LEARNING WHAT TO REFRESH: COUNTERFACTUAL OBSERVATION SELECTION FOR TOOL-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using language agents often act on cached observations collected earlier in the trajectory, even though the external world may have changed in the meantime. This induces a belief-repair problem: the agent's internal state remains anchored to historical evidence whose relevance depends on the current task objective and the evolving environment. Re-querying every source is prohibitively expensive, while observation age alone is not a reliable indicator of whether a refresh will improve the next decision. We study task-aware observation refresh: selecting a subset of previously observed sources to revisit before the agent continues execution. Rather than treating staleness as a binary signal or a recency heuristic, we learn refresh decisions from counterfactual interventions at the same task checkpoint, comparing downstream task utility under different refresh sets while holding the agent state fixed. A set-conditioned critic estimates the joint value of candidate refresh sets, allowing the policy to account for complementary and redundant observations while balancing expected utility against acquisition cost. The controller operates on agent-visible history, cached observations, timestamps, and query costs, with the language-model backbone kept frozen. We evaluate the approach on a controlled synthetic benchmark with stale observations and task-dependent source interactions, and compare against no-refresh, always-refresh, recency-based refresh, and additive-value baselines. Our goal is to characterize when selective refresh improves the success–cost trade-off and whether the learned policy transfers to unseen task structures and environment dynamics.

1 INTRODUCTION

Tool-using agents often operate over a partially stale world model. Gaia2 and SentinelBench evaluate agents in environments that change during execution (Froger et al., 2026; Maldaner et al., 2026). As agents accumulate observations, later evidence can invalidate stored beliefs or introduce temporal conflicts (Chao et al., 2026; Lyu et al., 2026). This creates a belief-repair problem: the agent's internal state is anchored to historical evidence whose relevance depends on the task and the evolving world. The challenge is to decide which previously observed sources are worth revisiting before the next action.

Observation age alone is not a sufficient signal. Information acquisition must balance decision value against cost (Li & Oliva, 2025; Kiyani et al., 2026). Two cached observations can have different downstream value depending on the task, remaining budget, and interactions among sources. Some are more informative together; others become redundant after a related update. Such interactions also motivate nongreedy active feature acquisition (Valancius et al., 2024). Refresh is therefore a task-aware subset selection problem, rather than a recency heuristic.

Tool-use and adaptive-retrieval methods address when to seek external evidence (Schick et al., 2023; Asai et al., 2024; Sun et al., 2026). Synchronization and event-triggered observation offer mechanisms for updating context (Zheng et al., 2026; Gaur & Lane, 2026). Jointly selecting which cached sources to refresh under a limited budget poses a further optimization problem. We study task-aware observation refresh: revisiting a subset of previously queried sources to repair the agent's world model and improve downstream task utility under a fixed acquisition budget.

Building on counterfactual comparisons of tool-use outcomes (Zou et al., 2026), we formalize observation refresh over replayable checkpoints. From the same agent state and cache, we compare

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.

refresh choices under a shared continuation policy to estimate their effects on task completion. The resulting supervision lets the controller learn which sources merit refreshing and which should be updated jointly. The language-model backbone remains frozen; only the refresh controller is trained.

The core technical challenge is that refresh value is compositional. A source may be low-value in isolation but critical when paired with another observation that constrains the same decision. To capture this, we learn a set-conditioned critic that estimates the joint value of a refresh set rather than assuming additive per-source utility. The controller then selects the highest-value refresh set subject to a cost budget.

Our contributions are threefold. First, we formalize selective observation refresh as a cost-sensitive belief-repair problem for tool-using agents in dynamic environments. Second, we learn refresh values from paired counterfactual rollouts from the same checkpoint, enabling budget-aware decisions over candidate refresh sets. Third, we evaluate the approach against baselines such as no-refresh, always-refresh, recency-based refresh, and additive-value policies, showing when joint refresh decisions improve the success–cost trade-off.

2 Related Work

Tool use and adaptive retrieval. Language models can gather external evidence while reasoning and acting. ReAct (Yao et al., 2023) interleaves reasoning traces with actions and observations, while Toolformer (Schick et al., 2023) learns when to invoke APIs and how to incorporate their outputs through self-supervision. Self-RAG (Asai et al., 2024) learns to retrieve passages on demand and assess their relevance to generation. More recently, When2Tool evaluates tool necessity, while Probe&Prefill uses pre-generation hidden states to steer tool invocation (Sun et al., 2026). These approaches establish mechanisms for acquiring evidence during execution. We consider a subsequent decision: which previously observed sources should be queried again when their cached values may no longer support the next action.

Observation refresh in dynamic environments. The temporal validity of observations has received increasing attention. TicToc examines whether agents' tool-use decisions over elapsed time agree with human preferences (Cheng et al., 2026). CORVUS synchronizes registered files with their current contents at each reasoning cycle (Zheng et al., 2026), while Signal-Driven Observation proposes rule-based triggers for re-observing web pages after relevant events (Gaur & Lane, 2026). AgentCheck and EnvTrustBench provide complementary evaluations of failures involving stale or unreliable environmental evidence (Mazumder & Lia, 2026; Sheng et al., 2026). Synchronization and event triggers offer concrete refresh mechanisms; our controller learns which subset to refresh by estimating its downstream task value under an acquisition budget. The target is evidence that may have become outdated as the environment changes; a new query is not assumed to correct a persistently unreliable source.

Counterfactual supervision. CoBRA (Zou et al., 2026) is a close methodological precedent: it learns tool-use boundaries from paired expert trajectories and counterfactual reward margins. Its reference-split construction compares internal and external branches under a shared reference policy. Our decision concerns subsets of cached sources at a replayable task checkpoint, including the choice to refresh none. We train a set-conditioned critic from their downstream returns and query costs while keeping the language-model backbone frozen. This use of downstream consequences also connects to AggreVaTe, which learns from expert cost-to-go estimates (Ross & Bagnell, 2014), and CF-GPS, which uses structural causal models to evaluate alternative policies on logged episodes (Buesing et al., 2018). Our supervision comes from replaying refresh interventions in the task environment rather than inferring alternative trajectories through a learned world model.

