Exploit More, Explore Smarter for Budget-Constrained Agentic Search

Haoyang Fang
Amazon AGI
haoyfang@amazon.com

Bernie Wang*
Amazon AGI
yuyawang@amazon.com

Abstract

Budget-constrained agentic search arises when an LLM agent must refine candidates under a small evaluation budget, because validation is expensive, generation requires multiple model calls, or both. In this regime, standard MCTS allocates budget poorly: exploration bonuses dominate at low visit counts, unpromising siblings are expanded before promising chains can deepen, and branching is independent of node quality. We introduce **ExTS**, a tree-search policy that treats expansion itself as a value-of-information decision. ExTS combines three mechanisms: discriminative reward shaping to separate candidates under narrow score distributions, a stochastic virtual child that estimates the value of creating a new branch from the parent's reward history, and quality-conditioned branching that expands only when a node's score justifies the budget cost. Across prompt optimization, code generation, molecular structure elucidation, and agentic workflow optimization, ExTS is competitive with or improves over task-specific tree-search baselines, with an average relative gain of **+5.5%** using a single fixed configuration. We further introduce pilot-run diagnostics that characterize what makes budget-constrained agentic search problems structurally different from one another, providing both understanding of the problem space and practical guidance for adaptation.

1 Introduction

A growing class of AI systems rely on *agentic search*: an LLM iteratively proposes and refines candidates, consuming budget on both generation and validation. These systems span prompt optimization (Agrawal et al., 2025; Opsahl-Ong et al., 2024), code generation (Zhang et al., 2023; Inoue et al., 2026), tool-augmented reasoning (Yao et al., 2022; Schick et al., 2023), multi-step planning (Yao et al., 2023; Besta et al., 2024), etc. Despite their diversity, they share a common constraint: the search budget is limited to tens or hundreds of calls. We call this regime *budget-constrained agentic search*.

Recent work on LLM search focuses predominantly on the *harness*: the framework wrapping the search, such as value functions, action spaces, or multi-agent evaluation (Shinn et al., 2023; Madaan et al., 2023; Zhou et al., 2023; Qi et al., 2024; Antoniades et al., 2025; Zhang et al., 2024; Hao et al., 2023; Liu et al., 2025; Li et al., 2025; Fang et al., 2026a). These systems retain linear search or standard UCT for selection and expansion (Appendix A), leaving substantial room for improvement in how budget is allocated within the tree. On the search algorithm side, AB-MCTS (Inoue et al., 2026) replaces UCT with Thompson sampling, GEPA (Agrawal et al., 2025) uses Pareto-frontier selection, and AFlow (Zhang et al., 2025) applies score-weighted random sampling, but each targets a single task type and none condition expansion on node quality or shape rewards for narrow score distributions. How to design selection and expansion policies that transfer across diverse agentic problems has received limited attention.

We introduce **ExTS**[1], a tree-search policy that *exploits more and explores smarter* by jointly redesigning both *how nodes are selected* and *whether expansion should occur*. The core insight is that under tight budgets, the decision of whether to expand is as consequential as which child to visit. Standard MCTS suffers from three structural limitations in this regime: exploration-bonus dominance at low visit counts, forced full expansion of unpromising siblings, and no quality gate on branching. ExTS addresses each through three mechanisms: **discriminative reward shaping** (Section 3.1) that produces effective selection signals when raw scores

*Work done at Amazon.

[1]Code will be released at https://github.com/amazon-science/ExTS. Until then, please contact us.

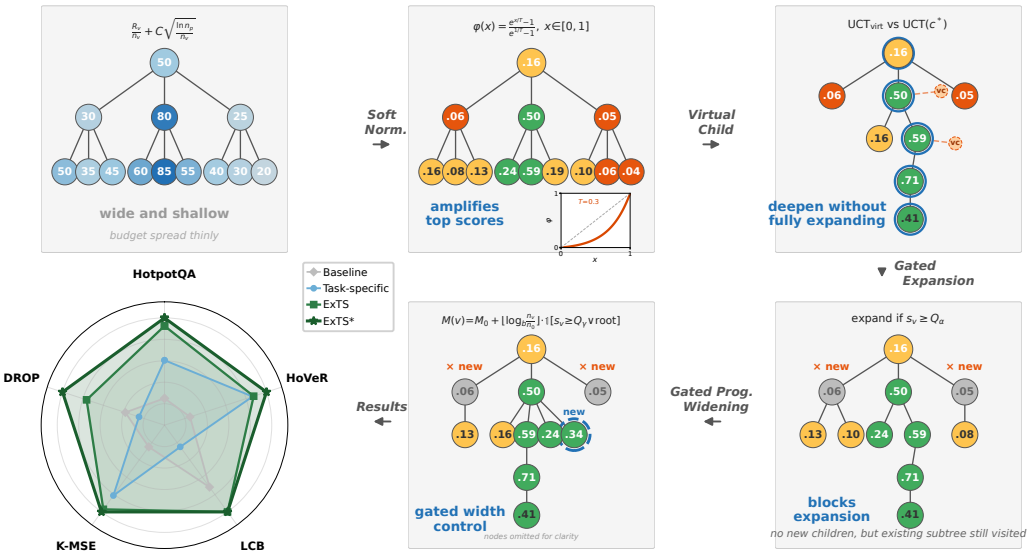

Figure 1: Standard UCT spreads budget across a wide, shallow tree via flat rewards and obligatory expansion. ExTS instead uses virtual children that frame expansion as a value-of-information decision and quality-conditioned branching gated on the reshaped score. Radar plot: normalized gains; ExTS* denotes pilot-adapted configurations.

cluster narrowly; a **virtual child heuristic** (Section 3.2) that estimates expansion value by sampling from the parent's reward history, making expansion a value-of-information (VOI) decision competing directly with deepening; and **quality-conditioned branching** (Section 3.3) that restricts expansion to nodes whose score justifies the budget cost. These mechanisms are inspired by classical MCTS ideas (progressive widening (Coulom, 2007), first-play urgency (Gelly and Wang, 2006), PUCT (Rosin, 2011)) and improve upon each for budget-constrained agentic search (Section 5.3).

With a single fixed configuration across prompt optimization (Yang et al., 2018; Jiang et al., 2020), code generation (Jain et al., 2024), molecular structure elucidation (Zhuang et al., 2025), and agentic workflow optimization (Dua et al., 2019), ExTS is competitive with or improves over task-specific methods (+10.8% on HotpotQA, +11.7% on LiveCodeBench hard, +1.3% on K-MSE, +3.5% on DROP, and +0.2% on HoVeR). ExTS* further improves performance by adjusting one or two hyperparameters based on pilot-run diagnostics. We additionally report diagnostics linking landscape structure to hyperparameter sensitivity, component ablations, budget-scaling and tree-shape analysis.

2 Preliminaries

2.1 Validation-Heavy Search

We formalize validation-heavy search as a tuple $(\mathcal{S}, \mathcal{A}, f, B)$ where $\mathcal{S}$ is the set of candidate solutions (e.g., prompts, code, or plans), $\mathcal{A} : \mathcal{S} \to \mathcal{S}$ is a stochastic refinement operator powered by an LLM, $f : \mathcal{S} \to \mathbb{R} \cup \{\perp\}$ is a validation function that returns a score or failure, and B is the total evaluation budget. The search builds a tree $\mathcal{T}$ rooted at an initial candidate x_0 and the goal is $\arg\max_{x \in \mathcal{T}} f(x)$ within budget B.

We identify four *pilot-run diagnostics* that characterize the search landscape of a validation-heavy task for a given model and scorer (formal definitions in Appendix C). Beyond guiding hyperparameter choices, these diagnostics characterize the axes along which agentic search problems structurally differ and help explain why no single search configuration dominates universally. These diagnostics are computed from a pilot tree built by running the *baseline* search method for each dataset, measuring properties of the landscape as seen by the task-native algorithm. The *refinement variance* $\hat{\sigma}_{\mathcal{A}}$ is the normalized standard deviation of f across independent refinements with the same input, quantifying the stochasticity of the LLM refinement operator. The *normalized score deviation* $\hat{\sigma}_f$ is the standard deviation of f across all successfully validated candidates in the completed tree, normalized to $[0, 1]$. The *score drift* κ is the mean absolute shift in normalized scores caused by evolving normalization bounds as new candidates are discovered; high drift indicates that the search frequently discovers candidates at the extremes of the score range. The *failure rate* ρ is the fraction of refinement at-

Table 1: Pilot-run diagnostics (mean±std, 3 seeds). $\hat{\sigma}_{\mathcal{A}}$: refinement variance; $\hat{\sigma}_f$: score spread; κ: score drift; ρ: failure rate.

