DECAF: COST-BASED DYNAMIC AGENTIC PLANNING UNDER BUDGET CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling and structured planning let LLM agents tackle long-horizon tool use a single forward pass cannot, but every reasoning step, candidate plan, and re-plan costs tokens and dollars. Under a bounded budget an agent can spend so much on preparatory thinking that it never reaches the answer: the task is solvable, yet the run ends with no response because the budget is gone. This failure, which we call budget exhaustion, is invisible to how we price agents, which counts tool calls and answer tokens but treats the reasoning and planning that drive them as free. Avoiding it is hard because spending well is three coupled decisions (how much to plan, when to re-plan, and which path to commit to), each depending on a task's difficulty, learned only by acting, and on the budget left. Existing structured planners fix these and are cost-blind; the closest plan-graph planners draft a fixed number of plans, re-plan on every mismatch, and carry no notion of price. We propose **DECAF** (Dynamic, Efficient Cost-control for Agentic planning under a Finite budget), a plan-graph method that instead makes all three decisions budget-aware: it scales breadth to observed difficulty, gates each re-plan by a priced divergence test, and selects the executed path with a priced graph-ILP that prefers the cheapest goal-reaching route. On five benchmarks DECAF gives the best accuracy-cost trade-off among structured planners: no baseline is both cheaper and more accurate, improving accuracy by 4.2 points on average over the strongest structured-planner baseline at $0.65\times$ its per-task cost; and on AppWorld it reaches at half the base budget the accuracy the strongest baselines need three times the budget to match.

1 INTRODUCTION

Test-time scaling and structured planning have transformed what LLM agents can attempt: by reasoning before they act, drafting and simulating candidate plans, and re-planning when execution diverges, agents now solve long-horizon, tool-using tasks that a single forward pass cannot (Yao et al., 2023b; Hao et al., 2023; Jeong et al., 2026). This progress carries a cost that grows with the very mechanisms that produce it. Every reasoning step, candidate plan, and re-plan consumes tokens and money, and a deployed agent runs under a bounded budget set by an API spend cap, a rate limit, or a latency deadline. Under such a budget it can spend so heavily on preparatory thinking that it never reaches the answer: the task is solvable and the trajectory looks reasonable, yet the run terminates with no response because the budget is already exhausted. This is a failure of allocation rather than capability: the budget is spent where it does not pay off.

Two things make the problem hard. First, it is largely invisible to how we measure agents: standard cost accounting charges the tool calls and answer tokens an agent emits while treating the reasoning and planning that drive them as free, so the spend that most determines whether an agent finishes goes uncounted. Second, spending well is not one decision but a coupled stack of three, namely how much to plan, when to re-plan, and which path to commit to. Each depends on a task's difficulty, which the agent discovers only by acting, and on how much budget remains, so prior planners that fix these decisions in isolation and blind to the budget waste on an easy task the budget a hard one needs and purchase re-plans that never pay off.

We treat an agent's deliberation as a resource to be allocated, weighing its cost against the accuracy it buys and letting its allocation respond to task difficulty and the remaining budget. We realize

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.

this in the Dynamic, Efficient Cost-control for Agentic planning under a Finite budget (**DECAF**) framework. Methodologically, DECAF stays close to the plan-graph method TAPE (Jeong et al., 2026): like TAPE it drafts candidate plans, simulates them into a merged plan graph, and repairs the plan when execution diverges. What differs is what each decision is conditioned on. TAPE (and structured planners more broadly (Yao et al., 2023a; Hao et al., 2023)) fixes the number of drafts, re-plans on every execution mismatch, and is blind to cost, so it cannot separate an easy task from a hard one or a cheap route from an expensive one; the budget-aware planners BATS and BAVT (Liu et al., 2025; Li et al., 2026) track a budget but cap every task uniformly rather than sizing planning to difficulty. DECAF instead ties all three decisions to observed difficulty and the budget still available: it scales planning breadth by escalation, adding candidate plans only when the current ones fail to reach the goal; it gates each re-plan by a priced divergence test, re-planning only when the expected gain outweighs the priced cost; and it selects the executed path with a priced graph-ILP that prefers the cheapest route that still reaches the goal. Because difficulty is read from the run itself through escalation rather than any pre-labeled category, DECAF needs no oracle or per-task annotation.

We evaluate DECAF on five benchmarks. Across all of them it gives the best accuracy-cost trade-off among structured planners: no baseline is simultaneously cheaper and more accurate, and DECAF improves accuracy by 4.2 points on average over the strongest structured-planner baseline while spending at $0.65\times$ its per-task cost. The advantage is not an artifact of a generous budget: in a budget sweep on AppWorld, DECAF at half the base budget already matches the accuracy the strongest baselines reach only at three times as much. Further analyses show that priced path selection matches the accuracy of the most expensive routes at near the cost of the cheapest, and that outcomes stay tightly clustered over 100 repeats per question.

2 Related Works

2.1 LLM Planning and Adaptive Test-Time Compute

LLM planners structure reasoning as explicit search over intermediate states, adding world-model simulation and feedback-driven re-planning in agentic settings (Yao et al., 2023a; Besta et al., 2024; Zhou et al., 2023; Hao et al., 2023; Shinn et al., 2023; Sun et al., 2023). This lineage is DECAF's substrate but is cost-blind: breadth is fixed, re-planning triggers on correctness rather than cost, and none models a budget that can be exhausted, even in the efficiency wave that meters search compute against a token cap rather than a priced ledger (Wang et al., 2024; Misaki et al., 2025). The closest instance is the plan-graph planner TAPE (Jeong et al., 2026), whose representation DECAF adopts but whose allocation policy (fixed breadth, re-plan on every mismatch, unpriced actions) DECAF replaces with a budget-aware one. A parallel line allocates inference-time compute by difficulty, from compute-optimal scaling (Snell et al., 2024) to per-problem thinking budgets (Muennighoff et al., 2025; Aggarwal & Welleck, 2025; Han et al., 2024; Shen et al., 2025) and confidence-gated stopping (Aggarwal et al., 2023; Li et al., 2024; Fu et al., 2024), but governs sampling for single-turn generation rather than a multi-step plan.

2.2 Cost-Aware Agents, Routing, and Planning

Query-level routers and cascades cut cost by picking a model before execution (Chen et al., 2023; Ong et al., 2024; Ding et al., 2024; Madaan et al., 2023) but decide once and never track a task budget. Closer to DECAF, within-task methods condition search on the remaining budget (Liu et al., 2025; Li et al., 2026) (though under a uniform per-task cap rather than sizing planning to instance difficulty), or learn stepwise tier escalation (Lee et al., 2025; Zheng et al., 2024), and classical cost-aware planning seeks the cheapest way to reach a goal it cannot fully afford (Keyder & Geffner, 2009; Smith, 2004; Boutilier et al., 1999), but each moves only one of DECAF's axes. DECAF instead couples adaptive plan-breadth sizing, a priced divergence replan gate, and priced path selection against one budget ledger, with difficulty read from in-loop escalation and an explicit budget-exhaustion failure mode.

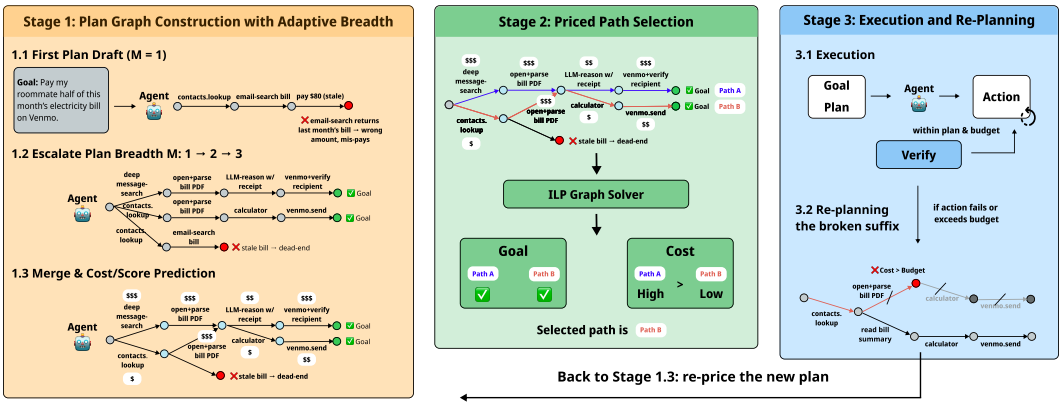

Figure 1: **Overview of DECAF** on an AppWorld task. Actions carry a price ($/$$/$$$; from the per-benchmark cost table, Table 3), nodes a score. **Stage 1:** escalate breadth (M:1→3) until a goal path appears, then merge drafts into one priced, scored plan graph. **Stage 2:** an ILP drops dead-ends and picks the goal-reaching path with the best score net of cost within budget (Path B). **Stage 3:** execute and verify while within plan and budget; on failure or overspend, re-plan only the broken suffix and reuse the prefix, then re-enter Stage 1 to re-price and re-select (bottom arrow).

3 METHOD

3.1 OVERVIEW

DECAF is a framework for an agent that solves a task through a sequence of tool calls under a *budget*, a ceiling on how much it may spend before returning an answer. This budget is *exogenous*: it is fixed at the operator level by deployment economics (an API spend cap, a rate limit, or a latency deadline) and inherited per task, rather than estimated from the task's difficulty, so setting it needs no query-complexity prior. In fact, having no such prior is DECAF's operating assumption: rather than sizing the cap to a task, DECAF takes whatever (typically uniform) budget it is given and decides how to spend it, discovering difficulty online through escalation (Stage 1). It treats the budget as a *ceiling, not a target*: it stops early on easy tasks and draws down only what a hard one needs, so even a loosely set, uniform cap does not force overspending. The central idea is that planning itself has a cost, and an agent should decide how much of it to pay the way it would spend a finite budget. DECAF runs a plan-graph cycle in three stages (Figure 1), pricing each decision against a shared budget ledger: it *builds* a plan graph whose breadth grows only as a task demands (Stage 1), *selects* the goal-reaching path with the best score net of cost (Stage 2), and *executes* the path, re-planning only when a fresh plan is worth its cost (Stage 3). Every action carries a price and all spend is charged to one cost tracker, so any planning DECAF does to save money is itself counted against the budget.