Cost-sensitive and joint information acquisition. Selective refresh is related to the decision-theoretic value of information (Howard, 1966). Active-Acquisition POMDPs couple observation acquisition with sequential task rewards (Li & Oliva, 2025), and work on time-varying features studies the assumptions needed to evaluate acquisition policies from retrospective data (von Kleist et al., 2025). For language-model agents, CAM-DF learns when to stop expanding a ranked tool prefix before execution (Feng et al., 2026); refresh instead acquires updated source values during execution. The acquisition conditioned oracle (ACO) addresses nongreedy feature acquisition and the

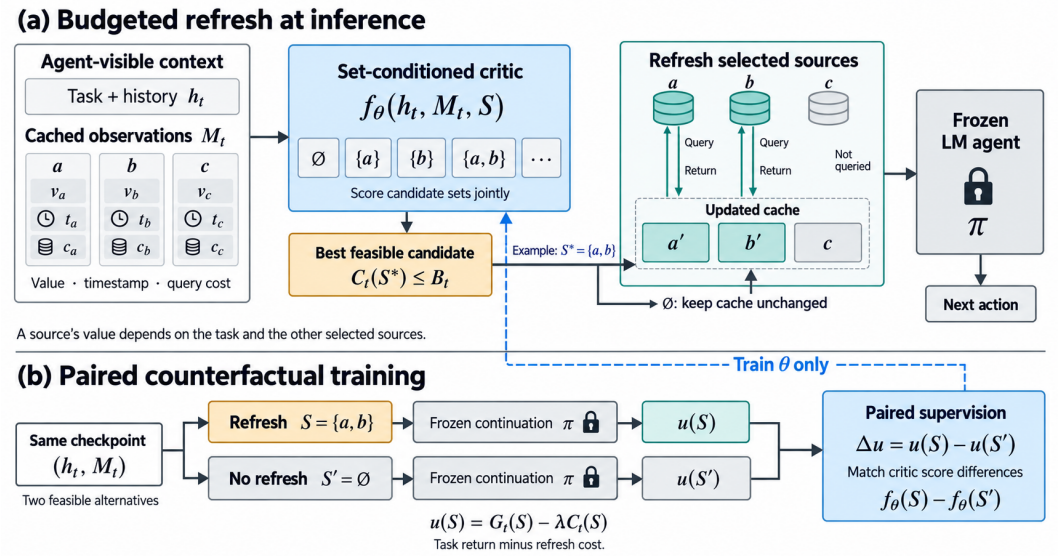

Figure 1: **Task-aware observation refresh.** (a) The set-conditioned critic scores candidate refresh sets using agent-visible history and cached observations, including timestamps and query costs. The controller selects the best feasible candidate under the refresh budget; the empty set retains the cache unchanged. Selected sources are re-queried before the frozen language-model agent continues. (b) Feasible alternatives are replayed from the same checkpoint under the same frozen continuation policy. Differences in observed return-minus-refresh-cost outcomes supervise differences between critic scores. Only the critic parameters θ are trained; the shared context is omitted from $f_\theta(S)$ for readability.

limitations of selecting features solely by their individual benefit (Valancius et al., 2024). Relatedly, Set Transformer models interactions among set elements while preserving permutation invariance (Lee et al., 2019). These works motivate evaluating observations jointly. Our critic scores candidate refresh sets directly, allowing the value of refreshing one source to depend on which other sources are refreshed.

Cache maintenance and agent memory. Learning to Crawl optimizes cache freshness under a bandwidth constraint while estimating unknown page-change rates (Upadhyay et al., 2020). In agent memory, STALE evaluates whether models recognize invalid memories after receiving relevant new evidence (Chao et al., 2026). Decision-aware memory cards select and compress existing context according to its influence on decisions (Guan et al., 2026), while PReM refreshes compressed context by reselecting evidence from a stored representation of the original context (Yu et al., 2026). Causal memory interventions compare behavior with absent, present, or perturbed memories to guide memory selection (Srivastava, 2026). These methods concern cache freshness or the use of information already available to the agent. We study the task value of obtaining new observations from previously queried external sources, with acquisition cost included in the refresh objective.

3 METHOD

Let h_t denote the agent-visible state at decision time t, including the interaction history, cached observations, and task context. Let M_t be the set of cached observations, where each object records its source identity, last observed value, timestamp, and query cost. Let I_t index the sources eligible for refresh. For a candidate subset $S \subseteq I_t$, the refresh controller must decide whether to revisit those sources before the next action. We view this as a budgeted belief-repair problem: the agent's internal state may be anchored to stale evidence, and the controller must select the subset that most improves the latent task state under a fixed refresh budget.

For a candidate refresh set S, we define its value as the expected downstream task return minus the acquisition cost:

$$Q_\lambda(h_t, M_t, S) = \mathbb{E}[G_t(S) - \lambda C_t(S) \mid h_t, M_t], \tag{1}$$

where $G_t(S)$ is downstream task return and $C_t(S)$ is the cost of re-querying the selected sources. The empty set corresponds to no refresh. The controller chooses the subset that maximizes this objective under the current budget.

This formulation differs from recency-based refresh rules. The decision depends on the current task state, the remaining budget, and the joint utility of multiple observations, not merely on whether an observation is old. In other words, refresh is treated as a selective intervention on the agent's world model, rather than a generic freshness check.

3.1 Counterfactual supervision

We train the controller from paired counterfactual rollouts generated from the same checkpoint. For each state (h_t, M_t), we keep the task context fixed and compare multiple refresh choices, including no refresh, single-source refreshes, and small multi-source refresh sets. Each branch continues under the same downstream policy, and we record the resulting reward and cost. This produces matched comparisons that isolate the effect of each refresh decision under the same task state.

For two feasible refresh sets S and S', their relative value is

$$A_\lambda(S, S' \mid h_t, M_t) = Q_\lambda(h_t, M_t, S) - Q_\lambda(h_t, M_t, S'). \tag{2}$$

A paired rollout estimates this difference by subtracting the two observed return-minus-cost outcomes. With $S' = \emptyset$, the target measures the incremental benefit of refreshing over continuing with the cache. Sharing the initial checkpoint controls for differences in starting task difficulty; it does not eliminate uncertainty in subsequent execution. The critic's score differences should therefore approximate relative utility, rather than encode a fixed preference for refreshing more sources. Costs here refer to the selected refresh intervention, not all subsequent tool use.

This is essential because refresh value is not a static property of an individual observation. It depends on the current decision context, remaining budget, and interactions among sources. A policy trained on age-only labels or generic freshness cues would fail to identify cases in which a small, coordinated refresh yields a large downstream gain, or cases in which an apparently stale source is irrelevant to the current objective.