Dataset	$\hat{\sigma}_{\mathcal{A}}$	$\hat{\sigma}_f$	κ	ρ
HotpotQA	.30±.05	.30±.09	.11±.03	.74±.04
HoVeR	.20±.06	.29±.01	.16±.11	.82±.01
LiveCodeBench	.32±.01	.23±.01	.14±.13	.57±.05
K-MSE	.09±.00	.18±.00	.05±.00	<.01
DROP	.04±.03	.23±.03	.02±.01	.03±.03

tempts where f returns $\perp$. Table 1 characterizes the datasets we study along these axes.

2.2 Standard MCTS and Its Limitations

In standard MCTS, the UCT policy (Kocsis and Szepesvári, 2006) selects child c of node v by maximizing

$$\mathrm{UCT}_{\mathrm{std}}(c) = \frac{R_c}{n_c} + C\sqrt{\frac{\ln n_v}{n_c}} \qquad (1)$$

where R_c/n_c is the average reward, n_c and n_v are visit counts, and C controls exploration. The theoretical default $C = \sqrt{2}$ is calibrated for rewards in $[0, 1]$ with sufficient budget for convergence (Browne et al., 2012).

Two assumptions underlying UCT are violated in validation-heavy search. First, reward shaping (Section 3.1) compresses the effective exploitation range, causing the exploration bonus to dominate and making all nodes appear equally promising. Second, budgets of tens to hundreds of evaluations are far below the asymptotic regime where UCT's convergence guarantees hold (Silver et al., 2016; Browne et al., 2012; Kocsis and Szepesvári, 2006). Together, these violations produce flat, wide trees that fail to develop deep refinement chains. The PUCT exploration term (Section 3.1) partially addresses the first issue by decaying linearly rather than logarithmically, reducing the dominance of exploration under tight budgets.

Classical extensions. Three classical MCTS ideas are relevant but do not transfer directly to this regime (Section 5.3). Progressive widening (Coulom, 2007; Chaslot et al., 2008) ties branching to visit count but is score-agnostic. First-play urgency (FPU) (Gelly and Wang, 2006) assigns a fixed value to unvisited actions but cannot adapt to non-stationary score distributions. PUCT (Rosin, 2011; Silver et al., 2017) uses a learned policy prior with linear exploration decay. ExTS redesigns each: progressive widening becomes quality-conditioned, FPU becomes a stochastic virtual child sampling from the parent's reward history, and PUCT-style decay operates without a learned prior.

3 Methodology

Our intuition is that, especially under tight budgets, adding a single scalar-weighted UCT or PUCT exploration term to the reward cannot by itself capture the balance between exploration and exploitation (Section 5.1). ExTS keeps this weighted exploration term, but conditions both selection and expansion decisions on the observed reward distribution at two scopes: local (a node's own subtree) and global (the whole search tree).

Concretely, each node v stores a candidate $x_v \in \mathcal{S}$ and maintains: n_v (total visits), n_v^+ (successful visits), $\mathcal{R}_v$ (reward observations from successful expansions in v's subtree), and $s_v = f(x_v)$ (validation score). The reward pool $\mathcal{R}_v$ implements mean backup: $\mathrm{ExUCT}(v)$ estimates the refinement productivity of expanding below v, not the quality of v's own candidate. Figure 1 illustrates how the three design choices transform the search tree. The complete search loop follows standard MCTS (select, expand, validate, backpropagate) and is given in Algorithm 2 (Appendix B).

3.1 Discriminative Reward Shaping

Both effective selection and meaningful expansion decisions require differentiating node quality. When raw validation scores cluster in a narrow range, as is common in validation-heavy domains, UCT's exploitation term makes all nodes appear equally attractive: selection becomes near-random, and any VOI comparison between expanding versus deepening becomes uninformative. The ExTS selection policy addresses this by scoring each visited node v ($n_v > 0$) as:

$$\mathrm{ExUCT}(v) = \left(\frac{n_v^+}{n_v}\right)^{\alpha} \cdot \overline{\varphi(r_i)} + \mathrm{explore}(n_{\mathrm{par}(v)},\, n_v) \qquad (2)$$

The exploitation term combines two signals: the shaped rewards $\overline{\varphi(r_i)}$, which separate candidates when raw scores cluster narrowly, and a success-rate weight $(n_v^+/n_v)^{\alpha}$, which discounts nodes whose subtrees produce mostly invalid outputs (α controls failure discounting). The shaping function φ maps raw scores through *global* normalization

and a temperature-controlled nonlinearity:

$$\varphi(r) = \frac{e^{\hat{r}/T} - 1}{e^{1/T} - 1}, \quad \hat{r} = \frac{r - s_{\min}}{s_{\max} - s_{\min}}, \tag{3}$$

$$\overline{\varphi(r_i)} = \frac{1}{|\mathcal{R}_v|} \sum_{r_i \in \mathcal{R}_v} \varphi(r_i). \tag{4}$$

For $T < 1$, φ is convex, compressing low scores and amplifying high scores. This is important in validation-heavy domains where scores cluster in a narrow range and linear normalization would make most nodes appear equally attractive.

The exploration term takes one of two forms. The **UCT** variant decays logarithmically:

$$\text{explore}_{\text{UCT}}(n_{\text{par}}, n_v) = C\sqrt{\frac{\ln n_{\text{par}}}{n_v}}, \tag{5}$$

while the **PUCT-style** variant decays linearly, providing faster convergence toward exploitation under tight budgets:

$$\text{explore}_{\text{PUCT}}(n_{\text{par}}, n_v) = C \cdot \frac{\sqrt{n_{\text{par}}}}{1 + n_v}. \tag{6}$$

Unlike standard PUCT (Rosin, 2011; Silver et al., 2017), which incorporates a learned policy prior $P(a|s)$, our variant uses no prior for simplicity, because such a prior must be designed task-specifically. The default ExTS configuration uses PUCT-style exploration with $C = 1.0$; we evaluate the UCT variant ($C = \sqrt{2}$) as an ablation in Section 5.3.

3.2 VOI Estimation via the Virtual Child

The core of ExTS's expansion-as-VOI principle is a mechanism for estimating the expected value of creating a new child at any internal node. In standard MCTS, expansion occurs only at leaf nodes, forcing the algorithm to fully expand each level before deepening. We instead make expansion a first-class decision at every internal node by introducing a *virtual child* that competes with real children during selection.

At each internal node v with m existing children, if v can still expand ($m < M$), we compute a virtual child score. First, we estimate the virtual child's fair-share visit count:

$$n_{\text{fair}} = \max\left(1, \frac{\sum_{c \in \text{children}(v)} n_c}{m}\right). \tag{7}$$

We then sample $k = \max(1, \lfloor n_{\text{fair}} \cdot n_v^+ / n_v \rfloor)$ scores with replacement from the parent's reward pool $\mathcal{R}_v \cup \{s_v\}$ and compute the estimated shaped mean $\overline{\varphi(\tilde{r})}$. The virtual child score mirrors Equation 2:

$$\text{ExUCT}_{\text{virt}} = \left(\frac{n_v^+}{n_v}\right)^{\alpha} \cdot \overline{\varphi(\tilde{r})} \; + \; \text{explore}(n_v, n_{\text{fair}}). \tag{8}$$

The stochastic sampling captures finite-sample uncertainty: fewer expected visits produce fewer samples and thus higher variance, naturally encouraging expansion when evidence is thin.

Let $c^* = \arg\max_{c_j} \text{ExUCT}(c_j)$ be the best real child. If $\text{ExUCT}_{\text{virtual}} > \text{ExUCT}(c^*)$, the node is selected for expansion; otherwise selection recurses into c^*. Appendix I works through this comparison on a real run.

3.3 Quality-Conditioned Branching

ExTS conditions branching on node quality at two granularities: whether a node may expand at all, and how many children it may accumulate.

Expansion gate. For non-root nodes, expansion is permitted only when the node's validation score exceeds the τ-quantile of the population:

$$s_v \geq Q_\tau\big(\{s_u : u \in \mathcal{T}\}\big). \tag{9}$$

Below-threshold nodes remain reachable and update their statistics but cannot spawn new children. The threshold adapts as the tree accumulates better candidates, becoming increasingly selective over time.

