance and environmental inspection. A minimal hit-rate equation (§3) relates guidance and inspection coverage to target discovery and predicts an r/ρ lift over uniform direct coverage under its stated assumptions. Realized as a single *explore action* with strong-actor scopes and frozen weak-scout cover, it improves an actor-only baseline across ALFWorld, WebShop, and SWE-Bench-Verified. The central design rule is to spend strong-model capability on selecting the search region and weak-model parallelism on covering it.

7 Discussion

Classical HRL options (Sutton et al., 1999) decompose actions; we decompose by *subproblem kind* and route across capability tiers. The parameter-update study shows that the same role assignment extends beyond frozen actors. Multi-tier scout pools, persistent experience banks, and retrieval-augmented scoping remain possible extensions.

Limitations

Scope-then-cover relies on task context that supports high-recall subspace selection and on scout observations grounded in the inspected environment. Table 2 reports scope fraction ρ and target inclusion r, while the return-content analysis in §4.4 characterizes the factual observations returned by scouts.

AI Use Statement

Yes, generative AI tools were used to assist with manuscript polishing, organization of experimental results, and translation. The authors are responsible for the final text, claims, code, and results.

References

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

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

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

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Wese: Weak exploration to strong exploitation for llm agents. *arXiv preprint arXiv:2404.07456*, 2024.

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

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? In *Advances in Neural Information Processing Systems 37*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-3818.

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

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, 2025. URL https://iclr.cc/virtual/2025/poster/30737.

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems 36*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://iclr.cc/virtual/2025/poster/31024.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

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

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

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023b.

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

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. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems 35*. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022.

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

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

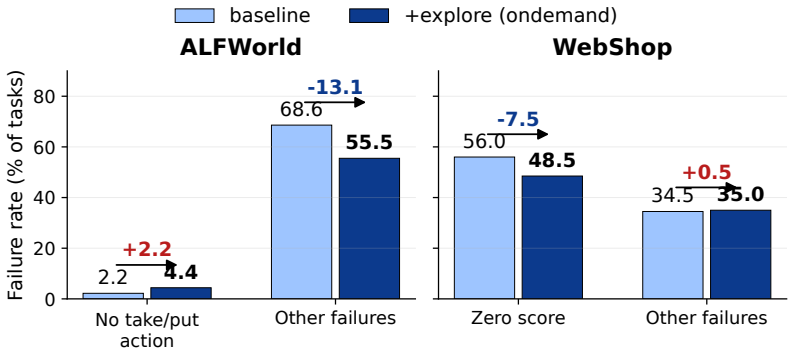


Figure 5: **Failure categories in the Qwen3-30B-A3B comparison.** ALFWorld separates failures with no take/put action from other failures; WebShop separates zero-score outcomes from other unsuccessful outcomes. Bars report percentages of all evaluated tasks. Light blue denotes the baseline and dark blue denotes +explore; arrow labels show percentage-point changes.

A Task Outcomes and Failure Categories

Figure 5 visualizes two descriptive failure categories in ALFWorld and WebShop. F1 denotes no `take/put` in ALFWorld or zero score in WebShop; F2 denotes the remaining failures. Rates use all tasks as the denominator. Each panel pairs the baseline with +explore and labels the change in percentage points.

B Proofs for the framework of §3

We restate the scope comparison using the notation of §3: candidate space $\mathcal{S}$, conditional target distribution P^*, actor belief b, its scope-conditioned form b', inspection budget T, and selected scope $\mathcal{S}_{\text{sub}}$. Its fractional size is $\rho = |\mathcal{S}_{\text{sub}}|/|\mathcal{S}|$, and its target inclusion probability is $r = P^*(s^* \in \mathcal{S}_{\text{sub}})$. Conditional probabilities below are given target inclusion in the selected scope and the information used to choose inspections; b' is b conditioned on $\mathcal{S}_{\text{sub}}$, not the true conditional target distribution.

Proposition 2 (Scope–cover split and uniform-reference lift)**.** *For reliable inspection of the T distinct candidates in $\mathcal{I}_T(b')$ inside a fixed scope, with $0 < T \le |\mathcal{S}_{\text{sub}}|$,*

$$\begin{aligned} P_{\text{scope}} &= r\, P_{\text{hit}}(b', T \mid s^* \in \mathcal{S}_{\text{sub}}) \\ &= r \sum_{s \in \mathcal{I}_T(b')} P^*(s^* = s \mid s^* \in \mathcal{S}_{\text{sub}}) \\ &\le r \min\left(1,\ T \max_{s \in \mathcal{S}_{\text{sub}}} P^*(s^* = s \mid s^* \in \mathcal{S}_{\text{sub}})\right). \end{aligned}$$