3.2 Set-conditioned critic

The key design choice is to score refresh sets jointly rather than summing independent source values. We use a set-conditioned critic $f_\theta(h_t, M_t, S)$ that evaluates the candidate subset directly. This allows the model to capture complementary and redundant observations, which are common in tool-using tasks where multiple cached sources constrain the same decision.

Refresh utility is non-additive: one source may be low-value alone but highly valuable jointly with another, while a second source may become unnecessary once a related update is obtained. The critic therefore accounts for pairwise and higher-order dependencies among cached observations, rather than assuming the total value is the sum of independent source scores. This set-level scoring is the central mechanism that distinguishes our method from additive refresh heuristics.

The relevant quantity is the marginal value of adding a source to an already selected set. For example, in delivery planning, updated inventory and transit time may jointly establish whether an order can arrive before a deadline; either update alone can leave the decision unresolved. Conversely, two sources reporting the same availability may be redundant. These illustrative cases motivate conditioning on both the task and the selected set, without assuming that every task benefits from interaction modeling.

3.3 Policy and training

At deployment time, the policy selects the refresh set maximizing the critic score under a budget constraint:

$$S^* = \arg \max_{S \in \mathcal{C}_t,\ C_t(S) \le B_t} f_\theta(h_t, M_t, S). \tag{3}$$

We restrict the candidate search to promising subsets, using exact enumeration in small cases and budgeted search in larger settings. This keeps the controller efficient while preserving the ability to reason over sets rather than greedily selecting single sources.

Here, $\mathcal{C}_t \subseteq 2^{I_t}$ is the searched candidate pool and includes the empty set. Unlike the unknown expected value Q_λ, f_θ is evaluated using agent-visible inputs. Including the empty set allows the policy to decline a refresh when its predicted benefit does not justify its cost. The hard budget limits feasible expenditure, while λ penalizes cost within that feasible region. Restricting the candidate pool introduces search error separately from critic estimation error; the selected set need not be optimal over all subsets.

Training proceeds by collecting replayable checkpoints, applying alternative refresh interventions from each checkpoint, and optimizing the critic against the observed downstream utility. The language model stays frozen; only the refresh controller is learned. The supervision signal is therefore tied directly to task outcomes, rather than to recency or local heuristics.

4 EXPERIMENTS

We evaluate task-aware observation refresh in a controlled benchmark that isolates the effect of stale cached observations in tool-using agents. The experiments examine task completion under limited refresh budgets, the contributions of task context and counterfactual supervision, robustness to environmental drift and distribution shift, and the computational cost of set-level selection.

4.1 EXPERIMENTAL SETUP AND EVALUATION PROTOCOL

We construct a benchmark with several task families, including route planning, inventory coordination, supplier triage, and delivery scheduling. These tasks are chosen because they naturally require multiple observations to be considered jointly. In many cases, a single stale source is not sufficient to determine the correct next action, while a small set of stale observations can jointly alter the action selection. This structure is essential for testing the main hypothesis of the paper: in non-stationary tool-use settings, refresh decisions should be formulated over observation sets rather than over isolated sources.

Each task includes a user objective, a partially stale cache of observations, a per-source refresh cost, a time-varying world state, and a fixed refresh budget. The benchmark also contains task-specific critical sources and pairwise interaction patterns so that the usefulness of refreshing one observation can depend on the state of another. This design allows us to compare source-wise refresh heuristics against set-level refresh policies under the same cost and execution conditions. The downstream planner and tool-execution policy are kept fixed across methods, so differences in performance are attributable to the refresh controller instead of to downstream reasoning or tool access.

We compare the following methods:

- **No Refresh**: the agent continues with the stale cache and makes no update.
- **Always Refresh**: the agent refreshes all stale observations in the current cache.
- **Recency-based Refresh**: the agent refreshes observations with the largest age or staleness score.
- **Additive Value Refresh**: the agent scores each source independently and selects a cost-feasible subset.
- **Set-conditioned Refresh (ours)**: the agent scores candidate refresh sets jointly using task context, source utility, pairwise interaction, and cost.

The synthetic benchmark run generates 200 tasks using seed 42, with each task assigned a budget in $\{3, 4, 5, 6\}$. The main comparison uses the 51 tasks assigned $B = 4$. Additive Value Refresh and Set-conditioned Refresh enforce the acquisition-cost constraint during selection. Always Refresh selects all sources, while Recency-based Refresh uses the budget as a limit on the number of selected sources rather than their total cost. These two baselines can therefore exceed the acquisition-cost budget. All policies are evaluated on the same task instances within each budget group; comparisons under a common cost constraint are restricted to the budget-feasible policies.

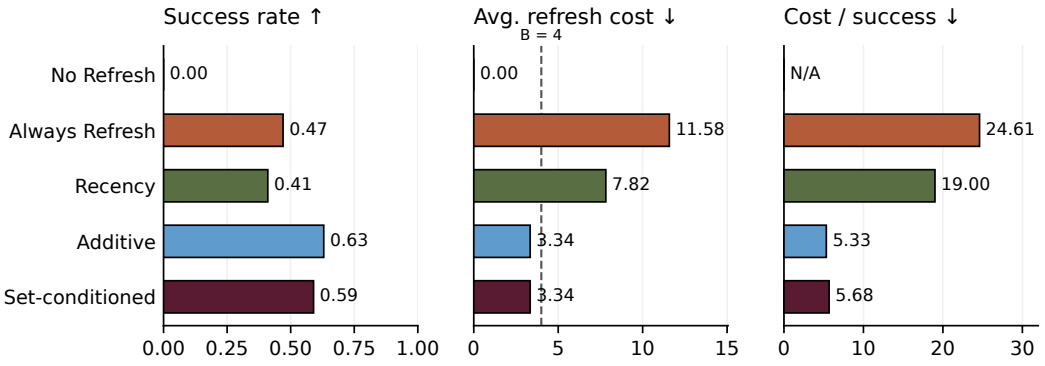

Figure 2: Synthetic benchmark results for the $B = 4$ task subset. Panels show success rate, average refresh cost, and cost per success. The dashed line marks the acquisition-cost budget enforced by Additive and Set-conditioned Refresh; Always Refresh and Recency are unconstrained in total cost. N/A denotes undefined cost per success.