3.2 STAGE 1: PLAN-GRAPH CONSTRUCTION

DECAF plans over a *plan graph*: several candidate plans merged into one graph that shares the steps they agree on. This graph-structured reasoning follows Graph of Thoughts (Besta et al., 2024) and takes the plan-graph form of Jeong et al. (2026), whose representation we adopt rather than contribute. Construction runs three model calls (Figure 1, Stage 1): the agent *drafts* M candidate plans, each a full action sequence from the current state to the goal; *simulates* them with the model as a world model, merging equivalent states into one node; and *labels* each node with a coarse score and each action with a cost. A single base model performs all three calls, so every result reflects the planner rather than the model. Two states merge when they agree on the task-relevant slots the benchmark exposes: held objects and location in ALFWorld and Sokoban, resolved entities in MuSiQue, and satisfied sub-goals in AppWorld and MCP-Universe. **Acyclicity** is a precondition, not a detail. Stage 2 finds the cheapest goal-reaching path, which is a finite longest-path problem only on a DAG, and a single back-edge would let paths loop and break the integer program and its topological solver (Appendix A.4). DECAF keeps the graph acyclic by construction: it admits a

merge only when it adds no edge into an ancestor, and leaves equivalent states separate whenever a merge would close a cycle. The simulation is a rollout, not real execution, so branching comes from drafts diverging and merging rather than from expanding a live node, and the executed *path* can combine drafts into a cheaper route than any single one. Following Tree-of-Thoughts (Yao et al., 2023a), the score is a coarse ordinal label, $+1$ promising, 0 uncertain, or -1 doomed, which language models assign reliably where a raw scalar would not; a ground-truth signal such as boxes on target in Sokoban overrides it when available. Because drafts reach the same state through different tools, the graph often offers several routes to a node at different prices, and the per-node scores and per-action costs are what let Stage 2 choose among them.

Budget-aware breadth. Prior plan-graph planners fix M and pay for the same number of drafts on every task. DECAF instead sets it by escalation: starting at $M = 1$, it commits once the graph contains a goal-reaching path and otherwise adds one more plan, carrying the failed drafts as context so the new one differs, up to $M_{\max} = 8$. Difficulty is thus never measured separately: it is simply how many rounds a task needs before a goal-reaching path appears. Escalation seeks a goal-reaching path, not the cheapest one. If the first draft reaches the goal by an expensive route, DECAF commits it rather than paying for more drafts whose expected saving rarely covers their priced cost. Stage 2 still picks the cheapest route already in the graph and Stage 3 re-plans only if that route overspends or diverges, so DECAF settles for a cheap-enough plan rather than the globally cheapest one. A *solvency gate* caps escalation: before adding a plan, DECAF checks the budget can still afford another round plus enough execution cost-to-go to finish, and otherwise commits the best plan it has. This is the first of two defenses against exhausting the budget before answering; the second is the Stage 3 re-plan gate.

3.3 Stage 2: Cost-based Path Selection

Several paths may reach the goal, and the highest-scoring one is often the most expensive. DECAF prefers the goal-reaching path with the best score net of priced cost, which favors a cheaper route whenever it does not sacrifice goal-reaching. Each action's cost $c(a_v)$ is what executing it bills, its token count times the tier's per-token rate plus any per-call tool fee, read from a fixed per-benchmark table (Table 3); an action is costlier when it does costlier work, so a dense-retrieval hop with an LLM rerank exceeds a sparse lookup. Where a benchmark already defines these prices we adopt them, and otherwise we assign relative values and confirm the result is insensitive to them (Appendix B.3). Because the plan graph is a DAG, choosing this path is a constrained longest-path problem, solved by a small integer program. With binary indicators $y_{v,\ell}$ that node v is taken at depth ℓ, node score $s(v)$, per-action cost $c(a_v)$, discount $\gamma \in (0, 1]$, dollar-to-value rate $\lambda_\$$, and remaining budget B,

$$\max_{y} \sum_{\ell} \sum_{v} \left(\gamma^{\ell-1}\, s(v) - \lambda_\$\, c(a_v)\right) y_{v,\ell} \quad \text{s.t.} \quad \sum_{\text{goals } g,\, \ell} y_{g,\ell} = 1, \qquad \sum_{v,\ell} c(a_v)\, y_{v,\ell} \le B,$$

with flow constraints linking the selected nodes into a single start-to-goal path; the full program is in Appendix A.4. Cost enters twice: the budget B rules out unaffordable paths, and $\lambda_\$$ makes cost a soft penalty inside the objective, so among affordable goal-reaching paths the solver trades a small drop in score for a large saving in cost. Score is discounted by depth while cost is not: a goal reached sooner is preferable other things equal, whereas each action's dollar cost is a resource spent in full regardless of where on the path it falls. This selection is inert when action costs are uniform, as in Sokoban and ALFWorld, and drives the savings where costs differ widely, as with the retriever tiers in MuSiQue. Hyperparameter values are in Appendix A.5.

3.4 Stage 3: Execution and Re-planning

DECAF runs the selected path while verifying, after each action, that the realized state and spend still match the plan (Figure 1, Stage 3). A re-plan is a full construction call that re-drafts, re-simulates, and rebuilds the graph, so a reflexive planner that re-plans on every mismatch pays this repeatedly. DECAF instead gates the re-plan with a *priced divergence test*, a cheap proxy for the metareasoning principle that a computation is worth running only when its expected value exceeds its cost (Russell & Wefald, 1991; Hansen & Zilberstein, 2001): rather than estimating the expected gain of re-planning, which we do not attempt, it thresholds how far the plan has drifted and weighs that drift against the priced cost of a fresh plan. On a mismatch it estimates the value-loss, how

much closer to the goal the plan predicted being than the real state is,

$$\text{value-loss} = \max\big(0,\ \text{progress}(\text{predicted}) - \text{progress}(\text{real})\big),$$

and re-plans only when an action fails outright, when its realized cost overruns what the budget can still afford, or when the value-loss clears the priced threshold:

$$\text{re-plan} \iff \text{action failed or over budget or } \big(\text{value-loss} \geq \delta_{\min} \wedge \text{value-loss} > \lambda_{\$}\, c_{\text{replan}} + \text{slack}\big).$$

$\text{progress}(\cdot) \in [0, 1]$ is a normalized goal-progress signal, so value-loss and the priced term $\lambda_{\$}\, c_{\text{replan}}$ live on one $[0, 1]$ scale through the same $\lambda_{\$}$ used in Stage 2. The remaining budget enters this decision through two hard triggers rather than the divergence threshold: the over-budget trigger fires only when the current route can no longer be afforded, and the Stage 1 solvency gate (Section 3.2) refuses any re-plan it cannot pay for and commits the plan in hand instead. The divergence threshold $\lambda_{\$}\, c_{\text{replan}} + \text{slack}$ is a fixed per-benchmark quantity; we do not additionally scale it with the budget still left, and found a fixed threshold sufficient in our experiments. When it does re-plan, DECAF keeps the executed prefix and repairs only the broken suffix: it re-enters Stages 1 and 2 from the divergence node, re-drafting and re-pricing only the actions after it and re-selecting a path, while the committed prefix is reused rather than re-paid. Per-benchmark definitions of $\text{progress}(\cdot)$ and the values of $\delta_{\min}$, slack, $\lambda_{\$}$, and c_{replan} are in Appendix A.5.

4 Experiments

4.1 Setup

All arms run on one base model, Claude Fable 5, through an identical agent loop and tool interface, so any difference reflects the planner rather than the model; Appendix B.1 repeats the comparison at a cheaper tier, Haiku 4.5 and Opus 4.5. We evaluate on five benchmarks that together span synthetic and real tools, reversible and irreversible dynamics, and uniform and heterogeneous action costs: **AppWorld** (Trivedi et al., 2024), **ALFWorld** (Shridhar et al., 2020), **Sokoban** (Racanière et al., 2017), **MuSiQue** (Trivedi et al., 2022), and **MCP-Universe** (Luo et al., 2025). We compare against seven planners that each fix one or more of the decisions DECAF makes budget-aware: **ReAct** (Yao et al., 2023b), **Plan-and-Act** (Erdogan et al., 2025), the plan-graph **TAPE** (Jeong et al., 2026), the tree-search **RAP** (Hao et al., 2023) and **CATS** (Zhang & Liu, 2025), and the budget-aware **BATS** (Liu et al., 2025) and **BAVT** (Li et al., 2026). Each runs at its reported-best search setting, detailed in Appendix A.2: ReAct and Plan-and-Act take a single trajectory, TAPE fixes breadth at $M{=}5$, RAP and CATS expand 16 rollouts per decision, and BATS and BAVT follow their default budget schedule; of these only BATS, BAVT, and DECAF track the remaining budget, and DECAF draws fewer candidate plans than the multi-rollout planners, so its gain reflects budget-aware allocation rather than more search. We report **accuracy**, each benchmark's native success metric, against **cost** in cents per task from a fixed per-benchmark table, and an LLM judge on a separate budget settles ambiguous answers. All spend, including the tokens DECAF burns while planning and gating, is charged to one tracker, so a planner that over-invokes its machinery reports as less cost-efficient; every arm that can use a progress signal receives the same one, the environment's where available and the shared LLM-judge estimate otherwise. Table 1 is uncapped, so its gaps reflect planning quality rather than budget avoidance; we set each benchmark's base $1\times$ budget to the mean cost of a successful uncapped run and scale it uniformly across arms in the sweep of Section 4.3. Appendix A.3 gives the cost tables, benchmark details, and the budget-exhaustion metric.

