

Budget-Aware Tool Use Enables Effective Agent Scaling

Tengxiao Liu^{1*}, Zifeng Wang^{2†}, Jin Miao³, I-Hung Hsu², Jun Yan², Jiefeng Chen²,
Rujun Han², Fangyuan Xu⁴, Yanfei Chen², Ke Jiang², Samira Daruki^{5‡}, Yi Liang³,
William Yang Wang¹, Tomas Pfister², Chen-Yu Lee^{2†}

¹University of California, Santa Barbara ²Google Cloud AI Research

³Google DeepMind ⁴New York University ⁵Reflection AI

tengxiao@ucsb.edu, {zifengw, chenyllee}@google.com

Abstract

Scaling test-time computation has been extended from language model reasoning to tool-augmented agents, where scaling involves not only thinking in tokens but also acting via tool calls that directly constrain environmental interaction. However, we found that simply increasing the tool-call budget fails to improve performance, as agents lack “budget awareness” and quickly hit a performance ceiling. We study how to scale such agents effectively under explicit tool-call budgets, focusing on web search agents. We first introduce the **Budget Tracker**, a lightweight plug-in that provides the agent with continuous budget awareness, enabling simple yet effective scaling. We further develop **BATS** (Budget Aware Test-time Scaling), an advanced framework that leverages this awareness to dynamically adapt its planning and verification strategy. To analyze cost-performance scaling in a controlled manner, we formalize a unified cost metric that jointly accounts for token and tool consumption. We provide the first systematic study on budget-constrained agents, showing that budget-aware methods produce more favorable scaling curves and push the cost-performance Pareto frontier. Our work offers empirical insights toward a more transparent and principled understanding of scaling in tool-augmented agents. Our code is available at <https://github.com/google-research/budget-aware-agent>.

1 Introduction

Scaling test-time compute in large language models (LLMs) helps improve the performance across a wide range of reasoning tasks (Wu et al., 2025b; Snell et al., 2024; Muennighoff et al., 2025; Chen et al., 2025b). Mainstream strategies such as sequential (Madaan et al., 2023) and parallel scaling (Wang et al., 2023; Brown et al., 2024) increase effective computation, enabling deeper reflection and output refinement with substantial gains in answer quality (Zhang et al., 2025a). These successes motivate recent efforts to extend test-time scaling to tool-augmented agents (Zhu et al., 2025b; Wang et al., 2025a), where LLMs are equipped with various tools to interact with the external environment.

Test-time scaling for tool-augmented agents expands both thinking (tokens) and acting (tool calls). While textual reasoning tasks are usually scaled by thinking with more tokens, solely increasing the internal thinking effort in agents proves to be suboptimal and does not always translate to performance scaling (Shen et al., 2025). Unlike controlling the token count in textual reasoning (Han et al., 2025b; Pu et al., 2025), in agent tasks such as web browsing, the number of tool calls directly determines the depth and breadth of exploration (Anthropic, 2025; Team et al., 2025b), defining the effective boundary of external information access.

However, simply granting agents more tool-call resources does not guarantee better performance. Our analysis reveals a critical bottleneck: *standard agents lack inherent budget*

*Work done during an internship at Google Cloud AI Research.

†Corresponding authors.

‡Work done while at Google.

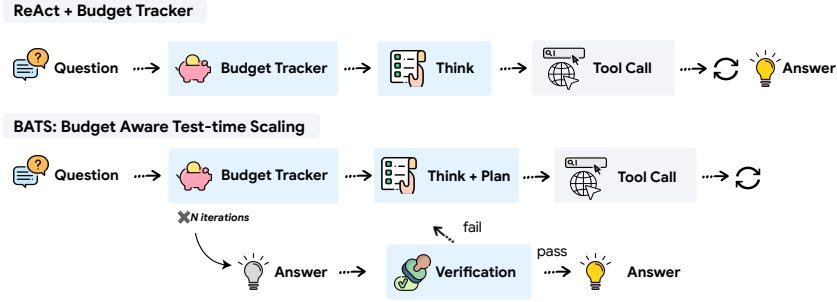


Figure 1: Budget Tracker is a lightweight plug-in that can be applied to both a standard ReAct agent (top) and more advanced orchestration frameworks like BATS (bottom). In this figure, blue boxes highlight modules that adapt to the budget.

awareness. Without explicit signals, they often perform shallow searches (Lu et al., 2025) and fail to utilize additional resources even when available. As our empirical findings later demonstrate, standard agents quickly hit a “performance ceiling”, saturating their capabilities regardless of extra budget allocated. The core challenge is not spending more, but managing diminishing and uncertain marginal gains from additional tool calls.

This unique complexity brings up a critical research question: How can tool-augmented agents scale *effectively* by making the best use of a given resource budget? We study this question in a budget-constrained setting, grounding our analysis in search agents equipped with search and browse tools, which are widely used in practice (Google, 2025; OpenAI, 2025) and inherently require extensive tool calls to collect external information. To ensure a fair and transparent comparison, we formalize a unified cost metric that jointly accounts for the economic costs of both internal token consumption and external tool interactions. This metric enables us to trace a true cost-performance scaling trend.

We first develop an intuitive, light-weight approach: **Budget Tracker** (Figure 1 (top)), which is a plug-and-play module compatible with ReAct-based agents (Yao et al., 2023). Budget Tracker provides the agent with a continuous signal of resource availability, proving to be a simple yet effective method. Through extensive experiments, we show that this simple plug-in improves performance across various budget constraints. Combining this explicit budget awareness with traditional test-time scaling strategies consistently enables more effective scaling behaviors while continuously pushing the cost-performance Pareto frontier.

To further break the scaling ceiling, we introduce **BATS** (Budget Aware Test-time Scaling) (Figure 1 (bottom)), a framework that fully internalizes resource constraints to maximize agent performance under any given budget. BATS maintains a continuous signal of remaining resources and uses it to dynamically adapt its behavior: a planning module adjusts stepwise effort to match the current budget, while a verification module decides whether to dig deeper into a promising lead or pivot to alternative paths.

Systematic experiments under varying budgets show that BATS achieves higher performance with fewer tool calls and lower overall cost than competing methods, producing more favorable scaling curves and better cost-performance trade-offs. These findings highlight the importance of explicitly accounting for cost in agent test-time scaling, enabling agents that are both effective and efficient.

We summarize our contributions as follows:

- (1) We provide the first systematic study of budget-constrained tool-use agents, formalizing agent test-time scaling with explicit tool-call budgets and introducing a unified cost metric.
- (2) We introduce **Budget Tracker**, a lightweight plug-in compatible with any agent orchestration framework that enables effective budget-aware tool use.
- (3) We develop **BATS**, a budget-aware framework that dynamically adapts planning and verification strategies based on real-time resource tracking, flexibly switching between deepening a lead and branching to alternatives.

- (4) We conduct systematic experiments under varying budgets and unified costs with search agents, demonstrating that BATS is more cost-effective and yields more favorable scaling curves and better cost–performance trade-offs.

2 Problem Formulation

2.1 Agent Test-time Scaling

We formulate agent test-time scaling as how an agent’s performance scales with its budget for external tool-call interaction, refining the broader concept of test-time interaction scaling in Shen et al. (2025). To make agent test-time scaling cost-effective, an ideal agent should be able to achieve its best possible performance under an arbitrary budget constraint on the scaling curve. To this end, our target is to design an effective and efficient agent framework, π , that maximizes answer accuracy while adhering to a strict tool-call budget.

Assume the agent is equipped with a set of K tools as $\mathcal{T} = \{t_1, \dots, t_K\}$. For question $x \in \mathcal{X}$, the agent works under a budget $\mathbf{b} = (b_1, \dots, b_K)$, where b_i is the max number of invocations of tool $t_i \in \mathcal{T}$. Let $\hat{y}_\pi(x)$ denote the agent’s predicted answer for question x with ground truth $y(x)$ and let $c_i(x; \pi)$ be the realized number of calls to tool t_i on x . We formulate the agent test-time scaling problem as a budget-constrained optimization objective:

$$\begin{aligned} \max_{\pi} \quad & \text{Acc}_{\mathbf{b}}(\pi) = \mathbb{E}_x[\mathbf{1}\{\hat{y}_\pi(x) = y(x)\}] \\ \text{s.t.} \quad & c_i(x; \pi) \leq b_i, \text{ for all } i = 1, \dots, K, \text{ and every } x \in \mathcal{X} \end{aligned} \quad (1)$$

Here the objective is the expected accuracy over all questions. The constraint ensures that for any given question, the number of realized calls for each tool never exceeds its allocated budget. By evaluating the agent performance at various budget levels, we can trace the performance-cost curve, which characterizes the agent’s test-time scaling behavior, showing how effectively it leverages budget resources to improve its problem solving capabilities.

Budget vs. Cost. We distinguish between the preset budget constraint, which specifies the maximum number of tool calls available to the agent, and the realized cost, which reflects the resources actually consumed during execution. While the budget imposes a hard upper limit, the final cost depends on the agent’s strategy. For consistent comparison, we introduce in Section 2.2 a unified post-hoc cost metric for analyzing agents’ test-time scaling.

Choice of Budget. We prioritize a tool-call budget over a token-based budget for its *relevance*: tool calls more directly constrain an agent’s ability to acquire external knowledge; *consistency*: this choice aligns with established practices (Shen et al., 2025; Team et al., 2025b); and *practicability*: pre-determining an accurate token budget for multi-step agentic tasks is often impractical. While the budget is defined by tool calls, we still incorporate token usage into our unified cost metric (Section 2.2) to ensure a more fair and transparent comparison.

2.2 Problem Instantiation with Search Agent

We instantiate test-time scaling through search agents: LLMs that reason over retrieved evidence to answer information-seeking questions x . This setting offers broad applicability and established benchmarks. Using a widely adopted ReAct-style loop (Yao et al., 2023), the agent alternates between reasoning and action using two tools:

- (1) **Search.** This tool helps perform a standard search engine query. Given a text query, it returns a list of search results, each including a title, a brief snippet, and a URL.
- (2) **Browse.** Given a specific URL, this tool scrapes the full content of the corresponding webpage, providing detailed information that is often unavailable in a search snippet.

Unified Cost Metric. We define total cost as the sum of token and tool call usage, mapped to their corresponding economic costs for a unified metric.

(1) **Token cost.** This represents the agent’s internal cognitive effort, including its reasoning, planning, and parametric knowledge processing. Token costs are calculated based on the

pricing of the model provider, distinguishing among input, output, and cache hit tokens. In multi-round iterative frameworks, the output of iteration i becomes part of the input for iteration $i + 1$. Any overlap is a cache hit, thereby lowering the token consumption cost.

(2) Tool call cost. This captures the agent’s active interaction with the environment through information-seeking actions. Each invocation (e.g., a search query or browsing request) incurs a cost determined by the corresponding API provider’s pricing.

The total unified cost, $C_{unified}(x; \pi)$, for solving a given question x under policy π is the sum of the token cost and the tool call cost. Let $c_i(x; \pi)$ be the number of actual invocations to tool t_i , and P_i be the economic cost per invocation of t_i . We define:

$$C_{unified}(x; \pi) = \underbrace{c_{token}(x; \pi)}_{\text{Token Cost}} + \underbrace{\sum_{i=1}^K c_i(x; \pi) \cdot P_i}_{\text{Total Tool Cost}} \quad (2)$$

Here, $c_{token}(x; \pi)$ represents the total cost incurred from token consumption during the reasoning process for question x . This formulation allows us to uniformly analyze the actual incurred costs. These two dimensions are inherently coupled: additional tool calls generally increase token consumption, as the agent must process and reason over the retrieved external information. Measuring this unified cost under varying budget constraints $\mathbf{b}$ enables a more comprehensive and consistent analysis across different policies.