We report three primary metrics: task success rate, refresh cost, and cost per successful task. For the synthetic benchmark, success is determined by a task-specific scoring rule over the selected sources, their utility, staleness, relevance, and acquisition cost. Refresh cost is the sum of the selected sources' costs, averaged over tasks; cost per success is total refresh cost divided by the number of successful tasks. These metrics characterize the simulator's success–cost trade-off rather than completion of an executed language-model action sequence.

The main synthetic comparison reports point estimates from seed 42. Always Refresh and Recency-based Refresh provide unconstrained reference points, with their full acquisition costs included in the averages. The supplementary protocol evaluates 20 additional seeds using shared task instances and paired seed-level comparisons. Appendix A.1 specifies the scoring rule, budget controls, and bootstrap procedure for these supplementary results.

We also report performance under multiple refresh budgets, stale regimes, task-dependence conditions, and longer-horizon settings. This allows us to distinguish a policy that works only under favorable conditions from a policy that is robust to realistic non-stationarity. A key protocol distinction is that the policy can only use information available to a deployed agent at decision time. We therefore separate three categories of information: (a) visible controller inputs, including task context, cached observations, age, cost, and quantities available from the observable history; (b) training labels computed from replayed continuations at shared checkpoints; and (c) hidden evaluation information, including the true latent world state and oracle outcomes used only for score computation and reporting. Source utility, staleness score, and pairwise interaction terms are treated as visible features only when they are derived from the observable history; they are never used as hidden test-time ground truth.

4.2 Main Results and Budget Trade-offs

Figure 2 reports success under the synthetic scoring rule. No Refresh selects the empty set, which receives zero utility and falls below the success threshold by construction. Additive Value Refresh achieves a success rate of 0.63, compared with 0.59 for Set-conditioned Refresh, with both average refresh costs rounding to 3.34. Thus, Additive has higher success in this task subset, while both value-based policies select sources within the same acquisition-cost budget and use similar average expenditure.

Always Refresh and Recency-based Refresh have average costs of 11.58 and 7.82, respectively, exceeding $B = 4$. These unconstrained references quantify expenditure under broad acquisition and age-based selection. Additive and Set-conditioned Refresh provide the comparison under a shared cost constraint.

Figure 3 compares the task groups assigned budgets $B \in \{3, 4, 5, 6\}$, containing 50, 51, 49, and 50 tasks, respectively. The additive baseline outperforms the set-conditioned method at $B = 3$,

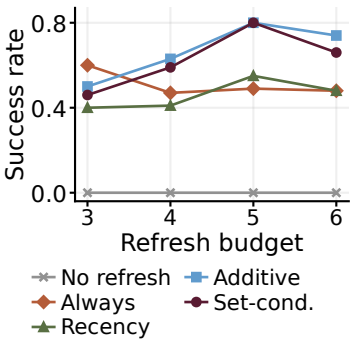

Figure 3: Success rate across refresh budgets. Additive and set-conditioned selection tie at $B = 5$.

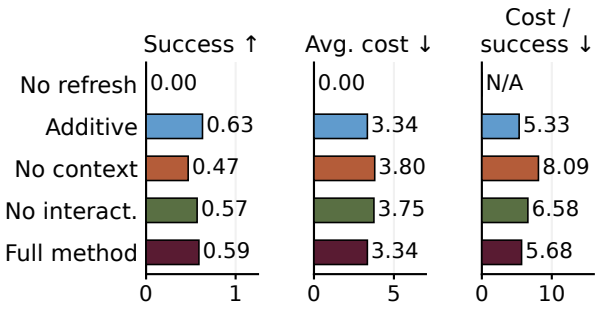

Figure 4: Reported module-ablation point estimates at $B = 4$. Additive selection has the highest reported success rate. N/A denotes undefined cost per success.

$B = 4$, and $B = 6$, while the two methods tie at $B = 5$. Because the groups contain different task instances, changes across budgets combine task-composition and policy-selection effects. We therefore interpret the plotted success rates together with realized expenditure, distinguishing the available budget from the amount actually spent on selected sources.

Across $B \in \{3, 4, 5, 6\}$, average set-conditioned costs are 2.42, 3.34, 4.37, and 5.22 (budget utilization: 80.7%, 83.6%, 87.4%, and 87.0%). At $B = 5$, it matches Additive's 0.80 success with 0.63 cost units unspent on average. Over all 200 tasks, mean cost is 3.83 and budget utilization is 85.3%. Appendices A.2 and A.3 report supplementary multi-seed budget and task-family comparisons.

4.3 Ablation and Mechanism Analysis

Task context and source interactions. Figure 4 summarizes the reported task-context and interaction ablations. Success is 0.59 for the full method, 0.47 without task context, and 0.57 without interaction terms. The full method also has a lower reported average refresh cost than either ablation, while the additive baseline remains stronger in aggregate. The descriptive differences are 12 percentage points for the task-context comparison and two points for the interaction comparison; each is reported as a point estimate.

The supplementary task-family breakdown provides a concrete comparison of observation requirements. In delivery coordination, inventory and shipping ETA form the critical source pair. Set-conditioned selection achieves 60.1% success versus 57.6% for Additive at average costs of 3.408 and 3.401, respectively. Appendix A.3 reports all four task families and paired seed-bootstrap intervals, placing this favorable setting alongside the other source requirements represented in the benchmark.

Independent, redundant, and complementary observations motivate different refresh choices. Independent sources inform distinct parts of the objective, redundant sources provide overlapping evidence, and complementary sources jointly determine an action. In the proposed formulation, the value of adding a source depends on the candidate set and the current task. This set dependence allows a critic to represent a coordinated update as one budgeted choice, with acquisition costs assessed for the complete selected set.

Counterfactual supervision. The supervision comparison contrasts paired counterfactual targets with absolute outcome fitting. Paired labels compare alternative refresh sets from a shared checkpoint under the same downstream policy, targeting their relative return-minus-cost value. Absolute outcome fitting instead predicts final-task scores directly. The paired objective expresses the training target as a within-checkpoint score difference. This matches the controller's decision rule: rank the feasible candidates for the current state and select the set with the highest predicted value.

In Figure 5, the reported selection regret is 0.87 for absolute fitting and 0.72 for paired supervision; oracle-set top-1 accuracy is 0.61 and 0.73, and task success is 0.58 and 0.72, respectively.