4.2 Main Results

DECAF gives the best accuracy-cost trade-off. Uncapped, no baseline in our experiments is both cheaper and more accurate than DECAF (Figure 2). Because DECAF also reaches the highest or tied-highest accuracy on all five benchmarks, this non-domination is expected on the accuracy axis; the substantive result is on the cost axis, where DECAF matches or beats each structured planner's accuracy while spending strictly less. Table 1 gives the numbers: its mean gain is $+4.2$ points over TAPE, the strongest structured planner, and $+3.9$ over the per-benchmark strongest baseline, with a tie on MCP-Universe, at $0.65\times$ TAPE's per-task cost.

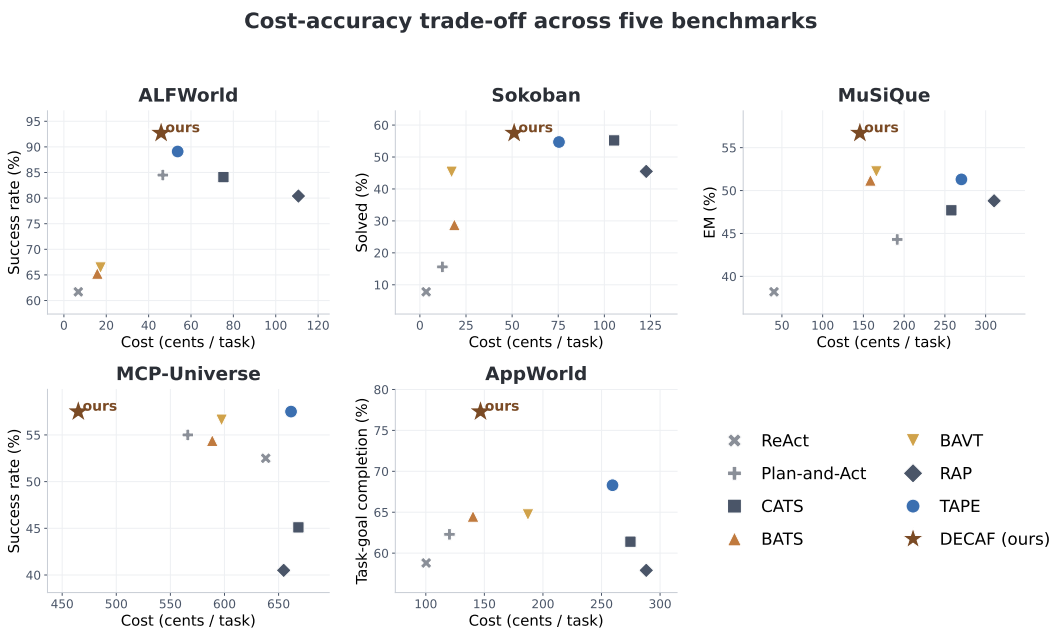

Figure 2: **Cost-accuracy trade-off across five benchmarks.** Per-task cost (x, cents) against accuracy (y, native measure); top-left is best. DECAF (⋆, "ours") is not dominated by any baseline in any panel. All arms share one base model and agent loop, so position reflects the planner, not the model.

Method	ALFWorld	Sokoban	MuSiQue	MCP-Universe	AppWorld
			Accuracy (↑)		
ReAct	61.7	7.8	38.2	52.5	58.8
Plan-and-Act	84.5	15.6	44.3	55.0	62.3
CATS	84.1	55.2	47.7	45.1	61.4
BATS	65.3	28.8	51.2	54.4	64.5
BAVT	66.4	45.3	52.2	56.6	64.7
RAP	80.4	45.5	48.8	40.5	57.9
TAPE	89.1	54.7	51.3	**57.5**	68.3
DECAF	**92.7**	**57.5**	**56.7**	**57.5**	**77.3**
			Cost (cents per task, ↓)		
ReAct	**6.8**	**3.4**	**40.3**	638.1	**100.3**
Plan-and-Act	46.7	12.2	191.5	566.1	120.2
CATS	75.3	105.3	257.9	668.3	274.7
BATS	15.8	18.7	158.6	588.7	140.3
BAVT	17.3	17.2	165.9	597.3	187.2
RAP	110.7	122.7	310.4	654.7	288.2
TAPE	53.7	75.4	270.3	661.5	259.4
DECAF	45.9	51.1	145.7	**464.9**	146.7

Table 1: **Cost-effectiveness across five benchmarks.** The upper block reports **accuracy** (each benchmark's native success measure, higher is better) and the lower block reports **cost** in cents per task (lower is better). Best accuracy and lowest cost among structured planners are in **bold**. Cost is the token-cost ledger in cents per task (the unified cross-benchmark metric of Appendix A.3).

How the baselines trade off cost and accuracy. DECAF is not the single cheapest arm: on every benchmark but MCP-Universe, ReAct spends less by barely planning, which leaves it low-cost and low-accuracy; on the longest horizon, MCP-Universe, even that frugality reverses as its trajectory outgrows DECAF's committed plan. CATS and RAP reach middling accuracy at the highest cost almost everywhere, since each rollout is a full model call (Hao et al., 2023; Zhang & Liu, 2025).

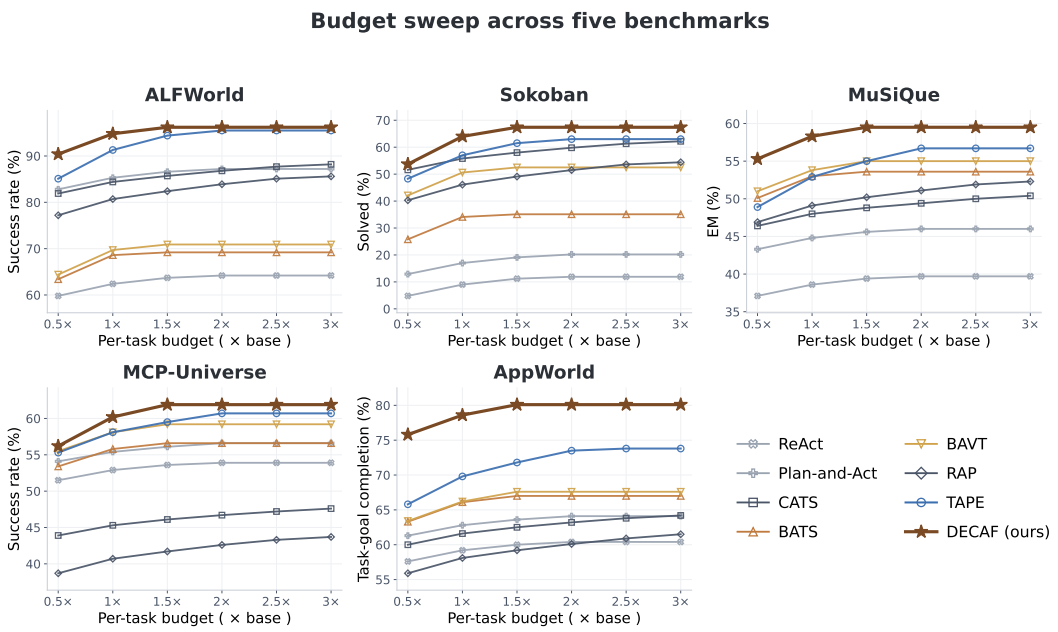

Figure 3: **Budget sweep across five benchmarks.** Per-task budget from $0.5\times$ to $3\times$ base (x) against accuracy (y). Each method plateaus once its actual spend saturates, so a larger cap cannot help it further. DECAF ($\star$, "ours") plateaus early yet stays highest at every budget. The search-based planners (CATS, RAP) keep climbing to nearly the full cap, since their tree search uses whatever budget it is given, and still fall short. A larger cap is headroom only the most expensive methods use; what matters is how efficiently the budget is spent, not how much is available.

TAPE spends 1.17–1.86$\times$ DECAF's budget with fixed breadth and no cost term, and the budget-aware BATS and BAVT are frugal on QA-like benchmarks but trail DECAF, sizing neither breadth to difficulty nor path to cost. The pattern holds at a cheaper tier (Haiku 4.5, Opus 4.5; Appendix B.1).

4.3 Robustness to the Budget Choice

A natural concern is that the base budget (Section 4.1) was chosen to favor DECAF, so that its lead is an artifact of one operating point. To address this, we vary the per-task budget from $0.5\times$ to $3\times$ its base value and re-run every arm at each setting (Figure 3).

DECAF leads at every budget. Across the sweep DECAF stays at or above every baseline (tying TAPE on MCP-Universe), so its lead does not depend on the budget chosen. Its curve is highest even at the generous $3\times$ end, so a larger budget does not let the baselines catch up, and it saturates early, reaching most of its accuracy well below the base budget. Read along the budget axis, this means the baselines need several times the budget to approach DECAF's base-budget accuracy: on AppWorld, DECAF at half the base budget already matches the strongest baselines given $3\times$ as much. The advantage is therefore a property of how efficiently DECAF spends, not of the operating point we chose.

A larger cap is headroom, not a requirement. A related objection is that if accuracy rises when the cap is tripled, DECAF must "need" $3\times$ the budget. It does not. Table 4 reports how much each method actually spends at a $3\times$ cap, relative to its base spend; a method that consumed the entire enlarged cap would report 3.0. DECAF reports about 1.3, so tripling the cap raises its actual spend by only about a third. Accuracy still rises with that increment, but the increment is spent almost entirely on the hard tail of instances that the base cap cut off, while easy instances stop early and leave the added budget unused. The cap is a ceiling, not a target: a larger cap lets a few hard instances finish rather than making every instance more expensive. The contrast with the search-

based planners is sharp, since CATS and RAP consume close to the full cap (2.7–2.9×), so their spend scales with whatever cap they are given, whereas DECAF's does not.

