SPEND LESS, REASON BETTER: BUDGET-ANNEALED TREE SEARCH FOR LLM AGENTS THAT SPEND ONLY ON NEW EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling lets tool-augmented LLM agents trade compute for accuracy, but much of that compute is wasted: agents re-issue tool calls they have already made, follow dead-end trajectories, and exhaust their tool and token budgets before reaching an answer. Addressing this waste requires jointly allocating the remaining budget across intermediate reasoning states and reusing evidence across branches, rather than treating budget control and evidence handling as separate decisions. We propose the Budget-Aware Value Tree (BAVT), a training-free inference framework that organizes reasoning and tool use as a tree scored by a step-level critic from the same backbone, made budget-aware by two mechanisms. *Budget-annealed selection* sharpens the sampling distribution over candidate nodes by the inverse of the remaining budget ratio, so the agent explores broadly while resources are ample and concentrates on the most promising branch as they run out. *Evidence reuse* charges the budget only for new evidence: same-parent exact repeats are merged before execution, while other eligible store hits reuse cached evidence and are re-scored in their new branch context, and every saved call is returned to the budget controller. For a capped selection variant, we prove a conditional finite-budget guarantee for generating a terminal answer. Across four multi-hop QA benchmarks, two model families and three budget tiers, BAVT outperforms parallel sampling, ReAct, LATS and BATS at every budget, using fewer paid tool calls than parallel sampling. With a five-call budget, it surpasses baselines given a $4\times$ larger budget while using about one fifth as many paid tool calls as parallel sampling; instantiated for browser agents on WebArena-Lite, it leads all baselines at every action budget, and the ordering carries over to three closed-weight models on both tasks. Allocating a fixed budget adaptively is thus more effective for reliable agents than brute-force test-time scaling.

1 INTRODUCTION

Tools have turned Large Language Models (LLMs) into agents that gather evidence and act in external environments (Yao et al., 2022; Schick et al., 2023; Deng et al., 2025; Luo et al., 2025; Jin et al., 2025), and test-time scaling is now the standard route to making them reliable: more samples, deeper search, longer loops (Snell et al., 2025; Zhu et al., 2025; Wang et al., 2022b; Yao et al., 2023; Shinn et al., 2023), up to systems that reason for hours or days (Yang et al., 2024; OpenClaw contributors, 2026; OpenAI, 2026). A deployed agent, however, runs on a finite budget. Tool calls are billed and add latency, output tokens are capped, and for search-augmented agents the retriever rather than the model dominates the per-question cost (over 90%; Table 7). Once the budget is fixed, the question is no longer how much to spend but how to spend it:

> *Given a finite inference budget, where should an agent spend it, and what should it pay for?*

Current agents answer neither question at the level where the budget is actually spent, and a fixed budget leaks in two ways:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Allocation waste: the balance between exploration and exploitation need not explicitly couple intermediate progress with the budget that remains. Parallel sampling keeps launching independent trajectories until the budget is gone, whether or not one of them is already ahead (Wang et al., 2022b); tree search allocates expansions by fixed width (Yao et al., 2023) or by UCB-style selection as in LATS (Zhou et al., 2023a); and budget-aware prompting, as in BATS (Liu et al., 2025), adapts planning and verification to the remaining budget. Unproductive paths can still consume a substantial share of resources (Cemri et al., 2025; Kim et al., 2025a), and extra compute can bring diminishing returns (Lu et al., 2025b; Liu et al., 2025). We therefore study an explicit allocation rule that combines step-level progress with the remaining budget to select among intermediate states.

Redundancy waste: the budget is charged per call rather than per piece of evidence, and search/browsing creates duplicates by construction. Siblings expanded from one promising parent share its context and can issue the same query or action, and a critic that sees only its own path rewards the repeated observation as progress. The agent thus pays repeatedly for evidence it already holds, on an instruct model, between a third and three quarters of its paid tool calls (Section 4), and a larger budget largely buys the same evidence again. What is missing is a notion of evidence equivalence.

In this paper, we introduce **Budget-Aware Value Tree (BAVT)**, a training-free framework that organizes agent inference as a tree over intermediate states, scores every step with a critic drawn from the same backbone, and makes the allocation of the remaining budget explicit through two mechanisms (Figure 1). First, to reduce allocation waste, *budget-annealed selection* determines **where to spend**: the next node is sampled with probability proportional to its value raised to $\alpha_t = 1/r_t$, where r_t is the remaining budget ratio. This keeps exploration broad when resources are ample and increasingly concentrates computation on high-value branches as the budget depletes, without a hand-designed annealing schedule. Second, to reduce redundancy waste, *evidence reuse* determines **what to pay for**: an interaction whose evidence the tree already holds is served from the tree rather than paid for again. In search-augmented QA, same-parent exact repeats are merged, while other eligible store hits reuse evidence and are re-scored, and only new evidence is charged; in browser agents the unit of evidence is the page observation (Section 4.4). The two mechanisms reinforce each other: reuse preserves the budget that drives annealing, extending exploration and redirecting saved resources toward promising branches that still require new information or actions.

For a capped selection variant, Theorem 1 gives a conditional finite-budget guarantee for generating a terminal answer. Across four multi-hop QA benchmarks and two open-weight model families, BAVT improves over the strongest matched-budget baseline by 3–14 EM points. With a five-call budget, it exceeds every baseline given a $4\times$ larger budget while using about one fifth as many paid tool calls as parallel sampling. It also leads in 17 of 18 closed-weight model–budget cells across QA and WebArena-Lite. Our contributions are summarized as follows:

- **Where to spend: a step-level value tree with budget-annealed selection.** We formulate agent test-time scaling under a tool-call budget with a token cap, score each step with a critic from the same backbone, and expand nodes by the budget-annealed rule $\alpha_t = 1/r_t$ over these values. We establish a conditional terminal-answer guarantee for a capped variant of the selection rule.
- **What to pay for: evidence reuse inside the tree.** We charge only for new evidence: a same-parent exact repeat is merged before execution and a parent that keeps repeating itself leaves the candidate pool, other eligible store hits are served from the tree-wide evidence store and re-scored, and every saved call stays in the budget that drives selection.
- **What it buys: matched-budget gains across models and environments.** BAVT outperforms all baselines at every budget on both open-weight families, leads in 17 of 18 closed-weight cells (QA and WebArena-Lite), and the full configuration uses 13–75% fewer paid tool calls than the configuration without reuse, with higher mean EM.

2 Related Work

Tool-augmented agents interleave reasoning and external actions (Yao et al., 2022; Schick et al., 2023; Nakano et al., 2021), including retrieval and browser interaction (Jin et al., 2025; Zhou et al., 2023b; Liu et al., 2024a). *Test-time scaling* improves accuracy through sampling, tree search, reflection and adaptive branching (Wang et al., 2022b; Yao et al., 2023; Zhou et al., 2023a; Shinn et al., 2023; Inoue et al., 2025), but these strategies typically do not condition selection on the remaining budget. Unlike LATS's visit-count-based UCT selection (Zhou et al., 2023a), BAVT explicitly conditions

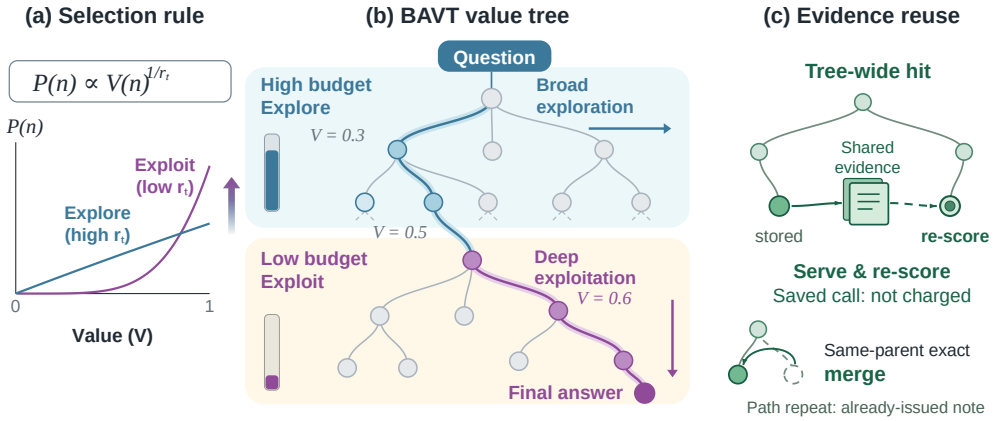

Figure 1: **Budget-Aware Value Tree (BAVT). (a)** Budget-annealed selection, $P(n) \propto V(n)^{1/r_t}$, sharpens toward higher-value nodes as the smaller remaining tool/token ratio r_t decreases (arrow). **(b)** Step-level critic values guide the tree from broad exploration to focused exploitation. **(c)** Same-parent exact repeats merge; path repeats return an already-issued note; other eligible tree-wide hits reuse stored evidence and are re-scored in the new branch, saving a paid tool call. Curves and node values are illustrative.

its node-sampling distribution on the remaining tool and token ratios. *Budget-aware inference* uses routing, cascading or trained controllers (Chen et al., 2023; Zhang et al., 2023; Han et al., 2025; Li et al., 2025); BATS instead uses budget prompts to guide planning, verification and transitions between attempts (Liu et al., 2025). *Evidence reuse* reduces repeated tool use through caching, pruning and state deduplication (Yang et al., 2026; Luo et al., 2026; Kim et al., 2025b; Wang et al., 2025a; Tian et al., 2024), while RL methods learn to make fewer tool calls (Java et al., 2026; Wang et al., 2025b; Wu et al., 2025). BAVT couples step-level budget-annealed selection with tree-wide reuse: duplicate proposals can retire exhausted parents, while near-duplicate hits retain separate nodes. Rather than treating a cache hit only as an execution shortcut, BAVT preserves the saved tool budget for subsequent selection and re-scores reused evidence in its new branch context. This connects evidence handling to budget allocation across reasoning branches without retraining the backbone. Appendix A provides detailed comparisons.

3 Methodology

3.1 Problem Formulation: Budget-Aware Agent Inference

We study test-time scaling for tool-augmented LLM agents under a hard budget constraint. Given a user question x, an agent interleaves reasoning and tool use before producing a final answer $\hat{y}$. We formalize this process as a resource-constrained decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{B}, \mathcal{C})$.

State Space ($\mathcal{S}$). Let $s_t \in \mathcal{S}$ denote the state at step t, containing the current context: the user query, prior actions, reasoning traces, and tool observations.