3 Budget Awareness

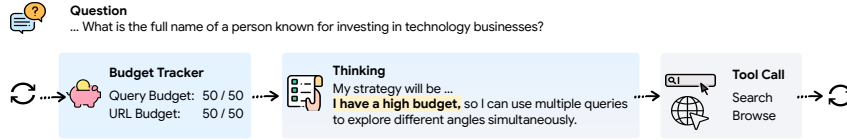


Figure 2: At each interaction round, the agent is provided with its current and remaining budget through the budget tracker before generating the next thinking and tool calls.

3.1 Budget Tracker

Each tool invocation introduces non-negligible costs in context and computation. We hypothesize that providing explicit budget signals enables the model to internalize resource constraints and adapt its strategy without additional training.

We introduce **Budget Tracker**, a lightweight, plug-and-play module that surfaces real-time budget states within the agent’s reasoning loop. At the first iteration, the tracker provides a brief policy guideline describing the budget regimes and corresponding tool-use recommendations (see Appendix D). At each subsequent iteration, it appends a budget status block showing the remaining and used budgets for each tool. Though simple, this mechanism allows the agent to condition its reasoning on the updated resource state:

```
<budget> Tooli Budget Used: ##, Tooli Budget Remaining: ##
Make the best use of the available resources. </budget>
```

3.2 Scaling with Budget Awareness

To better understand how agent behaviors scale under different budget constraints, we study two mainstream test-time scaling paradigms: *sequential scaling* and *parallel scaling*. In *sequential scaling*, we adopt the budget-forcing strategy (Muennighoff et al., 2025) by appending the message when the agent proposes an answer: “Wait, you still have remaining tool budget, use more search and browse tools to explore different information sources before concluding.” This encourages the model to reevaluate its reasoning and make better use of its available tool calls. In *parallel scaling*, we fix the per-run budget and conduct

multiple independent runs, reporting Majority Vote, Best-of-N and Pass@N results following standard test-time scaling practice (Wei et al., 2025). See Appendix C.4 for more details.

3.3 Results and Analysis

We evaluate Budget Tracker on three information-seeking benchmarks (BrowseComp (Wei et al., 2025), BrowseComp-ZH (Zhou et al., 2025) and HLE-Search (Phan et al., 2025; Han et al., 2025a); details in Section 5.1) and on conversational tasks (τ^2 -bench (Barres et al., 2025); Appendix C.2). Building on the ReAct framework, the Budget Tracker is inserted immediately after each tool response. Each tool is assigned a budget of 100, and the process stops when any budget is exhausted or the agent reaches a final answer.

Model	Method	BrowseComp	BrowseComp-ZH	HLE-Search
Gemini-2.5-Pro	ReAct	12.6 $\pm$ 1.2	31.5 $\pm$ 1.5	20.5 $\pm$ 0.5
	+ Budget Tracker	14.6$\pm$1.2	32.9$\pm$0.5	21.8$\pm$1.0
Gemini-2.5-Flash	ReAct	9.7 $\pm$ 0.5	26.5 $\pm$ 0.7	14.7 $\pm$ 1.1
	+ Budget Tracker	10.7$\pm$0.2	28.7$\pm$1.7	17.3$\pm$2.3
Claude-Sonnet-4	ReAct	12.2 $\pm$ 0.6	29.1 $\pm$ 1.7	20.5 $\pm$ 0.5
	+ Budget Tracker	14.0$\pm$0.5	31.1$\pm$1.2	23.0$\pm$0.5

Table 1: Results show consistent accuracy gains with Budget Tracker integration across different models and benchmarks. The results are averaged over 3 independent runs.

Budget Tracker enables higher performance under the same budget. Table 1 compares ReAct with and without the Budget Tracker under identical budget limits. Across different models, adding the tracker consistently improves accuracy on all the datasets. Making budget signals explicit encourages more strategic and positive agent tool-use behavior.

Method	Budget	Acc (%)	Consumed resources		
			# search	# browse	cost (€)
ReAct	10	10.3	7.87	0.60	5.2
	30	10.5	13.77	1.33	8.0
	100	12.6	14.24	1.36	9.9
+ Budget Tracker	10	12.8	8.48	1.09	6.8

Table 2: Comparison under different budgets using Gemini-2.5-Pro. With 10 $\times$ less budget (10 vs. 100), Budget Tracker (+ BT) achieves comparable accuracy while requiring fewer tool calls and lower total cost, highlighting the benefits of budget awareness.

Budget Tracker achieves lower resource usage and cost under similar accuracy. In Table 2, the consumed resources report the average number of search and browse tool calls, as well as the unified cost combining tool usage and token consumption per question. Increasing the tool-use budget in ReAct improves performance but also raises both tool call frequency and total cost. In contrast, Budget Tracker achieves comparable accuracy with less budget (10 vs. 100), while using 40.4% fewer search calls, 19.9% fewer browse calls, and reducing overall cost by 31.3%. This demonstrates that explicit budget awareness enables more efficient tool use without sacrificing accuracy.

Budget Tracker facilitates continued scaling by simply adding budget awareness. Figure 3 illustrates Gemini-2.5-Pro’s performance on

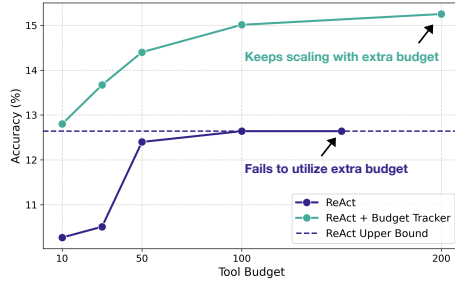


Figure 3: ReAct saturates and fails to utilize additional tool budget, reaching a performance ceiling, while Budget Tracker continues to scale effectively with larger budgets.

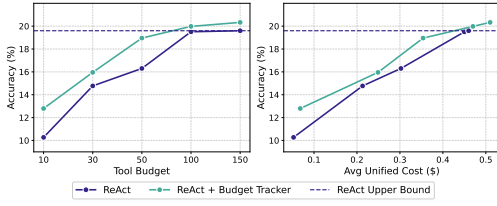


Figure 4: Sequential scaling comparison (Gemini-2.5-Pro). With explicit budget awareness, Budget Tracker overcomes early plateaus by adapting budget utilization to extend the cost-performance frontier.

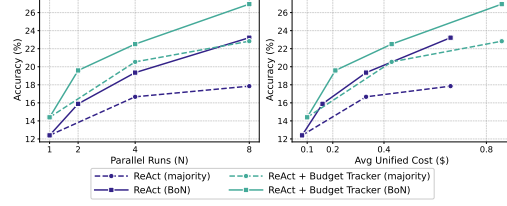


Figure 5: Parallel scaling comparison (Gemini-2.5-Pro). The left subfigure shows accuracy scaling with increasing parallel runs, while the right subfigure illustrates the corresponding cost-performance trend.

BrowseComp as tool budget increases. ReAct saturates at a budget of 100 on BrowseComp, terminating prematurely because it believes it has found a sufficient answer or is stuck, unaware of unused resources, even though the context window is not filled up. Budget Tracker breaks this ceiling by explicitly surfacing remaining resources, enabling the agent to leverage larger budgets and continue scaling.

3.4 Analysis on Test-time Scaling Strategies

Budget Tracker consistently pushes the Pareto frontier across different test-time scaling strategies. Sequential scaling prompts the agent to continue exploring until it refuses to invoke any tools for five consecutive iterations, where the agent concludes that no tool calls are necessary. In Figure 4, this strategy improves ReAct but eventually hits a ceiling. Budget Tracker sustains gains beyond this plateau and advances the cost-performance Pareto frontier, achieving both more effective scaling and more efficient tool utilization.

For parallel scaling, we set a fixed budget of 50 per run for Gemini-2.5-Pro, as empirical observations indicate that only 0.8% of the BrowseComp data requires more (see Appendix C.1). In Figure 5, we use identical sample sizes for both settings and report Majority Vote and Best-of-N results (Pass@N results in Appendix C.4). On the right, the figure illustrates the cost-performance scaling, where Budget Tracker consistently yields a superior curve.

4 BATS: Budget-Aware Test-time Scaling

Building on the effectiveness of budget awareness, we propose **BATS**, a **Budget-Aware Test-time Scaling** framework that we instantiate for search agents. As shown in Figure 6, given a question and a tool call budget, BATS begins with internal reasoning to formulate a structured action plan and decide which tools to invoke. Tool calls are triggered through special tokens (Hao et al., 2023), and the returned tool responses are appended to the reasoning sequence, expanding the context with new evidence. When the agent proposes a candidate answer, the verification module checks its validity and decides whether to continue the current sequence or initiate a new attempt with the remaining budget. The iterative process terminates once any budgeted resource is exhausted. Finally, an LLM-as-a-judge selects the best answer across all verified candidates. The complete algorithm pseudocode is in Appendix B; the full system instructions are provided in Appendix D.

4.1 Budget-Aware Planning

Planning in BATS incorporates both *constraint decomposition* and *structured dynamic planning*. Selecting a good starting point is critical: a well-chosen entry narrows the search space and conserves budget, while a poor choice quickly exhausts it (see example in Appendix F.1). To address this, the agent first decomposes question clues into two categories: (1) *exploration*, to expand the candidate space, and (2) *verification*, to validate specific properties. While a verification clue can sometimes provide a direct shortcut to the answer, relying on it prematurely is risky, as it may consume resources without guaranteeing progress.

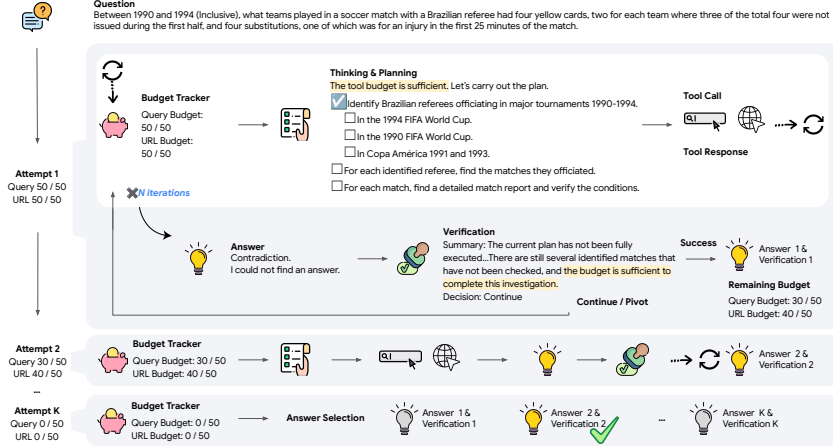


Figure 6: Overview of BATS. Given a question and per-tool budgets, the agent performs budget-aware thinking and planning via a checklist. It iterates by reasoning over new information and updated budgets. When an answer is proposed, verification decides to continue, pivot, or restart. BATS terminates when any budget is exhausted.

The agent maintains a persistent, **tree-structured plan** that acts as a dynamic checklist for resource allocation, where all completed, failed, or partial steps are retained to prevent redundant tool calls (full prompt in Appendix D.2). In Figure 6, each step represents a subtask requiring multiple tool calls. After each iteration, the module refines the tree by adding branches, resolving steps, or invalidating unproductive paths, adjusting exploration breadth and verification depth based on the remaining budget.