Quality-gated progressive widening. Classical progressive widening (Coulom, 2007; Chaslot et al., 2008) ties branching to visit count but is score-agnostic. We condition it on node quality: the maximum branching factor grows with visit count, but only for nodes whose quality justifies it:

$$M(v) = M_0 + \left\lfloor \log_b \tfrac{n_v}{n_0} \right\rfloor \cdot \mathbb{1}\left[s_v \geq Q_\gamma \vee v = v_{\text{root}}\right], \tag{10}$$

where M_0 is the initial maximum children, b is the widening base, n_0 is the visit threshold before widening begins, and Q_γ is the γ-quantile of node scores. Typically $\gamma \gg \tau$ (defaults: $\gamma = 0.75$, $\tau = 0.25$), reflecting that widening is a stronger commitment than a single expansion. Low-scoring nodes remain capped at M_0.

Gate semantics. The expansion gate (Algorithm 1, line 5) applies to *internal* nodes deciding whether to create additional children. A leaf node returned at line 2 is selected for its first expansion regardless of score; the gate restricts further

Algorithm 1 EXUCT-SELECT: Selection with Virtual Child

Require: Node v

if v is a leaf **then**
  return v
end if
can_expand $\leftarrow |\text{children}(v)| < M(v)$
if $v \neq$ root **and** can_expand **then**
  can_expand $\leftarrow s_v \geq Q_\tau(\{s_u : u \in \mathcal{T}\})$
end if
$c^* \leftarrow \arg\max_{c \in \text{children}(v)} \text{ExUCT}(c)$
if can_expand **then**
  $m \leftarrow |\text{children}(v)|$
  $n_{\text{fair}} \leftarrow \max\big(1, \sum_c n_c/m\big)$
  $k \leftarrow \max(1, \lfloor n_{\text{fair}} \cdot n_v^+/n_v \rfloor)$
  Sample $\tilde{r}_1, \ldots, \tilde{r}_k \sim \mathcal{R}_v \cup \{s_v\}$ with replacement
  $q \leftarrow (n_v^+/n_v)^\alpha \cdot \frac{1}{k}\sum_i \varphi(\tilde{r}_i)$
  $\text{ExUCT}_{\text{virtual}} \leftarrow q + \text{explore}(n_v, n_{\text{fair}})$
  if $\text{ExUCT}_{\text{virtual}} > \text{ExUCT}(c^*)$ **then**
    return v expand here
  end if
end if
return EXUCT-SELECT(c^*)

branching only after a node has at least one child and evidence of its quality. This means every node receives at least one expansion attempt before gating takes effect.

4 Evaluation

We evaluate ExTS across four domains spanning diverse failure rates, score variances, and drift levels (Table 1). All methods share the same evaluation budget B per task. ExTS uses a single fixed configuration across all domains (Table 10); each baseline is the best-known method for its respective task with a fully tuned, domain-specific configuration. We additionally report ExTS* variants that adjust one or two hyperparameters per task based on pilot-run diagnostics, a modest adaptation that remains fair given the domain-specific tuning already present in each baseline. Full formulation details are in Appendix D.

4.1 Prompt Optimization

We integrate ExTS into the GEPA framework (Agrawal et al., 2025) for optimizing DSPy (Khattab et al., 2023) instructions on HotpotQA (Yang et al., 2018) and HoVeR (Jiang et al., 2020) (300 test examples, 3 seeds, Qwen3-8B). This domain has the highest failure rate ($\rho = 0.74$–0.82) among our tasks.

Table 2: Prompt optimization: test accuracy (%) over 3 seeds. †Results from Agrawal et al. (2025). *Pilot-adapted (Section 5.2).

Method	HotpotQA	HoVeR
Baseline	41.44±1.25	36.11±0.79
GRPO†	43.33	38.67
MIPROv2†	55.33	47.33
GEPA (Pareto)	58.55±5.32	50.33±2.45
ExTS	**64.89**±0.83	50.45±2.18
ExTS*	**66.00**±0.98	**51.67**±0.98

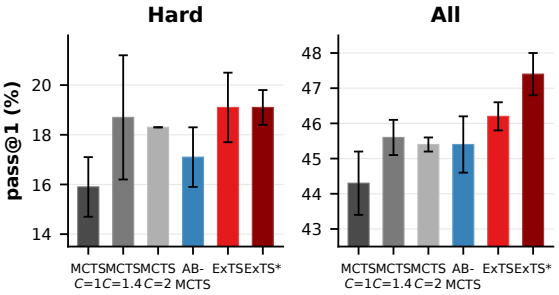

Figure 2: LiveCodeBench pass@1 (%) over 3 seeds.

Table 2 shows that ExTS improves over GEPA Pareto by +10.8% on HotpotQA with 6.4× lower variance. The pilot-adapted ExTS* further improves to 66.00% by increasing the success-rate exponent to $\alpha = 2.0$, which sharpens node selection in this high-failure-rate domain ($\rho = 0.74$; Section 5.2). On HoVeR, ExTS performs on par with GEPA Pareto (+0.2%, within noise). HoVeR exhibits the highest score drift ($\kappa = 0.16$) among our tasks, causing normalization bounds to shift frequently and re-rank nodes mid-search. The pilot-adapted ExTS* switches to UCT exploration with no gating, buffering against score re-ranking by maintaining sustained exploration (Section 5.2).

4.2 Code Generation

We integrate ExTS into the TreeQuest framework (Inoue et al., 2026) on LiveCodeBench (Jain et al., 2024) (182 problems, budget 128, Claude Sonnet 4), comparing against StandardMCTS and AB-MCTS-A. All three methods achieve identical public-test success rates on easy (100%) and medium (85.8%), making those splits non-discriminative; we therefore use the hard split as our primary metric while reporting overall performance for reference (Figure 2).

On the hard split, ExTS reaches 19.1% pass@1

Table 3: K-MSE molecular elucidation (Claude Sonnet 4.6, 216 molecules, 16 rollouts, 3 seeds). ACC (exact match) is the primary metric; fingerprint similarities are listed for reference. *Pilot-adapted (Section 5.2).

Method	ACC (%)	Morgan	MACCS	RDK
CoT	79.8±0.2	.887±.002	.942±.001	.904±.001
K-MSE	89.8±0.4	.952±.002	.976±.001	.962±.003
ExTS	**91.0**±0.6	.952±.004	.972±.005	.960±.004
ExTS*	**91.2**±0.4	.954±.004	.973±.003	.959±.004

Table 4: Workflow optimization on DROP (F1 × 100, 3 seeds). *Pilot-adapted (Section 5.2).

Method	F1 (mean±std)	Best	Cost ($)
CoT (linear)	89.28±1.78	91.10	36.70
AFlow	87.89±2.29	89.56	26.09
ExTS	**90.96**±0.93	**91.54**	**15.26**
ExTS*	**91.54**±1.92	**93.76**	**10.43**

versus 17.1% for AB-MCTS-A (+11.7%) and 18.1% for StandardMCTS (+5.5%), with the lowest cross-seed variance. The gains come from deeper refinement of promising candidates rather than broader coverage. Overall pass@1 is 46.2% (+1.3% over StandardMCTS). The pilot-adapted ExTS* ($T = 0.5$) further improves to 47.4% overall (+4.0% over StandardMCTS).

4.3 Molecular Structure Elucidation

We apply ExTS to K-MSE (Zhuang et al., 2025), where an LLM deduces molecular SMILES from NMR and IR spectra (216 molecules, Claude Sonnet 4.6). This domain has near-zero failure rate and stationary scores.

ExTS improves ACC by +1.3% over K-MSE's MCTSr (Table 3). The strong base model leaves limited headroom; gains come from exploitation shaping and the virtual child directing budget toward molecules that benefit from iterative refinement. The scorer measures embedding similarity rather than exact match, creating a reward-metric gap where structurally similar but incorrect molecules (e.g., isomers sharing NMR signatures) receive high rewards.

4.4 Agentic Workflow Optimization

We evaluate ExTS on DROP (Dua et al., 2019) within the AFlow framework (Zhang et al., 2025), which searches over LLM-based operator graphs (20 rounds, 3 seeds, Claude Sonnet 4.5 optimizer, Haiku 4.5 executor). The three methods differ in how they condition refinement: CoT always extends the deepest chain; AFlow conditions on a top-scoring round chosen by stochastic sampling; ExTS conditions on a tree-search-selected node.