Action Space ($\mathcal{A}$). Let $a_t \in \mathcal{A}$ denote the action at step t, which may be an internal reasoning step, an external tool invocation, or terminal answer generation.

Transition Dynamics ($\mathcal{T}$). The environment evolves deterministically as $s_{t+1} = \mathcal{T}(s_t, a_t)$, where the new action and resulting observation are appended to the context.

Budget State Space ($\mathcal{B}$). We track the remaining tool and token budgets by

$$b_t = (b_{\text{tool},t}, b_{\text{token},t}) \in \mathcal{B} \subseteq \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}, \tag{1}$$

initialized as $b_0 = (B_{\text{tool}}, B_{\text{token}})$, where B_{tool} and B_{token} are the initial tool-call and output-token budgets.

Cost Function ($\mathcal{C}$). Each action incurs a cost $\mathcal{C}(a_t) = \big(\mathcal{C}_{\text{tool}}(a_t), \mathcal{C}_{\text{token}}(a_t)\big)$, where $\mathcal{C}_{\text{tool}}(a_t) = 1$ for every action to call the tools, including failed or malformed calls, and 0 when the action is served

from the query's evidence store or merged into a sibling (Equation (5), Section 3.2.3); $\mathcal{C}_{\text{tool}}(a_t) = 0$ for reasoning steps, and $\mathcal{C}_{\text{token}}(a_t)$ is the number of output tokens generated at step t. The remaining budget is updated as $b_{t+1} = b_t - \mathcal{C}(a_t)$, so the remaining budget ratio r_t used for node selection (Section 3.2.2) is computed from paid calls only. The two budgets capture distinct resource constraints. For example, retrieval can dominate per-question cost (Section 1), making tool calls the primary bottleneck and generation tokens a secondary cap. BAVT therefore anneals according to the smaller remaining-budget ratio, allowing the more depleted resource to determine the effective constraint.

Inference Objective. A trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T)$ is induced by the transition dynamics $\mathcal{T}$. Rather than committing to a single trajectory, test-time scaling considers a tree of possible state transitions rooted at s_0. The objective is to define an inference policy π that allocates the finite budget across competing branches to maximize the quality of the final terminal answer $\hat{y}$.

3.2 Overview of Budget-Aware Value Tree (BAVT)

BAVT incrementally builds a tree of reasoning and tool-use states (Figure 1). A step-level critic scores the progress of every new node (Section 3.2.1); budget-annealed selection chooses the node to expand from these values and the remaining budget, and decides when inference terminates (Section 3.2.2); evidence reuse serves an interaction whose evidence the tree already holds without charging it (Section 3.2.3). Sections 3.2.1–3.2.3 describe the three components and Section 3.3 the guarantee.

3.2.1 Step-Level Value Estimation

Residual Value Prediction. The critic scores the most recent step rather than the absolute quality of the state. Absolute self-evaluation by LLMs is prone to overconfidence (Xiong et al., 2023), whereas relative judgments do not require an internally consistent scale and align better with human assessment (Zheng et al., 2023; Liu et al., 2024b). Given parent value $V(n)$, the critic returns a raw-scale integer delta $\Delta_t \in \{-4, \ldots, 4\}$. The normalized child value is

$$V(n') = \text{clip}_{[0.1,1]}(V(n) + \Delta_t/10)\,. \tag{2}$$

Here n' and n denote child and parent nodes. Division by 10 converts the critic's 1–10 scale to normalized units; clipping enforces $V \in [0.1, 1]$. A step that adds no information receives $\Delta_t \leq 0$, so its value cannot exceed its parent's, guiding the expansion rules below.

Value-Guided Step Instruction. These step-level values directly dictate the topological expansion of the inference tree. Let θ be the confidence threshold for answer generation; we use $\theta = 0.7$ throughout, and Appendix E.3 shows that results are insensitive to θ between 0.7 and 0.9:

- **Answer Generation** ($V(n') \geq \theta$): The node is marked for answering; when it is next selected, the model synthesizes a final answer and the path terminates.
- **Widening** ($V(n') \leq V(n)$): The recent action yields zero or negative information gain, so the agent proposes divergent thoughts or tool calls.
- **Deepening** ($V(n) < V(n') < \theta$): The branch is promising but incomplete, so the agent continues reasoning along it.

This structural guidance ensures the agent efficiently alternates between depth-first exploitation of promising chains and breadth-first exploration of uncertain pathways.

3.2.2 Budget-Annealed Node Selection

Another core mechanism of the inference is a dynamic, budget-aware node selection strategy. The inference tree is initialized with the original question as the root node, assigned a minimum starting value. At each step, the framework samples a node to expand from the pool of existing candidate nodes, strictly excluding terminal answer nodes.

Exploration to Exploitation Shift. Standard Upper Confidence Bound (UCB) rules are designed for asymptotic exploration over an unbounded horizon and do not adapt to a depleting budget. We instead use a budget-aware stochastic sampling rule. Let $r_t \in (0, 1]$ represent the effective remaining budget ratio at step t, defined as the minimum between the remaining tool budget ratio and the remaining

token budget ratio: $r_t = \min\left(\frac{b_{\text{tool},t}}{B_{\text{tool}}}, \frac{b_{\text{token},t}}{B_{\text{token}}}\right)$. Because $b_{\text{tool},t}$ is decremented only for paid calls (Section 3.1), every tool call saved by reuse (Section 3.2.3) keeps r_t higher and the selection broader. We define a dynamic scaling exponent α_t, which is inversely proportional to this limiting budget ratio: $\alpha_t = 1/r_t$. For each candidate node $n_i \in \mathcal{N}_{\text{cand}}$ with an accumulated state value $V(n_i)$, we compute an unnormalized selection weight w_{n_i} via a power-based scaling function: $w_{n_i} = V(n_i)^{\alpha_t}$ The probability $\mathbb{P}(n_i)$ of selecting node n_i for expansion is determined by normalizing these weights across all N candidate nodes in the pool:

$$\mathbb{P}(n_i) = \frac{w_{n_i}}{\sum_{j=1}^{N} w_{n_j}}. \tag{3}$$

The sampling rule of Equation (3) induces a budget-dependent shift in the agent's behavior. When the budget is abundant ($r_t \approx 1$), $\alpha_t \approx 1$, yielding a sampling distribution roughly proportional to the raw node values and promoting exploration of the search space. As the budget decreases ($r_t \to 0$), α_t increases, magnifying value differences and concentrating probability mass on higher-valued nodes. In the limit, the distribution approaches a near-deterministic selection of the highest-valued node. Thus, the policy transitions from broad exploration in early stages to increasingly exploitative behavior as resources become scarce.

Termination and answer extraction. A node with $V \geq \theta$ is marked for answering; when selected again, it generates a terminal answer. Inference continues within the remaining budget to collect further answers. If no terminal answer exists when the tool budget is exhausted or the remaining token ratio falls below $\eta = 0.2$, the highest-valued candidate is instructed to answer from its current context. The final prediction is the majority vote over $\mathcal{N}_{\text{ans}}$, with ties broken by node value.

3.2.3 Paying Only for New Evidence: Evidence Reuse

The value tree can expand the same high-value state several times, so different branches revisit the same or similar interactions, and a critic that sees only its own path treats the repeated observation as new progress. BAVT addresses this through *evidence reuse*. We call the observation returned by an external interaction its *evidence* (the passages returned by a search; the page observation returned by a browser action): the tool budget is charged only for evidence the tree does not yet hold, and evidence already in the tree is reused at no cost.

Evidence store and interaction equivalence. BAVT maintains a tree-wide evidence store

$$\mathcal{E}_t = \{(s_j, a_j, o_j)\}_{j \leq t}, \tag{4}$$

where s_j is the state from which action a_j was executed and o_j is the evidence it returned. The store is shared across all branches of the tree. Two interactions are *identical* when their normalized actions coincide, and *equivalent*, written $(s, a) \simeq (s', a')$, when executing the new one is expected to return the same or sufficiently similar evidence; identical interactions are equivalent, and the criterion for non-identical ones is environment-dependent (Appendix E.2).

Cost under reuse. The tool cost $\mathcal{C}_{\text{tool}}$ of Section 3.1 is incurred only when the tree does not already contain an equivalent interaction:

$$\mathcal{C}_{\text{tool}}(a_t) = \mathbb{1}[\nexists\, (s_j, a_j, o_j) \in \mathcal{E}_t : (s_t, a_t) \simeq (s_j, a_j)]\,. \tag{5}$$

A newly executed interaction adds its evidence to $\mathcal{E}_t$, whereas an equivalent interaction reuses the stored evidence without consuming tool budget. Since the remaining-budget ratio r_t in Equation (3) is computed from resources actually consumed, every call saved by reuse keeps r_t higher and the selection broader. This holds for any store under paid-call accounting; what is specific to BAVT is how the two cases below are handled inside the tree.

Two reuse rules. Handling depends on both the match and its location in the tree:

1. *Same-parent and path repeats: merge or return a note.* If the proposed action is identical to an action already issued from the same parent n, the proposal is merged into the existing child without being executed, and the parent's merge count $m(n)$ is incremented; when $m(n) \geq 2$ the parent leaves $\mathcal{N}_{\text{cand}}$, since it has shown that it has nothing new to propose. If the identical interaction already occurs on the path from the root to n, the call is answered, without execution or charge, by a note that it was already issued, and the node is treated as having made a no-tool turn (two such turns make it terminal). An identical interaction found elsewhere in the tree is an exact store hit and follows Rule 2.

2. *Other store hits: reuse, then re-score.* For an exact hit elsewhere in the tree or a non-identical equivalent interaction, BAVT returns the stored evidence, creates the child in the current branch and lets the critic score it, since the same evidence may have a different utility in a different reasoning context; only the external call is free. At the smallest budget ($B_{\text{tool}} = 5$) at most two free hits of the store are granted per question so that the store does not substitute for exploration (Table 5); a further equivalent proposal is executed and charged like a new one, and merges under Rule 1 do not count toward this cap.

The asymmetry between the two rules is deliberate. An identical repeat from the same parent carries no information beyond its sibling, so the branch is not worth keeping, and the repeated proposal is what signals an exhausted parent; an eligible store hit in a different branch can still lead to a different next step, so the child is kept and re-scored while the external call is saved. A store without the merge rule would still save the call but leave the exhausted parent in the pool, and merging equivalent branches outright would discard those continuations.