4.2 Budget-Aware Self-verification

Once the agent proposes an answer, the self-verification module re-evaluates the reasoning trajectory and corresponding resource usage (full prompt in Appendix D.3). It performs a constraint-by-constraint backward check: using the extracted exploration and verification clues, the module assesses each constraint as satisfied, contradicted, or unverifiable, grounding the answer directly against the question’s requirements.

Based on this analysis, the module then makes a verification decision considering budget status. If all constraints are satisfied, the answer is marked as a *SUCCESS*. If several constraints remain unverifiable but the trajectory is promising and budget permits deeper exploration, the outcome is *CONTINUE*. If contradictions are found or remaining budget cannot support further investigation, the agent terminates low-yield directions early and *PIVOTS* to avoid wasting resources while alternatives remain viable.

When the decision is to *CONTINUE* or *PIVOT*, the module generates a concise summary replacing the raw trajectory in context, including key reasoning steps, intermediate findings, failure causes, and suggestions for optimization to avoid redundant exploration. By compressing the reasoning trajectory into a compact and informative summary, the verifier reduces context length while keeping subsequent steps grounded in prior findings.

Together, constraint checking, explicit decision rules, and budget-aware summarization allow BATS to terminate low-yield trajectories early and continue promising ones efficiently.

5 Experiments and Results

5.1 Setup

Datasets. We evaluate our methods on 3 information seeking benchmarks: BrowseComp (Wei et al., 2025), a dataset of 1,266 web-browsing questions requiring persistent

retrieval; BrowseComp-ZH (Zhou et al., 2025), a 289-question Chinese benchmark designed to test agents’ performance in region-specific web environments; and HLE-Search (Han et al., 2025a), a curated 200-question subset of Humanity’s Last Exam (Phan et al., 2025) focusing on queries that explicitly require search rather than pure reasoning. We also evaluate Budget Tracker on 114 tasks from τ^2 -bench (Barres et al., 2025) retail subset which focuses on domain-specific user-agent conversations. The details are provided in Appendix C.2.

Baselines. We compare BATS against a range of models and agentic frameworks, including both general-purpose base models called directly without tool augmentation (Jaech et al., 2024; Comanici et al., 2025; Anthropic, 2025a;b) and those specifically fine-tuned for agentic search tasks (Li et al., 2025b; Liu et al., 2025a). Specifically, **SLIM** periodically summarizes the running trajectory into a concise state that is carried forward to facilitate longer horizons and more efficient scaling (Yen et al., 2025). To evaluate the final answer accuracy, we use Gemini-2.5-Flash as the judge model and adopt the evaluation prompt from Phan et al. (2025). Baseline results for HLE-Search are scraped from Han et al. (2025a). We use a temperature of 0.7 for execution and 0.0 for evaluation; implementation details and temperature analysis are in Appendix A.2 and C.5.

To analyze scaling behavior, we primarily compare BATS with parallel scaling. This ensures a direct, resource-equivalent comparison: increasing parallel sample size scales total tool consumption in the same way BATS’s iterative attempts do. By default, we use Gemini-2.5-Flash to select the most common answer via a majority vote (Wang et al., 2023) (see Appendix C.4 for details). For experiments in Section 5.2, to enforce the budget constraint for each question, we continue sampling new sequences until the budget is fully consumed. Thus the number of sampled sequences may differ across questions.

5.2 Results

Table 3 shows the performance comparison across different web search agents. Under the strict budget constraints of 100 tool uses for BrowseComp, BATS consistently achieves better results than baselines, obtaining 24.6% on BrowseComp, 46.0% on BrowseComp-ZH and 27.0% on HLE-Search using Gemini-2.5-Pro. Crucially, while many comparing agents rely on extensive task-specific training to boost performance, BATS is entirely training-free. This result highlights the effectiveness of our budget-aware design, which maximizes the efficiency of every tool call to achieve superior results in resource-constrained settings without the need for additional fine-tuning.

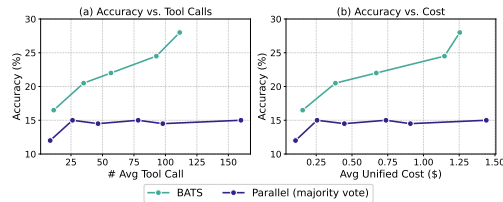


Figure 7: Scaling behaviors along (a) total number of tool calls and (b) average unified cost, evaluated on a 200-example subset of BrowseComp using Gemini-2.5-Pro.

BATS achieves higher performance under the same budget constraint. We vary the tool-call budget and evaluate performance on a random subset of 200 examples from BrowseComp for a manageable analysis. Figure 7 (left) shows the accuracy against the average number of tool calls, including both search and browse. Across all budget levels, BATS consistently outperforms the parallel majority-vote baseline, demonstrating that budget-aware adaptation leads to more effective use of limited tool calls.

BATS achieves higher performance when accounting for unified costs. Beyond tool-call counts, we measure performance under a unified cost metric that incorporates both token usage and tool-call expenses. In Figure 7 (right), BATS achieves more favorable scaling curves, delivering higher accuracy at comparable or lower costs. BATS not only improves effectiveness under budget constraints but also yields better cost–performance trade-offs.

Method	Training	BrowseComp	BrowseComp-ZH	HLE-Search
<i>Model Only (w/o tools)</i>				
Claude-3.7-Sonnet*	✗	2.3	11.8	-
Gemini-2.5-Flash	✗	2.7	23.9	2.8
Gemini-2.5-Pro	✗	6.3	27.8	8.6
<i>Training-based Agents (w/ tools)</i>				
WebSailor*	✓	12.0	30.1	-
WebExplorer*	✓	15.7	32.0	-
OpenAI Deep Research*	✓	51.5	42.9	29.1
<i>Budget-constrained (w/ tools)</i>				
Claude-Sonnet-4 ReAct	✗	12.2	29.1	20.5
Claude-Sonnet-4 SLIM	✗	11.0	25.3	15.0
Claude-Sonnet-4 BATS (Ours)	✗	19.1	41.5	29.0
Gemini-2.5-Flash ReAct	✗	9.7	26.5	14.7
Gemini-2.5-Flash SLIM	✗	10.6	26.3	14.5
Gemini-2.5-Flash BATS (Ours)	✗	14.3	34.3	19.5
Gemini-2.5-Pro ReAct	✗	12.6	31.5	20.5
Gemini-2.5-Pro SLIM	✗	13.4	31.5	14.5
Gemini-2.5-Pro BATS (Ours)	✗	24.6	46.0	27.0

Table 3: Performance comparison across web search agents. Results cited from other publications are marked with *. The “Training” column specifies whether the model has been specifically trained on agentic web search tasks. For our budget-constrained setting, each agent is provided a budget of 100 tool uses per tool.

Method	BrowseComp	BrowseComp-ZH	HLE-Search
BATS	18.7	39.1	23.0
w/o Planning	17.0	34.6	20.0
w/o Verification	15.4	37.7	22.0
w/o Planning & Verification	14.6	32.9	21.5

Table 4: Ablation results of BATS with Gemini-2.5-Pro (budget = 100 per tool). Removing different modules leads to various performance drops across datasets.

5.3 Analysis

Ablations. We present an ablation analysis of the key modules within BATS using Gemini-2.5-Pro in Table 4. Across all settings, the agent is allocated a budget of 100 tool uses per tool. To provide a clearer view of the orchestration framework, we focus on BATS with the early-stopping mechanism, where generation terminates once an answer is verified as success. The gap between these results and Table 3 demonstrates that continuing to initiate new attempts with the remaining budget after an initial verified answer leads to substantially higher accuracy.

Results indicate that removing the planning leads to a moderate performance decrease. However, removing the verification module causes a more significant drop on BrowseComp (from 18.7% to 15.4%). This suggests that the verification module is crucial for enabling the agent to navigate the solution space and accurately assess its current progress. Removing both modules results in lower performance (14.6% on BrowseComp), while BrowseComp-ZH and HLE-Search exhibit some variance, likely due to the smaller dataset size. In addition, questions in HLE-Search are typically shorter and contain fewer details that can be verified, which limits the contribution of the verification module. Overall, the trend confirms that both modules contribute positively to the orchestration framework’s effectiveness.

Additional analysis. In the Appendix, we further analyze a single-attempt early stopping setting where the agent may terminate before exhausting its budget (C.6), resource consumption breakdowns (C.7), and case studies (F). We also provide an analysis of the behavioral mechanisms underlying these improvements in Appendix E.

6 Related Work

6.1 Test-time Scaling

Test-time scaling (TTS) (Snell et al., 2024; Zhang et al., 2025a) strategies include sequential refinement (Madaan et al., 2023; Zhang et al., 2025b; Muennighoff et al., 2025; Liu et al., 2025b), parallel sampling with aggregation (Brown et al., 2024; Wang et al., 2023), and hybrid approaches (Chen et al., 2025a;b; Wan et al., 2025; Li et al., 2024). While these focus on text-only reasoning, we extend TTS to tool-augmented agents, where scaling accounts for both tokens and tool calls under budget constraints. A complementary line of work constrains the scaling effort via tokens, sampled sequences, or FLOPs (Nayab et al., 2024; Welleck et al., 2024; Damani et al., 2025; Pu et al., 2025). AgentTTS (Wang et al., 2025a) optimizes LLM size and sampling under a unified FLOPs budget, while SLIM (Yen et al., 2025) uses periodic summarization to manage context growth. In contrast, we formalize and constrain the tool-call budget, shifting the focus to budget-constrained scaling of tool-augmented agents.

6.2 Web Search Agents

Web search agents use search and browse tools to solve complex, multi-hop queries (Chen et al., 2025c; Wong et al., 2025; Team et al., 2025a; Han et al., 2025a; Li et al., 2025a). One research direction builds training data and applies various training methods to specialize the models (Jin et al., 2025; Wu et al., 2025a; Li et al., 2025b; Liu et al., 2025a; Tao et al., 2025; Ye et al., 2025; Li et al., 2025c). MiroThinker (Team et al., 2025b) shows scaling interactive tool use is an effective dimension, but it often yields redundant or inefficient tool calls. Another explores inference-time strategies (Li et al., 2025d; Zhu et al., 2025a; Qiao et al., 2025; Qin et al., 2025), such as incorporating programmatic execution to perform multiple tool call actions (Pang et al., 2025), finding an optimal, statically efficient configuration (Wang et al., 2025b), or exploring various design choices when scaling test-time compute (Zhu et al., 2025b). Instead, our work focuses on dynamic, cost-effective performance, providing the first analysis of agent scaling behavior under explicit budget constraints.

7 Conclusion

In this work, we present the first systematic study of budget-constrained tool-use agents and their test-time scaling behaviors. We introduce the **Budget Tracker**, a lightweight plug-in that provides budget awareness, and **BATS**, a framework that dynamically adapts planning and verification based on real-time resource status. Experiments show that budget awareness enables stronger scaling and pushes the cost-performance Pareto frontier. By formalizing tool-call budgets as a critical yet overlooked scaling dimension, our work offers a principled foundation for building efficient and adaptive agents.

Ethics Statement

Our work does not involve human subjects, personally identifiable information, or any data with sensitive ethical implications. Our framework design promotes more efficient compute utilization and reduces the unnecessary consumption of tokens and tool calls. This contributes to the broader goal of making large-scale AI systems more sustainable and reducing the environmental footprint of long-horizon reasoning.

Disclosure of LLM Usage

We acknowledge the use of LLMs (ChatGPT, Gemini and Claude) exclusively for editing the text to correct grammatical errors and improve clarity and flow. All core scientific content and research ideas were authored solely by the human authors.