CoT achieves strong accuracy by accumulating history but incurs the highest cost as context grows linearly. AFlow reduces cost by avoiding deep chains but sacrifices robustness through stateless selection. ExTS is Pareto-dominant: tree search identifies productive nodes at moderate depths while quality gates prevent overcommitment to deep chains, achieving the best accuracy, lowest variance, and lowest cost (Table 4). We report cost only for this domain because the optimizer receives the parent workflow as context, making cost proportional to conditioning depth; in other domains (code generation, molecular elucidation), cost per iteration is fixed regardless of search depth.

5 Understanding ExTS

5.1 The Exploration Constant Is Not Enough

A natural question is whether simply tuning the exploration constant C in standard MCTS could replicate ExTS's gains. On LiveCodeBench (Figure 2), sweeping C from 1.0 to 2.0 produces only marginal differences (44.3–45.6% overall), with no configuration approaching ExTS (46.2%). The exploration-exploitation ratio is a single scalar that uniformly scales the bonus across all nodes; it cannot address flat reward signals, forced full expansion, or score-agnostic branching. Lowering C shifts budget toward exploitation but still expands unpromising nodes unconditionally; raising C broadens coverage but wastes budget on shallow siblings that never deepen. Neither direction addresses the core issue: the tree policy lacks the structural mechanisms to decide *where* expansion is worthwhile. ExTS's improvement is structural: it reshapes *how* the tree grows, not merely *how much* it explores.

5.2 Pilot-Run Findings

We examine how pilot-run diagnostics (Table 1) characterize the structural differences among agentic search problems and relate these differences to hyperparameter sensitivity. The high cost of agentic search experiments does not permit exhaustive grid search, so the ExTS* configurations below may not represent optimal settings. Nonetheless, even coarse adaptation guided by these diagnostics yields ExTS* variants that outperform all task-specific baselines (Tables 2, 3, 4 and Figure 2). Appendix G condenses these findings into a com-

pact diagnostic guide (Table 13). Three patterns emerge.

Low $\hat{\sigma}_{\mathcal{A}}$/low κ or unstable validation $\rightarrow$ soften T. When refinement variance and drift are both low, scores cluster tightly and aggressive exploitation (low T) over-commits to gaps that may not predict test improvement. Relaxing T to 0.5 hedges against this. On K-MSE ($\hat{\sigma}_{\mathcal{A}} = 0.09$, $\kappa = 0.05$), $T = 0.5$ yields 91.2% ACC (+0.2 pp; Table 3). On DROP ($\hat{\sigma}_{\mathcal{A}} = 0.04$, $\kappa = 0.02$), raising $\tau = 0.5$ with $M_0 = 4$ yields 91.54 F1 (+0.58 pp; Table 4). LiveCodeBench ($\hat{\sigma}_{\mathcal{A}} = 0.32$, $\kappa = 0.14$) does not exhibit this signature, yet also benefits from $T = 0.5$ (47.4%, +1.2 pp; Figure 2) because its validation signal is inherently sparse: a few public test cases serve as proxy for the full hidden suite, creating per-sample noise that similarly rewards softer exploitation. In all three cases, moderate T-softening improves performance when the validation scorer is a weak proxy for the true metric.

High score drift (κ) $\rightarrow$ early exploration. High κ means normalization bounds shift frequently, making early shaped reward rankings unreliable. UCT's exploration term is larger than PUCT's when the tree is small, providing stronger early exploration before rankings stabilize. Disabling the score gate ($\tau_{\text{gate}} = 0$) avoids blocking potentially good parents based on transiently low scores. On HoVeR ($\kappa = 0.16$, highest), switching to UCT with $\tau_{\text{gate}} = 0$ yields 51.67% (+1.22 pp, 2.2$\times$ lower variance; Table 2). Conversely, HotpotQA ($\kappa = 0.11$) strongly prefers PUCT (+6.45 pp over UCT; Table 7), where fast decay concentrates budget on chains whose rankings are stable enough to trust.

High failure rate (ρ) $\rightarrow$ sensitive to α. Intuitively, when most expansions fail, the success-rate exponent α becomes critical for separating productive nodes from unproductive ones. We verify this on HotpotQA ($\rho = 0.74$): setting $\alpha = 2.0$ yields 66.00% (+1.11 pp; Table 2), while reducing α to 1.0 causes -7.78 pp with 6.8$\times$ higher variance (Table 5). In low-failure tasks, α has negligible effect.

5.3 Ablation Studies

We isolate each design choice via leave-one-out ablation on HotpotQA (high ρ, high κ). LiveCodeBench ablations are in Table 12 (Appendix F).

Table 5 confirms that all components contribute. Failure penalty has the largest impact (-7.78 pp), followed by the virtual child (-3.22 pp); without the latter, the algorithm reverts to wide-tree behavior. Gated progressive widening contributes -2.34 pp by restricting expansion to nodes whose quality justifies the budget cost.

Table 5: Leave-one-out ablation on HotpotQA (3 seeds). LiveCodeBench ablations in Table 12.

Configuration	Acc.	Configuration	Acc.
ExTS (full)	**64.89**$\pm$0.83	w/o Soft norm.	63.44$\pm$2.20
w/o Virtual child	61.67$\pm$1.96	w/o Gated expan.	63.22$\pm$1.10
w/o Fail. penalty ($\alpha = 1$)	57.11$\pm$5.67	w/o Gated PW	62.55$\pm$2.08

Table 6: Classical mechanisms vs. ExTS redesigns (3 seeds).

Configuration	Score
DROP (F1 $\times$ 100)	
ExTS (virtual child)	**90.96**$\pm$0.93
Fixed FPU = 0.5	88.95$\pm$0.81
Fixed FPU = 0.8	89.34$\pm$3.23
LiveCodeBench (pass@1 %)	
ExTS (quality-gated PW)	**46.2**$\pm$0.4
Score-agnostic PW	45.0$\pm$0.9

Classical mechanisms do not transfer directly. Table 6 replaces ExTS components with their classical counterparts: fixed FPU (Gelly and Wang, 2006) instead of the virtual child, and score-agnostic progressive widening (Coulom, 2007) instead of quality-gated widening. On DROP, both FPU constants underperform ExTS (-2.0 pp and -1.6 pp), and the gap between FPU = 0.5 and FPU = 0.8 (+0.4 pp) illustrates a fundamental limitation: fixed FPU is sensitive to the constant's value, which interacts with exploration C, shaping temperature T, and domain score scale. The virtual child sidesteps this coupling by sampling from observed rewards, adapting automatically as the score distribution evolves. Score-agnostic widening wastes budget expanding low-quality nodes.

Exploration type: PUCT vs. UCT. Table 7 compares the default PUCT (linear decay) against UCT (logarithmic decay). HotpotQA strongly prefers PUCT (+6.45 pp), where fast decay concentrates budget on proven chains. HoVeR prefers UCT (+0.88 pp), where sustained exploration buffers against score drift. This task-dependent sensitivity motivates the diagnostic analysis in Section 5.2.

5.4 Tree-Shape Diagnostics

We verify that ExTS redirects budget from breadth-first exploration into selective refinement by comparing tree shape on K-MSE (216 molecules, $B =$

Table 7: Exploration type ablation: PUCT (default) vs. UCT, 3 seeds.

Dataset	PUCT	UCT
HotpotQA	**64.89**±0.83	58.44±2.20
HoVeR	50.45±2.18	**51.33**±1.44
K-MSE	**91.0**±0.6	89.0±1.5

Table 8: Tree-shape diagnostics on K-MSE ($B=16$, 3 seeds). Non-trivial: 92 molecules where at least one method improves beyond the root.

Method	Leaf dep.	Best dep.	Scorer	ACC (%)
All tasks ($n=216$)				
MCTSr	4.89±.08	0.44±.07	72.2±0.3	89.8±0.4
ExTS$_{M_0=2}$	5.49±.13	0.48±.05	72.5±0.2	90.4±0.4
ExTS$_{M_0=3}$	4.91±.14	0.47±.03	**72.7**±0.1	**91.0**±0.6
Non-trivial tasks ($n=92$)				
MCTSr	4.43±.13	1.03±.16	69.5±0.6	77.2±0.9
ExTS$_{M_0=2}$	4.44±.09	1.12±.11	69.6±0.4	78.6±1.0
ExTS$_{M_0=3}$	3.73±.22	1.09±.06	**70.1**±0.1	**80.1**±1.4

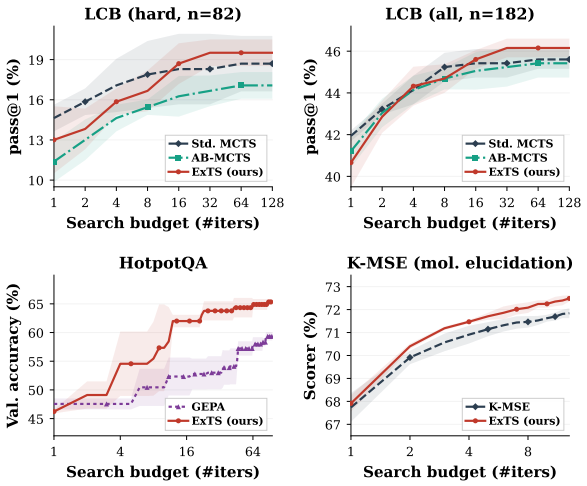

Figure 3: Budget-scaling behavior across three domains, averaged over 3 seeds ($\pm 1\sigma$ bands). The x-axis is search budget. ExTS's advantage is most pronounced on hard instances (top left) and in prompt optimization (bottom left), where iterative refinement matters most.