Exhaustion under tight budgets. To measure how often each method exhausts its budget before answering, we re-run every arm under per-task caps and separate *bankrupt* runs, which breach a meter and end with no answer, from runs that breach but still answer. Table 5 reports the bankruptcy rate at $0.5\times$ and $1\times$ the base budget across all five benchmarks. DECAF bankrupts least everywhere, on 1–4% of runs at $1\times$ and 6–13% at $0.5\times$, against 4–33% and 14–61% for the baselines, and the gap widens as the cap tightens. The tree-search arms (CATS, RAP) exhaust most, since they spend a fixed rollout budget before acting; the budget-aware siblings BATS and BAVT are the closest baselines but still bankrupt two to three times as often, because a uniform per-task cap cannot shift spend toward the instances that need it. Exhaustion is highest on the long-horizon, high-cost benchmarks (MCP-Universe, MuSiQue) and lowest on the cheap ones (Sokoban, ALFWorld), but DECAF's ordering holds on every one: its two defenses, the Stage 1 solvency gate and the Stage 3 priced divergence gate, convert runs that would otherwise bankrupt into answers.

4.4 Where the Savings Come From

The main results raise a natural question: is DECAF more cost-effective because it avoids running out of budget, or because it plans better? Table 1 is uncapped, so no run can exhaust its budget there; its gaps must come from how DECAF plans. We isolate that planning advantage below, comparing against TAPE throughout, since the metrics here (candidate-plan count, valid-plan ratio, re-plan recovery) are defined only for methods that build a plan graph, and TAPE is the one baseline that shares DECAF's plan-graph structure.

Fewer plans, and cheaper recovery. DECAF drafts far fewer candidate plans than the fixed-breadth planner and wastes far less of what it drafts (Table 6): on AppWorld it draws a mean of 2.3 candidate plans per task versus 7.0, and a larger share of them are valid, meaning executable to the goal (78.8% versus 42.5%). Recovery is also cheaper (Table 6): TAPE re-plans on every execution mismatch, while DECAF's priced divergence gate re-plans only when the recovered value justifies the cost, so on Sokoban it makes fewer than half as many re-plan attempts, recovers on a larger fraction of them, and spends roughly $11\times$ less per recovery. Localizing where runs fail confirms that the residual accuracy gap is a planning-quality effect rather than a recovery one: on Sokoban 80% of failures occur at the plan stage, where no goal-reaching plan was found, and DECAF fails there far less often than TAPE. DECAF wins by producing goal-reaching plans more often and repairing them more cheaply, not by outlasting the budget.

Each axis contributes on its own. A leave-one-axis-out ablation (Table 7) confirms that no single mechanism accounts for the result. Removing adaptive breadth, the priced divergence gate, or priced path selection each lowers cost-effectiveness, and the axes act in different regimes: adaptive breadth and the priced divergence gate mainly cut cost on the spatial and long-horizon benchmarks where they fire, while priced path selection matters most on MuSiQue, whose retriever tiers differ widely in cost. Only the full three-axis method gives the best accuracy-cost trade-off across all benchmarks.

5 Analyses

5.1 Does Priced Path Selection Help?

Stage 2 selects the goal-reaching path that maximizes score net of cost, but this raises the question of whether a cheaper path costs accuracy. To isolate the effect, we fix the plan graph and vary only the path-selection rule, comparing DECAF's priced selection against four alternatives: pick the highest-score path, the most expensive path, the cheapest path (ignoring score), or a random goal-reaching path. Figure 4 plots the result on the three benchmarks whose actions differ in cost; on the uniform-cost benchmarks (Sokoban, ALFWorld) every rule reduces to shortest-path and the comparison is degenerate, so we omit them. The priced selection sits at the top-left of every panel: it reaches essentially the same accuracy as the highest-score and most-expensive rules while spending far less. On MuSiQue, for example, it matches the highest-score rule within 0.7 points of accuracy at roughly half the cost (145.7 versus 268.9 cents). Selecting the cheapest path outright is not a

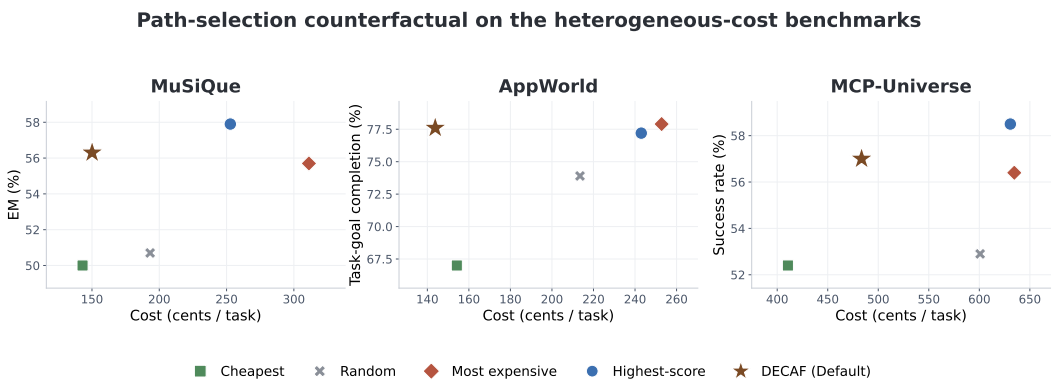

Figure 4: **Path-selection counterfactual.** On the same plan graph, we vary only the path-selection rule; DECAF (Default) selects the best score-net-of-cost path within budget. DECAF matches the accuracy of the highest-score and most-expensive rules at far lower cost, while selecting the cheapest path outright loses accuracy, so priced selection is what buys efficiency without sacrificing accuracy.

substitute, since it drops accuracy sharply (to 48.9 on MuSiQue, 68.4 on AppWorld, 50.8 on MCP-Universe) by taking routes that are cheap but do not reliably reach the goal. Priced selection is therefore what converts a plan graph into an efficient plan: it keeps the accuracy of the expensive routes at close to the cost of the cheapest one, rather than trading accuracy for cost or paying for accuracy it does not need.

5.2 How Deterministic Is DECAF's Planning?

Because the plan graph is built by a language model, a fair concern is that DECAF's results depend on lucky sampling: a different run might draft different plans and reach a different outcome. We do not claim an identical plan every time; we show that the *outcome* is stable. We run each question 100 times and record the resulting cost and accuracy (Figure 5). The runs form a tight cluster on every benchmark: the standard deviation of accuracy is small (roughly 1.3 to 1.6 points) and that of cost is a few percent of the mean, so repeating the same question rarely changes where DECAF lands on the cost-accuracy plane. The plan text may differ from run to run, but the budget-aware decisions that produce it, how far to escalate breadth and when to re-plan, converge to the same regime, which is why the reported numbers are representative rather than a single fortunate draw.

5.3 Further Analyses

Appendix B.1–B.9 stress-test the result: the advantage holds across model tiers and families, is significant over $N{=}5$ seeds, survives perturbing the cost table, $\lambda_\$$, and γ, and is explained neither by capping alone, an uncharged progress signal, nor answer emission at exhaustion; the three axes compose super-additively, with difficulty-sized breadth a real gain over a uniform cap.

6 Conclusion

We presented DECAF, a plan-graph framework that treats an agent's planning as a finite budget to allocate, pricing three coupled decisions, how much to plan, when to re-plan, and which path to commit to, against one shared budget ledger. Across five benchmarks DECAF gives the best accuracy-cost trade-off among structured planners, improving accuracy by 4.2 points on average over the strongest structured-planner baseline at $0.65\times$ the per-task cost, and its lead holds across a full budget sweep. Our analyses trace the win to the coupling of the three axes: each helps alone, and combining them against one budget adds more than their sum. DECAF currently prices actions from a fixed cost table rather than predicting cost online, and reads difficulty from escalation rather than a learned estimate; learning these signals, and extending the budget view to latency alongside dollars, are natural next steps.

AI USE STATEMENT

In this work, we used generative AI tools for assisting the writing of the paper and code. We have not used generative AI tools for conducting the experiments and the rest of the required disclosure tasks are not applicable to this work. We have reviewed all AI-assisted work. For example we reviewed the text and citations that there are any other hallucinated text and citations. And we also did sanity check of the code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies how a planning agent allocates a finite compute and cost budget across the stages of solving a task, and it raises no concerns under the ICLR Code of Ethics. The research involves no human subjects, no user studies, and no collection of personal or sensitive data, so no IRB review or consent process applies. All experiments use publicly available academic benchmarks under their respective licenses and intended research use; we release no new dataset and scrape no private information. DECAF is a cost-aware planning layer over an existing language model and introduces no new capability for generating harmful, deceptive, or unsafe content beyond that of the underlying model; our contribution is to make an agent spend its budget more efficiently, which if anything reduces wasted computation and its associated energy and monetary cost. The method does not target or profile individuals and has no application to surveillance, discrimination, or other high-risk settings, so we do not foresee fairness, bias, privacy, or security harms arising from it. We report all costs on a single instrumented ledger and disclose our pricing assumptions, hyperparameters, and evaluation protocol in the appendix in the interest of research integrity and reproducibility. The authors declare no conflicts of interest or external sponsorship that could have influenced this work.