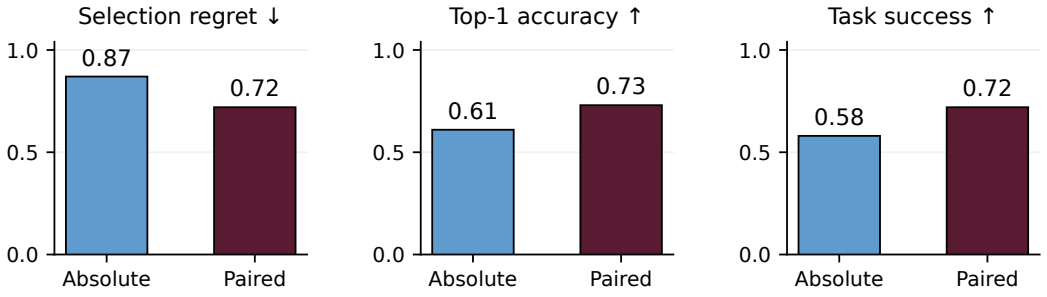

Figure 5: Absolute outcome fitting versus paired counterfactual supervision at $B = 4$. Arrows indicate the preferred direction.

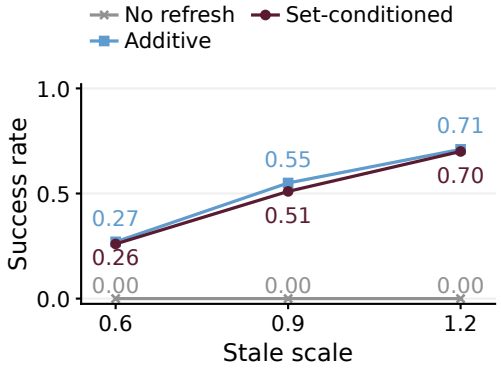

Figure 6: Success under increasing stale scales. Set-conditioned selection remains close to the additive baseline.

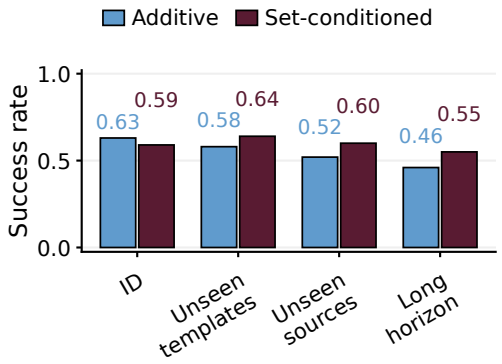

Figure 7: Generalization and long-horizon success. ID denotes in-distribution tasks; unseen sources denotes unseen source combinations.

Selection regret measures the average gap to the oracle-best feasible set under the same budget and candidate pool; top-1 accuracy measures agreement with that oracle choice. The paired variant has better reported point estimates on all three metrics. These values describe the supervision comparison separately from the main evaluation, where the full method's success is 0.59. Regret and oracle agreement characterize candidate ranking, while success characterizes the selected set's task outcome.

4.4 Robustness and Generalization

Stale observation regimes. Figure 6 compares synthetic task pools generated at stale scales 0.6, 0.9, and 1.2. Each scale uses a distinct random seed and therefore a different pool of task instances. The set-conditioned method remains slightly below the additive baseline at every reported scale. Both methods' success rates increase with stale scale. In the simulator, staleness positively scales the utility attributed to selected sources, so larger stale scales increase their potential refresh value. The plotted trend reflects this scoring rule together with differences between the generated task pools.

At stale scales 0.6, 0.9, and 1.2, set-conditioned average costs are 3.73, 3.81, and 3.82, with success rates of 0.260, 0.505, and 0.700. Cost per success is 14.35, 7.55, and 5.45, respectively. Higher success accompanies similar expenditure across these settings. Appendix A.4 reports a supplementary comparison using matched task instances.

Unseen tasks and long-horizon execution. We evaluate in-distribution tasks, unseen task templates, unseen source combinations, and longer trajectories with repeated world-state drift. Models

Table 1: Computational overhead by method. Total cost is normalized to No Refresh.

Method	Controller time	Total cost	Search overhead
No Refresh	0.00	1.00	0.00
Recency-based	0.02	1.15	0.02
Additive	0.04	1.24	0.06
Pairwise	0.07	1.35	0.10
Set-conditioned (ours)	0.09	1.42	0.16

are frozen at test time, and splits are defined over task templates and source combinations to avoid leakage from repeated world instances.

Figure 7 shows a different pattern from the main in-distribution comparison. Although the set-conditioned method trails the additive baseline in distribution, its reported success is higher on unseen templates (0.64 versus 0.58), unseen source combinations (0.60 versus 0.52), and long-horizon drift (0.55 versus 0.46). The corresponding point-estimate differences are six, eight, and nine percentage points. These settings respectively vary task templates, source combinations, and trajectory length.

4.5 Computational Cost and Discussion

Table 1 compares reported computational costs, with total cost normalized to No Refresh (1.00). Set-conditioned selection has a normalized total of 1.42, compared with 1.24 for Additive. Its workload includes constructing feasible candidates and evaluating their scores; candidate counts directly characterize this search effort.

Joint scoring adds candidate-generation and evaluation work to source-wise scoring. Appendix A.5 reports candidate counts and search agreement. In the six-source benchmark, the default four-source cap scores an average of 14.855 feasible candidates at $B = 4$ and 32.452 at $B = 6$. It reproduces full-search selections on 100% and 99.925% of the corresponding task instances under the fixed set score.

The supplementary synthetic evaluation relates selection quality to acquisition cost under a common budget. At $B = 4$, set-conditioned selection exceeds Budgeted Recency by 34.08 percentage points across 20 seeds, with average costs of 3.382 and 3.394. Additive achieves higher aggregate success, while delivery coordination favors the set-conditioned selector at similar expenditure. The budget, task-family, and candidate-cap comparisons jointly characterize resource allocation and search coverage for the fixed-score selector.

5 Conclusion

We presented task-aware observation refresh as a budgeted decision problem for tool-using agents operating with partially stale evidence. The proposed framework uses a set-conditioned critic to evaluate candidate refresh sets and counterfactual comparisons to define supervision for their relative value. It connects source selection to the current task, the cached evidence, and the acquisition budget, while keeping the downstream language-model policy fixed. The synthetic evaluation characterizes the resulting success–cost trade-offs. In the supplementary experiments, the fixed-score selector outperforms Budgeted Recency under a shared cost constraint and improves on Additive in delivery coordination, while Additive retains higher aggregate success.