16, 3 seeds), where MCTSr and ExTS differ only in tree policy. We measure mean leaf depth and depth of the best-scoring node, partitioning molecules into *trivial* (124/216 solved at root) and *non-trivial* (92/216). Full setup details are in Appendix E.

On the non-trivial partition, ExTS ($M_0=3$) produces shallower leaf depths (3.73 vs. 4.43) while finding its best node *deeper* (1.09 vs. 1.03), the signature of selective refinement. This structural shift yields +2.9 pp ACC on the non-trivial subset (80.1% vs. 77.2%), where the overall +1.2 pp gain is concentrated. MCTSr distributes budget uniformly across depths; ExTS concentrates children at promising nodes and deepens only subtrees that improve, placing the best node at the depth where refinement transitions from improving to diminishing returns.

5.5 Budget-Scaling Behavior

We measure performance as a function of evaluation budget B across LiveCodeBench, HotpotQA, and K-MSE (setup details in Appendix E).

Figure 3 shows that ExTS converts budget into performance more efficiently across all domains. On LiveCodeBench, ExTS trails at $B=1$ (all methods reduce to a single sample) but overtakes Standard MCTS at $B\approx 8$ and reaches 46.2% vs. 45.6% (Standard) and 45.4% (AB-MCTS) at $B=128$; on hard problems the gap widens to 19.1% vs. 17.1% (AB-MCTS). On HotpotQA, ExTS reaches 64.9% at $B=70$ versus 58.0% for GEPA Pareto with tighter confidence intervals. On K-MSE, ExTS leads from $B=1$ onward and maintains a growing gap (72.5 vs. 71.8 at $B=13$). The crossover pattern reflects ExTS's calibration overhead: progressive widening and virtual-child scoring require initial samples, but this investment pays off once quality-conditioned pruning can redirect evaluations away from unpromising subtrees. The steeper scaling on hard problems (+6.5 pp from $B=1$ to $B=32$ vs. +3.7 pp for MCTS) confirms that the advantage concentrates where search depth matters most.

6 Conclusion

We introduced ExTS, a tree-search policy for budget-constrained LLM agent search that jointly redesigns both selection and expansion. Rather than applying flat UCT selection and expanding unconditionally, ExTS makes both decisions quality-aware: shaping rewards for narrow score distributions, framing expansion as a value-of-information decision, and gating branching on node quality. Across four diverse domains, ExTS is competitive with or improves over task-specific baselines using a single fixed configuration, with gains concentrated in high-failure-rate and moderate-difficulty regimes. The same fixed configuration further generalizes to an additional domain, GPU kernel optimization (Appendix H). We additionally show that pilot-run diagnostics characterize the structural differences among agentic search problems, explain why different tasks respond to different configurations, and offer practical guidance for adaptation.

Limitations

We use a single model per domain due to the substantial API costs of tree search (each experiment requires hundreds of LLM calls per task instance across multiple seeds) and do not study how model scale alters the search landscape; because the pilot-run diagnostics are conditioned on the model, scorer, and budget, they describe the landscape *as seen by that configuration* rather than intrinsic domain properties, and their transferability across model families is untested. Finally, the diagnostic approach requires a pilot tree that, in deployment on a new task, consumes budget not applied to the final search.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, and 1 others. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. 2025. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *Preprint*, arXiv:2410.20285.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

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

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Guillaume M Jb Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. 2008. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357.

Rémi Coulom. 2007. Computing "elo ratings" of move patterns in the game of go. *ICGA journal*, 30(4):198–208.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378.

Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Turkmen, Tony Hu, Huzefa Rangwala, and 1 others. 2026a. Mlzero: A multi-agent system for end-to-end machine learning automation. *Advances in Neural Information Processing Systems*, 38:69001–69070.

Haoyang Fang, Wei Zhu, Boran Han, Alex Zhang, Zhenyu Pan, Shuo Yang, Shuai Zhang, Jiading Gai, Peng Tang, Cuixiong Hu, and 1 others. 2026b. Llmzero: Discovering adaptive training strategies for rl post-training via llm agents. *arXiv preprint arXiv:2606.18388*.

Jiading Gai, Shuai Zhang, Kaj Bostrom, Jin Huang, Vihang Patil, Haoyang Fang, Bernie Wang, Huzefa Rangwala, and George Karypis. 2026. Optimizing cuda like a human: Micro-profiling tools as expert surrogates for llm-based gpu kernel optimization. *arXiv preprint arXiv:2606.26453*.

Sylvain Gelly and Yizao Wang. 2006. Exploration exploitation in go: Uct for monte-carlo go. In *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*.

Kehan Guo, Bozhao Nan, Yujun Zhou, Taicheng Guo, Zhichun Guo, Mihir Surve, Zhenwen Liang, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Can llms solve molecule puzzles? a multimodal benchmark for molecular structure elucidation. In *Advances in Neural Information Processing Systems*, volume 37, pages 134721–134746. Curran Associates, Inc.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *Preprint*, arXiv:2305.14992.

Zelin He, Haotian Lin, Boran Han, Wei Zhu, Haoyang Fang, Bernie Wang, Xuan Zhu, Runze Li, and Matthew Reimherr. 2026. Reskill: Reconciling skill creation with policy optimization in agentic rl. *arXiv preprint arXiv:2606.01619*.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. 2026. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. *Advances in Neural Information Processing Systems*, 38:35448–35484.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Live-

codebench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. 2020. Hover: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3441–3460.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, and 1 others. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. 2026. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pages 74026–74078.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. 2025. S*: Test time scaling for code generation. *Preprint*, arXiv:2502.14382.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Preprint*, arXiv:1603.06560.

Yushu Li, Wenlong Deng, Jiajin Li, and Xiaoxiao Li. 2026. Spend less, reason better: Budget-aware value tree search for llm agents. *Preprint*, arXiv:2603.12634.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, and 1 others. 2026. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, and 1 others. 2025. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *Preprint*, arXiv:2408.06195.

Christopher D Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734.

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

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and 1 others. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, and 1 others. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, volume 2024, pages 23967–24001.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068.

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. 2024. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *Preprint*, arXiv:2410.02884.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2025. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pages 34040–34077.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

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

Xiang Zhuang, Bin Wu, Jiyu Cui, Kehua Feng, Xiaotong Li, Huabin Xing, Keyan Ding, Qiang Zhang, and Huajun Chen. 2025. Boosting llm's molecular structure elucidation with knowledge enhanced tree search reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22561–22576.

Contents

A Related Work

Search for LLM reasoning and generation. Search strategies for LLM generation can be organized by whether their primary contribution lies in the *harness* (the framework surrounding the search) or the *search algorithm* itself (selection and expansion policies).

On the harness side, many systems wrap standard MCTS with LLM-specific components while retaining vanilla UCT for selection and expansion. Some operate as linear refinement chains without tree structure (Shinn et al., 2023; Madaan et al., 2023; Fang et al., 2026a,b). LATS (Zhou et al., 2023) integrates MCTS with LLM value functions and self-reflection but does not modify the tree policy. rStar (Qi et al., 2024) defines rich reasoning actions with mutual verification between two SLMs. RAP (Hao et al., 2023) repurposes the LLM as a world model within standard MCTS. SWE-Search (Antoniades et al., 2025) introduces a multi-agent evaluation framework with a hybrid value function for software engineering tasks. PromptAgent (Wang et al., 2024) frames prompt optimization as MCTS-style strategic planning with error feedback. LLaMA-Berry (Zhang et al., 2024) replaces raw value estimates with pairwise preference aggregation for mathematical reasoning but retains standard UCT selection. S* (Li et al., 2025) combines parallel sampling with sequential refinement and execution-grounded selection for code generation, contributing a scaling pipeline rather than a tree policy. MLMaster (Liu et al., 2025) applies vanilla UCT to data science automation. In each case, the primary innovation is the surrounding framework or evaluation mechanism; the internal selection and expansion dynamics remain standard or unmodified, representing a gap where better search policies could yield further gains.