3.3 Theoretical Analysis: A Capped Selection Variant

For analysis, consider the capped variant $\alpha_t = \min\{1/r_t, \bar{\alpha}\}$, where $1 \leq \bar{\alpha} < \infty$ is fixed independently of the initial budgets. The result below concerns generation of a terminal answer, not correctness. It is conditional on the following assumptions; the reported experiments use the uncapped rule of Section 3.2.2.

Assumption 1 (Reachable Progress)**.** *There is a trajectory $\mathcal{P}^*$ whose frontier, when selected, advances to its next state with conditional probability at least $q_{\min} > 0$. Each advance adds at least $\delta > 0$ to its value until θ is reached. A selected frontier with $V \geq \theta$ produces a terminal answer with probability at least $q_{\min}$. These bounds hold for every history before an answer is generated.*

Assumption 2 (Preserved Frontier)**.** *Until an answer is generated, the current frontier of $\mathcal{P}^*$ remains active. Retirement, merging and reuse preserve an active representative with the same continuation and value lower bound; other value updates do not erase accumulated progress. Clipping is the identity on $[v_{\min}, v_{\max}]$, with $\theta < v_{\max}$.*

Assumption 3 (Uniform Bounds)**.** *The candidate pool has at most $N_{\max}$ nodes, with values in $[v_{\min}, v_{\max}]$ and $v_{\min} > 0$. Every expansion, including generation and scoring, has bounded tool and token costs. All constants above are independent of the initial budgets, and inference continues while resources permit and no terminal answer has been generated.*

Theorem 1 (Terminal-Answer Generation with Capped Selection)**.** *Under Assumptions 1–3, for every $\epsilon \in (0, 1)$, finite tool and token budgets suffice for the capped variant to generate a terminal answer with probability at least $1 - \epsilon$.*

The fixed cap gives a budget-independent frontier-selection bound $p_{\min} = N_{\max}^{-1}(v_{\min}/v_{\max})^{\bar{\alpha}}$. The proof uses progress probability $p_{\min} q_{\min}$ and includes the additional selection needed to generate an answer (Appendix C.1). Frontier preservation and a uniform pool bound are assumptions, not consequences of duplicate retirement; this theorem does not establish an unconditional guarantee for the empirical implementation.

4 Experiments

4.1 Experimental Setup

Datasets. We use four multi-hop reasoning benchmarks: HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). We evaluate on the full evaluation splits (HotpotQA 7,405; 2WikiMultihopQA 12,576; MuSiQue 2,417; Bamboogle 125), i.e., 22,523 questions per model and budget. We report exact match (EM) macro-averaged over the four datasets as mean±sd over three seeds.

Models Evaluated. We evaluate two open models: GPT-OSS-20B (Agarwal et al., 2025), a reasoning model with native tool use, and Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025), an instruction-following model. We additionally evaluate three closed-weight models served through a commercial API (Claude Sonnet 5, GPT-5.6 and DeepSeek V3.2) on a 100-question random subset of each QA dataset and on WebArena-Lite (Table 4; setup in Appendix F).

Method	GPT-OSS-20B			Qwen3-30B-A3B		
	B_{tool}=5	B_{tool}=10	B_{tool}=20	B_{tool}=5	B_{tool}=10	B_{tool}=20
ReAct	16.7±0.8 (3.8)	18.6±1.1 (5.8)	19.9±0.2 (8.1)	27.2±0.2 (1.5)	27.2±0.3 (1.5)	27.2±0.1 (1.5)
LATS	19.4±0.8 (4.9)	21.7±0.2 (9.7)	24.8±1.1 (18.2)	28.5±0.1 (4.7)	28.8±0.7 (9.0)	32.8±1.0 (13.7)
BATS	18.6±0.9 (4.5)	22.8±0.6 (8.9)	25.3±0.6 (17.6)	35.8±0.4 (2.6)	36.2±0.8 (4.9)	37.0±1.0 (8.7)
Parallel	20.9±0.8 (4.9)	27.4±1.4 (9.9)	34.2±0.8† (19.8)	27.4±0.2 (5.0)	28.1±0.2 (9.9)	28.3±0.4 (19.9)
BAVT w/o reuse	33.7±1.4 (4.7)	38.0±0.3 (9.2)	40.6±0.7 (18.0)	38.4±0.4 (4.7)	39.2±0.3 (8.9)	38.4±0.5 (16.9)
BAVT	**35.0±0.1**† (4.1)	**39.0±0.2** (6.6)	**42.0±1.1** (11.7)	**39.1±0.3** (3.0)	**40.1±0.8** (3.4)	**40.0±0.4** (4.3)
Δ config.	$+1.3/-13\%$	$+1.0/-29\%$	$+1.4/-35\%$	$+0.7/-36\%$	$+0.9/-62\%$	$+1.6/-75\%$

Table 1: Multi-hop QA exact match (EM, %; mean±sd over three seeds), with mean paid *tool calls per question in parentheses*. Δ config. compares full BAVT with the evaluated configuration without reuse (EM difference / call reduction). †BAVT at B_{tool}=5 exceeds parallel sampling at B_{tool}=20.

Budget Configurations. We evaluate three budget tiers defined by a tool-call budget B_{tool} and a token budget B_{token}. Since reasoning models produce longer traces, they are assigned larger token budgets. *Low:* $B_{\text{tool}} = 5$, $B_{\text{token}} = 2000/1000$ (reasoning/instruct); *Middle:* $B_{\text{tool}} = 10$, $B_{\text{token}} = 4000/2000$; *High:* $B_{\text{tool}} = 20$, $B_{\text{token}} = 8000/4000$. For fairness, the reported token budget reflects the full end-to-end inference cost of each method. For QA, paid tool calls count retriever invocations charged to the budget, averaged per question.

Baselines. We compare with ReAct (Yao et al., 2022), a single reasoning–acting trajectory; LATS (Zhou et al., 2023a), MCTS-style search under the shared tool-call budget; BATS (Liu et al., 2025), budget-aware tool use with trajectory restarts; and budget-constrained parallel sampling with majority vote. We also report the BAVT configuration without evidence reuse. Methods share the backbone, retriever, and corpus; paid calls include failed tool calls. Appendix B.1 gives baseline details.

Implementation Details. We implement BAVT within the Inspect AI (UK AI Safety Institute, 2024) framework. At the root node, we add a budget hint to prompt a single high-level plan (Section H.3); the resulting abstract outline and estimated tool usage are shared across all subsequent nodes. For retrieval-augmented tool execution, we follow Search-R1 (Jin et al., 2025) using the 2018 Wikipedia dump (Karpukhin et al., 2020) as the knowledge corpus and the E5 model (Wang et al., 2022a) as the dense retriever, with five retrieved passages per query. Generation uses each family's default sampling settings (Appendix E). Evidence reuse follows Section 3.2.3: the near-duplicate check uses the retriever's own E5 encoder with the threshold and numeric guard of Appendix E.2, so no additional model is loaded. Further details are provided in Appendix E.

4.2 Main Results

Table 1 reports EM (in %) and paid tool calls per question for all methods at the three budgets, and we further plot EM against the tool-call budget in Figure 2 in the appendix.

The largest gains occur at tight budgets. BAVT leads every baseline at all three budgets on both open-weight families (Table 1). At five calls, GPT-OSS-20B reaches 35.0 EM with 4.1 paid tool calls, exceeding parallel sampling at twenty calls (34.2 EM, 19.8 paid tool calls). Qwen3-30B-A3B shows the same cross-budget advantage, outperforming the strongest twenty-call baseline with five calls. The cross-budget comparison measures attained accuracy against actual retrieval use, rather than the allowed maximum: the GPT-OSS configuration uses about one fifth as many paid tool calls as parallel sampling. At the matched twenty-call budget, BAVT also improves EM by 7.8 points while using 41% fewer paid tool calls.

Higher budgets help the two families differently. GPT-OSS-20B improves from 35.0 to 42.0 EM as BAVT's budget increases from five to twenty calls. Qwen3-30B-A3B stays near 40 EM; its parallel-sampling baseline is nearly flat despite quadrupling its paid tool calls. For GPT-OSS-20B, the ten-to-twenty tier still brings a 3.0-point EM gain, accompanied by an increase from 6.6 to 11.7 paid tool calls. Qwen's corresponding change is -0.1 points despite more paid tool calls, so its highest tier provides no clear accuracy benefit in these measurements. The empirically useful budget range therefore depends on the model family; it cannot be inferred from the nominal call limit alone. Because both tool and token caps increase across tiers, these trends describe the combined budget

Components				EM (%)					Paid
Tree	Value	Budget	Reuse	HotpotQA	2Wiki	MuSiQue	Bamboogle	AVG	tool calls/q
×	×	×	×	29.0±2.0	19.7±1.2	14.0±1.7	44.7±2.9	26.8±1.2	9.77
✓	×	×	×	27.3±2.5	28.3±2.5	12.7±3.5	39.0±1.0	26.8±1.6	9.98
✓	✓	×	×	38.0±1.0	38.7±0.6	17.3±2.1	48.7±0.6	35.7±0.9	9.51
✓	✓	✓	×	39.7±2.1	43.0±1.7	17.0±3.0	**55.7±0.6**	38.8±0.1	9.00
✓	✓	✓	✓	**41.0±1.0**	**43.3±1.2**	**20.0±4.6**	53.3±0.6	**39.4±0.8**	**6.34**

Table 2: Component configurations on GPT-OSS-20B at $B_{\text{tool}} = 10$: EM (%, mean±sd over three seeds on 100 questions per dataset) and mean paid tool calls per question. Checkmarks mark active components. The tree-only configuration selects both expansion and final-answer nodes uniformly; the final row is full BAVT.

setting rather than an isolated effect of additional retrieval. This contrast motivates measuring both accuracy and resource use rather than treating a larger budget as evidence of useful computation.

The full configuration uses fewer paid tool calls. Relative to BAVT w/o reuse, it uses 13–35% fewer paid tool calls on GPT-OSS-20B and 36–75% fewer on Qwen3-30B-A3B, with mean EM higher by 0.7–1.6 points. These are comparisons of the evaluated configurations; the small accuracy differences alone do not establish an isolated effect of reuse. The efficiency claim here concerns paid retrieval, not a proportional reduction in total inference latency or token use. Verifying and re-scoring a reused observation still requires model computation, so the call savings should be read alongside EM rather than treated as a complete accounting of all inference costs.