Across 20 supplementary seeds, the selector respects the acquisition budget on every evaluated task. At the lowest stale scale, it attains nearly identical success to Additive with 2.6% lower average acquisition cost. The four-source candidate cap also preserves full-search selections on at least 99.925% of evaluated tasks at budgets four and six. These results identify useful operating conditions for task-aware selection and a compact search configuration for the six-source benchmark.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.11511.

Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau, and Nicolas Heess. Woulda, coulda, shoulda: Counterfactually-guided policy search, 2018. URL https://arxiv.org/abs/1811.06272.

Hanxiang Chao, Yihan Bai, Rui Sheng, Tianle Li, and Yushi Sun. STALE: Can LLM agents know when their memories are no longer valid?, 2026. URL https://arxiv.org/abs/2605.06527.

Yize Cheng, Arshia Soltani Moakhar, Chenrui Fan, Parsa Hosseini, Kazem Faghih, Zahra Sodagar, Wenxiao Wang, and Soheil Feizi. Your LLM agents are temporally blind: The misalignment between tool use decisions and human time perception. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37082–37104, 2026. doi: 10.18653/v1/2026.findings-acl.1848. URL https://aclanthology.org/2026.findings-acl.1848/.

Yicheng Feng, Yan Zhang, Yan Cheng, and Wei Qi. Scores are not decisions: Cost-aware stopping for tool acquisition in LLM agents, 2026. URL https://arxiv.org/abs/2607.27083.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=9gw03JpKK4.

Shubham Gaur and Ian Lane. Signal-driven observation for long-horizon web agents, 2026. URL https://arxiv.org/abs/2606.06708.

Xinyu Guan, Qianyang Zhao, and Yuming Deng. Decision-aware memory cards: Counterfactual-inspired context selection and compression for tool-using LLM agents, 2026. URL https://arxiv.org/abs/2606.08151.

Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966. doi: 10.1109/TSSC.1966.300074. URL https://ieeexplore.ieee.org/document/4082064/.

Shayan Kiyani, Sima Noorani, George Pappas, and Hamed Hassani. Strategic decision support for AI agents, 2026. URL https://arxiv.org/abs/2606.12587.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Yang Li and Junier Oliva. Towards cost sensitive decision making. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 3601–3609. PMLR, 2025. URL https://proceedings.mlr.press/v258/li25h.html.

Yaxing Lyu, Shengjie Zhou, Binbin Toh, Pengyu Zhu, and Lijun Li. KC-Bench: A dynamic interactive benchmark for evaluating knowledge conflicts in LLM agents, 2026. URL https://arxiv.org/abs/2609.03588.

Matheus Kunzler Maldaner, Adam Fourney, Amanda Swearngin, Hussein Mozannar, Gagan Bansal, Maya Murad, Rafah Hosn, and Saleema Amershi. SentinelBench: A benchmark for long-running monitoring agents, 2026. URL https://arxiv.org/abs/2606.05342.

Aritra Mazumder and Nusrat jahan Lia. AgentCheck: A reproduce-intervene-mitigate workbench for LLM agents over MCP, 2026. URL https://arxiv.org/abs/2607.11098.

Stephane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning, 2014. URL https://arxiv.org/abs/1406.5979.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Strick Sheng, Ziyue Wang, and Liyi Zhou. When agents overtrust environmental evidence: An extensible agentic framework for benchmarking evidence-grounding defects in LLM agents, 2026. URL https://arxiv.org/abs/2605.08828.

Saksham Sahai Srivastava. Causal intervention-based memory selection for long-horizon LLM agents, 2026. URL https://arxiv.org/abs/2605.17641.

Chung-En Sun, Linbo Liu, Ge Yan, Zimo Wang, and Tsui-Wei Weng. LLM agents already know when to call tools – even without reasoning, 2026. URL https://arxiv.org/abs/2605.09252.

Utkarsh Upadhyay, Róbert Busa-Fekete, Wojciech Kotłowski, Dávid Pál, and Balázs Szörényi. Learning to crawl. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6046–6053, 2020. doi: 10.1609/aaai.v34i04.6067. URL https://ojs.aaai.org/index.php/AAAI/article/view/6067.

Michael Valancius, Maxwell Lennon, and Junier Oliva. Acquisition conditioned oracle for non-greedy active feature acquisition. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48957–48975. PMLR, 2024. URL https://proceedings.mlr.press/v235/valancius24a.html.

Henrik von Kleist, Alireza Zamanian, Ilya Shpitser, and Narges Ahmidi. Evaluation of active feature acquisition methods for time-varying feature settings. *Journal of Machine Learning Research*, 26 (60):1–84, 2025. URL https://jmlr.org/papers/v26/23-1635.html.

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. URL https://arxiv.org/abs/2210.03629.

Bohan Yu, Lei Shen, Chenxi Zhou, Chen Han, Junlin Liu, Wenbo Su, Yu Cheng, and Bo Zheng. PReM: Learning what to preserve and when to refresh for context compression, 2026. URL https://arxiv.org/abs/2607.14327.

Mingwei Zheng, David OBrien, Siwei Cui, Pardis Pashakhanloo, Rajdeep Mukherjee, Myeongsoo Kim, and Sachit Kuhar. CORVUS: Context optimization and reduction via underlying synchronization for LLM coding agents, 2026. URL https://arxiv.org/abs/2607.22711.

Wenhao Zou, Xianglong Liu, Wendong Bi, Hanjie Wang, Simin Zhao, and Gong Zhi. CoBRA: Learning tool-use boundaries via counterfactual margins, 2026. URL https://arxiv.org/abs/2609.00967.

A Supplementary Synthetic Experiments

A.1 Protocol and Reproducibility

These experiments examine the fixed-score implementation of budgeted refresh selection in the supplied synthetic benchmark. We generate 200 tasks for each of 20 seeds, $100, \ldots, 119$, giving 4,000 task instances with 1,000 instances per task family. All policies share the same task instances. The budget sweep evaluates every instance at each $B \in \{3, 4, 5, 6\}$; task composition is therefore fixed across budgets. The complete settings are specified in `supplement/protocol.json`, and all settings in that file are reported.