Reproducibility Statement

To ensure reproducibility, we provide the complete algorithm pseudocode for BATS (Appendix B), full system prompts for all modules (Appendix D), and detailed implementation configurations including model settings and tool providers (Appendix A.2). All evaluation benchmarks used in this work are publicly available. We have also released our code.

References

- Anthropic. Claude 3.7 sonnet system card. <https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf>, 2025a.
- Anthropic. Claude 4 system card: Claude opus 4 & claude sonnet 4. <https://www-cdn.anthropic.com/6d8a8055020700718b0c49369f60816ba2a7c285.pdf>, 2025b.
- Anthropic. Effective context engineering for ai agents. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>, September 2025.
- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL <https://doi.org/10.48550/arXiv.2506.07982>.
- Bradley C. A. Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *CoRR*, abs/2407.21787, 2024. doi: 10.48550/ARXIV.2407.21787. URL <https://doi.org/10.48550/arXiv.2407.21787>.
- Jianhao Chen, Zishuo Xun, Bocheng Zhou, Han Qi, Qiaosheng Zhang, Yang Chen, Wei Hu, Yuzhong Qu, Wanli Ouyang, and Shuyue Hu. Do we truly need so many samples? multi-llm repeated sampling efficiently scales test-time compute. *CoRR*, abs/2504.00762, 2025a. doi: 10.48550/ARXIV.2504.00762. URL <https://doi.org/10.48550/arXiv.2504.00762>.
- Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, and Sercan Ö. Arik. SETS: leveraging self-verification and self-correction for improved test-time scaling. *CoRR*, abs/2501.19306, 2025b. doi: 10.48550/ARXIV.2501.19306. URL <https://doi.org/10.48550/arXiv.2501.19306>.
- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Hao-ran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhui Chen, and Jimmy Lin. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *CoRR*, abs/2508.06600, 2025c. doi: 10.48550/ARXIV.2508.06600. URL <https://doi.org/10.48550/arXiv.2508.06600>.
- Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, In-derjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, Krishna Haridasan, Ahmed Omran, Nikunj Saunshi, Dara Bahri, Gaurav Mishra, Eric Chu, Toby Boyd, Brad Hekman, Aaron Parisi, Chaoyi Zhang, Kornraphop Kawintiranon, Tania Bedrax-Weiss, Oliver Wang, Ya Xu, Ollie Purkiss, Uri Mendlovic, Ilai Deutel, Nam Nguyen, Adam Langley, Flip Korn, Lucia Rossazza, Alexandre Ramé, Sagar Waghmare, Helen Miller, Nathan Byrd, Ashrith Sheshan, Raia Hadsell Sangnie Bhardwaj, Pawel Janus, Tero Rissa, Dan Horgan, Sharon Silver, Ayzaan Wahid, Sergey Brin, Yves Raimond, Klemen Kloboves, Cindy Wang, Nitesh Bharadwaj Gundavarapu, Ilia Shumailov, Bo Wang, Mantas Pajarskas, Joe Heyward, Martin Nikoltchev, Maciej Kula, Hao Zhou, Zachary Garrett, Sushant Kafle, Sercan Arik, Ankita Goel, Mingyao Yang, Jiho Park, Koji Kojima, Parsa Mahmoudieh, Koray Kavukcuoglu, Grace Chen, Doug Fritz, Anton Bulyenov, Sudeshna Roy, Dimitris Paparas, Hadar Shemtov, Bo-Juen Chen, Robin Strudel, David Reitter, Aurko Roy, Andrey Vlasov, Changwan

- Ryu, Chas Lechner, Haichuan Yang, Zelda Mariet, Denis Vnukov, Tim Sohn, Amy Stuart, Wei Liang, Minmin Chen, Praynaa Rawlani, Christy Koh, JD Co-Reyes, Guangda Lai, Praseem Banzal, Dimitrios Vytiniotis, Jieru Mei, and Mu Cai. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261, 2025. doi: 10.48550/ARXIV.2507.06261. URL <https://doi.org/10.48550/arXiv.2507.06261>.
- Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=6qUUGw9bAZ>.
- Google. Gemini deep research — your personal research assistant. <https://gemini.google/overview/deep-research/>, 2025.
- Rujun Han, Yanfei Chen, Zoey CuiZhu, Lesly Miculicich, Guan Sun, Yuanjun Bi, Weiming Wen, Hui Wan, Chunfeng Wen, Solène Maître, George Lee, Vishy Tirumalashetty, Emily Xue, Zizhao Zhang, Salem Haykal, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. Deep researcher with test-time diffusion. *CoRR*, abs/2507.16075, 2025a. doi: 10.48550/ARXIV.2507.16075. URL <https://doi.org/10.48550/arXiv.2507.16075>.
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 24842–24855. Association for Computational Linguistics, 2025b. URL <https://aclanthology.org/2025.findings-acl.1274/>.
- Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/8fd1a81c882cd45f64958da6284f4a3f-Abstract-Conference.html.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O’Connell, Ian Osband, Ignasi Clavera Gilaberte, and Ilge Akkaya. Openai o1 system card. *CoRR*, abs/2412.16720, 2024. doi: 10.48550/ARXIV.2412.16720. URL <https://doi.org/10.48550/arXiv.2412.16720>.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQm66>.

- Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL <https://doi.org/10.48550/arXiv.2503.09516>.
- Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, Xixi Wu, Xuanzhong Chen, Yida Zhao, Zhen Zhang, Zhengwei Tao, Zhongwang Zhang, Zile Qiao, Chenxi Wang, Donglei Yu, Gang Fu, Haiyang Shen, Jiayin Yang, Jun Lin, Junkai Zhang, Kui Zeng, Li Yang, Hailong Yin, Maojia Song, Ming Yan, Peng Xia, Qian Xiao, Rui Min, Ruixue Ding, Runnan Fang, Shaowei Chen, Shen Huang, Shihang Wang, Shihao Cai, Weizhou Shen, Xiaobin Wang, Xin Guan, Xinyu Geng, Yingcheng Shi, Yuning Wu, Zhuo Chen, Zijian Li, and Yong Jiang. Tongyi deepresearch technical report. *CoRR*, abs/2510.24701, 2025a. doi: 10.48550/ARXIV.2510.24701. URL <https://doi.org/10.48550/arXiv.2510.24701>.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. Websailor: Navigating super-human reasoning for web agent. *CoRR*, abs/2507.02592, 2025b. doi: 10.48550/ARXIV.2507.02592. URL <https://doi.org/10.48550/arXiv.2507.02592>.
- Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic RL. *CoRR*, abs/2508.13167, 2025c. doi: 10.48550/ARXIV.2508.13167. URL <https://doi.org/10.48550/arXiv.2508.13167>.
- Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *CoRR*, abs/2501.05366, 2025d. doi: 10.48550/ARXIV.2501.05366. URL <https://doi.org/10.48550/arXiv.2501.05366>.
- Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=ndR8Ytrzh>.
- Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhui Chen, Pengyu Zhao, and Junxian He. Webexplorer: Explore and evolve for training long-horizon web agents. 2025a. URL <https://api.semanticscholar.org/CorpusID:281204359>.
- Licheng Liu, Zihan Wang, Linjie Li, Chenwei Xu, Yiping Lu, Han Liu, Avirup Sil, and Manling Li. A simple "try again" can elicit multi-turn LLM reasoning. *CoRR*, abs/2507.14295, 2025b. doi: 10.48550/ARXIV.2507.14295. URL <https://doi.org/10.48550/arXiv.2507.14295>.
- Rui Lu, Zhenyu Hou, Zihan Wang, Hanchen Zhang, Xiao Liu, Yujiang Li, Shi Feng, Jie Tang, and Yuxiao Dong. Deepdive: Advancing deep search agents with knowledge graphs and multi-turn rl, 2025. URL <https://arxiv.org/abs/2509.10446>.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegraffe, Uri Alon, Nouha Dziri, Shrimai Prabhunoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *CoRR*, abs/2303.17651, 2023. doi: 10.48550/ARXIV.2303.17651. URL <https://doi.org/10.48550/arXiv.2303.17651>.

- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *CoRR*, abs/2501.19393, 2025. doi: 10.48550/ARXIV.2501.19393. URL <https://doi.org/10.48550/arXiv.2501.19393>.
- Sania Nayab, Giulio Rossolini, Giorgio C. Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on LLM reasoning and cost. *CoRR*, abs/2407.19825, 2024. doi: 10.48550/ARXIV.2407.19825. URL <https://doi.org/10.48550/arXiv.2407.19825>.
- OpenAI. Introducing deep research. <https://openai.com/index/introducing-deep-research/>, 2025.
- Xianghe Pang, Shuo Tang, Rui Ye, Yuwen Du, Yaxin Du, and Siheng Chen. Browse-master: Towards scalable web browsing via tool-augmented programmatic agent pair. *CoRR*, abs/2508.09129, 2025. doi: 10.48550/ARXIV.2508.09129. URL <https://doi.org/10.48550/arXiv.2508.09129>.
- Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Daron Anderson, Tung Nguyen, Mobeen Mahmood, Fiona Feng, Steven Y. Feng, Haoran Zhao, Michael Yu, Varun Gangal, Chelsea Zou, Zihan Wang, Jessica P. Wang, Pawan Kumar, Oleksandr Pokutnyi, Robert Gerbicz, Serguei Popov, John-Clark Levin, Mstyslav Kazakov, Johannes Schmitt, Geoff Galgon, Alvaro Sanchez, Yongki Lee, Will Yeadon, Scott Sauers, Marc Roth, Chidozie Agu, Søren Riis, Fabian Giska, Saiteja Utpala, Zachary Giboney, Gashaw M. Goshu, Joan of Arc Xavier, Sarah-Jane Crowson, Mohinder Maheshbhai Naiya, Noah Burns, Lennart Finke, Zerui Cheng, Hyunwoo Park, Francesco Fournier-Facio, John Wydallis, Mark Nandor, Ankit Singh, Tim Gehringer, Jiaqi Cai, Ben McCarty, Darling Duclosel, Jungbae Nam, Jennifer Zampese, Ryan G. Hoerr, Aras Bacho, Gautier Abou Loume, Abdallah Galal, Hangrui Cao, Alexis C. Garretson, Damien Sileo, Qiuyu Ren, Doru Cojoc, Pavel Arkhipov, Usman Qazi, Lianghui Li, Sumeet Motwani, Christian Schröder de Witt, Edwin Taylor, Johannes Veith, Eric Singer, Taylor D. Hartman, Paolo Rissone, Jaehyeok Jin, Jack Wei Lun Shi, Chris G. Willcocks, Joshua Robinson, Aleksandar Mikov, Ameya Prabhu, Longke Tang, Xavier Alapont, Justine Leon Uro, Kevin Zhou, Emily de Oliveira Santos, Andrey Pupasov Maksimov, Edward Vendrow, Kengo Zenitani, Julien Guillod, Yuqi Li, Joshua Vendrow, Vladyslav Kuchkin, and Ng Ze-An. Humanity’s last exam. *CoRR*, abs/2501.14249, 2025. doi: 10.48550/ARXIV.2501.14249. URL <https://doi.org/10.48550/arXiv.2501.14249>.
- Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. THOUGHTTERMINATOR: benchmarking, calibrating, and mitigating overthinking in reasoning models. *CoRR*, abs/2504.13367, 2025. doi: 10.48550/ARXIV.2504.13367. URL <https://doi.org/10.48550/arXiv.2504.13367>.
- Zile Qiao, Guoxin Chen, Xuanzhong Chen, Donglei Yu, Wenbiao Yin, Xinyu Wang, Zhen Zhang, Baixuan Li, Huifeng Yin, Kuan Li, Rui Min, Minpeng Liao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webresearcher: Unleashing unbounded reasoning capability in long-horizon agents. 2025. URL <https://api.semanticscholar.org/CorpusID:281325175>.
- Tianrui Qin, Qianben Chen, Sinuo Wang, He Xing, King Zhu, He Zhu, Dingfeng Shi, Xinxin Liu, Ge Zhang, Jiaheng Liu, Yuchen Eleanor Jiang, Xitong Gao, and Wangchunshu Zhou. Flash-searcher: Fast and effective web agents via dag-based parallel execution. *CoRR*, abs/2509.25301, 2025. doi: 10.48550/ARXIV.2509.25301. URL <https://doi.org/10.48550/arXiv.2509.25301>.
- Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. Thinking vs. doing: Agents that reason by scaling test-time interaction. *CoRR*, abs/2506.07976, 2025. doi: 10.48550/ARXIV.2506.07976. URL <https://doi.org/10.48550/arXiv.2506.07976>.

- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024. doi: 10.48550/ARXIV.2408.03314. URL <https://doi.org/10.48550/arXiv.2408.03314>.
- Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingen Zhou. Webshaper: Agentically data synthesizing via information-seeking formalization. *CoRR*, abs/2507.15061, 2025. doi: 10.48550/ARXIV.2507.15061. URL <https://doi.org/10.48550/arXiv.2507.15061>.
- Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.
- MiroMind Team, Song Bai, Lidong Bing, Carson Chen, Guanzheng Chen, Yuntao Chen, Zhe Chen, Ziyi Chen, Jifeng Dai, Xuan Dong, Wenhan Dou, Yue Deng, Yunjie Fu, Junqi Ge, Chenxia Han, Tammy Huang, Zhenhang Huang, Jerry Jiao, Shilei Jiang, Tianyu Jiao, Xiaoqi Jian, Lei Lei, Ruilin Li, Ryan Luo, Tiantong Li, Xiang Lin, Ziyuan Liu, Zhiqi Li, Jie Ni, Qiang Ren, Pax Sun, Shiqian Su, Chenxin Tao, Bin Wang, Hellen Wang, Haonan Wang, James Wang, Jin Wang, Jojo Wang, Letian Wang, Shizun Wang, Weizhi Wang, Zixuan Wang, Jinfan Xu, Sen Xing, Chenyu Yang, Hai Ye, Jiaheng Yu, Yue Yu, Muyan Zhong, Tianchen Zhao, Xizhou Zhu, Yanpeng Zhou, Yifan Zhang, and Zhi Zhu. Mirothinker: Pushing the performance boundaries of open-source research agents via model, context, and interactive scaling, 2025b. URL <https://arxiv.org/abs/2511.11793>.
- UncleCode. Crawl4ai: Open-source llm friendly web crawler & scraper. <https://github.com/unclecode/crawl4ai>, 2024.
- Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 3613–3635. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.184. URL <https://doi.org/10.18653/v1/2025.naacl-long.184>.
- Fali Wang, Hui Liu, Zhenwei Dai, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zheng Li, Xianfeng Tang, Qi He, and Suhang Wang. Agenttts: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. *CoRR*, abs/2508.00890, 2025a. doi: 10.48550/ARXIV.2508.00890. URL <https://doi.org/10.48550/arXiv.2508.00890>.
- Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Efficient agents: Building effective agents while reducing cost. *CoRR*, abs/2508.02694, 2025b. doi: 10.48550/ARXIV.2508.02694. URL <https://doi.org/10.48550/arXiv.2508.02694>.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL <https://openreview.net/forum?id=1PL1NIMMrw>.
- Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *CoRR*, abs/2504.12516, 2025. doi: 10.48550/ARXIV.2504.12516. URL <https://doi.org/10.48550/arXiv.2504.12516>.

- Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilya Kulikov, and Zaïd Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL <https://openreview.net/forum?id=eskQMcIbMS>.
- Ryan Wong, Jiawei Wang, Junjie Zhao, Li Chen, Yan Gao, Long Zhang, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and Ke Wang. Widesearch: Benchmarking agentic broad info-seeking. *CoRR*, abs/2508.07999, 2025. doi: 10.48550/ARXIV.2508.07999. URL <https://doi.org/10.48550/arXiv.2508.07999>.
- Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webdancer: Towards autonomous information seeking agency. *CoRR*, abs/2505.22648, 2025a. doi: 10.48550/ARXIV.2505.22648. URL <https://doi.org/10.48550/arXiv.2505.22648>.
- Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL <https://openreview.net/forum?id=VNckp7JEHn>.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://arxiv.org/abs/2405.15793>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL <https://doi.org/10.48550/arXiv.2406.12045>.
- Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. Agentfold: Long-horizon web agents with proactive context management. *CoRR*, abs/2510.24699, 2025. doi: 10.48550/ARXIV.2510.24699. URL <https://doi.org/10.48550/arXiv.2510.24699>.
- Howard Yen, Ashwin Paranjape, Mengzhou Xia, Thejas Venkatesh, Jack Hessel, Danqi Chen, and Yuhao Zhang. Lost in the maze: Overcoming context limitations in long-horizon agentic search. *CoRR*, abs/2510.18939, 2025. doi: 10.48550/ARXIV.2510.18939. URL <https://doi.org/10.48550/arXiv.2510.18939>.
- Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. What, how, where, and how well? A survey on test-time scaling in large language models. *CoRR*, abs/2503.24235, 2025a. doi: 10.48550/ARXIV.2503.24235. URL <https://doi.org/10.48550/arXiv.2503.24235>.
- Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew C. Yao. Cumulative reasoning with large language models. *Trans. Mach. Learn. Res.*, 2025, 2025b. URL <https://openreview.net/forum?id=grW15p4eq2>.
- Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, Yuxin Gu, Sixin Hong, Jing Ren, Jian Chen, Chao Liu, and Yining Hua. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *CoRR*, abs/2504.19314, 2025. doi: 10.48550/ARXIV.2504.19314. URL <https://doi.org/10.48550/arXiv.2504.19314>.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Yi Yao, Hanhao Li, Ningning Wang, Pai Liu, Tianhao Peng, Xin Gui, Xiaowan Li, Yuhui Liu, Yuchen Eleanor Jiang, Jun Wang, Changwang Zhang, Xiangru Tang, Ge Zhang, Jian Yang, Minghao Liu, Xitong Gao, Jiaheng Liu, and Wangchunshu Zhou. Oagents: An empirical study of building effective agents. *CoRR*, abs/2506.15741, 2025a. doi: 10.48550/ARXIV.2506.15741. URL <https://doi.org/10.48550/arXiv.2506.15741>.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for LLM agents. *CoRR*, abs/2506.12928, 2025b. doi: 10.48550/ARXIV.2506.12928. URL <https://doi.org/10.48550/arXiv.2506.12928>.

A Experiment Details

A.1 Resource Cost

The cost of tool calls is determined by the pricing of the providers. To standardize billing, we’ve established a unified rate of \$0.001 per invocation for both search API calls and web browsing actions. Because different URLs produce varying amounts of text and tokens, this per-call rate is derived as an average computed from post hoc statistics over all our experiments. The consumption of tokens is billed separately, adhering to the official pricing models of the API provider pricing¹.

A.2 Implementation Details

We use Gemini-2.5-Flash, Gemini-2.5-Pro (Comanici et al., 2025) and Claude-Sonnet-4 (Anthropic, 2025b) as the backbone models in our framework. By default, we disable the thinking mode by setting the thinking budget as 0 for Gemini-2.5-Flash and 1024 for Gemini-2.5-Pro models. The maximum number of new tokens for generation was set to 65,536 for Gemini models and 64,000 for Claude. We use a temperature of 0.7 during agent execution to encourage exploration, and use a deterministic temperature of 0.0 for final answer selection and answer evaluation. We use the Google Custom Search JSON API² and SERPER³ for search tools, Jina.ai⁴ and Crawl4AI (UncleCode, 2024) for web browsing. When reproducing SLIM (Yen et al., 2025), we set the maximum number of iterations to 100, the maximum token count to 64K, and the summary interval to 50 following the default settings.

To keep the context length manageable, we employ several simple context management strategies. For each browse tool call, the fetched webpage is truncated to 150,000 characters before being sent to the model. At every iteration, we discard tool outputs from previous steps and retain only the most recent tool response, preventing the context from growing with accumulated tool results. In BATS’s verification module, we further control context size by periodically replacing the historical trajectory with summary. Since the agent determines when to activate the verification module, we perform a check during each invocation: if more than K iterations have passed since the last update (with $K = 10$ in our experiments), we replace the older reasoning trace with a concise summary derived from the verification outputs.

B Algorithm

Algorithm 1 provides the complete pseudocode for BATS. Given a question x and per-tool budgets $\mathbf{b} = (b_1, \dots, b_K)$, the agent iteratively reasons, plans, and invokes tools while the

¹<https://cloud.google.com/vertex-ai/generative-ai/pricing>

²<https://developers.google.com/custom-search/v1/overview>

³<https://serper.dev/>

⁴<https://jina.ai/>

Budget Tracker maintains real-time resource states. When a candidate answer is proposed, the self-verification module evaluates it and decides to *CONTINUE*, *PIVOT*, or declare *SUCCESS*. Upon pivoting, the trajectory is compressed into a summary and a new attempt begins with the remaining budget. The process terminates when any budget is exhausted, and a final judge selects the best answer among all verified candidates.

Algorithm 1 BATS: Budget-Aware Test-time Scaling

Require: Question x , tool set $\mathcal{T} = \{t_1, \dots, t_K\}$, per-tool budgets $\mathbf{b} = (b_1, \dots, b_K)$
Ensure: Final answer $\hat{y}$

```

1:  $\mathcal{A} \leftarrow \emptyset$  ▷ Verified answer set
2:  $\mathbf{u} \leftarrow (0, \dots, 0)$  ▷ Per-tool usage counters
3:  $\mathcal{S} \leftarrow \emptyset$  ▷ Trajectory summaries from previous attempts
4:  $\hat{y}_{\text{last}} \leftarrow \text{None}$  ▷ Most recent candidate answer
5: Decompose  $x$  into exploration and verification constraints
6: while  $u_i < b_i$  for all  $i = 1, \dots, K$  do ▷ Budget not exhausted
7:   // Budget-Aware Planning & Reasoning
8:   Inject budget status  $(\mathbf{b} - \mathbf{u})$  via Budget Tracker
9:   Generate reasoning conditioned on  $x, \mathcal{P}, \mathcal{S}$ , and budget status
10:  Update plan  $\mathcal{P}$ : add branches, resolve or invalidate steps
11:  if agent decides to invoke tool  $t_i$  with arguments  $a$  then
12:     $r \leftarrow t_i(a)$  ▷ Execute tool call
13:     $u_i \leftarrow u_i + 1$ 
14:    Append response  $r$  to reasoning context
15:  else if agent proposes candidate answer  $\hat{y}_c$  then
16:     $\hat{y}_{\text{last}} \leftarrow \hat{y}_c$ 
17:    // Budget-Aware Self-Verification
18:     $d \leftarrow \text{VERIFY}(\hat{y}_c, x, \text{trajectory}, \mathbf{b} - \mathbf{u})$ 
19:    if  $d = \text{SUCCESS}$  then
20:       $\mathcal{A} \leftarrow \mathcal{A} \cup \{\hat{y}_c\}$ 
21:      Summarize trajectory  $\rightarrow s$ ;  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$ 
22:      Reset reasoning context; start new attempt
23:    else if  $d = \text{CONTINUE}$  then
24:      Summarize trajectory  $\rightarrow s$  (if context exceeds threshold)
25:      Continue current reasoning path
26:    else if  $d = \text{PIVOT}$  then
27:      Summarize trajectory  $\rightarrow s$ ;  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$ 
28:      Reset reasoning context; abandon current direction in  $\mathcal{P}$ 
29:    end if
30:  end if
31: end while
32: if  $\mathcal{A} \neq \emptyset$  then
33:    $\hat{y} \leftarrow \text{JUDGESELECT}(\mathcal{A})$  ▷ LLM-as-judge picks best answer
34: else
35:    $\hat{y} \leftarrow \hat{y}_{\text{last}}$  ▷ Fallback to last candidate
36: end if
37: return  $\hat{y}$ 