4.3 Ablation Studies

Table 2 compares successive configurations on GPT-OSS-20B at ten calls, using 100 questions per dataset over three seeds. A uniformly sampled tree matches parallel sampling at 26.8 EM. The tree-only row improves 2Wiki but declines on the other three datasets, so retaining alternative paths is not sufficient when both expansion and answer selection are uniform. Adding step-level values raises the mean to 35.7. This 8.9-point increase is the largest adjacent-row gain, with improvements on all four datasets, consistent with the importance of discriminating among candidate states. Budget-annealed selection reaches 38.8 EM with 9.00 paid tool calls, compared with 9.51 for the value-only tree. This stage improves HotpotQA, 2Wiki and Bamboogle, whereas MuSiQue is essentially unchanged (17.3 versus 17.0 EM). Full BAVT reaches 39.4 EM with 6.34 paid tool calls, a 30% reduction relative to the preceding configuration; its 0.6-point EM difference is small compared with seed variation. The higher macro-average also hides a decline on Bamboogle, from 55.7 to 53.3; the full configuration is therefore not uniformly best across datasets. Its lower average retrieval cost is more pronounced than the final EM increment. These comparisons support the system's accuracy–cost trade-off without isolating every component interaction. Threshold sensitivity is modest between $\theta = 0.7$ and 0.9 (Appendix E.3).

4.4 Beyond QA: Browser Agents on WebArena-Lite

Instantiation. For browser agents, nodes correspond to browser states and external cost is measured in browser actions. BAVT applies the same budget-annealed selection and evidence-reuse principles, with the page observation as the unit of evidence in place of retrieved passages; implementation details are given in Appendix G.

Setup. We evaluate on 129 WebArena-Lite (Liu et al., 2024a) tasks spanning four self-hosted sites (shopping, shopping-admin, GitLab, and Reddit; Appendix G), using the official WebArena evaluators. We use the same two open-weight backbones and generation settings as in Section 4.1, with each model serving as both actor and verifier. Each task is given a tool-call budget $B_{\text{tool}} \in \{5, 10, 20\}$ and an output-token cap of $400B_{\text{tool}}$; here, B_{tool} counts browser actions. We compare against parallel sampling, ReAct, and ported LATS and BATS using a shared actor model and actor prompt; results are reported as success rate (%, mean±sd) in Table 3.

Findings. BAVT has the highest mean success rate at every budget for both models (Table 3), exceeding parallel sampling by 2.7–4.9 percentage points and ReAct by 1.8–4.4 percentage points.

Method	GPT-OSS-20B			Qwen3-30B-A3B		
	B_{tool}=5	B_{tool}=10	B_{tool}=20	B_{tool}=5	B_{tool}=10	B_{tool}=20
LATS	6.2±1.3	8.0±0.5	10.6±1.6	5.2±3.5	6.5±3.2	6.5±2.0
BATS	10.3±0.4	13.4±0.4	15.8±1.2	10.1±3.4	13.7±4.5	15.0±0.5
ReAct	11.0±3.3	15.5±1.7	18.6±1.7	11.0±0.8	14.6±1.3	17.7±1.3
Parallel	11.8±1.8	16.3±1.5	17.7±1.3	11.1±1.3	16.2±1.7	16.8±1.1
BAVT	**14.6±1.7**	**19.6±0.9**	**20.4±1.9**	**15.4±1.3**	**18.9±1.9**	**21.7±1.3**

Table 3: WebArena-Lite success rate (%, mean±sd) on 129 tasks under three action budgets. GPT-OSS-20B and Qwen3-30B-A3B each serve as both actor and verifier. Results use six seeds, except ported LATS and BATS with three; environment and protocol details are in Appendix G.

Method	Claude Sonnet 5			GPT-5.6			DeepSeek V3.2		
	5	10	20	5	10	20	5	10	20
Multi-hop QA: EM % (paid tool calls per question); budget B_{tool}									
ReAct	33.5 2.2	34.5 2.9	36.0 4.3	36.8 2.5	38.8 3.5	40.8 4.4	23.8 4.3	35.5 6.7	40.2 9.3
LATS	30.5 3.9	30.0 6.0	31.8 8.6	38.2 4.8	41.5 9.2	**45.2** 15.2	28.5 5.0	28.5 10.0	33.8 19.6
BATS	22.8 1.9	28.3 3.4	29.0 5.8	35.2 3.9	34.8 7.5	37.8 14.0	25.0 4.6	27.0 8.7	30.0 15.9
Parallel sampling	37.0 4.7	37.8 9.3	39.8 18.7	37.2 4.7	40.4 9.4	42.9 18.9	22.5 5.0	34.4 10.0	41.2 20.0
BAVT (ours)	**39.2** 3.2	**41.8** 4.7	**41.9** 7.7	**41.7** 3.0	**43.1** 4.2	43.8 6.2	**37.8** 5.0	**41.5** 9.5	**45.1** 18.2
WebArena-Lite: success rate %; budget B_{tool}									
ReAct	24.8	33.3	39.5	17.1	30.2	32.6	13.2	22.5	28.7
LATS	22.5	25.6	31.8	15.5	17.1	24.0	9.3	11.6	19.4
BATS	23.3	28.7	30.2	20.2	27.9	32.6	13.2	15.5	17.1
Parallel sampling	25.6	34.9	38.0	17.8	29.5	31.0	12.4	20.9	24.8
BAVT (ours)	**30.2**	**40.3**	**43.4**	**24.0**	**37.2**	**34.9**	**30.2**	**27.9**	**31.0**

Table 4: Closed-weight models served through an API (setup in Appendix F). Top: multi-hop QA on a 100-question random subset of each dataset, exact match (%) with paid tool calls per question in grey; BAVT and parallel sampling are three-seed means (sd $\leq$ 1.9), the other rows single runs. Bottom: WebArena-Lite success rate (%) on 129 tasks, single seed, with the same model as actor and verifier; ReAct runs without the per-step budget hint. Bold marks the best entry in each column of a band.

The ported LATS and BATS variants trail parallel sampling in all six model–budget settings. These results support the browser instantiation's use of step-level verification and budget-conditioned search under short action horizons.

Closed-weight models. BAVT leads in eight of nine QA cells and all nine browser cells (Table 4; Appendix F). On QA, Claude Sonnet 5 and GPT-5.6 use substantially fewer paid tool calls than parallel sampling, whereas DeepSeek V3.2 uses nearly as many. The exception is GPT-5.6 at twenty calls: LATS is 1.4 EM points ahead, using 2.4 times as many paid tool calls.

Scope of the evidence. Budgets above twenty calls were not tested. Qwen3-30B-A3B saturates early, and the small accuracy differences between BAVT configurations should not be attributed to reuse alone. Browser margins of 2–5 points accompany seed standard deviations around 1–3 points, while the closed-weight browser results are single-seed measurements.

5 Conclusion

BAVT allocates finite tool and token budgets through step-level values, budget-annealed selection and evidence reuse. Across four QA benchmarks and two open-weight families, its five-call configuration exceeds every baseline given twenty calls; the browser instantiation and API-model results extend this evidence to other settings. A capped selection variant admits a conditional finite-budget guarantee for generating an answer, without guaranteeing correctness. The results support allocating computation according to progress while avoiding redundant tool calls. Larger budgets, changing tool outputs and stronger component controls remain directions for further study.

ETHICS STATEMENT

This work studies budget-aware inference for tool-augmented LLM agents using public multi-hop QA benchmarks and existing open or publicly documented models. It does not involve human subjects, private data, or real-world autonomous deployment. Our goal is to improve the reliability and efficiency of agent inference under fixed token and tool-use budgets, which may help reduce unnecessary computation and external API usage. At the same time, methods that improve agent planning and search can also increase the effectiveness of autonomous systems more broadly. We therefore encourage evaluation in controlled settings and the use of appropriate safeguards in deployment, including tool restrictions, budget limits, and human oversight for high-stakes applications.

AI USE STATEMENT

Generative AI tools were used to improve the clarity and readability of the manuscript and to assist with editing software code. All AI-assisted text and code were reviewed and verified by the authors, who take full responsibility for the final content.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Chaokun Chang, Yukun Zhou, Kaihua Fu, Dakai An, Tianyu Feng, Hanfeng Lu, Sheng Yao, Pu Guo, Yinghao Yu, Yizhou Shan, Bo Li, Binhang Yuan, and Wei Wang. From LLM inference to agentic workloads: Characterization and implications for serving systems. *arXiv preprint arXiv:2608.15127*, 2026.

Harrison Chase. Langchain. https://github.com/langchain-ai/langchain, 2022.

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

Wenlong Deng, Yushu Li, Boying Gong, Yi Ren, Christos Thrampoulidis, and Xiaoxiao Li. On grpo collapse in search-r1: The lazy likelihood-displacement death spiral. *arXiv preprint arXiv:2512.04220*, 2025.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, 2025.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. *arXiv preprint arXiv:2503.04412*, 2025.

Abhinav Java, Srivathsan Koundinyan, Nagarajan Natarajan, and Amit Sharma. FrugalRAG: Less is more in RL finetuning for multi-hop question answering. In *The Fourteenth International Conference on Learning Representations*, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective. *arXiv preprint arXiv:2506.04301*, 2025a.

Joongho Kim, Xirui Huang, Zarreen Reza, and Gabriel Grand. Chopping trees: Semantic similarity based dynamic pruning for tree-of-thought reasoning. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025b. arXiv preprint arXiv:2511.08595.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. *arXiv preprint arXiv:2505.11274*, 2025.

Bang Liu, Linglong Kong, and Jian Pei. Phase transition for budgeted multi-agent synergy. *arXiv preprint arXiv:2601.17311*, 2026.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024a.

Yinhong Liu, Han Zhou, Zhijiang Guo, Ehsan Shareghi, Ivan Vulić, Anna Korhonen, and Nigel Collier. Aligning with human judgement: The role of pairwise preference in large language model evaluators. *arXiv preprint arXiv:2403.16950*, 2024b.

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025a.

Ruofan Lu, Yichen Li, and Yintong Huo. Exploring autonomous agents: A closer look at why they fail when completing tasks. *arXiv preprint arXiv:2508.13143*, 2025b.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

Zhizhao Luo, Zhaojing Luo, Meihui Zhang, and Rui Mao. TabTracer: Monte carlo tree search for complex table reasoning with large language models. *arXiv preprint arXiv:2602.14089*, 2026.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

OpenAI. Introducing GPT-5.3-Codex. https://openai.com/index/introducing-gpt-5-3-codex/, 2026. Accessed: 2026-02-23.