Inputs and selection. Each task contains six sources: inventory, shipping ETA, supplier risk, price, warehouse temperature, and demand. The generator supplies source utility u_i, staleness d_i, relevance r_i, gain multiplier g_i, and query cost c_i. These attributes are inputs to the synthetic selector. The default set score is

$$f_{\mathrm{syn}}(S) = \sum_{i \in S} u_i d_i r_i g_i + \sum_{\{i,j\} \subseteq S} \widetilde{b}_{ij} + \mathbb{I}[S \neq \emptyset] - 0.5 \sum_{i \in S} c_i. \tag{A1}$$

The controller enumerates subsets of at most four sources and chooses the highest-scoring set with $\sum_{i \in S} c_i \leq B$, including the empty set. Pair coefficients $\widetilde{b}_{ij}$ use the original implementation's dictionary lookup on sorted source identifiers; a missing key contributes zero. All selector parameters and lookup conventions are held fixed across the sweeps.

Outcome and metrics. For the task's critical-source set K, the evaluator computes

$$R(S) = \sum_{i \in S} w_i u_i d_i r_i - 0.6 \sum_{i \in S} c_i + 2.5 \frac{|S \cap K|}{|K|}, \qquad w_i = \begin{cases} 1.4, & i \in K, \\ 0.25, & i \notin K. \end{cases} \tag{A2}$$

Success is $\mathbb{I}[R(S) > 6.5]$. Average cost measures selected-source expenditure, and cost per success divides total expenditure by the number of successful tasks. No Refresh selects the empty set, for which $R(\emptyset) = 0$. The experiments keep the original success rule and cost penalties throughout.

Controls and uncertainty. Alongside the original policies, we include *Budgeted Recency*: sources are ranked by staleness and added greedily when their cost fits the remaining budget. Budgeted Recency, Additive, and Set-conditioned selection share the same acquisition-cost constraint. Always Refresh and the original source-count Recency are unconstrained reference policies. Descriptive 95% intervals use 10,000 bootstrap resamples of the 20 seed-level averages, with bootstrap seed 8675309. Comparisons use paired seed-level differences; family-level intervals are reported without adjustment for multiple comparisons.

Table A1: Supplementary $B = 4$ comparison on 4,000 shared tasks. Intervals resample seeds. Over budget is the percentage of tasks exceeding the acquisition-cost budget. N/A denotes undefined cost per success.

Policy	Success (%; 95% CI)	Avg. cost	Cost/success	Over budget (%)
No Refresh	0.00 [0.00, 0.00]	0.000	N/A	0
Always Refresh	54.23 [53.07, 55.42]	12.032	22.19	100
Recency	52.10 [50.83, 53.23]	8.013	15.38	100
Budgeted Recency	24.78 [23.95, 25.65]	3.394	13.70	0
Additive	61.80 [60.75, 62.98]	3.394	5.49	0
Set-conditioned	58.85 [57.75, 60.00]	3.382	5.75	0

A.2 Paired Budget Sweep and Acquisition Efficiency

Figure A1 and Table A2 report the same 4,000 tasks at every budget. Set-conditioned selection exceeds Budgeted Recency in success at all four budgets, with similar average acquisition costs. At $B = 4$, the paired success difference is 34.08 percentage points (95% interval: 32.85–35.38), and cost per success is 5.75 versus 13.70. Both policies satisfy the budget on every evaluated task. This comparison connects the improvement to the selection rule under a common expenditure constraint.

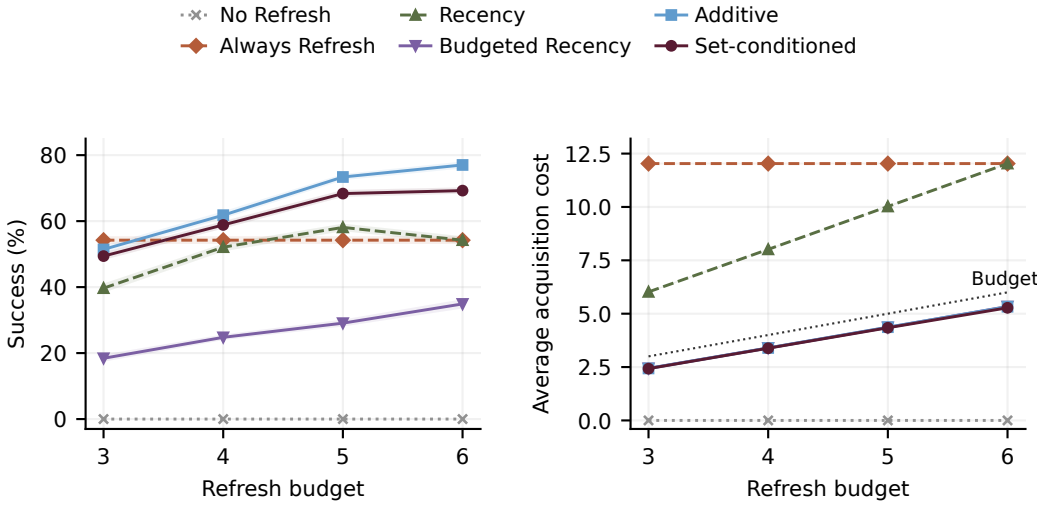

Figure A1: Paired budget sweep over 20 seeds and 4,000 tasks per budget. Shading shows descriptive 95% seed-bootstrap intervals. The dotted diagonal in the cost panel marks the acquisition budget. Always Refresh and the original Recency are unconstrained reference policies.

Table A2: Complete budget grid. Each cell gives success percentage / average acquisition cost. Every column uses the same task instances.

Policy	$B = 3$	$B = 4$	$B = 5$	$B = 6$
No Refresh	0.00 / 0.000	0.00 / 0.000	0.00 / 0.000	0.00 / 0.000
Always Refresh	54.23 / 12.032	54.23 / 12.032	54.23 / 12.032	54.23 / 12.032
Recency	39.67 / 6.021	52.10 / 8.013	58.10 / 10.022	54.23 / 12.032
Budgeted Recency	18.42 / 2.441	24.78 / 3.394	29.07 / 4.359	34.90 / 5.336
Additive	51.42 / 2.435	61.80 / 3.394	73.38 / 4.368	77.03 / 5.327
Set-conditioned	49.40 / 2.426	58.85 / 3.382	68.35 / 4.341	69.25 / 5.277

At $B = 4$, set-conditioned selection refreshes an average of 1.96 of the six sources. Its average cost of 3.382 is 71.9% below Always Refresh and 57.8% below the original Recency policy. Their success rates are 54.23% and 52.10%, compared with 58.85% for set-conditioned selection. These reference comparisons quantify the savings from selective acquisition in the simulator.