REPRODUCIBILITY STATEMENT

We have taken several steps to make DECAF reproducible. The full three-stage algorithm is specified in Section 3, with the priced path-selection integer program and its flow constraints written out in Appendix A.4 and every hyperparameter ($\lambda_\$$, γ, $\delta_{\min}$, slack, $M_{\max}$, and c_{replan}) listed in Appendix A.5. The experimental setup is described in Section 4.1: all methods share one base model and one agent loop, and we report the exact benchmarks, their evaluation metrics, and the per-benchmark cost tables and pricing assumptions used to compute every reported figure. The LLM-judge protocol used to score open-ended answers is given in Appendix A.6. All benchmarks are publicly available and used under their standard settings, and the data-processing and prompting details needed to reproduce each run are provided in the appendix. Anonymized source code for the DECAF planner, the cost tracker, and the evaluation harness is included in the supplementary materials.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *ArXiv*, abs/2503.04697, 2025. URL https://api.semanticscholar.org/CorpusID:276813519.

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:258823191.

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.

Craig Boutilier, Thomas L. Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artif. Intell. Res.*, 11:1–94, 1999. URL https://api.semanticscholar.org/CorpusID:5297450.

Lingjiao Chen, Matei A. Zaharia, and James Y. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *ArXiv*, abs/2305.05176, 2023. URL https://api.semanticscholar.org/CorpusID:258564349.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *ArXiv*, abs/2404.14618, 2024. URL https://api.semanticscholar.org/CorpusID:269303119.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2503.09572.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yi Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling llm reasoning with certaindex. 2024. URL https://api.semanticscholar.org/CorpusID:275133390.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyun Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:274992044.

Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1–2):139–157, 2001.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:258865812.

Jongwon Jeong, Jungtaek Kim, and Kangwook Lee. Tape: Tool-guided adaptive planning and constrained execution in language model agents. *ArXiv*, abs/2602.19633, 2026. URL https://api.semanticscholar.org/CorpusID:285973454.

Emil Ragip Keyder and Hector Geffner. Soft goals can be compiled away. *J. Artif. Intell. Res.*, 36: 547–556, 2009. URL https://api.semanticscholar.org/CorpusID:10651521.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. *ArXiv*, abs/2511.06190, 2025. URL https://api.semanticscholar.org/CorpusID:282911823.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. *ArXiv*, abs/2401.10480, 2024. URL https://api.semanticscholar.org/CorpusID:267060971.

Yushu Li, Wenlong Deng, Jiajin Li, and Xiaoxiao Li. Spend less, reason better: Budget-aware value tree search for llm agents. *ArXiv*, abs/2603.12634, 2026. URL https://api.semanticscholar.org/CorpusID:286517499.

Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. Budget-aware tool-use enables effective agent scaling. *ArXiv*, abs/2511.17006, 2025. URL https://api.semanticscholar.org/CorpusID:283225884.

Ziyang Luo, Zhiqi Shen, Wei Yang, et al. Mcp-universe: Benchmarking large language models with real-world model context protocol servers, 2025. arXiv:2508.14704.

Aman Madaan, Pranjal Aggarwal, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Mausam, and Manaal Faruqui. Automix: Automatically mixing language models. *ArXiv*, abs/2310.12963, 2023. URL https://api.semanticscholar.org/CorpusID:264306222.

Kou Misaki, Yuichi Inoue, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. *ArXiv*, abs/2503.04412, 2025. URL https://api.semanticscholar.org/CorpusID:276813234.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Fei-Fei Li, Hanna Hajishirzi, Luke S. Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://api.semanticscholar.org/CorpusID:276079693.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, Mohammed Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *ArXiv*, abs/2406.18665, 2024. URL https://api.semanticscholar.org/CorpusID:270764307.

Sébastien Racanière, Théophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Manfred Otto Heess, Yujia Li, Razvan Pascanu, Peter W. Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. *ArXiv*, abs/1707.06203, 2017. URL https://api.semanticscholar.org/CorpusID:13301124.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1–3):361–395, 1991.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. Dast: Difficulty-adaptive slow-thinking for large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2322–2331, 2025.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems 36*, 2023. URL https://api.semanticscholar.org/CorpusID:258833055.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768, 2020. URL https://api.semanticscholar.org/CorpusID:222208810.

David E. Smith. Choosing objectives in over-subscription planning. In *International Conference on Automated Planning and Scheduling*, 2004. URL https://api.semanticscholar.org/CorpusID:36790171.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *ArXiv*, abs/2408.03314, 2024. URL https://api.semanticscholar.org/CorpusID:271719990.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *ArXiv*, abs/2305.16653, 2023. URL https://api.semanticscholar.org/CorpusID:258947337.

H. Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Raj Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *ArXiv*, abs/2407.18901, 2024. URL https://api.semanticscholar.org/CorpusID:271516633.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 10:539–554, 2022. arXiv:2108.00573.

Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. Litesearch: Efficacious tree search for llm. *ArXiv*, abs/2407.00320, 2024. URL https://api.semanticscholar.org/CorpusID:270869427.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601, 2023a. URL https://api.semanticscholar.org/CorpusID:258762525.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. arXiv:2210.03629.

Zihao Zhang and Fei Liu. Cost-augmented monte carlo tree search for llm-assisted planning, 2025. arXiv:2505.14656.

Yuanhang Zheng, Peng Li, Mingshi Yan, Ji Zhang, Fei Huang, and Yang Liu. Budget-constrained tool learning with planning. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:267938223.

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*, abs/2310.04406, 2023. URL https://api.semanticscholar.org/CorpusID:263829963.

A Implementation details

A.1 Benchmark Details

We evaluate on five benchmarks chosen to span the demands a planner must handle: synthetic and real tools, reversible and irreversible dynamics, and uniform and highly heterogeneous action costs.

AppWorld (Trivedi et al., 2024) runs multi-step tasks across simulated apps and APIs, where a single wrong plan can waste many downstream calls, stressing long-horizon tool use. **ALFWorld** (Shridhar et al., 2020) is a text-embodied household environment with long, partially observable action sequences. **Sokoban** (Racanière et al., 2017) is a grid puzzle whose irreversible dead-ends punish shallow planning, so a mistaken push cannot be undone. **MuSiQue** (Trivedi et al., 2022) is multi-hop question answering in which each hop is a priced retrieval step, and the retrievers span a wide cost range, which is what makes its path selection non-trivial. **MCP-Universe** (Luo et al., 2025) runs live tool use over real Model Context Protocol servers; we use its financial-analysis domain over yfinance, which needs no external API keys.

A.2 Baseline Details

Each baseline fixes one or more of the three decisions DECAF makes budget-aware. **ReAct** (Yao et al., 2023b) interleaves one reasoning step and one action per turn with no plan graph. **Plan-and-Act** (Erdogan et al., 2025) commits to a single upfront plan and executes it without re-planning. **TAPE** (Jeong et al., 2026), the closest comparison, is a plan-graph planner with fixed breadth that re-plans on every execution mismatch, so it isolates the value of DECAF's adaptive breadth and priced divergence gate. **RAP** (Hao et al., 2023) and **CATS** (Zhang & Liu, 2025) run Monte Carlo tree search, maximizing task reward and cost-folded reward respectively, but neither allocates a planning budget against a solvency constraint, so they are the cost-in-search foils for DECAF's priced path selection. **BATS** (Liu et al., 2025) and **BAVT** (Li et al., 2026) are the closest budget-aware siblings: both condition on the budget still left, but apply a uniform per-task cap rather than sizing planning to instance difficulty, and neither selects a priced goal-reaching path over a plan graph. For a fair cost comparison we set the tunable search knobs of each baseline to values commonly used in prior work: RAP and CATS run with 16 rollouts per decision and TAPE uses breadth $M{=}5$ (its reported best); these search knobs are held fixed across all runs. The per-task *budget*, by contrast, is what the sweep varies: in each sweep cell every budget-aware arm (BATS, BAVT, and DECAF) receives that cell's budget, so BATS and BAVT can respond to a larger cap exactly as DECAF does, and their plateau reflects their allocation policy rather than a frozen budget input.

A.3 Cost Model and Metrics

Model	Input ($/1M)	Output ($/1M)
Haiku 4.5	1.00	5.00
Opus 4.5	5.00	25.00
Fable 5	10.00	50.00

Table 2: Token pricing used for the unified token-cost ledger: USD per one million tokens for each model. The token-cost of a run is its measured input and output tokens multiplied by these rates.

Two ledgers. Every action carries a price $c(a)$ read from a fixed per-benchmark cost table, computed as its token count times the tier's per-token rate plus any per-call tool fee. Because the price is a table lookup rather than a learned prediction, the cost of a run is exact and re-computable: a logged trajectory can be re-priced for another region or model without re-executing it. We report two ledgers. **Token cost** is the pure LLM-token dollars every arm spends, and we use it as the unified efficiency metric across benchmarks. A **per-benchmark in-task cost** prices each action type, such as a tool call, a retrieval hop, or a grid move, and it is the cost the budget is set in and that triggers budget exhaustion. Where prior work on a benchmark already defines action costs we adopt its settings; where it does not, we assign the missing prices ourselves, chosen to be reasonable relative to one another. Table 2 gives the per-tier token rates behind the token-cost ledger, and Table 3

Benchmark	Cost regime	Priced action type	Unit cost (¢)
AppWorld	heterogeneous	app read / query (list, get)	0.5
		app write / action (send, post, pay)	1.0
		cross-app search / document parse	3.0
ALFWorld	uniform	environment action	1.0
Sokoban	uniform	grid move / push	1.0
MuSiQue	heterogeneous	BM25 sparse retrieval	0.5
		dense retrieval	2.0
		dense retrieval + LLM read/rerank	6.0
MCP-Universe	heterogeneous	single-endpoint quote lookup	0.5
		historical / multi-point data pull	1.5
		multi-endpoint aggregation + synthesis	4.0

Table 3: **Per-benchmark in-task action costs.** Each priced action type carries a unit cost in cents.

lists the per-benchmark in-task action costs. Reasoning and planning steps that emit no tool call are priced by tokens alone (Table 2).