```

C Additional Analysis

C.1 Tool Usage Breakdown

In Table 5, we examine how the ReAct baseline utilizes tools under different budget constraints on the BrowseComp dataset. As expected, increasing the allowed tool budget enables the agent to interact more with the environment, leading to improved accuracy. The Over-Budget% column shows the proportion of examples that exhaust the allocated tool budget. For Gemini-2.5-Pro, only 0.8% of the data requires more than 50 tool calls, whereas Gemini-2.5-Flash shows a higher reliance on tool usage, with 2.6% of the queries exceeding

Model	Budget	Acc %	Query		URL	
			Avg. #	Over-Budget %	Avg. #	Over-Budget %
Gemini-2.5-Pro	30	10.5	13.77	9.24	1.33	0.00
	50	12.4	13.95	0.79	1.31	0.00
	100	12.6	14.24	0.00	1.36	0.00
Gemini-2.5-Flash	30	9.0	22.77	42.81	1.04	0.00
	50	9.7	28.88	14.22	1.24	0.00
	100	10.0	29.93	2.60	1.36	0.00

Table 5: Resource usage statistics of the ReAct baseline using Gemini-2.5-Pro and Gemini-2.5-Flash on the BrowseComp dataset. Over-Budget % denotes the percentage of data that reaches the given tool budget limit.

a budget of 100. Notably, Flash models issue roughly twice as many search queries as Pro models. This suggests that the stronger parametric knowledge of Gemini-2.5-Pro allows it to navigate the search space more efficiently, reducing reliance on externally gathered evidence and enabling it to solve queries with fewer tool calls.

C.2 Generalization to Conversational Agents

To verify the generalizability of budget awareness beyond search agents, we extend our experiments to general conversational tasks via τ^2 -bench (Barres et al., 2025). Unlike search agents, agents in this benchmark must engage in multi-turn dialogues with users and invoke various tools to resolve complex, real-world tasks.

Experiment Details. A key advantage of our Budget Tracker plugin is its high modularity and ease of integration into existing open-source frameworks. We implement all experiments in this section by plugging the tracker into the official τ^2 -bench public codebase⁵ with minimal modifications. For the experimental setup, we utilize Gemini-2.5-Flash (32K context window) as the backbone for both the agent and the user simulator, with temperatures set to 0.7 and 0.0, respectively. To reflect realistic resource constraints, we apply a unified global budget rather than per-tool limits. We set total tool-call limits of 10, 30, and 50, allowing the agent to flexibly allocate its available resources across each task execution. We use the Retail subset which contains 114 data in total.

Metrics. We use the metrics from the official codebase and report Avg@ k and Pass@ k scores (Yao et al., 2024; Barres et al., 2025). Pass@ k reflects the rate at which a task succeeds in all k independent attempts, averaged across the task set. This metric highlights reliability and consistency, which are important for conversational customer service tasks in this dataset.

Results. Table 6 demonstrates the effectiveness of budget awareness across two key dimensions. First, **under the same budget constraints, Budget Tracker consistently delivers better performance.** At a budget of 10, it improves Avg@4 from 47.8% to 55.5% and Pass@4 from 21.9% to 27.2%. Second, **under similar or higher performance levels, Budget Tracker utilizes lower costs and fewer tool calls.** Specifically, our method at a budget of 10 already surpasses the Baseline’s Pass@4 performance at a budget of 50 (27.2% vs. 26.3%), while requiring approximately 18.4% fewer tool calls (5.38 vs. 6.59) and lower total cost. This indicates that budget-aware agents achieve higher reliability through strategic resource management rather than simply relying on increased tool call invocations.

C.3 Generalization to Coding Agents

To further evaluate whether budget awareness generalizes beyond search and conversational settings, we conduct an additional experiment on all 500 tasks in SWE-bench Verified (Jimenez et al., 2024) using Gemini-3-Flash-Preview within the mini-swe-agent

⁵<https://github.com/sierra-research/tau2-bench>

Method	Budget	Performance (%)		Consumed resources	
		Avg@4	Pass^4	# tool	cost (¢)
Baseline	10	47.8	21.9	6.29	2.64
	30	59.0	26.3	6.59	2.89
	50	59.0	26.3	6.59	2.89
+ Budget Tracker	10	55.5	27.2	5.38	2.72
	30	61.4	28.9	6.66	3.00
	50	63.2	36.0	6.82	3.04

Table 6: Generalizability evaluation on conversational tasks via τ^2 -bench Retail subset using Gemini-2.5-Flash. Under unified global budgets (10, 30, 50 tool-call limits), the modular Budget Tracker consistently enhances reliability (Pass^4) and average performance (Avg@4) compared to the baseline.

framework (Yang et al., 2024). Coding agents in this setting must inspect and modify software repositories through iterative Bash tool use, providing a substantially different tool-use environment from the web search tasks studied in the main paper.

Experiment Details. We integrate Budget Tracker into mini-swe-agent and use the same budget-awareness prompt as in our main experiments, with only minor wording changes that adapt the instructions from search and browse tools to Bash tool use. We impose a global tool-call budget for each task and compare the standard baseline against the baseline augmented with Budget Tracker.

Metrics. We report the resolve rate on SWE-bench Verified, the mean number of tool calls consumed per task, and the mean token cost in cents. Table 7 summarizes the results.

Method	Budget	Performance (%)	Consumed resources	
		Resolve Rate	# tool	cost (¢)
Baseline	50	36.8	46.0	18.50
	60	53.2	50.7	20.40
	70	63.0	53.6	21.59
+ Budget Tracker	50	55.4	45.7	18.88
	70	70.0	55.1	23.31

Table 7: Generalizability evaluation on all 500 SWE-bench Verified tasks using Gemini-3-Flash-Preview within the mini-swe-agent framework. Budget Tracker improves resolve rate while maintaining comparable tool usage and token cost.

Results. Table 7 shows that the benefits of budget awareness extend to software engineering agents. First, **under the same budget constraints, Budget Tracker substantially improves task resolution at comparable resource usage.** At a budget of 50, Budget Tracker increases the resolve rate by 18.6 percentage points (55.4% vs. 36.8%) while using a similar number of tool calls (45.7 vs. 46.0) and incurring a similar token cost (18.88¢ vs. 18.50¢).

Second, **Budget Tracker produces a clear Pareto improvement over a higher-budget baseline.** Budget Tracker with a budget of 50 outperforms the baseline with a budget of 60 on every measured axis: it uses 9.9% fewer tool calls (45.7 vs. 50.7), reduces token cost by 7.5% (18.88¢ vs. 20.40¢), and improves resolve rate by 2.2 percentage points (55.4% vs. 53.2%). These results reinforce our main conclusion: across web search, conversational, and coding tasks, explicit budget awareness helps agents make more effective use of available tool calls across diverse tool-use domains.

C.4 Parallel Scaling Details

For Majority Vote, we aggregate the final answers across different samples and use a judge model (Gemini-2.5-Flash) to identify the consensus answer. For Best-of-N, we provide all response trajectories to the judge model and select the most promising one. For Pass@N, the score is calculated as 1 if any of the sampled responses contain the correct answer. The prompts used for these evaluations are provided in Appendix D.4. In Figure 8, we additionally report the Pass@N results for ReAct and Budget Tracker under parallel scaling settings. Budget Tracker consistently achieves higher overall accuracy across varying budgets and cost levels.

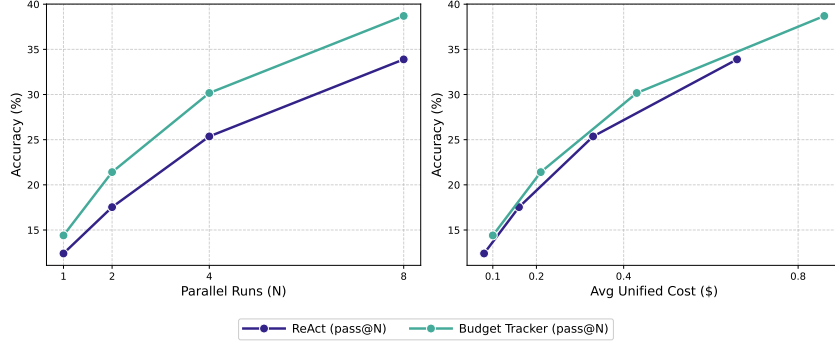


Figure 8: Pass@N results of Budget Tracker and ReAct in parallel scaling. The left subfigure shows accuracy scaling with increasing parallel runs, while the right subfigure illustrates the corresponding cost-performance trend.

C.5 Temperature Sensitivity

For model generation, we adopt a temperature of $T = 0.7$ for all tasks, which facilitates the diverse reasoning paths necessary for exploration. For final answer selection, we use a temperature of $T = 0.0$ to maintain deterministic scoring and ensure reproducibility. To validate these hyperparameter choices, we conduct a comparison using a 200-sample subset of the BrowseComp dataset with Gemini-2.5-Flash. As shown in Table 8, the model exhibits stability across various decoding configurations, with $T = 0.7$ providing the optimal balance of exploration and precision.

Temperature	0.0	0.3	0.7	1.0
Acc (%)	15.0	14.0	16.0	14.0

Table 8: Comparison of different temperature settings on model accuracy. Results are based on a 200-sample subset of BrowseComp using Gemini-2.5-Flash.

C.6 Early Stopping Analysis

We analyze how effectively agents can solve questions under budget constraints when allowed to stop early, without exhausting all tool calls. This setting reflects realistic scenarios where efficient, confident answers are preferable to prolonged exploration. To isolate this behavior, we focus on the agent’s *first attempt* only. For BATS, this is the first answer that successfully passes self-verification. For baselines that lack verification, we use the output from a single generation pass as their first attempt. If any tool budget is exhausted before this first attempt completes, the agent will stop and output a final answer based on its progress so far. This evaluation captures both efficiency and robustness, testing whether agents can solve questions quickly without being forced to use their full budgets.

Budget awareness enables BATS to scale effectively with increased resources. Figure 9 presents the early stopping performance on BrowseComp-ZH using Gemini-2.5-Pro. The

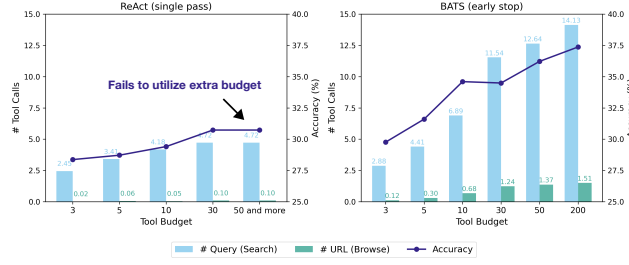


Figure 9: Performance and resource utilization with early stopping on BrowseComp-ZH using Gemini-2.5-Pro. The bars represent the average number of Search and Browse tool calls (left axis), and the line plots indicate accuracy (right axis). While ReAct plateaus and fails to utilize additional budget, BATS effectively scales up tool usage to achieve higher accuracy as the budget increases.

x-axis denotes the predefined tool call budget for both search and browse tool calls. Bars show the average number of tool calls actually used, while the line plot reports the resulting accuracy. The ReAct baseline (Figure 9 left) exhibits poor budget utilization: it consistently underuses the browse tool (fewer than 0.1 calls on average) and reaches a performance plateau early, with accuracy capped at 30.7% for all budgets of 30 and above. This behavior shows its inability to benefit from additional resources. In contrast, BATS (Figure 9 right) shows strong budget-adaptive behavior. As the budget increases, it strategically raises its use of both search and browse tools, leading to steady gains from 29.8% (budget=3) to 37.4% (budget=200). Notably, with a budget of 5, BATS already surpasses the baseline’s best achievable accuracy. This demonstrates BATS’s ability to make more strategic and cost-effective decisions, delivering higher accuracy even with fewer resources.