The paired budget sweep also shows how the selector uses additional capacity: its average refresh count increases from 1.43 at $B = 3$ to 2.87 at $B = 6$. Success increases from 49.40% to 69.25% across those endpoints, while mean expenditure remains below the allocated budget. Additive is included throughout as a strong cost-feasible value-based comparator; the complete grid records both policies' outcomes at every budget.

A.3 TASK-FAMILY BREAKDOWN

Table A3 decomposes the $B = 4$ comparison into all four task families. Each row contains 1,000 tasks, equally distributed across the 20 seeds. This breakdown relates refresh decisions to the task's observation requirements while keeping the budget and source-generation procedure fixed.

Table A3: Success percentages by task family at $B = 4$. The final column gives Set-conditioned minus Additive in percentage points, with a paired seed-bootstrap interval. All four generated task families are included.

Task family	Budgeted Recency	Additive	Set-conditioned	Paired Δ [95% CI]
Route planning	46.5	99.7	94.2	-5.5 [-6.6, -4.5]
Inventory allocation	11.6	34.9	30.4	-4.5 [-6.1, -3.0]
Supplier triage	17.7	55.0	50.7	-4.3 [-6.7, -1.9]
Delivery coordination	23.3	57.6	60.1	+2.5 [+1.0, +4.0]

Delivery coordination provides a favorable setting for set-conditioned selection: success is 60.1%, compared with 57.6% for Additive. The paired difference is 2.5 percentage points, with a descriptive interval of 1.0–4.0 points. Their average costs are close, at 3.408 and 3.401. This task family uses inventory and shipping ETA as its critical pair, with price also assigned substantial utility. The family-level comparison identifies a concrete operating setting in which the set-conditioned selector improves success at similar expenditure. The complete breakdown preserves the variation in outcomes across task families.

A.4 STALENESS SWEEP WITH SHARED TASK INSTANCES

We regenerate each seed at stale scales $\alpha \in \{0.6, 0.9, 1.2\}$ and fix $B = 4$. Within a seed, the task identities, source costs, utilities, relevance values, gain multipliers, and pair coefficients are identical across scales; only the generated staleness attributes change. This produces 4,000 matched instances per scale. The generator applies the scale before rounding source attributes. All six policies are evaluated, and Table A4 compares the three nontrivial policies that enforce the acquisition-cost budget; No Refresh has zero success and cost throughout.

Table A4: Matched staleness sweep at $B = 4$. Each cell gives success percentage / average acquisition cost. Costs and non-staleness source attributes are shared across scales.

Stale scale	Budgeted Recency Success / cost	Additive Success / cost	Set-conditioned Success / cost
0.6	6.33 / 3.394	24.25 / 3.394	24.30 / 3.306
0.9	19.27 / 3.394	52.62 / 3.394	50.58 / 3.373
1.2	36.35 / 3.394	71.47 / 3.394	70.35 / 3.404

At $\alpha = 0.6$, Set-conditioned and Additive achieve nearly identical success, 24.30% and 24.25%, while their average costs are 3.306 and 3.394. Set-conditioned selection reduces average expenditure by 2.6% and cost per success by 2.8% (13.60 versus 14.00). The paired cost difference is -0.088, with a 95% interval of $[-0.106, -0.070]$. Across all three scales, set-conditioned average costs lie between 3.306 and 3.404. These observations characterize a stable expenditure range alongside the change in simulated refresh value induced by the staleness multiplier.

A.5 Candidate-Set Size and Search Coverage

We vary the maximum selected-set size $k \in \{1, 2, 3, 4, 6\}$ at budgets four and six, using the same 4,000 tasks and fixed set score. With six cached sources, $k = 6$ searches all subsets. Agreement is the fraction of task instances for which a restricted search returns exactly the same source set as this full search. The feasible-set count includes the empty set and counts candidates surviving the cost constraint. Task success is evaluated separately using $R(S)$.

Table A5: Candidate-cap comparison on 4,000 tasks per budget. Feasible sets is the average number of candidate sets scored. Agreement compares selected sets with full six-source enumeration under the same score.

Budget	Cap k	Success (%)	Avg. cost	Feasible sets	Agreement (%)
4	1	47.55	1.939	7.000	12.525
4	2	59.33	3.300	14.471	91.700
4	3	58.85	3.382	14.855	100.000
4	4	58.85	3.382	14.855	100.000
4	6	58.85	3.382	14.855	100.000
6	1	47.55	1.939	7.000	0.000
6	2	74.15	3.938	22.000	23.300
6	3	68.92	5.149	31.875	89.625
6	4	69.25	5.277	32.452	99.925
6	6	69.25	5.278	32.454	100.000

The default cap of four reproduces full-search selections on all 4,000 tasks at $B = 4$ and on 3,997 of 4,000 tasks at $B = 6$ (99.925%). In both budget settings, its success outcome agrees with full search on every task. It scores an average of 14.855 cost-feasible candidates at $B = 4$ and 32.452 at $B = 6$. Before cost filtering, the four-source cap enumerates at most $\sum_{j=0}^{4} \binom{6}{j} = 57$ subsets, compared with 64 for full search. These measurements support the default cap as a compact implementation of the fixed-score selection rule in this six-source benchmark.

A.6 Reproduction Files and Validation

The supplementary directory includes a snapshot of the original generator and selector, the complete protocol, per-task selections, per-seed metrics, aggregate results, and source/output hashes. The snapshot preserves the original selection and evaluation logic; its graph-library import is deferred so the numerical benchmark can run independently of graph execution. Reproduction requires Python and NumPy; figure generation additionally uses Matplotlib.

```
python3 scripts/run_appendix_experiments.py
python3 scripts/build_appendix_assets.py
```

The runner first reproduces the existing seed-42 aggregate: set-conditioned success is 0.625 and average cost is 3.83325. It then executes the fixed supplementary grid and writes 208,000 policy-task records, covering the budget, staleness, and candidate-cap comparisons. The recorded outputs include every policy and setting specified in the protocol. The manuscript tables and vector figure are generated directly from the saved summaries. Per-task checks verify acquisition-budget feasibility for the cost-constrained selectors and confirm the candidate-cap agreement results.