Charging the planner's own spend. All spend, including the tokens DECAF itself burns while drafting plans, simulating them, and running its gates, is charged to a single cost tracker. A planner that over-invokes its own machinery therefore reports as less cost-efficient rather than getting a free pass. This planning spend counts against the same budget on which exhaustion is defined, so a planner that over-thinks can bankrupt itself; this is exactly the failure mode DECAF's solvency gate and priced divergence gate are designed to prevent.

Metrics. **Accuracy** is each benchmark's native success measure: task-goal completion for AppWorld, task success rate for ALFWorld and MCP-Universe, fraction of puzzles solved for Sokoban, answer EM/F1 for MuSiQue. Exact-match scoring runs first, and an LLM judge with its own isolated budget settles ambiguous answers without ever drawing on the agent's task budget. **Cost** is reported on the two ledgers above, lower being better. We additionally track the **budget-exhaustion rate**, the fraction of tasks that spend the whole budget before producing an answer, which separates an allocation failure from a capability failure: an exhausted run fails by running out of budget rather than by being unable to solve the task.

A.4 Priced-Path ILP: Variables and Flow Constraints

This appendix expands the Stage 2 integer program. Let $G = (V, E)$ be the plan-graph DAG produced by Stage 1, with a single start node v_0 and a set of goal nodes $G^\star \subseteq V$. Depth $\ell \in \{1, \dots, L\}$ indexes position along the executed path, with L an upper bound on path length taken from the graph diameter. We use two binary families: $y_{v,\ell} \in \{0, 1\}$ indicates node v is visited at depth ℓ, and $x_{e,\ell} \in \{0, 1\}$ indicates edge $e = (u \to v)$ is traversed from depth ℓ to $\ell + 1$. Node score $s(v)$ is the three-way $\{-1, 0, +1\}$ tag emitted by the Stage 1 simulator, rescaled to $[0, 1]$; $c(a_v)$ is the priced cost of the action attached to v.

The full program is

$$\max_{y} \sum_{\ell} \sum_{v} \left(\gamma^{\ell-1}\, s(v) - \lambda_{\$}\, c(a_v)\right) y_{v,\ell}$$

subject to

$$y_{v_0,1} = 1, \qquad \text{(start pinned)} \tag{1}$$

$$\sum_{g \in G^\star} \sum_{\ell=1}^{L} y_{g,\ell} = 1, \qquad \text{(exactly one goal)} \tag{2}$$

$$\sum_{v \in V} y_{v,\ell} \le 1 \qquad \forall\, \ell, \qquad \text{(one node per depth)} \tag{3}$$

$$y_{v,\ell} \; - \sum_{e:\, \mathrm{tgt}(e)=v} x_{e,\ell-1} = 0 \qquad \forall\, v \neq v_0,\ \ell \ge 2, \qquad \text{(enter node via one edge)} \tag{4}$$

$$\sum_{e:\, \mathrm{src}(e)=v} x_{e,\ell} \; - \; y_{v,\ell} = 0 \qquad \forall\, v \notin G^\star,\ \ell < L, \qquad \text{(leave node via one edge)} \tag{5}$$

$$\sum_{\ell, v} c(a_v)\, y_{v,\ell} \le B. \qquad \text{(budget)} \tag{6}$$

Line (4) and (5) together form the flow-conservation constraints that link chosen nodes into a single start-to-goal walk: a non-goal, non-start node is entered by exactly one edge iff it is left by exactly one edge, and a goal node is entered but not required to be left. Under uniform action costs, inactive B, and $\gamma = 1$, the program reduces to a longest-path solve on the DAG; for $\gamma < 1$ the depth discount makes it a discounted-score maximization over goal-reaching paths. We solve it with CBC branch-and-bound for graphs up to $|V| \approx 10^2$ and $L \le 12$, sizes well inside those produced by Stage 1; a topological-order dynamic program on the linearized DAG returns the same optimum when a solver is unavailable.

A.5 Hyperparameters and Cost Settings

Values used in all reported runs. The dollar-to-value rate $\lambda_\$$ is the only parameter tying the Stage 2 objective and the Stage 3 gate to the same $[0, 1]$ scale; we fix it per benchmark to the reciprocal of the base uncapped mean cost, so one unit of node score is worth roughly one base-run budget. This makes the priced term non-trivial without letting cost dominate score.

Symbol	Value	Where used	Role
γ	0.5	Stage 2 objective	depth discount on node score
$M_{\max}$	8	Stage 1	cap on plan-graph breadth
$\lambda_\$$	$1/\bar{c}_{1\times}$ per bench.	Stages 2, 3	normalizes value against cost
$\delta_{\min}$	0.15	Stage 3 gate	suppresses harmless drift
slack	0.05	Stage 3 gate	hysteresis on borderline mismatches
c_{replan}	measured per bench.	Stage 3 gate	priced cost of one Stage 1 build

$\bar{c}_{1\times}$ is the per-benchmark mean cost of a successful uncapped base-model run, matching the definition of the $1\times$ budget in Appendix A.3. c_{replan} is measured as the mean planner cost of one Stage 1 build on the same benchmark. All hyperparameters are set once and then frozen. The global constants (γ, $M_{\max}$, $\delta_{\min}$, slack) take the same value across all five benchmarks and were fixed on a small held-out development subset disjoint from the evaluation instances. The per-benchmark quantities ($\lambda_\$$, the $1\times$ budget, and c_{replan}) are derived from $\bar{c}_{1\times}$, the mean cost of a successful uncapped base-model run measured on that held-out split rather than on the test set, so no evaluation-set outcome enters the tuning and DECAF's operating point is not calibrated on the data it is scored on.

Progress signal per benchmark. $\mathrm{progress}(\cdot) \in [0, 1]$ is rescaled from the closest native goal-progress measurement each benchmark exposes. Sokoban: fraction of boxes on target. ALFWorld: fraction of declared subgoals completed. AppWorld and MuSiQue: fraction of task-required entities and sub-actions completed, as scored by the LLM judge described in Appendix A.6. MCP-Universe: the benchmark's own graded success signal. Where a benchmark exposes no native progress measure, the Stage 1 three-way node score $\{-1, 0, +1\}$ is mapped to $\{0, 0.5, 1\}$ and used in its place.

Method	ALFW.	Soko.	MuSiQue	MCP-U	AppW.
ReAct	1.7	1.7	1.7	1.8	1.8
Plan-and-Act	1.8	1.8	1.8	1.9	1.9
CATS	2.8	2.8	2.9	2.9	2.9
BATS	1.1	1.1	1.2	1.3	1.3
BAVT	1.2	1.2	1.4	1.4	1.5
RAP	2.7	2.7	2.8	2.8	2.9
TAPE	1.7	1.7	2.0	2.0	2.1
DECAF	1.3	1.3	1.4	1.4	1.5

Table 4: **Actual mean spend at a $3\times$ budget cap, as a multiple of the base ($1\times$) spend.** A method that used the entire enlarged cap would report 3.0. Search-based planners (CATS, RAP) consume almost the whole cap, whereas DECAF and the other budget-aware methods spend far below it.

	ALFWorld		Sokoban		MuSiQue		MCP-U		AppWorld	
Method	$0.5\times$	$1\times$	$0.5\times$	$1\times$	$0.5\times$	$1\times$	$0.5\times$	$1\times$	$0.5\times$	$1\times$
ReAct	28	10	24	8	33	13	38	16	35	14
Plan-and-Act	24	8	21	7	29	11	34	14	30	11
CATS	46	22	43	19	55	28	61	33	52	26
RAP	43	19	40	17	51	25	58	30	48	23
BATS	18	5	16	4	24	8	29	11	22	7
BAVT	16	4	14	4	22	7	27	10	20	6
TAPE	25	9	22	8	31	13	36	16	28	12
DECAF	**7**	**2**	**6**	**1**	**10**	**3**	**13**	**4**	**9**	**2**

Table 5: **Bankruptcy rate across five benchmarks** (% of runs that breach a meter and end with no answer), at per-task caps of $0.5\times$ and $1\times$ the base budget. Lower is better. DECAF bankrupts least on every benchmark, and its gap to every baseline widens as the cap tightens; the tree-search arms (CATS, RAP) exhaust most, while the budget-aware siblings (BATS, BAVT) are the closest baselines yet still bankrupt two to three times as often.

A.6 LLM-Judge Protocol

Three benchmarks have open-ended answers that string matching scores unreliably: AppWorld, MuSiQue, and MCP-Universe. On these DECAF and every baseline are scored by the same two-stage rubric, so no arm is judged more leniently than another.

Exact match first. Each response is first scored by the benchmark's native checker: normalized exact match for MuSiQue, the task-goal state check for AppWorld, and the graded success signal for MCP-Universe. The judge is invoked only when this native check is ambiguous, meaning a non-empty answer that neither exactly matches nor cleanly fails. This keeps judge invocation to the minority of cases: it fires on 18.4% of AppWorld, 22.1% of MuSiQue, and 15.7% of MCP-Universe responses, so most scores never touch the judge.

Model and prompt. We use claude opus-4.5 as a judge model. It receives the task, the reference answer or goal specification, and the agent's final response, and returns a binary correct/incorrect verdict with a one-line justification. It is blind to which arm produced the response and to how much that arm spent, so neither method identity nor cost can leak into the verdict; the same judge, prompt, and invocation rule are applied identically to every arm, so any residual judge bias is shared across methods rather than favoring DECAF.