On the search algorithm side, fewer works redesign how budget is allocated across candidates. AB-MCTS (Inoue et al., 2026) replaces UCT with Thompson sampling for code generation. GEPA (Agrawal et al., 2025) uses Pareto-frontier selection over scored candidates for prompt optimization. AFlow (Zhang et al., 2025) applies score-weighted random sampling from recent rounds for workflow optimization. BAVT (Li et al., 2026) introduces budget-conditioned node selection via a power-law exponent that shifts from exploration to exploitation as budget depletes, evaluated on multi-hop QA tasks. BAVT is not open-sourced, precluding direct comparison.

ExTS departs from most of these methods by making the exploration/exploitation balance *reward-aware*: instead of a fixed exploration constant, a reward-agnostic first-play-urgency value, or progressive widening tied to visit count alone, it lets observed validation rewards drive both the value of expansion and how wide a node may branch. Its virtual child is the clearest example: where first-play urgency (Gelly and Wang, 2006) assigns the same fixed value to every unexpanded child, the virtual child is a node-specific, data-dependent estimate resampled from the parent's shaped-reward pool and scored at a fair-share visit count, which we treat as a value-of-information-style estimate of expansion value. Its closest relative is the adaptive branching of AB-MCTS (Inoue et al., 2026) (one of our baselines), which likewise lets a hypothetical branch compete with existing children. However, AB-MCTS-A relies on parametric Bayesian posteriors with conjugate priors, while AB-MCTS-M dynamically estimates two posterior distributions via MCMC, which can be less effective when the search budget is small; the virtual child, by contrast, is a nonparametric, prior-free bootstrap coupled with quality-gated widening. We compare against AB-MCTS-A (Gaussian), the strongest AB-MCTS variant on LiveCodeBench. Finally, whereas each of these baselines targets a single task type, ExTS redesigns both selection and expansion and holds a single fixed configuration across four structurally distinct domains, plugging into each framework's existing search interface (GEPA, TreeQuest, K-MSE, AFlow). Table 9 summarizes this positioning against prior LLM tree-search methods.

Test-time compute allocation and sampling baselines. Several lines of work study inference-time budget allocation from complementary perspectives. Snell et al. (2024) characterize when repeated sampling versus sequential revision is compute-optimal for reasoning tasks, providing scaling predictions at the problem level. ExTS addresses a different use case, agentic search with iterative refinement, but its mechanisms for efficient budget allocation could in principle be applied within test-time scaling frameworks. Best-of-N sampling (Brown et al., 2024) generates N independent candidates and returns the best. ExTS instead exploits parent-child refinement structure, achieving better efficiency when iterative improvement is produc-

tive. Sequential halving and Hyperband (Li et al., 2018) efficiently eliminate unpromising configurations but assume independent candidates. ExTS handles tree-structured refinement where parent quality predicts child quality.

Table 9: Positioning of ExTS relative to prior LLM tree-search methods. "Focus" indicates whether the primary contribution is the search harness (H) or the search algorithm (A).

Method	Focus	Selection	Expansion	Fail.-aware
Linear	H	Sequential	N/A	No
ToT	H	BFS/DFS	Full	No
LATS	H	UCT	Full	No
rStar	H	UCT	Full	No
RAP	H	UCT	Full	No
SWE-Search	H	UCT + hybrid value	Full	No
LLaMA-Berry	H	UCT + pairwise	Full	No
S*	H	Parallel + refine	Pipeline	No
MLMaster	H	UCT	Full	No
PromptAgent	H	UCT	Full	No
GEPA	A	Pareto	Stateless	No
AFlow	A	Score-weighted	Random	No
AB-MCTS	A	Thompson	Adaptive	No
BAVT	A	Budget-conditioned	Structural	No
ExTS	A	ExUCT	Gated + adaptive	Yes

Evolutionary search for LLM systems. ShinkaEvolve (Lange et al., 2026) and EvoX (Liu et al., 2026) are population-based evolutionary optimizers that run at a far larger budget than ExTS's tight per-query regime (ShinkaEvolve uses roughly 150 evaluations for its headline result, whereas several of our settings run well below this, e.g., 16 rollouts on K-MSE and 20 rounds on AFlow), so they differ from ExTS in search family, budget, and search space. Even GEPA (Agrawal et al., 2025), one of our prompt-optimization baselines, is itself a Genetic-Pareto evolutionary method, yet ExTS improves over it by 10.8% on HotpotQA at 6.4× lower variance. ExTS is instead a complementary, plug-and-play tree policy whose value-of-information view of expansion is orthogonal to how candidates are generated, and matched-budget comparisons between tree and evolutionary search (and hybrids of the two) are a valuable direction for future work.

Prompt and Skill optimization. OPRO (Yang et al., 2024) uses LLMs as optimizers over scored solution histories. MIPROv2 (Opsahl-Ong et al., 2024) applies Bayesian surrogate optimization to DSPy instruction tuning. For skill optimization, ReSkill (He et al., 2026) selects among candidate skills via Thompson sampling in agentic RL. ExTS can serve as a drop-in selection component within such frameworks.

Self-improving agents. Reflexion (Shinn et al., 2023) uses verbal self-reflection for iterative improvement but follows a linear chain rather than a tree, missing the opportunity to explore alternative refinement paths. ReAct (Yao et al., 2022) synergizes reasoning and acting but uses greedy selection. These approaches are complementary, and ExTS could serve as the search backbone for systems that currently rely on linear or greedy strategies.

Automated workflow optimization. AFlow (Zhang et al., 2025) automates agentic workflow generation by searching the space of LLM-based operator graphs. We directly compare against AFlow's search strategy in Section 4.4.

Molecular structure elucidation. K-MSE (Zhuang et al., 2025) already applies MCTS with a neural molecule-spectrum scorer to molecular structure elucidation, making it a directly relevant baseline. Our K-MSE evaluation uses the same scorer and knowledge base, isolating the effect of the selection/expansion policy.

B Default Configuration and Complete Algorithm

Algorithm 2 gives the complete ExTS search loop and Table 10 lists the default hyperparameters used across all experiments.

Algorithm 2 ExTS: Complete Search Loop

Require: x_0, refinement operator $\mathcal{A}$, validator f, budget B

Initialize tree $\mathcal{T}$ with root v_0, $x_{v_0} \leftarrow x_0$, $s_{v_0} \leftarrow f(x_0)$
while budget B not exhausted **do**
  $v \leftarrow$ ExUCT-Select(v_0) Alg. 1
  $x' \leftarrow \mathcal{A}(x_v)$ LLM refinement
  $r \leftarrow f(x')$
  if $r = \perp$ **then**
    Backprop-Failure(v); **continue**
  end if
  Create child u: $x_u \leftarrow x'$, $s_u \leftarrow r$
  children$(v) \leftarrow$ children$(v) \cup \{u\}$
  Backprop-Success(v, r)
end while
return $\arg\max_{v \in \mathcal{T}} s_v$

Table 10: ExTS default configuration, fixed across all experiments.

Parameter	Default	Rationale
Exploration type	PUCT	Linear decay for tight budgets
Exploration C	1.0	Conservative; no policy prior
Success-rate α	$\sqrt{2}$	Amplified failure discounting
Temperature T	0.3	Top-20% emphasis
Initial children M_0	3	Moderate branching
Widening base b	2	$\log_2$ growth
Widening threshold n_0	32	Delay widening
Widening quantile γ	0.75	Top-quartile widening
Score gate τ	0.25	Mild gating

C Formal Definitions of Pilot-Run Diagnostics

Section 2.1 introduces four measurable pilot-run diagnostics that characterize the search landscape of validation-heavy domains. We provide rigorous definitions below, expressed in terms of the search tree $\mathcal{T}$ built during a pilot run of budget B. Table 11 summarizes the notation used throughout.

Table 11: Notation used in formal property definitions.