OpenClaw contributors. OpenClaw: Your own personal AI assistant. https://github.com/openclaw/openclaw, 2026. Accessed: 2026-02-23.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*, 2025.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. Toward self-improvement of LLMs via imagination, searching, and criticizing. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

UK AI Safety Institute. Inspect: A framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024.

Ante Wang, Linfeng Song, Ye Tian, Dian Yu, Haitao Mi, Xiangyu Duan, Zhaopeng Tu, Jinsong Su, and Dong Yu. Don't get lost in the trees: Streamlining LLM reasoning by overcoming tree search exploration pitfalls. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23946–23959, Vienna, Austria, July 2025a. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1167. URL https://aclanthology.org/2025.acl-long.1167/.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently. *arXiv preprint arXiv:2504.14870*, 2025b.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022a.

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

Peilin Wu, Mian Zhang, Xinlu Zhang, Xinya Du, and Zhiyu Zoey Chen. Search wisely: Mitigating sub-optimal agentic searches by reducing uncertainty. *arXiv preprint arXiv:2505.17281*, 2025.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shuo Yang, Soyeon Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hovy. ToolTree: Efficient LLM agent tool planning via dual-feedback Monte Carlo tree search and bidirectional pruning. In *International Conference on Learning Representations (ICLR)*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

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*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Jieyu Zhang, Ranjay Krishna, Ahmed H Awadallah, and Chi Wang. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*, 2023.

Yaocheng Zhang, Haohuan Huang, Zijun Song, Yuanheng Zhu, Qichao Zhang, Zijie Zhao, and Dongbin Zhao. Criticsearch: Fine-grained credit assignment for search agents via a retrospective critic. *arXiv preprint arXiv:2511.12159*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

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

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, et al. Scaling test-time compute for llm agents. *arXiv preprint arXiv:2506.12928*, 2025.

A Detailed Related Work

A.1 Tool-Augmented LLM Agents

The integration of external tools has significantly advanced the capabilities of Large Language Models (LLMs), transitioning them from static text generators to active agents capable of interacting with dynamic environments. Foundational frameworks such as ReAct (Yao et al., 2022), Toolformer (Schick et al., 2023), and WebGPT (Nakano et al., 2021) have demonstrated the efficacy of interleaving reasoning traces with tool actions to solve complex queries. More recently, the development of robust orchestration frameworks, such as LangChain (Chase, 2022), and advanced agentic evaluation toolkits, like Inspect AI (UK AI Safety Institute, 2024) and OctoTools (Lu et al., 2025a), have standardized the deployment and testing of these complex multi-hop agents. While recent RL-based approaches attempt to optimize tool-use directly during training, they often suffer from severe instability and high computational overhead (Jin et al., 2025; Zhang et al., 2025; Sun et al., 2025; Deng et al., 2025). Consequently, deployed agents typically rely on naive autonomous loops that assume infinite resources, frequently trapping them in costly dead ends (Cemri et al., 2025; Kim et al., 2025a). Browser agents extend this loop to realistic websites, where benchmarks such as WebArena (Zhou et al., 2023b) and its WebArena-Lite subset (Liu et al., 2024a) demand long action sequences under partial observability and state changes that cannot be undone. Search over browser trajectories has been studied with LATS (Zhou et al., 2023a), which grows an MCTS-style tree with LLM value estimates, and with BATS (Liu et al., 2025), which gives the agent awareness of its remaining tool budget and adapts its planning and verification to it; Section 4.4 ports both to the same browser-action budget as BAVT.

A.2 Test-Time Scaling

To overcome the limitations of linear decoding, recent literature has systematically shifted toward test-time scaling—allocating more computational resources during inference to optimally improve reasoning robustness (Snell et al., 2025). Expanding upon these foundational scaling laws, recent works have specifically investigated scaling test-time compute for LLM agents (Zhu et al., 2025), emphasizing the unique performance gains achieved when autonomous systems are granted extended inference budgets for multi-step tool interactions. Methodologies such as Self-Consistency (Wang et al., 2022b), Tree of Thoughts (ToT) (Yao et al., 2023), Graph of Thoughts (GoT) (Besta et al., 2024), and Language Agent Tree Search (LATS) (Zhou et al., 2023a) formulate the reasoning process as a search problem over a vast state space. Furthermore, recent advancements explore dynamically optimizing the geometry of these search spaces, such as adaptively balancing whether to expand wider or search deeper during inference (Inoue et al., 2025). Drawing inspiration from reinforcement learning, recent works increasingly employ actor-critic paradigms during inference, while reflection mechanisms and prompt-based critics serve as value functions to evaluate intermediate states and facilitate self-correction (Shinn et al., 2023). While these search-based and value-guided algorithms achieve strong performance, they are predominantly accuracy-driven and operate under the assumption of unbounded computational resources. They generally lack internal mechanisms to penalize expensive actions or adapt their search geometry based on resource depletion.

A.3 Budget-Aware Inference

As the economic and computational costs of deploying LLMs become a critical bottleneck, a growing subfield has focused on budget-aware inference. Initial strategies, such as model cascading (Chen et al., 2023) and routing systems like EcoAssistant (Zhang et al., 2023), reduce costs by intelligently directing queries to cheaper models. Recent efforts in budget-aware inference have explored dynamic resource allocation for general LLM reasoning (Han et al., 2025; Li et al., 2025). However, these approaches are largely confined to static, closed-book problems rather than multi-hop agent workflows, and they typically rely on computationally expensive post-training to align the model's policy with resource constraints. However, autonomous agents pose a unique challenge due to the high financial cost of iterative environment interactions, such as web searches. Budget-Aware Test-time Scaling (BATS) (Liu et al., 2025) addresses this for agents through prompted planning and self-verification conditioned on the remaining budget, retaining a dynamic plan and summaries of intermediate findings; theoretical work identifies performance phase transitions under strict multi-agent constraints (Liu et al., 2026). BATS adapts its behavior through planning and verification prompts. BAVT instead

uses step-level values to define an explicit budget-conditioned node-sampling distribution, shifting from exploration to exploitation as resources shrink.

A.4 Evidence Reuse and Redundancy

Redundant tool calls inside LLM tree search have been addressed by caching, pruning and deduplication. TabTracer (Luo et al., 2026) caches tool results behind a state hash so that an identical invocation is not executed twice; ToolTree (Yang et al., 2026) runs MCTS over tool plans with dual-feedback evaluation and prunes low-value plans before they are executed; SSDP (Kim et al., 2025b), Fetch (Wang et al., 2025a) and AlphaLLM (Tian et al., 2024) detect equivalent or near-equivalent states and merge them so that the same state is not expanded twice. These mechanisms operate within one rollout or merge branches outright, and none of them is connected to a budget. A training-based alternative teaches the policy to search less: FrugalRAG (Java et al., 2026), OTC-PO (Wang et al., 2025b) and Search Wisely (Wu et al., 2025) shape the reward so that a retrieval-augmented agent issues fewer or better-targeted searches; these methods require RL training and reduce the number of searches per trajectory rather than sharing evidence across trajectories. Production deep-research agents (Zheng et al., 2025) and agent-system benchmarks (Chang et al., 2026) use query-level search caches to cut API cost and latency, without a search tree. Reuse in BAVT differs in three respects: it is tree-wide, so a query issued in one branch serves any later branch of the same question; it is coupled to the budget, since the remaining ratio r_t counts paid calls only and every saved call is returned to the annealed selection; and near-duplicate queries are verified rather than merged, i.e., the child node is created and scored while only the retrieval is free.

B Additional Details

B.1 Baseline Details

All baselines run in the same implementation as BAVT, with the same backbone, retriever, corpus and budget tiers, and under the same accounting: every issued search is charged one tool call, including malformed or failed calls, and every generated token of every component counts against the token budget. Paid tool calls per question are averaged over all questions of a run.

ReAct. A single reasoning–acting trajectory (Yao et al., 2022) with the shared actor prompt and the search tool; it stops when it emits an answer or when either budget is exhausted, in which case its last tentative answer is used.

LATS. The port keeps the algorithm of Zhou et al. (2023a): Monte-Carlo tree search over reasoning states with UCT selection, LLM-sampled expansions, an LLM value for each new state and backpropagation of the values; the search ends when the shared budget is exhausted and returns the highest-valued answer node. Every search issued during expansion or rollouts is charged to the shared tool budget and every LLM call to the shared token budget. **BATS.** Following Liu et al. (2025), sequential attempts with a fresh context share the tool budget; each attempt is told the budget remaining, a planner call decides whether it continues or is abandoned, and an abandoned attempt restarts from the question with the budget that is left. The tokens of the planner and of the verifier count against the same token budget, and the answers of the attempts are aggregated by majority vote.

Parallel sampling. The parallel sampling baseline is formulated as budget-constrained majority voting. Under identical resource constraints, this baseline executes m independent reasoning trajectories concurrently. Let $\{\tau_i\}_{i=1}^{m}$ denote the set of successfully completed trajectories, where each trajectory τ_i incurs a resource cost $\mathcal{C}(\tau_i)$, measured in tool calls and generated tokens. The generation process continues, maximizing the total number of trajectories m, until the cumulative cost exhausts the predefined budget limit B:

$$\sum_{i=1}^{m} \mathcal{C}(\tau_i) \leq B \tag{6}$$

Once the budget B is fully consumed, let $A(\tau_i)$ denote the terminal answer extracted from trajectory τ_i. The final predicted answer $\hat{y}$ is determined by applying a majority voting function across all valid

trajectories, formally defined as:

$$\hat{y} = \arg\max_{y} \sum_{i=1}^{m} \mathbb{I}(A(\tau_i) = y) \tag{7}$$

where $\mathbb{I}(\cdot)$ is the indicator function. This formalizes a rigorous scaling baseline that natively consumes the exact same computational budget as our proposed dynamic tree search.

C Theorem and Proof

C.1 Proof of Theorem 1

Proof. Fix $\bar{\alpha}$, $N_{\max}$, $v_{\min}$, $v_{\max}$, $q_{\min}$ and δ independently of the initial budgets. Let $\mathcal{F}_{t-1}$ be the history before expansion t, and n_t^* the current frontier of $\mathcal{P}^*$ (or its active representative). By Assumption 2, this node remains eligible until a terminal answer is generated. Under capped selection,

$$\mathbb{P}(n_t^* \text{ selected} \mid \mathcal{F}_{t-1}) \geq \frac{1}{N_{\max}} \left(\frac{v_{\min}}{v_{\max}}\right)^{\alpha_t} \geq \underbrace{\frac{1}{N_{\max}} \left(\frac{v_{\min}}{v_{\max}}\right)^{\bar{\alpha}}}_{p_{\min}} > 0. \tag{8}$$

Selection alone is insufficient: the generator must also make the prescribed advance. Assumption 1 bounds its conditional probability below by $q_{\min}$, so each expansion makes progress with probability at least $\rho = p_{\min} q_{\min} > 0$.