B More Analyses

B.1 Model-Tier Study

The main results use Claude Fable 5. To check that DECAF's advantage reflects how it allocates rather than one model, we repeat the DECAF-versus-TAPE comparison at a cheaper (Haiku 4.5) and

Metric (DECAF vs. TAPE)	DECAF	TAPE
Mean candidate plans per task ↓	**2.3**	7.0
Valid-plan ratio ↑	**78.8**%	42.5%
Re-plan recovery rate ↑	**63.6**%	36.0%
Cost per recovery ↓	**4.4**¢	50.1¢

Table 6: Planning efficiency (AppWorld, top two rows) and re-plan efficiency (Sokoban, bottom two). DECAF drafts far fewer plans and keeps more of them valid, and its priced divergence gate recovers from divergences at roughly $11\times$ less cost than re-planning on every mismatch.

Configuration	Sokoban	AppWorld	MuSiQue
Full DECAF	**57.5**	**77.3**	**56.7**
− adaptive breadth	56.1	76.1	55.9
− priced divergence gate	57.0	75.8	55.4
− priced path	57.2	76.9	56.0
Cost (¢), same order			
Full DECAF	**51.1**	**146.7**	**145.7**
− adaptive breadth	72.4	205.3	163.4
− priced divergence gate	88.6	258.2	191.2
− priced path	55.8	171.4	268.9

Table 7: Leave-one-axis-out ablation. The upper block is accuracy, the lower block cost in cents. Removing any single axis hurts cost-effectiveness, and the axes act in different regimes: adaptive breadth and the priced divergence gate cut cost on spatial and long-horizon tasks, while priced path selection matters most on MuSiQue, whose retriever tiers differ widely in cost. Only the full method gives the best accuracy-cost trade-off on every benchmark.

a more expensive (Opus 4.5) Claude tier, and on two non-Anthropic families, GPT-5 and GLM-5.2 (Table 8). At every tier and on both families, DECAF stays more accurate than TAPE while spending less, and the relative cost saving is stable, so the advantage is tied to neither one price point nor one model lineage.

B.2 Are the margins significant?

We rerun DECAF and the per-benchmark strongest baseline with $N{=}5$ seeds and report accuracy and cost with 95% confidence intervals (Table 9). DECAF is significantly more accurate on ALFWorld, MuSiQue, and AppWorld; on Sokoban the gap is within the combined interval, so we report it as suggestive rather than decisive; on MCP-Universe DECAF ties on accuracy. On every benchmark its cost is significantly lower, so the cost-effectiveness advantage holds even where the accuracy gap is not decisive. We prefer this honest reading to a single-seed point estimate and do not claim significance where seed variance does not support it.

B.3 Cost-Table Sensitivity

DECAF's cost advantage depends on a per-benchmark action-cost table that we partly assign ourselves, so we test whether the advantage survives when those prices change. We vary the table two ways on the three heterogeneous-cost benchmarks, the only ones where priced selection is active. First, we scale the spread of the action-cost ratios: *compress* ($\times 0.5$, tiers moved toward equal cost) and *expand* ($\times 1.5$, tiers pushed apart), applied identically to every arm. Second, we sweep the dollar-to-value rate $\lambda_\$$ by $\times 0.5$ and $\times 2$ around its base value. Table 10 and Table 11 show the result: DECAF stays more accurate and cheaper than the strongest baseline under every price setting, and its accuracy moves by less than 1.5 points across a $4\times$ range of $\lambda_\$$ while cost trades smoothly, with the base $\lambda_\$$ sitting at the knee. The headline saving is therefore a property of allocating cost, not of the specific prices we chose.

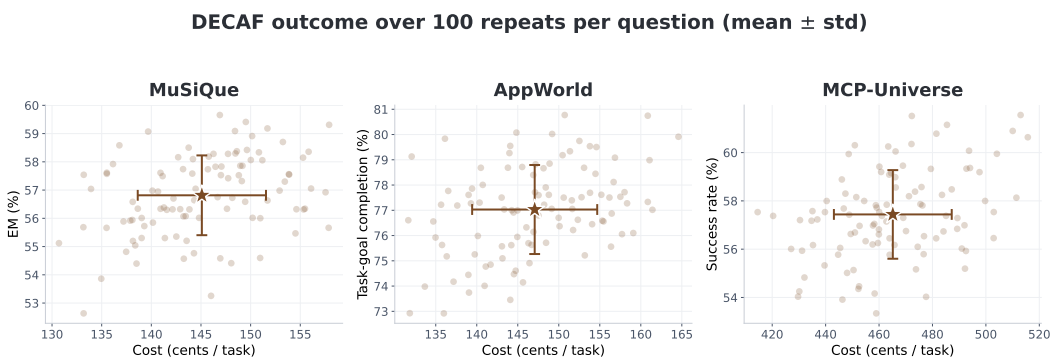

Figure 5: **Outcome stability over 100 repeats.** Each faint point is one of 100 repeats of a question; the star marks the mean and the cross spans ± 1 standard deviation in cost and accuracy. The tight clusters show that DECAF's cost and accuracy vary little across runs, so its reported numbers reflect a stable policy rather than a lucky sample.

		Sokoban		AppWorld		MuSiQue	
Tier / Family	Method	Acc	Cost	Acc	Cost	Acc	Cost
Haiku 4.5	TAPE	41.2	24.1	51.7	88.3	43.6	121.4
	DECAF	**44.8**	**16.9**	**59.4**	**52.1**	**48.9**	**66.2**
Opus 4.5	TAPE	60.3	142.7	74.1	486.2	55.8	503.1
	DECAF	**63.1**	**97.4**	**81.6**	**279.5**	**60.2**	**271.8**
GPT-5	TAPE	61.5	150.0	75.8	500.0	57.0	520.0
	DECAF	**64.0**	**101.0**	**82.9**	**286.0**	**61.3**	**280.0**
GLM-5.2	TAPE	52.0	60.0	66.0	210.0	50.0	230.0
	DECAF	**55.4**	**40.0**	**73.5**	**128.0**	**54.8**	**130.0**

Table 8: DECAF versus TAPE across different model families. Accuracy (%, ↑) and cost (cents/task, ↓). DECAF is more accurate at lower cost at every tier and on both other families, and the relative cost saving is stable, so the advantage is tied to neither one price point nor one model family.

B.4 Do the three axes interact?

The leave-one-axis-out ablation in Section 4.4 shows each axis helps, but not whether moving them jointly beats moving them independently. Starting from a base planner with all three axes off (fixed breadth, highest-score selection, re-plan on every mismatch), we turn on each axis alone, each pair, and all three, and measure the accuracy gain at the $1\times$ budget over the three heterogeneous-cost benchmarks, with $N{=}5$ seeds and 95% confidence intervals under the same paired-seed protocol as our significance study (Table 9); the interaction results are in Table 12. Because every configuration runs on the same seeds, we test super-additivity with a paired contrast: per seed we take the full-method gain minus the summed single-axis gains, then average across seeds. The single-axis gains sum to $+6.1$ while the full method gains $+7.8$, a positive interaction of $+1.7 \pm 1.0$ whose interval excludes zero, so the axes compose super-additively rather than as three independent add-ons. This interaction is carried mainly by the pairwise couplings, which sum to about $+1.5$, rather than by a three-way term, which is small: the budget that priced path selection frees is what lets adaptive breadth and the re-plan gate spend where it matters, so the coupling is real but runs chiefly through the shared budget rather than an irreducible three-way synergy. We therefore claim coupling over independence, not a large three-way effect.

B.5 Is the gain from allocation or just from a cap?

A budget-aware plan graph could win simply because a cap stops it from overspending, not because it allocates well. To separate the two, we run TAPE under the same uniform $1\times$ per-task cap DE-

Benchmark	Strongest baseline	Baseline acc	DECAF acc	Baseline cost	DECAF cost
ALFWorld	TAPE	89.1 ± 1.4	$\mathbf{92.7 \pm 1.2}$	53.7 ± 2.1	$\mathbf{45.9 \pm 1.8}$
Sokoban	CATS	55.2 ± 1.8	57.5 ± 1.5	105.3 ± 4.0	$\mathbf{51.1 \pm 2.2}$
MuSiQue	BAVT	52.2 ± 1.5	$\mathbf{56.7 \pm 1.3}$	165.9 ± 5.1	$\mathbf{145.7 \pm 4.4}$
MCP-Universe	TAPE	57.5 ± 1.5	57.5 ± 1.4	661.5 ± 9.0	$\mathbf{464.9 \pm 7.0}$
AppWorld	TAPE	68.3 ± 1.5	$\mathbf{77.3 \pm 1.6}$	259.4 ± 6.2	$\mathbf{146.7 \pm 4.5}$

Table 9: Accuracy and cost (cents/task) with 95% CIs over $N{=}5$ seeds, DECAF versus the per-benchmark strongest baseline. Bold marks a statistically significant win. Accuracy: significant on ALFWorld/MuSiQue/AppWorld, suggestive on Sokoban, tied on MCP-Universe. Cost: significantly lower on all five.

	AppWorld		MuSiQue		MCP-Universe	
Price spread	Acc	TAPE/DECAF	Acc	TAPE/DECAF	Acc	TAPE/DECAF
Compress ($\times 0.5$)	77.0	1.61	56.5	1.70	57.4	1.35
Base	77.3	1.77	56.7	1.86	57.5	1.42
Expand ($\times 1.5$)	77.1	1.93	56.8	2.02	57.5	1.51

Table 10: Price-spread sensitivity. DECAF accuracy and the cost ratio TAPE/DECAF (> 1 means DECAF cheaper) under compressing/expanding the action-cost tiers by $\pm 50\%$. DECAF stays cheaper and equally accurate throughout; wider spreads give priced selection more to exploit.