If the conditional target distribution is uniform, this becomes

$$P_{\text{scope}} = r\frac{T}{\rho|\mathcal{S}|}.$$

For $T < \rho|\mathcal{S}|$, comparison with uniform direct coverage gives

$$\frac{P_{\text{scope}}}{P_{\text{uniform}}(T)} = \frac{r}{\rho}.$$

Proof. The law of total probability gives the factor r; coverage is confined to the selected scope. Conditional on target inclusion, reliable inspection of distinct candidates succeeds exactly when the target occupies one of those candidates. Summing their conditional target probabilities gives the second equality. Each term is bounded by the largest conditional target probability, and the sum cannot exceed one. Under residual uniformity, each of the T terms is $1/|\mathcal{S}_{\text{sub}}|$, giving the stated equality. Division by $P_{\text{uniform}}(T) = T/|\mathcal{S}|$ gives r/ρ. □

Remark. Distinct inspection means that the sum contains no repeated candidates; it does not make their target probabilities equal. Under a non-uniform residual target distribution, the conditional hit probability depends on which set $\mathcal{I}_T(b')$ the belief induces. Approximate residual uniformity yields an approximate lift, not the exact equality above.

General direct-policy comparison. An informed direct policy has hit probability $P_{\text{hit}}(b, T)$. Scope-then-cover improves on it exactly when

$$r\, P_{\text{hit}}(b', T \mid s^* \in \mathcal{S}_{\text{sub}}) > P_{\text{hit}}(b, T).$$

This condition does not require uniform target probabilities. The closed-form r/ρ comparison uses the uniform direct reference and the proposition's residual-uniformity assumptions.

Proposition 3 (Budget-to-coverage leverage)**.** *Fix an exploration budget B and let q_m be model m's measured per-inspection cost. Under reliable inspection and residual uniformity, model m inspects $T_m = \lfloor B/q_m \rfloor$ candidates and achieves*

$$P_m = r \min\left(1, \frac{T_m}{\rho|\mathcal{S}|}\right).$$

For two models with $q_w < q_a$, the weak scout covers at least as many candidates under the same budget, with a strict hit-probability increase whenever $T_w > T_a$ and the scope is unsaturated.

Proof. Since $q_w < q_a$, $T_w \geq T_a$. The hit probability is monotone in T_m and strictly increasing before saturation. □

Remark (allocation). Scope quality determines where to inspect; per-inspection cost determines how much of that scope can be covered within the exploration budget. Together, they motivate capable-actor scoping and parallel weak-scout coverage.

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 scout 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 Evaluation Sets and Scoring

For ALFWorld, we evaluate all methods on the same 274 tasks with random seed 42. In table order, the five policies solve 68, 109, 102, 108, and 124 tasks. For WebShop, we use the same 200 task instances with seed 42 and report the benchmark's native attribute-match score. Every policy has 200 unique completed records. The SWE-Bench main table uses the full 500-instance Verified set with seed 42.

C.3 Strong Actor (test-time setting)

The SWE-Bench experiments in Section 4.1 use a frozen Qwen3-Coder-Next-80B-A3B actor. The actor parameters remain fixed throughout evaluation.

C.4 Strong Actor (online-learning setting)

The online-learning experiments in Section 4.5 use Qwen3-4B-Instruct-2507 as the trainable strong actor. The actor is updated through on-policy context distillation following the OEL protocol (Ye et al., 2026).

C.5 Weak Scout Configuration

The scout parameters remain fixed during evaluation and trajectory collection.

C.6 Operator Returns (test-time variants)

The content ablation in Section 4.4 evaluates a structural scan baseline and six scout-return variants. The variants differ in both the type of information returned to the actor and whether the scout provides explicit repair advice. The returned information includes factual summaries, source-code excerpts, test-code excerpts, caller information, and combinations of these contents. All variants use the same scout and evaluation protocol.

C.7 Environment Harnesses

The online-learning experiments in Section 4.5 use the text-game harness provided by the OEL codebase. The SWE-Bench main evaluation uses the full 500-instance Verified set, while the return-content ablation uses a separate fixed subset of 50 instances. All evaluations use the same task instances and per-task budgets within each comparison.

C.8 Explore Action Design and Implementation

Motivation. We borrow the high-level/low-level split of the *options* framework (Sutton et al., 1999), but repurpose it across capability tiers rather than time horizons: the strong actor plays the high-level controller and decides *where* to look, while frozen weak scouts act as inner sub-policies that actually *look*. Orchestrator/worker LLM patterns (Shen et al., 2023; Wu et al., 2023) route by task type; our division of labor follows the scope-quality analysis in §3: the strong actor selects the candidate scope, and weak scouts perform the inspections.

The explore action. At each decision step, the actor either takes an environment action or calls `explore(scope, query)`. The `scope` defines a candidate subspace $\mathcal{S}_{\text{sub}} \subseteq \mathcal{S}$, and the `query` states what evidence to collect there. The harness dispatches candidate inspections to frozen weak scouts in parallel, subject to the inspection budget T. Each scout receives an assigned candidate and the query, performs the required environment actions, and returns its findings.