Symbol	Description
$\mathcal{T}$	Search tree rooted at initial candidate x_0
$V(\mathcal{T})$	Set of all nodes in $\mathcal{T}$
$I(\mathcal{T})$	Set of internal (expanded) nodes: $\{v \in V(\mathcal{T}) : \lvert\text{children}(v)\rvert > 0\}$
n_v	Visit count of node v (incremented via backpropagation from all expansions in the subtree rooted at v)
n_v^+	Successful visit count (backpropagated from expansions where $f(x') \neq \perp$)
n_v^-	Failed visit count: $n_v^- = n_v - n_v^+$
s_v	Validation score $f(x_v)$ for node v's candidate x_v
S^+	Set of successfully validated nodes: $\{v \in V(\mathcal{T}) : s_v \text{ is defined}\}$
$s_{\min}, s_{\max}$	$\min_{v \in S^+} s_v$ and $\max_{v \in S^+} s_v$ respectively

Definition 1: Failure rate (ρ). The fraction of expansion attempts that do not produce a successfully validated candidate:

$$\rho = \frac{n_{v_0}^-}{n_{v_0}} \tag{11}$$

where v_0 is the root. Since backpropagation increments all ancestors after each expansion, n_{v_0} equals the total number of expansion attempts across the tree and $n_{v_0}^-$ equals the total failures. An attempt counts as failed if f returns $\perp$ (e.g., syntax error, runtime error, timeout, or a domain-specific rejection criterion). By construction, $\rho \in [0, 1]$. Domains with $\rho > 0.5$ spend more than half their budget on failed attempts, making failure-aware mechanisms (dual backpropagation, success-rate weighting) essential.

Definition 2: Normalized score deviation ($\hat{\sigma}_f$). The spread of validation scores across all successfully evaluated candidates, normalized by the observed range:

$$\hat{\sigma}_f = \frac{1}{s_{\max} - s_{\min}} \sqrt{\frac{1}{|S^+| - 1} \sum_{v \in S^+} (s_v - \bar{s})^2} \tag{12}$$

where $\bar{s} = \frac{1}{|S^+|} \sum_{v \in S^+} s_v$. If $s_{\max} = s_{\min}$ or $|S^+| < 2$, we define $\hat{\sigma}_f = 0$. Low values indicate a flat score landscape where exploitation shaping must work harder to differentiate candidates.

Definition 3: Score drift (κ). Score drift quantifies the non-stationarity of normalized node scores as the tree grows. At each iteration t a new node with score s_{new} is added. Let $s_{\min}^t$ and $s_{\max}^t$ denote the running minimum and maximum scores among all nodes at iteration t. Let $\mathcal{T}_\Delta = \{t : s_{\min}^t \neq s_{\min}^{t-1} \text{ or } s_{\max}^t \neq s_{\max}^{t-1}\}$ be iterations where the normalization bounds change. The normalized score of node v at time t is $\hat{s}_v^t = (s_v - s_{\min}^t)/(s_{\max}^t - s_{\min}^t)$. Then:

$$\kappa = \frac{1}{|\mathcal{T}_\Delta|} \sum_{t \in \mathcal{T}_\Delta} \frac{1}{|V_t|} \sum_{v \in V_t} \left| \hat{s}_v^t - \hat{s}_v^{t-1} \right| \tag{13}$$

where V_t is the set of nodes existing before iteration t (excluding the newly added node). If $|\mathcal{T}_\Delta| = 0$ or the score range is zero, we define $\kappa = 0$. High κ indicates that newly discovered candidates frequently shift the normalization bounds, changing the relative exploitation values of existing candidates.

Definition 4: Refinement variance ($\hat{\sigma}_\mathcal{A}$). The refinement variance measures the stochasticity of the refinement operator $\mathcal{A}$ when applied to the same parent state. For each internal node $v \in I(\mathcal{T})$ with at least two successfully validated children, let $C_v^+ = \{c \in \text{children}(v) : s_c \text{ is defined}\}$ and $\bar{f}_v = \frac{1}{|C_v^+|} \sum_{c \in C_v^+} s_c$. The per-node refinement variance is:

$$\sigma_{\mathcal{A},v}^2 = \frac{1}{|C_v^+| - 1} \sum_{c \in C_v^+} (s_c - \bar{f}_v)^2 \tag{14}$$

Let $I_2 = \{v \in I(\mathcal{T}) : |C_v^+| \geq 2\}$. The global refinement variance is:

$$\hat{\sigma}_\mathcal{A} = \frac{1}{s_{\max} - s_{\min}} \sqrt{\frac{1}{|I_2|} \sum_{v \in I_2} \sigma_{\mathcal{A},v}^2} \tag{15}$$

Distinction from $\hat{\sigma}_f$. $\hat{\sigma}_f$ measures the spread of scores across *all* candidates in the tree, reflecting the combined effect of different parent states,

depths, and refinement histories. In contrast, $\hat{\sigma}_{\mathcal{A}}$ isolates the variance attributable to the LLM's sampling stochasticity by conditioning on the parent. A domain may have low $\hat{\sigma}_f$ (flat overall landscape) but high $\hat{\sigma}_{\mathcal{A}}$ (diverse candidates from any single parent), or vice versa. High $\hat{\sigma}_{\mathcal{A}}$ indicates that resampling from the same node is productive, providing implicit exploration; low $\hat{\sigma}_{\mathcal{A}}$ indicates refinements are largely deterministic and explicit exploration via higher C is needed.

Computation details for Table 1. All properties are computed from pilot trees built by running the baseline method for each dataset with budget B, averaged across 3 seeds ($\{0, 42, 1024\}$).

ρ: We read n_{v_0} and $n^+_{v_0}$ directly from the root node's backpropagated statistics stored in the pilot tree.

$\hat{\sigma}_f$: We collect the score s_v of every node $v \in S^+$ in the pilot tree and compute the sample standard deviation divided by $s_{\max} - s_{\min}$. For K-MSE, which runs 216 independent molecule searches per seed, we pool all node scores across molecules before computing a single $\hat{\sigma}_f$ per seed.

κ: We replay the tree construction in node-creation order, tracking $s^t_{\min}$ and $s^t_{\max}$ after each insertion. At each iteration where bounds change, we compute the mean absolute shift in normalized scores across existing nodes and average over all such events.

$\hat{\sigma}_{\mathcal{A}}$: We estimate refinement variance from sibling scores in the pilot tree: for each internal node v with $|C^+_v| \geq 2$ successfully validated children, we compute $\sigma^2_{\mathcal{A},v}$ from the children's scores and aggregate via Equation 15. This treats siblings as approximate samples from $\mathcal{A}(x_v)$, which is valid when the tree context does not meaningfully change between sibling expansions. For LiveCodeBench, where the pilot does not save per-node tree structure, we instead run $K = 10$ i.i.d. refinements from a fixed parent state for 5 representative problems and aggregate.

D Detailed Experimental Setup

D.1 Prompt Optimization

We evaluate ExTS within the GEPA prompt optimization framework (Agrawal et al., 2025), which optimizes natural-language instructions in DSPy programs (Khattab et al., 2023). We use two multi-hop reasoning benchmarks: HotpotQA (Yang et al., 2018), a question answering task with a 4-predictor DSPy program and ColBERTv2 retrieval (Santhanam et al., 2022) over 5.2M Wikipedia abstracts, and HoVeR (Jiang et al., 2020), a fact verification task. Both use a test set of 300 examples with accuracy (%) as the evaluation metric, and all experiments are run across seeds $\{0, 42, 1024\}$ using Qwen3-8B (Yang et al., 2025) via vLLM (Kwon et al., 2023) (TP=4, temperature 0.6, top-p 0.95, thinking enabled).

In the ExTS formulation, the candidate set $\mathcal{S}$ consists of DSPy instruction configurations. The refinement operator $\mathcal{A}$ is an LLM instruction proposer conditioned on failure feedback. The validator f evaluates on the full train and validation set, returning $\perp$ if the new score does not exceed the parent's. Score stationarity κ is high because aggregate scores shift when new programs join the Pareto frontier.

We compare four conditions: Baseline (unoptimized seed program), GEPA Pareto (task-native Pareto-frontier selection (Agrawal et al., 2025)), ExTS default (Table 10), and per-dataset ExTS* (HotpotQA: $\alpha = 2.0$; HoVeR: UCT exploration, $\tau = 0$), with the optimization budget matched across methods.

D.2 Code Generation

We evaluate ExTS within the TreeQuest framework (Inoue et al., 2026) on LiveCodeBench (Jain et al., 2024), which contains 182 problems from release v6 (January to April 2025) split into 45 easy, 55 medium, and 82 hard problems. Public test cases serve as the search reward signal and private test cases determine the final pass@1 metric. We use Claude Sonnet 4 with temperature 0.6, a budget of 128 steps per problem, and report results over seeds $\{0, 42, 1024\}$.