BATS pushes the cost-performance Pareto frontier even further by leveraging early stopping. To provide a direct and transparent comparison of cost-efficiency, Figure 10 shows accuracy against the actual unified cost. BATS demonstrates a much steeper performance curve, indicating that it achieves higher accuracy for a lower cost compared to the parallel majority vote baseline. BATS reaches over 37% accuracy for approximately \$0.23, while the parallel baseline requires more than double that cost (over \$0.50) to achieve a comparable result. This efficiency benefits from its budget-aware verification module, which enables early termination upon finding a satisfactory answer and minimizes unnecessary spending.

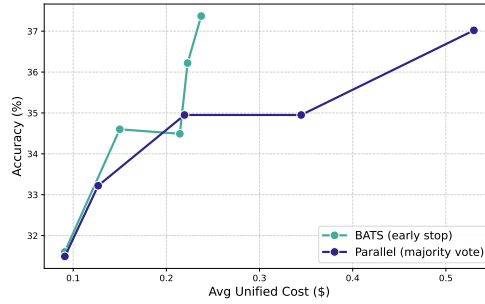


Figure 10: Average unified cost analysis on BrowseComp-ZH using Gemini-2.5-Pro.

C.7 Resource Usage Breakdown

Table 9 reports the detailed breakdown of tool calls and token consumption in BATS using Gemini-2.5-Pro on BrowseComp. We compare our approach against the variant BATS-response, which keeps the full tool responses from previous rounds. In contrast, BATS removes these responses to optimize context length. In the table, token usage is reported as a triplet: (Input / Output / Cache). The results demonstrate that removing historical tool responses does not compromise performance (maintaining similar accuracy) but significantly reduces computational overhead. This efficiency is evidenced by the substantial decrease in cache tokens and the reduction in overall cost.

	Acc (%)	# search	# browse	Total tokens	Veri. tokens	Cost (\$)
BATS	24.6	87.3	13.6	32.1 / 6.9 / 39.3	1.1 / 1.2 / 7.6	1.1
BATS-response	24.3	84.4	14.6	33.6 / 6.5 / 91.8	1.0 / 1.3 / 17.1	1.2

Table 9: Resource usage comparison. Token counts are reported as Input / Output / Cache (in 10,000). BATS achieves comparable accuracy to the full-context baseline (BATS-response) while significantly reducing cache token usage and cost.

D Prompts

Our prompt for web search agents is developed based on [Jin et al. \(2025\)](#) and [Li et al. \(2025b\)](#).

D.1 ReAct+ Budget Tracker

ReAct + Budget Tracker

```

You are an AI reasoner with Google Search and Browsing tools. Solve the question by iterating: think,
tool_code, tool_response, answer.

## Tools
You have access to 2 tools: search and browse.
{
  "name": "search",
  "description": "Performs batched web searches: supply an array 'query'; the tool retrieves the top 10
    results for each query in one call.",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {
        "type": "array",
        "items": {
          "type": "string"
        }
      },
      "description": "Array of query strings. Include multiple complementary search queries in a
        single call."
    }
  },
  "required": [
    "query"
  ]
},
{
  "name": "browse",
  "description": "Visit webpage(s) and return the summary of the content.",
  "parameters": {
    "type": "object",
    "properties": {
      "url": {
        "type": "array",
        "items": {
          "type": "string"
        }
      },
      "description": "The URL(s) of the webpage(s) to visit. Can be a single URL or an array
        of URLs."
    },
    "goal": {
      "type": "string",
      "description": "The specific information goal for browsing webpage(s)."
    }
  },
  "required": [
    "url",
    "goal"
  ]
}
}

You should start with one or more cycles of (thinking about which tool to use -> performing tool code ->
waiting for tool response), and end with (thinking about the answer -> answer of the question).
The thinking processes, tool codes, tool responses, and answer are enclosed within their tags.
There could be multiple thinking processes, tool codes, tool call parameters and tool response
parameters.

## Budget
You have two independent budgets:
- Query Budget (for search)

```

```

- URL Budget (for browse)
Each string in 'query' or 'url' consumes 1 unit respectively.

After each <tool_response>, a <budget> tag shows remaining units.
You must ADAPT your strategy dynamically to the current budget state.

### HIGH Budget (>=70% remaining)
- Search: 3-5 diverse queries in one batch.
- Browse: up to 2-3 high-value URLs.
- Goal: Broad exploration, build context fast.

### MEDIUM Budget (30%-70%)
- Search: 2-3 precise, refined queries per cycle.
- Browse: 1-2 URLs that close key knowledge gaps.
- Goal: Converge; eliminate uncertainty efficiently.

### LOW Budget (10%-30%)
- Search: 1 tightly focused query.
- Browse: at most 1 most promising URL.
- Goal: Verify a single critical fact or finalize answer.

### CRITICAL (<10% remaining or 0 in one budget)
- Avoid using the depleted tool.
- Only perform 1 minimal-cost query or browse if absolutely essential.
- If uncertainty remains and no tool use is possible, output <answer>None</answer>.

## Step syntax

<think>
Thinking process. Analyze the query, your internal knowledge, and search results to build your reasoning.
Always justify tool choices based on the remaining budgets.
</think>
<tool_code>
{"name": "tool name here", "arguments": {"parameter name here": parameter value here, "another parameter
name here": another parameter value here, ...}}
</tool_code>
<tool_response>
tool_response here
</tool_response>
<budget>
Query Budget Used: [number], Query Budget Remaining: [number], URL Budget Used: [number], URL Budget
Remaining: [number]
</budget>

Repeat <think><tool_code> until you have the final answer.
<answer> Final solution only. </answer>

## About answers

* Only write the final answer inside <answer> and </answer>.
* If you cannot find the answer, write <answer>None</answer>.

```

Figure 11: Prompts used in ReAct + Budget Tracker.

D.2 Planning Module

Budget-aware Planning Module in BATS

```

## About questions

Questions contain two types of constraints: exploration and verification.
* Exploration: Broad, core requirements (e.g., birthday, profession). Use these for initial searches to
surface candidates. You may combine 1-2 to form stronger queries.
* Verification: Narrow, specific details. Apply these only after you have candidates, to confirm or
filter them. Never begin with verification constraints.
Start with exploration queries, then use verification to validate the results.

## About planning

Maintain a tree-structured checklist of actionable steps (each may require several tool calls).
- Mark each step with its status: [ ] pending, [x] done, [!] failed, [~] partial.
- Use numbered branches (1.1, 1.2) to represent alternative paths or candidate leads.
- Log resource usage after execution: (Query=#, URL=#).
- Keep all executed steps, never delete them, retain history to avoid repeats.
- Update dynamically as you reason and gather info, adding or revising steps as needed.
- Always consider current and remaining budget when updating the plan.

```

Figure 12: Prompts used in planning module of BATS.

D.3 Self-verification Module

Budget-aware Self-verification Module in BATS

You are an AI Strategic Verifier. Your primary goal is to evaluate a proposed answer, assess the viability of the current problem-solving plan, and decide the best course of action: declare success, continue with the current plan, or pivot to a new one.

Given Inputs

- * Question: The original user question. An answer is believed to exist.
- * Trajectory: The sequence of reasoning steps and tool calls taken so far in the current attempt.
- * Current Answer: The final answer produced by the current attempt.
- * Budget Status: Information on current tool call budget utilization and remaining budget, including search queries and browsing urls.

Your Task: A 3-Step Process

You must proceed in the following order:

Step 1: Conduct Verification Analysis

First, perform a strict verification of the `Current Answer`.

- * Go through each constraint from the original Question one by one.
- * For each constraint, compare it against the `Current Answer` and the `Trajectory`.
- * State your finding for each constraint: satisfied, contradicted, or unverifiable.

Step 2: Make a Strategic Decision

Based on your verification and the budget, make one of three decisions.

1. SUCCESS: If the verification in Step 1 passed (all constraints are satisfied). The task is complete.
2. CONTINUE: If the verification failed because few constraints are unverifiable, but the overall plan is still sound and salvageable. This is the choice if **both** of these conditions are true:
 - * Promising Path: The `Trajectory` is generally sound, and the failure was due to a correctable error.
 - * Sufficient Budget: There is enough `Remaining Budget` to attempt a correction on this path.
3. PIVOT: If the verification failed, signal to abandon the current plan and switch to another one. You should pivot if any of these conditions are true:
 - * Dead End: The `Trajectory` reveals a fundamental flaw in the current plan's logic that cannot be easily fixed.
 - * Failed Tool Calls: The Trajectory shows repeated, unsuccessful attempts to find certain info.
 - * Insufficient Budget: The `Remaining Budget` is too low to make another meaningful attempt or correction within the *current* plan.

Step 3: Summarize for the Next Step

This is the most critical step for guiding future actions.

You need to first provide a **trajectory summary**: Summarize the agent's reasoning trajectory into a concise narrative. Explain its initial goal, the logical steps taken, key findings and the final conclusion, emphasizing how key findings or contradictions caused the agent to change its strategy.

Then, provide additional details tailored to your decision in Step 2.

- * If the decision is SUCCESS:
 - * No further detail needed.
- * If the decision is CONTINUE / PIVOT:
 - * Failure Analysis: Diagnose the root cause of the failure. Identify the critical flaw (e.g., poor query design, flawed logic, misinterpreted evidence) and name the general failure pattern to prevent its recurrence.
 - * Useful information: Any useful intermediate findings or results from the current `Trajectory` that could be valuable inputs for the next attempt. This prevents redundant work.
 - * Strategic Recommendations: Provide actionable advice for the agent's next attempt. Suggest strategic pivots, new angles of investigation, or different ways to combine the problem's constraints. Explicitly state if it should backtrack to and resume from a specific step in the previous plan to avoid re-doing work.