In ALFWorld and WebShop, scouts inspect candidates in separate environment sessions. ALFWorld scouts initialize the same task and navigate to the assigned receptacle; WebShop scouts replay the actor's browsing actions in their own sessions before inspecting the assigned product. The actor's environment remains at its pre-call state. The harness combines the per-target reports into one observation, preserving the association between each candidate and its findings. The actor adds this observation to its context and chooses its next action. Algorithm 1 (Appendix C.9) gives the per-rollout procedure.

What constitutes a valid scope depends on the environment. We define valid scopes from groupings already exposed by the environment, with each group containing units that scouts can inspect independently. In our experiments, these are receptacle clusters in ALFWorld, product categories in WebShop, and directory subtrees in SWE-Bench. During a rollout, the actor selects or combines these groups according to the current task. We choose the grouping granularity on a small validation set, seeking high target recall r with a small subspace fraction ρ, the scope-quality tradeoff characterized by Proposition 1.

Transfer beyond frozen actors. The main method is the in-rollout explore action used by a frozen actor. We also apply this division of strong guidance and weak collection during iterative experience collection, where trajectories collected by scouts are consolidated into parameter updates for the actor (§4.5).

C.9 Per-Rollout Pseudocode

Algorithm 1 Test-time scope-then-cover explore action (§3.4)

Input: actor π_a, frozen scout π_w, environment $\mathcal{E}$, inspection budget T
for each task **do**
  repeat
    $a \leftarrow \pi_a(\text{context})$
    if $a =$ `explore(scope, query)` **then**
      $\mathcal{S}_{\text{sub}} \leftarrow$ `scope`
      Assign up to T candidate inspections within $\mathcal{S}_{\text{sub}}$ to scouts
      Execute the inspections in parallel
      $o \leftarrow$ `aggregate`(scout reports)
      Update the context with o
    else
      Execute a in $\mathcal{E}$
    end if
  until task is complete
end for

C.10 Diversity Experiment Setup (§4.3)

We evaluate scout selection diversity on 100 ALFWorld pick-and-place tasks. Each task is repeated under multiple scout models and prompt conditions, with a fixed inspection budget and no repeated inspections within a run. The prompt conditions either specify the target object or ask the scout to explore the layout. We keep the sampling procedure, candidate enumeration, and returned notepad format fixed across conditions.

For each task, we measure the mean pairwise Jaccard distance between the sets of candidates selected in independent runs:

$$d(V_i, V_j) = 1 - \frac{|V_i \cap V_j|}{|V_i \cup V_j|}.$$

Higher values indicate greater diversity in candidate selection.

D WebShop Controlled-Experiment Details

D.1 WebShop Evaluation Protocol

All five control policies use the same 200 task IDs, Qwen3-30B-A3B actor, Qwen3-4B scout, sampling configuration, 20-turn actor budget, environment, action interface, and native attribute-match scorer. Table 1 reports the complete seed-42 evaluation, with 200 unique task records per policy. Evaluation details are provided in Appendix C.2.

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 $T \in \{30, 12, 8\}$ for 0.6B / 4B / 8B are chosen so that $T \times$ 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 T 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, actor-only 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 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, actor-only 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 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 Exploration with different actor models

Exploration improves ALFWorld success rates for both actor models (Table 4). With Qwen3-8B-Instruct, success increases from 29.6% without scouts to 42.7% with the explore action. With Qwen3-30B-A3B-Instruct, it increases from 29.2% to 40.1%. The gains from exploration are retained across these two actor models.

Table 4: **ALFWorld actor-scale ablation.** Same harness, same wirings, same sampling temperature 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 scout)	29.6	29.2	—	—
+ prefetch	42.7	38.0	+13.1	+8.8
+ midway	36.5	35.8	+6.9	+6.6
+ explore	42.7	40.1	+13.1	+10.9

SWE-Bench-mini content-ablation 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.

E.2 Qualitative Example

SWE-Bench-mini content-ablation example. This example belongs to the 50-instance content ablation in Section 4.4. **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.

The actor enters its decision loop with evidence from the relevant repository 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 Mechanism Measurement Definitions

For each evaluated exploration call i, let $\mathcal{S}_i$ denote the candidate space, $\mathcal{S}_{\mathrm{sub},i}$ the scope selected by the actor, and G_i the set of task-relevant targets determined from environment ground truth. We define scope inclusion as

$$I_i = \mathbf{1}[G_i \cap \mathcal{S}_{\mathrm{sub},i} \neq \emptyset],$$

and scope fraction as

$$\rho_i = \frac{|\mathcal{S}_{\mathrm{sub},i}|}{|\mathcal{S}_i|}.$$

Scope recall is the mean of I_i over the evaluated calls.

For scout-capability comparisons, we keep the pre-call state, selected scope, query, and inspection budget fixed. For scope-selection comparisons, we keep the candidate space and inspection procedure fixed.