Let

$$K = \max\left\{0, \left\lceil \frac{\theta - V(s_0)}{\delta} \right\rceil \right\}, \qquad J = K + 1. \tag{9}$$

After at most K successful advances the frontier reaches θ: clipping cannot prevent crossing because $\theta < v_{\max}$, and other updates preserve the accumulated lower bound. One additional successful selection and generation produces the terminal answer, hence the use of J rather than K.

Before a terminal answer exists, let $X_t = 1$ when the selected frontier advances along $\mathcal{P}^*$, or, after reaching θ, produces the terminal answer. After any terminal answer has been generated, extend the sequence by setting $X_t = 1$. Then $\mathbb{P}(X_t = 1 \mid \mathcal{F}_{t-1}) \geq \rho$. Sequential coupling with independent uniforms shows that $\sum_{t=1}^{M} X_t$ stochastically dominates $Z \sim \mathrm{Binomial}(M, \rho)$; independence of the actual expansions is not required. No terminal answer by step M implies $\sum_{t=1}^{M} X_t < J$. With $x = M\rho > J$, the Chernoff bound gives

$$\mathbb{P}(\text{no terminal answer by } M) \leq \mathbb{P}(Z < J) \leq \exp\left(-\frac{(x-J)^2}{2x}\right). \tag{10}$$

Writing $L = \log(1/\epsilon)$, a sufficient choice is

$$M = \left\lceil \frac{J + \sqrt{2JL} + 2L}{\rho} \right\rceil. \tag{11}$$

Indeed, the numerator is at least $J + L + \sqrt{L^2 + 2JL}$, which makes the preceding exponential at most ϵ.

Finally, let c_{tool} and c_{token} bound the complete cost of an expansion, including action generation, execution and scoring. Finite budgets

$$B_{\mathrm{tool}} \geq M c_{\mathrm{tool}} + 1, \qquad B_{\mathrm{token}} \geq \left\lceil \frac{M c_{\mathrm{token}} + 1}{1 - \eta} \right\rceil \tag{12}$$

keep tool budget positive and the token ratio above the answer-generation trigger $\eta < 1$ throughout the first M expansions. Thus resource exhaustion cannot interrupt those attempts before a terminal answer exists. All constants in ρ and J are budget-independent, so these budgets are finite and establish the claimed probability. This proves generation of an answer, not its correctness. □

Algorithm 1 Budget-Aware Value Tree (BAVT)

Require: Query x, Initial budgets $B_{\text{tool}}, B_{\text{token}}$
Ensure: Final predicted answer $\hat{y}$

Init: root $s_0 \leftarrow x$, budget $b_0 \leftarrow (B_{\text{tool}}, B_{\text{token}})$, $\mathcal{G}, \mathcal{N}_{\text{cand}} \leftarrow \{s_0\}$, $\mathcal{N}_{\text{ans}} \leftarrow \emptyset$
while $b_{\text{tool},t} > 0 \wedge b_{\text{token},t} > 0$ **do**

— 1. Budget-Aware Node Selection —

  $r_t \leftarrow \min(b_{\text{tool},t}/B_{\text{tool}}, b_{\text{token},t}/B_{\text{token}})$; $\alpha_t \leftarrow 1/r_t$
  Sample expansion node $n \sim P(n_i) \propto V(n_i)^{\alpha_t}$ from $\mathcal{N}_{\text{cand}}$

— 2. Action Generation and Transition —

  Issue structural instruction I_t based on $V(n)$ ▷ Widen, Deepen, or Answer
  Generate action $a_t \sim P(\cdot|n, I_t)$
  if a_t is a search whose query duplicates a sibling's **then**
    Merge into the sibling: no call, no child, no charge; retire n from $\mathcal{N}_{\text{cand}}$ after its second merged duplicate; **continue**
  else if the query hits the per-question cache (exact or near-duplicate) **then**
    Serve the cached result as the observation; no charge
  else
    Call the retriever and charge one tool call
  end if
  Observe transition $\mathcal{T}(n, a_t) \to n'$; update budget $b_{t+1} \leftarrow b_t - \mathcal{C}(a_t)$; append n' to $\mathcal{G}$ ▷ tokens always charged

— 3. Value Critic —

  if n' is a terminal answer node **then**
    $V(n') \leftarrow V(n)$; $\mathcal{N}_{\text{ans}} \leftarrow \mathcal{N}_{\text{ans}} \cup \{n'\}$ ▷ Inherit value, skip candidate pool
  else
    Predict raw residual $\Delta_t \in \{-4, \ldots, 4\}$; set $V(n') \leftarrow \text{clip}_{[0.1,1]}(V(n) + \Delta_t/10)$
    $\mathcal{N}_{\text{cand}} \leftarrow \mathcal{N}_{\text{cand}} \cup \{n'\}$
  end if

— 4. Backstop Answer Generation —

  if $\mathcal{N}_{\text{ans}} = \emptyset \wedge (b_{\text{tool},t} = 0 \vee b_{\text{token},t}/B_{\text{token}} \leq \eta)$ **then**
    Synthesize answer n^*_{ans} from $n^* = \arg\max_{n \in \mathcal{N}_{\text{cand}}} V(n)$
    $V(n^*_{\text{ans}}) \leftarrow V(n^*)$; $\mathcal{N}_{\text{ans}} \leftarrow \mathcal{N}_{\text{ans}} \cup \{n^*_{\text{ans}}\}$; $\mathcal{G} \leftarrow \mathcal{G} \cup \{n^*_{\text{ans}}\}$
  end if
end while
return $\hat{y} \leftarrow$ majority vote over the answers of $\mathcal{N}_{\text{ans}}$, ties broken by the higher V

D BAVT ALGORITHM

The complete execution flow of Budget-Aware Value Tree (BAVT) is detailed in Algorithm 1. Initialized with a root query and strict computational constraints, the framework iteratively constructs a search tree. At each step, a candidate node is sampled from a budget-annealed distribution, where the scaling exponent α_t smoothly transitions the agent's policy from broad exploration to greedy exploitation as resources deplete. Following node selection, the framework issues a deterministic, value-guided instruction (widen, deepen, or generate answer) to orchestrate the generator's action. The resulting state is then evaluated by the prompt-based critic to predict a residual information delta (Δ_t), updating the node's accumulated value. Before a search is executed, the reuse rules of Section 3.2.3 are applied: a query that duplicates a sibling's is merged without a call, a hit in the per-question cache (exact or near-duplicate) is served without charge, and only a new query is sent to the retriever and charged. The search terminates upon budget exhaustion, and the final answer is the majority vote over the terminal answers, with ties broken by the higher node value (Section 3.2.2).

E Implementation Details and Reproducibility

E.1 Hyperparameter Configurations

To ensure full reproducibility of the Budget-Aware Value Tree (BAVT) framework, we detail the core hyperparameters utilized across all experiments in Table 5. The dynamic search parameters were held constant across all datasets to demonstrate the generalization of the framework. Generation uses the default sampling settings of each family: temperature 0.7, top-p 0.8 and top-k 20 for Qwen3-30B-A3B; temperature 1.0, top-p 1.0 and no top-k for GPT-OSS-20B.

Value Scaling and Normalization. To maintain stable exponentiation during the budget-annealed node sampling, we strictly bound all state values. The prompt-based critic operates on a raw discrete scale from 1 to 10. At each step, it predicts an integer residual $\Delta_t \in [-4, +4]$ on this same raw scale. The child raw value is updated by adding the clipped delta to the parent raw value and then bounding the result within [1,10]. The bounded raw value is subsequently normalized by dividing by 10, yielding the operational state value $V(n) \in [0.1, 1.0]$. Equivalently, Equation 2 adds $\Delta_t/10$ to the normalized parent value before clipping to $[0.1, 1]$.

Parameter	Value
Raw Critic Value Scale	$[1, 10]$
Residual Delta Bounds (Δ_t, raw scale)	$[-4, +4]$
Normalized State Value Range $V(n)$	$[0.1, 1.0]$
Terminal Confidence Threshold (θ)	0.7
Budget Backstop Threshold (η)	0.2
Near-duplicate cosine threshold	0.95
Numeric-token guard	required
Free reuses per question at $B_{\text{tool}} = 5$	2
Parent retirement after duplicate proposals	2
Max Output Tokens per Call	512

Table 5: Core hyperparameters for the BAVT framework and LLM generation.

E.2 Environment-Specific Evidence Equivalence

Section 3.2.3 defines interaction equivalence $(s, a) \simeq (s', a')$ abstractly. We instantiate this relation differently for retrieval and browser environments.

Retrieval interactions. For a retrieval action a, let $q(a)$ denote its normalized query (lower-cased and whitespace-folded), and let ϕ be the encoder used by the dense retriever. Two retrieval interactions are considered equivalent if their queries are identical or sufficiently close in the retriever's embedding space:

$$q(a) \simeq q(a') \iff q(a) = q(a') \ \vee \ \left[\cos\big(\phi(q(a)), \phi(q(a'))\big) \geq \tau \ \wedge \ \mathrm{num}(q(a)) = \mathrm{num}(q(a'))\right], \tag{13}$$

where $\mathrm{num}(\cdot)$ extracts numeric tokens such as years, quantities, and ordinals, and we use $\tau = 0.95$. The numeric-token guard prevents queries that are semantically similar but refer to different quantities or dates from being reused.

Browser interactions. Browser evidence consists of task-local page observations. Repeated observations are recognized within the same site-state context; a shared URL alone does not establish state equivalence. Page values and intermediate observations guide subsequent verification as described in Appendix G.