Output Requirement

Your final output must be a single JSON object with the following structure. Do not add any text before or after this JSON block.

```
```json
{
 "verification": "Verification analysis",
 "decision": "SUCCESS | CONTINUE | PIVOT",
 "justification": "A concise explanation for your strategic decision. Why is it a success, a dead end, or a correctable error?",
 "trajectory_summary": "The informative trajectory summary.",
 "details": "A JSON object containing the additional details required by Step 3. For a SUCCESS decision, this can be an empty object {}."
}
```

Figure 13: Prompts used in self-verification module of BATS.

#### D.4 Answer Selection

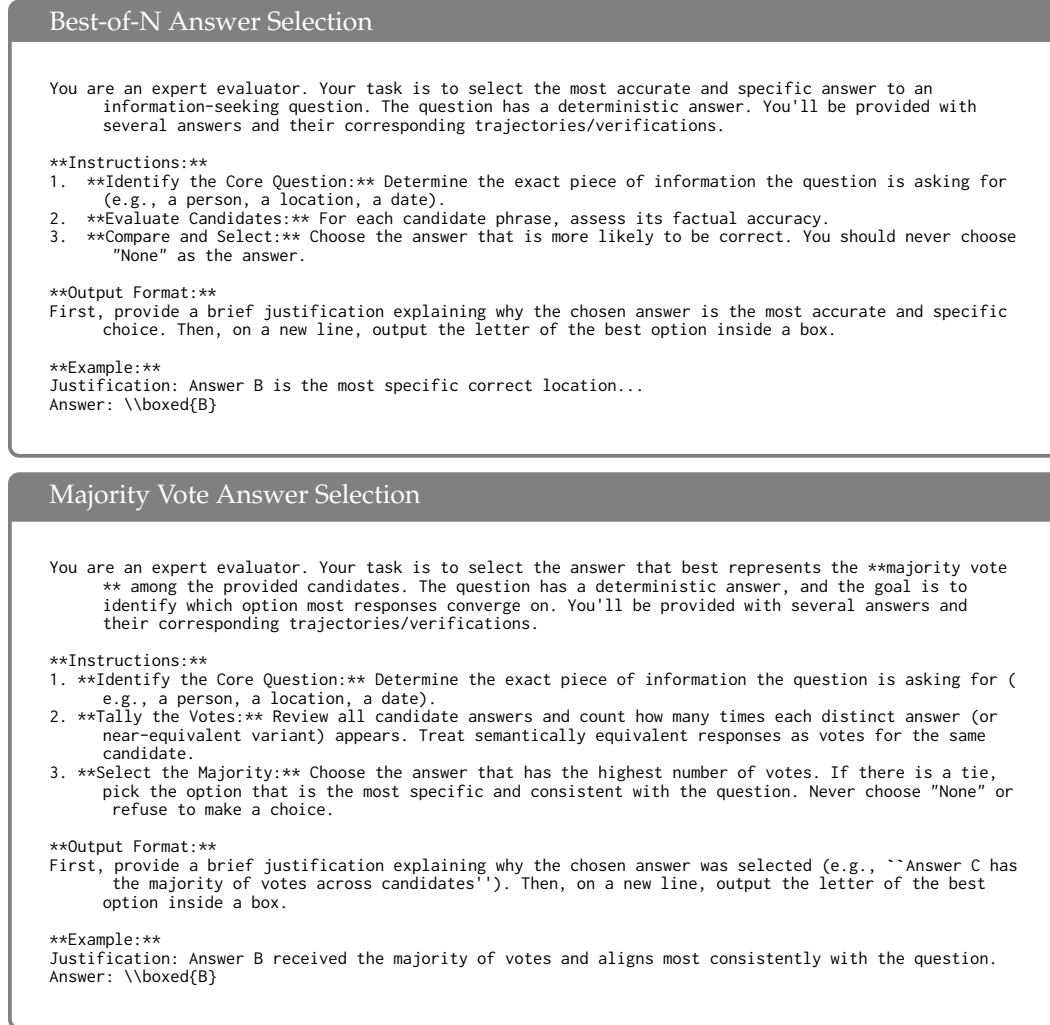


Figure 14: Prompts used in answer selection.

## E Why Does Budget Awareness Work?

The previous sections demonstrate that budget awareness improves performance; here we consolidate evidence to explain *why*. The key insight is not that budget-aware agents use more tools, but that they manage resources *more strategically*—adapting their search strategy to the available budget and making higher-quality decisions about *what* to do with each tool call.

### E.1 The Default Failure Mode: Rigid Strategy Regardless of Budget

Standard agents apply a fixed behavioral pattern irrespective of the resource budget. Table 5 shows that Gemini-2.5-Pro with ReAct uses nearly identical numbers of tool calls whether the budget is 30 or 100 (13.77 vs. 14.24 search calls), and its accuracy barely changes. The agent follows the same rigid strategy, issuing a fixed set of queries and terminating upon finding a plausible answer, without any awareness of how many resources remain or

whether deeper investigation is warranted. This rigidity, rather than the raw quantity of tool calls, is the fundamental bottleneck.

## E.2 Behavioral Shift 1: Adaptive Strategy Selection

Budget awareness enables agents to condition their search strategy on the current resource state, choosing qualitatively different approaches under different budget regimes.

The case studies in Figures 15 and 16 provide direct evidence. Under high budget, the budget-aware agent explicitly reasons about its resource abundance and selects a broad, exploratory strategy. Under low budget, the same agent shifts to a precision-oriented approach, crafting highly targeted queries and prioritizing the most promising leads. In contrast, the ReAct baseline uses the same brute-force, narrow-first strategy regardless of available resources, which happens to be suboptimal in both regimes: too narrow when resources are plentiful (missing relevant evidence), and too wasteful when resources are scarce (exhausting budget on low-yield queries).

This adaptivity is the primary driver behind Table 2’s result: Budget Tracker with budget=10 achieves comparable accuracy to ReAct with budget=100 (12.8% vs. 12.6%) while using 40.4% fewer search calls and reducing cost by 31.3%. The improvement comes not from spending more, but from spending differently.

## E.3 Behavioral Shift 2: More Balanced Tool Allocation

Beyond adapting the overall strategy, budget-aware agents allocate resources more effectively across tool types. From Table 2, we compute the browse-to-search ratio as a proxy for exploration depth:

- ReAct (Acc= 12.6%):  $1.36/14.24 = 9.6\%$
- Budget Tracker (Acc=12.8%):  $1.09/8.48 = 12.9\%$

Budget Tracker devotes a proportionally larger share of its tool calls to browsing (deep reading of full webpages) rather than surface-level search. This is a meaningful shift: browse calls retrieve detailed content that search snippets cannot provide and are critical for answering the verification-heavy questions in BrowseComp. The budget-aware agent, knowing its remaining resources, makes more deliberate decisions about when a query result warrants deeper investigation, rather than defaulting to repeated shallow searches.

## E.4 Behavioral Shift 3: Budget-Conditioned Verification in BATS

BATS’s verification module further leverages budget awareness for strategic decision making. Rather than blindly continuing or stopping, it chooses among *SUCCESS*, *CONTINUE*, and *PIVOT* based on both answer quality and remaining resources.

This budget-conditioned decision making is critical for resource efficiency. The gap between BATS’s early-stopping accuracy (18.7%, Table 4) and its full accuracy (24.6%, Table 3) shows that verification-guided retries contribute a 5.9 percentage-point gain. Importantly, these retries are not blind repetitions: as illustrated in Figure 17, the verifier diagnoses the specific failure cause and, recognizing sufficient remaining budget, redirects the agent toward a different strategy. This targeted pivoting avoids wasting resources on redundant exploration.

## E.5 Summary

Budget awareness improves performance through a clear mechanism: it shifts agents from rigid, budget-agnostic behavior to adaptive, resource-aware decision making. Specifically, agents (1) select qualitatively different strategies based on remaining budget, achieving better results with fewer resources; (2) allocate tool calls more deliberately across tool types; and (3) in BATS, make principled verification decisions that balance answer confidence

against resource availability. The consistent gains across models, benchmarks, and scaling strategies stem from this fundamental behavioral shift—not from simply using more tools, but from using each tool call more wisely.

## F Case Study

This section presents outputs from different agents. To avoid contaminating the internet with the cases shown, we have removed essential details and display only snippets of the agent trajectories.

### F.1 Adaptive Thinking and Planning

We demonstrate how Budget Tracker enables the agent to dynamically adapt its behavior based on budget awareness, choosing different strategies under high (Figure 15) or low (Figure 16) budget. These cases are collected from Gemini-2.5-Pro trajectories on BrowseComp. Without budget awareness, ReAct uses a brute-force planning strategy to exhaust direct searches. This causes it to quickly consume all available resources, failing to answer the question correctly.

**High Budget**



**Question**

What is the full name of a person known for investing in technology businesses? [Other constraints]...

**ReAct [Wrong answer]**

The most unique and searchable clue is the blog post titled "Mxx" about a trip to Rxx. **This is a very specific combination of terms. I will start by searching for this.**

**ReAct + Budget Tracker [Correct answer]**

My strategy will be to start with the most specific clue: the blog post. I will combine it with other keywords to narrow down the results. **I have a high budget, so I can use multiple queries to explore different angles simultaneously.** 

Figure 15: Adaption to high budget constraints. Being aware of the high budget, the agent chooses to expand its search queries starting from a specific keyword, whereas ReAct immediately applies narrow conditions and fails.

**Low Budget**



**Question**

In what year did the event occur that led to the loss of lives [Other constraints]...

**ReAct [Wrong answer]**

This requires a multi-pronged search. I will look for famous monuments in each city and then check their details... At this point, I **have exhausted the direct search** for a monument that fits all criteria.

**ReAct + Budget Tracker [Correct answer]**

This is too many queries. I will try to be more **efficient**. [performs search]  
**I have a low query budget, so this must be precise.** 

Figure 16: Adaption to low budget constraints. Under a low budget, ReAct fails due to an exhaustive search strategy. In contrast, budget tracker helps the agent adapt by prioritizing query efficiency and precision, successfully solving the task.

### F.2 Adaptive Verification

In Figure 17, we demonstrate how self-verification module makes budget aware decision based on the trajectory and the resource status. It decides to pivot from the current pass because the remaining budget still allows for further investigation. These cases are collected from Gemini-2.5-Pro trajectories on BrowseComp.

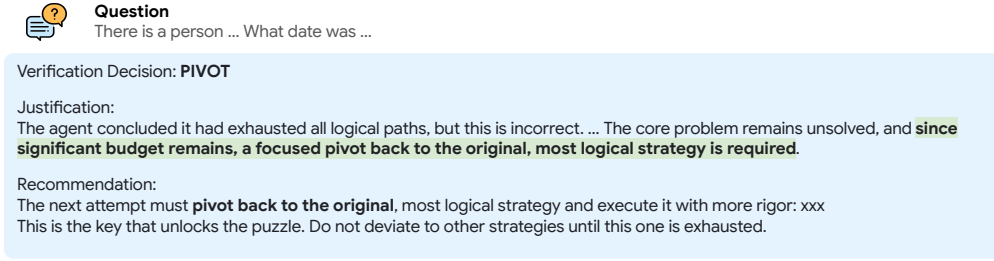


Figure 17: In BATS, the verification process dynamically adapts to budget constraints. It successfully identifies the cause of failure and provides strategic recommendations based on the current trajectory. Following this advice allows the agents to better utilize the budget and arrive at the correct answer in the next attempt.

## G Limitations

**More resource constraints.** While our study presents the first empirical analysis of tool-call budgets, a more realistic and challenging scenario involves managing multiple resource constraints jointly. Examples include token limits, inference latency, and tool-call budgets. Understanding and controlling agent behavior under multi-dimensional constraints is important for deploying scalable systems in real environments.

**Computational cost.** Running tool-augmented agents at scale incurs substantial costs from both API token consumption and tool-call fees, which limits our ability to conduct more extensive repeated experiments or hyperparameter search.

**Resource allocation.** Although we observe that budget awareness enables more effective scaling, we do not explore how an agent should allocate its available resources. Our empirical analysis suggests that models often underestimate their actual resource consumption, which can lead to suboptimal performance. Developing accurate resource estimation and principled budget allocation strategies is a promising direction for future work.

**More intelligent context management.** Although we adopt several simple context control techniques such as removing tool responses and summarizing intermediate trajectories, more advanced context engineering remains largely unexplored. Designing more effective memory formats and identifying the right balance between context length and performance are important open challenges for building robust and efficient agents.