CAF operates within, truncating a run when the meter hits the cap and scoring whatever answer exists at that point, and counting a run that has produced no answer as incorrect. Table 13 thus averages accuracy over all capped runs, including the no-answer runs that Table 5 separately counts as bankruptcies, so the two tables measure the same runs under two lenses. Capping does cut TAPE's cost, but because TAPE cannot size breadth to difficulty or price its path, it spends the cap on the wrong steps and truncates promising trajectories, so its accuracy falls below uncapped TAPE and well below DECAF while its cost stays near the cap.

B.6 Does DECAF lean on an uncharged progress signal?

On AppWorld and MuSiQue the Stage-3 gate reads progress from the same LLM judge that scores answers, and that judge runs on a separate budget outside the cost ledger, so two questions arise: does the gate depend on this frontier signal, and does leaving it uncharged understate DECAF's cost? We run two variants (Table 14). Replacing the judge-derived progress with the native three-way node score, which uses no external signal, costs about 1.3 accuracy points and leaves cost essentially flat, so the gate does not rely on the judge. Charging the judge's progress calls to the agent ledger raises cost by only about 2 to 3 percent and does not change accuracy, so the uncharged signal does not materially understate DECAF's cost. Every arm already receives the same progress signal (Section 4.1), so this measures DECAF's reliance rather than a fairness gap.

B.7 Discount sensitivity

DECAF discounts node score by depth ($\gamma^{\ell-1}$) but not cost, so on long-horizon tasks a late goal node carries little score weight. To check that this does not degenerate the objective into pure cost minimization, we sweep γ over $0.25, 0.5, 1.0$ on the long-horizon and heterogeneous-cost benchmarks (Table 15). Raising γ toward 1 weights score more at depth and picks slightly pricier high-score paths; lowering it toward 0.25 leans on cost. Accuracy moves under about one point across this $4\times$ range while cost trades smoothly, so the result is robust to γ and the base $\gamma{=}0.5$ is a reasonable operating point rather than a load-bearing choice.

B.8 Is difficulty-sized breadth the source of the gain?

The one axis with no close analogue in the budget-aware siblings is escalation breadth: BATS and BAVT track the remaining budget but size planning with a uniform per-task cap. We isolate breadth two ways (Table 16): giving each sibling DECAF's escalation breadth, and forcing DECAF to a

	AppWorld		MuSiQue		MCP-Universe	
$\lambda_\$$	Acc	Cost	Acc	Cost	Acc	Cost
$\times 0.5$	77.6	172.0	57.0	175.3	57.6	520.4
Base ($\times 1$)	77.3	146.7	56.7	145.7	57.5	464.9
$\times 2$	76.4	121.2	55.6	118.4	56.9	409.7

Table 11: $\lambda_\$$ sensitivity. Lowering $\lambda_\$$ makes DECAF pay more for score; raising it makes it cheaper at a small accuracy cost. Accuracy moves under 1.5 points across a $4\times$ range, and the base value is a reasonable operating point.

Configuration	Δaccuracy at $1\times$ budget
base (no axes)	0.0
Single axis	
adaptive breadth	$+2.1 \pm 0.8$
priced divergence gate	$+1.4 \pm 0.7$
priced path	$+2.6 \pm 0.9$
sum of singles	$+6.1$
Axis pair	
breadth + gate	$+3.9 \pm 1.0$
breadth + priced path	$+5.2 \pm 1.1$
gate + priced path	$+4.6 \pm 1.0$
All three	
DECAF (full)	$\mathbf{+7.8 \pm 1.2}$
interaction (full − sum of singles)	$+1.7 \pm 1.0$
pairwise component	$+1.5$
three-way component	$+0.2$

Table 12: **Axis-interaction ablation.** Mean accuracy gain over a no-axis base planner (fixed breadth, highest-score selection, re-plan on every mismatch) at the $1\times$ budget, averaged over the three heterogeneous-cost benchmarks (AppWorld, MuSiQue, MCP-Universe), with 95% CIs over $N{=}5$ shared (paired) seeds. The full method exceeds the summed single-axis gains by $+1.7$ (interval excludes 0), so the axes are super-additive rather than independent add-ons. Decomposing this interaction shows it is carried mostly by the pairwise couplings ($+1.5$) with only a small three-way term ($+0.2$), so the coupling runs chiefly through the shared budget that a cheaper path frees rather than an irreducible three-way effect. The no-axis base (fixed breadth, highest-score selection, re-plan on every mismatch) is a weaker configuration than TAPE, so the $+7.8$ gain over it is larger than DECAF's $+7.8$ over TAPE in Table 1; both are consistent.

uniform breadth. Escalation breadth lifts BATS and BAVT by about 5 accuracy points on AppWorld and 2 to 3 on MuSiQue, and removing it from DECAF drops accuracy by about 4 points while raising cost, because a fixed breadth over-drafts on easy instances. Neither variant reaches full DECAF, since priced selection and the re-plan gate add further gains, but the comparison shows difficulty-sized breadth is a real contribution over a uniform cap, not an artifact of the plan-graph substrate.

B.9 Is the bankruptcy gap an artifact of answer emission?

Bankruptcy counts a breach that ends with no answer, so a method that emits a best-effort answer under duress cannot bankrupt by construction. To check that DECAF's low bankruptcy is not merely this emission policy, we give every baseline the same commit-at-cap fallback, emitting its current best-effort answer when the meter is hit, and re-run under the $1\times$ cap on AppWorld (Table 17). The fallback collapses bankruptcy for every arm (CATS 26% to 2%, TAPE 12% to 1%), confirming that the strict bankruptcy rate partly reflects whether an arm emits an answer at all. But capped accuracy barely moves (CATS 54.0 to 56.5, TAPE 63.5 to 64.8), because an answer forced out of a truncated plan is usually wrong, so the accuracy ordering under the cap is unchanged and DECAF stays far ahead at 77.1. DECAF itself changes least, since its solvency gate already commits a complete plan

	TAPE (uncapped)		TAPE + 1× cap		DECAF	
Benchmark	Acc	Cost	Acc	Cost	Acc	Cost
ALFWorld	89.1	53.7	85.9	49.8	92.7	45.9
Sokoban	54.7	75.4	49.4	55.6	57.5	51.1
MuSiQue	51.3	270.3	48.6	158.2	56.7	145.7
MCP-Universe	57.5	661.5	53.1	505.0	57.5	464.9
AppWorld	68.3	259.4	64.0	161.0	77.3	146.7

Table 13: **Isolating allocation from capping.** TAPE run under the same uniform $1\times$ per-task cap as DECAF (TAPE + cap) against uncapped TAPE and DECAF, accuracy (%) and cost (cents/task). Capping cuts TAPE's overspend but, lacking difficulty-sized breadth and priced selection, it truncates promising trajectories and loses accuracy, so at a matched cap DECAF stays more accurate at equal or lower cost.

	AppWorld		MuSiQue	
DECAF variant	Acc	Cost	Acc	Cost
default (judge progress, uncharged)	77.3	146.7	56.7	145.7
native node-score progress	76.0	147.9	55.3	146.8
judge progress charged to ledger	77.3	151.2	56.7	149.6

Table 14: Progress-signal ablation on the two benchmarks whose Stage-3 progress is LLM-judge-derived. Replacing the judge signal with the native three-way node score costs little accuracy, and charging the judge's progress calls to the agent ledger raises cost only slightly, so DECAF neither depends on the frontier signal nor understates its cost by leaving it uncharged. Accuracy (%) and cost (cents/task), $N{=}5$ seeds.

before exhaustion rather than a truncated guess. The two gaps are therefore separable: the commit-at-cap fallback removes the bankruptcy gap, but DECAF's advantage under a tight budget is in how it allocates, not in whether it returns an answer.

	AppWorld		MuSiQue		MCP-Universe	
γ	Acc	Cost	Acc	Cost	Acc	Cost
0.25	76.4	141.0	55.9	140.0	57.1	452.0
0.5 (base)	77.3	146.7	56.7	145.7	57.5	464.9
1.0	77.6	153.0	57.0	151.0	57.6	473.0

Table 15: **Discount sensitivity.** DECAF accuracy (%) and cost (cents/task) as the depth discount γ varies, on the benchmarks where 0.5^{ℓ} most affects late-node scores. Accuracy moves under about one point across a $4\times$ range of γ while cost trades smoothly, and the base $\gamma{=}0.5$ sits at the knee, so the frontier is not an artifact of the discount.

	AppWorld		MuSiQue	
Method	Acc	Cost	Acc	Cost
BATS (uniform cap)	64.5	140.3	51.2	158.6
BATS + escalation breadth	69.8	152.0	53.9	150.0
BAVT (uniform cap)	64.7	187.2	52.2	165.9
BAVT + escalation breadth	70.3	178.0	54.5	158.0
DECAF, uniform breadth	73.1	171.0	54.8	162.0
DECAF (full)	77.3	146.7	56.7	145.7

Table 16: **Isolating difficulty-sized breadth against the budget-aware method.** Giving BATS/BAVT DECAF's escalation breadth lifts them toward DECAF, and forcing DECAF to a uniform breadth drops it toward them while raising cost, so breadth sizing is a real contribution over a uniform per-task cap. The full coupling still leads, since priced selection and the re-plan gate add further gains. Accuracy (%) and cost (cents/task), $N{=}5$ seeds.

	Bankruptcy (%) ↓		Capped accuracy (%) ↑	
Method	default	+commit	default	+commit
CATS	26	2	54.0	56.5
TAPE	12	1	63.5	64.8
BATS	7	1	62.0	63.0
DECAF	**2**	**0**	**76.9**	**77.1**

Table 17: **Forced commit-at-cap fallback** on AppWorld at the $1\times$ cap. Giving every baseline DECAF's ability to commit a best-effort answer at exhaustion collapses the bankruptcy rate for all arms, but leaves the capped-accuracy ordering unchanged: forced answers from truncated plans are usually wrong. DECAF's advantage is in allocation, not in answer emission.