E.3 Sensitivity to the Termination Threshold

Table 6 varies the termination threshold θ, above which a node is allowed to answer, over $\{0.6, 0.7, 0.8, 0.9\}$ for BAVT w/o reuse on both open models, using a 100-question random subset of

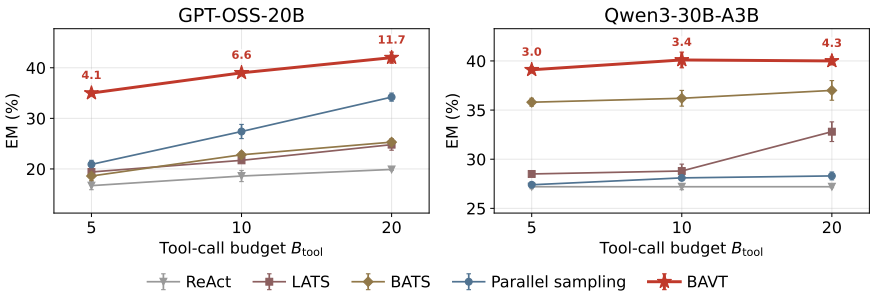

Figure 2: Exact match (%, macro-average over the four datasets, three seeds; error bars are ±sd) against the tool-call budget $B_{\text{tool}} \in \{5, 10, 20\}$ for GPT-OSS-20B (left) and Qwen3-30B-A3B (right). Red numbers above the BAVT markers are its paid tool calls per question. On both families BAVT reaches higher EM at $B_{\text{tool}} = 5$ (4.1 and 3.0 paid calls) than every baseline reaches at $B_{\text{tool}} = 20$.

each dataset (400 questions per model and budget) and three seeds. Between 0.7 and 0.9 the macro EM moves within about two points at every budget with no consistent direction, and the number of paid tool calls is essentially unchanged, so the choice of default is not critical. A permissive threshold of 0.6 is the one setting that matters: the instruct model then accepts weak answers early and falls two to four points below the default at every budget (six points below the best setting at $B_{\text{tool}} = 20$), while the reasoning model moves by less than two points.

	B_{tool}=5		B_{tool}=10		B_{tool}=20	
θ	EM	Calls	EM	Calls	EM	Calls
GPT-OSS-20B						
0.6	32.4±1.0	4.6	37.1±2.5	9.0	42.1±1.7	17.6
0.7 (default)	34.2±0.6	4.6	38.8±0.1	9.0	40.5±1.3	17.5
0.8	33.7±0.9	4.6	37.8±2.5	9.1	41.7±0.8	17.5
0.9	33.2±0.9	4.6	39.1±1.1	9.1	39.9±0.9	17.6
Qwen3-30B-A3B						
0.6	36.4±1.1	4.7	38.4±0.6	9.0	35.3±0.4	17.3
0.7 (default)	39.6±0.3	4.7	40.1±0.9	8.9	39.7±1.4	17.1
0.8	40.2±1.1	4.6	41.6±1.4	8.8	40.2±0.4	16.3
0.9	39.8±1.7	4.6	41.3±1.1	8.7	41.2±1.0	16.4

Table 6: Sensitivity of BAVT w/o reuse to the termination threshold θ: exact match (%, macro-averaged; mean over three seeds, standard deviation in small type) and paid tool calls per question (grey) on a 100-question random subset of each dataset (400 questions per model and budget). The shaded row is the default used in all other experiments.

E.4 Cost Estimation Analysis

To contextualize the economic efficiency of the BAVT framework during deployment, we estimate the maximum theoretical cost per sample across our three predefined budget configurations.

Performance against the tool-call budget As shown in Figure 2, exact match (%, macro-average over the four datasets, three seeds; error bars are ±sd) against the tool-call budget $B_{\text{tool}} \in \{5, 10, 20\}$ for GPT-OSS-20B (left) and Qwen3-30B-A3B (right). Red numbers above the BAVT markers are its paid tool calls per question. On both families BAVT reaches higher EM at $B_{\text{tool}} = 5$ (4.1 and 3.0 paid calls) than every baseline reaches at $B_{\text{tool}} = 20$.

Pricing and Assumptions. For our estimation, we assume the token generation budget (B_{token}) represents the maximum output tokens. In multi-hop reasoning tasks with retrieved contexts, input prompts are significantly longer than generated outputs; therefore, we conservatively estimate input token volume to be $10\times$ the output token volume. Token costs are calculated using standard API

pricing per 1M tokens[1]: GPT-OSS-20B (Reasoning Model) at $0.03 input / $0.14 output, and Qwen3-30B-A3B (Instruct Model) at $0.08 input / $0.28 output. The external search API is priced at a flat rate of $0.005 per query.

As detailed in Table 7, the analysis shows a large gap between LLM inference and tool utilization. Across all budget tiers and model architectures, the cost of executing external search queries accounts for over 90% of the cost per sample. The table bounds the cost by the maximum number of tool calls; the paid tool calls used by each method are reported in Table 1, and the per-call search price is an assumption of this estimate.

Budget	Max Tool	Search	Reasoning Model (GPT-OSS)		Instruct Model (Qwen)	
Tier	Calls	Cost ($)	Token Cost ($)	Total Cost ($)	Token Cost ($)	Total Cost ($)
Low	5	0.025	0.00088	0.02588	0.00108	0.02608
Middle	10	0.050	0.00176	0.05176	0.00216	0.05216
High	20	0.100	0.00352	0.10352	0.00432	0.10432

Table 7: Estimated maximum cost per sample (in USD) across budget tiers. Token costs assume an input-to-output token ratio of 10:1. Search operations constitute the vast majority of the total cost.

F Closed-Weight Models: Setup

Models. We evaluate three closed-weight models served through a commercial API: Claude Sonnet 5, GPT-5.6 (the Luna variant offered by the provider) and DeepSeek V3.2. As for the open models, a single model plays every role: on QA it acts, proposes the plan at the root and scores nodes as the step-level critic; in the browser it acts and verifies pages. All prompts are those used for the open models, and decoding settings are unchanged from the open-model runs (Appendix E).

QA protocol. The four datasets, the retriever and corpus, the budget accounting and the baselines of Section 4.1 are unchanged. To bound the number of API calls we evaluate a fixed random subset of 100 questions per dataset (400 questions per model and budget) instead of the full splits; the subset is the same for every method and model. All three models receive the reasoning-model token tiers (2000/4000/8000 output tokens at $B_{\text{tool}} = 5/10/20$), because under the instruct tiers these models exhausted the token budget before answering on a large share of questions at the smallest budget. BAVT and parallel sampling are run with three seeds and reported as mean$\pm$sd; ReAct, LATS and BATS are single runs. BAVT uses the hyperparameters of Table 5 without change ($\theta = 0.7$, $\tau = 0.95$, at most two free reuses at $B_{\text{tool}} = 5$).

Browser protocol. The WebArena-Lite runs follow Appendix G: the same 129 tasks on the same self-hosted sites, restored from the official images before each pass; $B_{\text{tool}} \in \{5, 10, 20\}$ browser actions per task with the output-token cap of $400B_{\text{tool}}$ charged to every component; the same model as actor and verifier; success measured by the WebArena evaluators; and the same ported LATS and BATS. Parallel sampling is the arm of Table 3; ReAct runs a single rollout without the per-step remaining-budget hint. Each cell is a single seed, and no sample errored.

Results. Table 4 reports every method. On QA, BAVT is ahead of every baseline in eight of the nine model–budget cells (by 1.6–9.3 EM points); the exception is LATS with GPT-5.6 at twenty calls, 1.4 points ahead at 2.4 times as many paid tool calls. Relative to parallel sampling, BAVT uses about one third as many paid tool calls on GPT-5.6 at twenty calls, but approximately as many on DeepSeek V3.2. In the browser, BAVT leads every baseline in all nine cells, and with five actions it solves 1.2–2.3 times as many tasks as the best baseline.

[1]API pricing rates were retrieved from https://deepinfra.com.

G WebArena-Lite Details

G.1 Environment and Measurement Protocol

Tasks. WebArena-Lite (Liu et al., 2024a) is a 165-task subset of WebArena (Zhou et al., 2023b). We self-host the shopping, shopping-admin, GitLab and Reddit sites from the official WebArena images and keep the 129 tasks that target exactly one of them (shopping 45, shopping-admin 35, GitLab 30, Reddit 19); the other 36 tasks either span two sites (5) or involve the map or Wikipedia sites (31). Outcomes are scored with the WebArena evaluators (string match, URL match and page-state checks), and the success rate is the percentage of the 129 tasks solved. Actor and verifier are the same model, GPT-OSS-20B or Qwen3-30B-A3B-Instruct-2507, with the sampling settings of Section 4.1 and low reasoning effort for GPT-OSS.

Instantiation. Browser nodes represent page observations, valued by the same model acting as verifier. The remaining budget controls candidate-action breadth and verification of tentative answers. When a chain converges, intermediate pages provide starting points for additional checks, selected using $V(n)^{1/r_t}$. Evidence reuse is based on task-local page observations and their site-state context.

Budgets. We use browser-action budgets $B_{\text{tool}} \in \{5, 10, 20\}$ and an output-token cap of $400B_{\text{tool}}$ per task. Tokens generated by the actor, verifier and planner all count toward this cap.

Protocol. Tasks mutate the sites (carts, posts, issues, settings), so all site containers are restored from the official images before the runs. Errored samples (at most 20% per run) are scored as failures.

G.2 BAVT and Baseline Implementations

BAVT. The actor first follows a single greedy chain of actions for an opening share of the budget and records any answer it produces as a candidate with a verifier score; if the opening chain stalls (repeated actions or unchanged pages), the actor proposes several candidate actions per step, with the number of proposals annealed by the remaining budget ratio, and one verifier call selects the action and scores the current page. When the chain has converged, the remaining budget decides how the answer is checked: if enough actions remain, an independent verification branch is started from an intermediate page of the chain, sampled with probability proportional to $V(n)^{1/r_t}$, and asked to confirm the tentative answer by a different route; with fewer actions the same trajectory refines its answer; with fewer still the answer is submitted after the verifier has re-checked the top candidates against their source pages. The actor is told the remaining budget at every step. The final answer is the evidence-checked candidate when that check ran and otherwise a vote over all candidates, weighted by verifier value when a verification branch was run.

Parallel sampling (budget-aware). Independent ReAct rollouts with the shared actor prompt are started one after another until the shared action or token budget is exhausted. Each rollout is told once, before its first action, "You have a budget of B'_{tool} browser actions in total for this task," where B'_{tool} is the part of the budget still unspent when the rollout starts; each rollout that volunteers an answer casts one vote and the majority answer is submitted. **ReAct (budget-aware).** A single ReAct trajectory that may spend the whole budget and receives the same opening prompt with $B'_{\text{tool}}{=}B_{\text{tool}}$.

LATS (ported). The port keeps the algorithm of Zhou et al. (2023a): UCT selection ($c{=}1.4$), expansion of two candidate actions at the selected node, an LLM value in $[0, 1]$ for the page of every new node, a greedy rollout of depth at most six from the best new child, mean-value backpropagation, and submission of the highest-valued answer node; the search stops when the budget is exhausted or an answer node reaches value 0.8. **BATS (ported).** Sequential fresh-context ReAct attempts (at most three) share the action budget; each attempt starts from the task's start page and, after every three executed actions, one planner call judges it *continue* or *kill*, in which case the attempt is abandoned and a new one begins if at least two actions remain. Answers volunteered by the attempts are aggregated by majority vote, ties broken toward the latest. Both ports use BAVT's actor model, actor prompt and budget tiers.

G.3 Budget Curves

Figure 3 plots the success rate of the five GPT-OSS-20B methods of Table 3 against the browser-action budget.

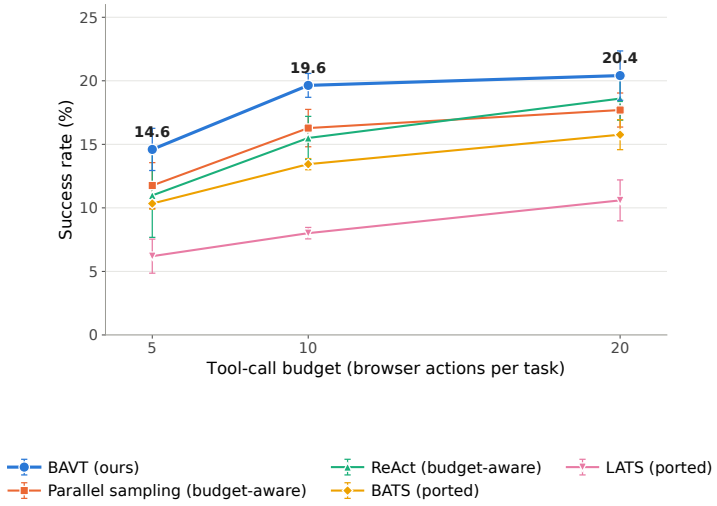

Figure 3: WebArena-Lite success rate (%) against the browser-action budget B_{tool} for the five GPT-OSS-20B methods of Table 3 (mean±sd; seeds as in Table 3). BAVT is the highest curve at every budget, and both ported tree searches stay below budget-aware parallel sampling at every budget.

G.4 Browser Prompts

All methods share the browser actor system prompt below. BAVT adds the two budget messages, the verifier prompt that selects among proposals and values the page, and the branch and refine prompts; branch answers are scored by a short verifier confidence prompt on the same 1–10 scale. Placeholders in braces are filled at run time.

System Prompt: Browser Actor (shared by all methods)

```
You are a careful web agent operating a web browser to complete a
    task
on a website (the task states which site).

On each turn take exactly ONE action:
- call exactly one web_browser tool (go / click / type /
    type_submit /
  scroll / back / forward / refresh), or
- output the final answer.

Work under a strict budget of browser actions. Be efficient:
    navigate
directly toward the information the task needs and avoid redundant
    clicks.

When you are confident, output the final answer on a single line,
    exactly:
<answer>FINAL_ANSWER</answer>
Put only the answer inside the tag, with no extra commentary after
    it.
```

Budget Messages (BAVT)

```
Opening note (once, before the first action):
```

```
You have a budget of {b} browser actions total; try to reach the
    answer
within {0.6 b} actions.

Opening-phase step line (appended to every opening turn, always on):
[budget] browser actions used: {used} / {0.6 b} (remaining:
    {remaining}).
(when at most one opening action remains, followed by:)
You MUST output your final answer as <answer>...</answer> NOW (no
    more tools).

Per-step update (appended to every turn after the opening phase,
    not kept
in the history):
Budget update: {remaining} browser actions remaining.
```

System Prompt: BAVT Verifier (proposal selection and page value)

```
You are supervising a web agent working on a website task. You will
see the task, the current page observation, the recent action
    history, and
a numbered list of candidate next actions proposed by the agent.

Return TWO judgments:

1. "choice": the 1-based index of the single BEST candidate action
    -- the
   one most likely to bring answer-relevant information into view
    fastest.
   Penalize candidates that repeat an action from the history, undo
    recent
   progress, or will clearly hit an error / login wall.

2. "value": the ABSOLUTE value (integer 1-10) of the CURRENT state:
   - 1-2: not started, irrelevant page, error, login wall, or stuck
    in a loop.
   - 3-4: navigating in the right area, but the needed information
    is not visible yet.
   - 5-6: on a relevant page; some of the needed information is now
    visible.
   - 7-8: the current page shows most of the information needed to
    answer.
   - 9-10: the answer is directly readable / derivable from the
    current page.

Output ONLY a JSON object (no Markdown, no extra text):
{"choice": 1, "value": 5}
Do NOT include any other fields.
```

Instruction: Verification Branch (leftover ¿= K)

```
CASE FILE from your earlier exploration of this task:
{recent actions and observations}
Tentative answer recorded: {answer}
You are now back on an earlier page ({url}). Using the case file as
background, verify or correct the tentative answer via a DIFFERENT
    route
from before. IMPORTANT: if the tentative answer contains specific
```

```
numbers/names, only change them if you find a page that EXPLICITLY
    shows
different values; copy exact values from pages, never re-derive
    from memory.
Current page: {observation}
When confident, reply <answer>...</answer>.
```

Instruction: Refine (3 ¡= leftover ¡ K)

```
You still have {remaining} browser actions left.
Your recorded answer so far: {answer}
Use the remaining actions to DIG DEEPER in the current area to
    verify or
refine this answer (check exact values, complete lists, confirm page
details). Then reply <answer>...</answer> with the verified/refined
    answer.
```

H System Prompt Templates

To ensure complete reproducibility of the BAVT framework, we provide the exact system prompts used in the planning, generation, and evaluation phases.

H.1 Generator (Actor) System Prompt

System Prompt: BAVT Generator

```
You are a precise AI assistant for multi-hop question answering.

Rules:
- Think step by step and explicitly show your reasoning.
- In each turn, do exactly one action: either call one tool or
    provide the final answer.
- You may use tools multiple times and reason across multiple
    rounds if needed.
- Submit exactly ONE final answer.
- The final answer must be strictly one line, with no extra text
    and no newline.

Final Answer Output Format:
<answer>FINAL_ANSWER</answer>

{dynamic_instruction}
```

H.2 Step-Level Critic Prompt

System Prompt: BAVT Critic

```
You are a strict evaluator. Your job is NOT to solve the problem,
    but to judge the quality of the Agent's latest action and score
    it.

### Value Score (Delta)
- The absolute value is in the range [1, 10]. Node 0 starts at 1.
- 10 means a correct final answer is certain given current
    information.
```

```
- 1 means a correct final answer is impossible given current
    information.
- You will be given the previous absolute value and the full step
    history (actions, observations, values).
- Output ONLY the delta relative to the previous value (e.g., +1,
    -1, 0). Do NOT output an absolute value.
- The delta MUST be an integer in [-4, +4].
- Keep deltas conservative; if evidence is weak/ambiguous, avoid
    positive deltas. If no meaningful gain, use <= -1.
- Ensure the implied new value stays within [1, 10].

### Score Calibration (5 tiers; be conservative)
Score meaning (absolute value in [1,10]): how likely a fully
    correct final answer is, given current grounded info.
- 1-2 (Tier 1: No basis / impossible): almost no grounded support,
    major contradictions, or blocked path.
- 3-4 (Tier 2: Weak / early): some relevant signals, but key facts
    missing; multiple plausible answers remain.
- 5-6 (Tier 3: Partial / uncertain): several key facts covered, but
    at least one major gap/uncertainty remains.
- 7-8 (Tier 4: Strong / likely): most essential facts covered; only
    small residual uncertainty.
- 9-10 (Tier 5: Near-certain / certain): all essential facts
    directly supported; no plausible alternative. 10 should be rare.

### Strict Evaluation Criteria (be harsh)
1) Correctness and verifiability of observations (penalize
    speculation or errors).
2) Direct relevance to the user question and current plan.
3) Information gain vs. repetition (no credit for redundant steps).
4) Coverage and completeness progress toward a full answer.
5) Reasoning quality (no unjustified leaps from evidence).
6) Efficiency vs. information gain (low gain for repeated effort is
    negative).
7) Risk and uncertainty (if ambiguity remains, avoid positive
    deltas).
8) If uncertain, choose deepen.

### Output Format
Output ONLY a JSON object (no Markdown, no extra text):
{''delta'': 0}
Do NOT include any other fields.
```

H.3 High-Level Planner Prompt

System Prompt: BAVT Planner

```
You are a high-level Multi-hop QA planner. Provide an outline plan
    only; do NOT answer the question.
- This conversation is planning-only. Never provide a final answer
    to the user question.
- Do not invent facts, entities, names, dates, or sources.
- Keep the plan abstract: do NOT list specific searches, tools,
    websites, queries, or data sources.
- Keep to 2-5 steps. Can also give alternative paths.
- Keep each line $\le$ 120 characters.
- State what must be established at each hop (the kind of
    fact/relationship), not how to retrieve it.

{budget_hint}
```

```
Question:
{question}

Follow this template to give a high-level plan:
This turn is planning-only: do NOT provide the final answer.
- Hop 1: <what link must be established>
- Hop 2: <next link>
...
Estimated tool calls: <how many tools will be called to answer the
    question>
Stop condition: <when the plan is sufficient to produce the final
    answer>
```

H.4 Dynamic Structural Instructions

During tree expansion, the placeholder {dynamic_instruction} in the Generator prompt is replaced by one of the following deterministic commands, based on the Critic's value estimates:

- **Answer:**

Dynamic Instruction: Answer

```
Instruction: Answer now. Do NOT call any tools. Return exactly
    one line in this format:
<answer>FINAL_ANSWER</answer>.
```

- **Widen ($V(n') \leq V(n)$):**

Dynamic Instruction: Widen

```
Instruction: This step did not produce a positive value gain. Try
    a different query or line of reasoning. Avoid repeating the
    previous path and do not answer yet.
```

- **Deepen ($V(n) < V(n') < \theta$):**

Dynamic Instruction: Deepen

```
Instruction: Deepen with one next step. Do exactly one action: if
    evidence is already sufficient, answer now with
    <answer>...</answer> and do NOT call tools; otherwise call
    exactly one tool.
```