In the ExTS formulation, $\mathcal{S}$ consists of code solutions, $\mathcal{A}$ is an LLM code editor conditioned on test failure traces, and f is the full public test suite execution (returning $\perp$ on syntax errors or compilation failures). Score stationarity κ is low because existing solution scores do not change. We compare StandardMCTS (UCT with $C = \sqrt{2}$, samples per action of 5), AB-MCTS-A (Thompson sampling with Gaussian conjugate prior (Inoue et al., 2026)), and ExTS default (Table 10). ExTS* sets $T = 0.5$.

D.3 Molecular Structure Elucidation

We evaluate ExTS on the K-MSE molecular structure elucidation task (Zhuang et al., 2025), in which an LLM must deduce a molecule's SMILES rep-

resentation from infrared (IR) and nuclear magnetic resonance (NMR) spectral data together with a molecular formula. The evaluation set consists of 216 molecules from the MolPuzzle dataset (Guo et al., 2024), each paired with its IR spectrum image, carbon-13 and proton NMR spectra, and molecular formula. The primary metrics are Morgan fingerprint Tanimoto similarity (FTS) and exact-match accuracy (ACC), where ACC canonicalizes both predicted and ground-truth SMILES via RDKit before comparison. We use Claude Sonnet 4.6 with a pre-trained molecule-spectrum alignment scorer for reward computation.

In the ExTS formulation, $\mathcal{S}$ consists of SMILES strings, $\mathcal{A}$ is an LLM that critiques the current prediction against spectral data and a retrieved knowledge base of 593 molecular substructures and then proposes a revised SMILES, g checks chemical validity via RDKit, and f is a neural scorer that computes cosine similarity between the predicted molecule's embedding and the target spectrum's embedding scaled to $[0, 100]$. We compare ExTS against K-MSE's original MCTSr (Zhuang et al., 2025) and chain-of-thought prompting (CoT; Wei et al., 2023), using 16 rollouts per molecule and 3 seeds ($\{0, 42, 1024\}$). ExTS uses the default configuration (Table 10). ExTS* sets $T = 0.5$.

D.4 Agentic Workflow Optimization

We evaluate ExTS within the AFlow framework (Zhang et al., 2025) on the DROP reading comprehension benchmark (Dua et al., 2019). DROP consists of paragraphs with questions requiring discrete reasoning (counting, sorting, arithmetic) over text; the test set contains 800 problems scored by token-level F1.

In the AFlow formulation, a "workflow" is a Python function that orchestrates one or more LLM calls (using operators such as Generate, Format, Review, Ensemble) to answer a question given a passage. The search space $\mathcal{S}$ consists of these workflow programs. The refinement operator $\mathcal{A}$ is an optimizer LLM (Claude Sonnet 4.5) that proposes mutations to the workflow graph conditioned on execution logs and past experience. The validator f executes the workflow on a validation split of 200 problems over 5 rounds and returns the mean F1 score. The failure rate is low ($\rho \approx 0.05$) because workflows almost always produce parseable output; failures are rare execution errors.

We compare three search strategies sharing the same optimizer LLM, executor LLM (Claude Haiku 4.5), validation protocol, and test evaluation:

- **AFlow** (Zhang et al., 2025): Score-weighted random parent sampling from recent rounds. No tree structure or visit statistics.
- **CoT (linear)**: Sequential refinement chain that always extends the deepest node. No branching or selection policy.
- **ExTS**: Default configuration (Table 10). ExTS* sets $\tau = 0.5$, $M_0 = 4$.

Budget is 20 search rounds per seed, with seeds $\{0, 42, 1024\}$.

E Analysis Setup Details

Tree-shape diagnostics (Section 5.4). We compare MCTSr (binary expansion, no gating) against ExTS at two branching caps: $M_0 = 2$ (matching MCTSr's effective branching) and $M_0 = 3$ (the default). All methods share the same scorer, retriever, and LLM (Claude Sonnet 4.6). Molecules are partitioned into *trivial* (best node at depth 0 for all methods in all seeds; 124/216) and *non-trivial* (at least one method in one seed improves beyond the root; 92/216). On the full dataset, all methods achieve similar leaf depths ($\sim$5) because 57% of molecules are trivially solved and MCTS exhausts budget on already-solved problems. The non-trivial partition isolates the algorithmic difference. The scorer gap on non-trivial molecules (+0.6 over MCTSr) translates to a meaningful ACC improvement because this partition selects precisely the molecules where the scorer provides actionable signal.

Budget-scaling behavior (Section 5.5). On LiveCodeBench we run Standard MCTS, AB-MCTS, and ExTS with B=128 (182 problems, 3 seeds) and extract pass@1 at each intermediate budget from the cumulative score trajectory: a problem is solved at budget b if its best public-test solution within the first b evaluations also passes all held-out private tests. On HotpotQA we compare ExTS and GEPA Pareto over 78 search iterations (3 seeds), tracking cumulative best validation accuracy. On K-MSE we compare ExTS and MCTSr over 13 evaluation steps (3 seeds, 216 molecules), tracking cumulative best scorer score.

F More Ablations on LiveCodeBench

Table 12 isolates two components on LiveCodeBench. Removing gated progressive widening costs -1.3 pp overall and -1.6 pp on hard problems, confirming that quality-conditioned branching prevents budget waste on unpromising subtrees.

Removing root exclusion (allowing the score gate to block root expansion) costs -1.0 pp overall and -2.0 pp on hard, with notably higher variance, because gating the root can starve the tree of initial diversity when early candidates score poorly.

Table 12: Leave-one-out ablation on LiveCodeBench (pass@1 %, 3 seeds).

Configuration	All (%)	Hard (%)
ExTS (full)	**46.2**$\pm$0.4	**19.1**$\pm$1.4
w/o Gated prog. widening	44.9$\pm$0.7	17.5$\pm$1.4
w/o Root exclusion	45.2$\pm$1.5	17.1$\pm$2.5

G Compact Diagnostic Guide

Table 13 condenses the pilot-run findings of Section 5.2 into a compact diagnostic guide. We intend it as both a deployment aid and an analytic contribution: the four diagnostics characterize the axes along which agentic search landscapes structurally differ (failure rate, score spread, drift, and refinement variance), and each axis maps to a concrete hyperparameter adjustment. These full-dataset diagnostics are an analytic study of why agentic search landscapes differ, not a deployment requirement: the fixed default configuration needs no pilot and already matches or beats every task-specialized baseline, while the ExTS* refinements adjust only one or two hyperparameters for modest gains (typically 0.2–1.2 pp). When adaptation is desired, the diagnostics are cheap landscape statistics estimable from a small pilot subset, and most are collected simply by running the baseline method (Section 2.1), a run practitioners would perform anyway, so they largely reuse existing computation rather than new budget.

The default configuration, used when no pilot is run, is PUCT with $C = 1.0$, $\alpha = \sqrt{2}$, $T = 0.3$, $M_0 = 3$, and $\tau = 0.25$ (Table 10).

H Further Evaluation on an Additional Domain: GPU Kernel Optimization

To further probe the generalizability of ExTS, we evaluate it on an additional domain beyond the four in the main text: GPU kernel optimization. Using the same fixed default configuration (Table 10), with no tuning and no diagnostics, we follow the setup of Gai et al. (2026), using Claude Opus 4.6 as the refinement model and their original configuration for all other settings, and reproduce their pipeline on 40 tasks, running each task on a single NVIDIA A100 40GB GPU, and compare against their optimized MCTS baseline, which we label MCTS (optimized). Owing to the substantial compute cost of this domain, we report results from a single run rather than the multiple runs used elsewhere in the paper. In this run, ExTS improves over MCTS (optimized) on every metric, raising the geometric-mean speedup from $4.15\times$ to $4.36\times$, the median speedup from $3.09\times$ to $3.81\times$, and the per-category speedups across all three difficulty levels (Table 14). We note that the original speed validation of Gai et al. (2026) contained an issue that admitted invalid (spurious) speedups; the numbers reported here are the corrected results that exclude those invalid speedups. These results indicate that ExTS's budget-allocation mechanisms generalize to this additional domain without any domain-specific adaptation.

Table 13: Compact diagnostic guide: mapping pilot-run signatures to hyperparameter adjustments, condensing Section 5.2.

Diagnostic signature	What it says about the task	Relevant hyperparameter	Evidence
High failure rate ρ (e.g. >0.5)	most expansions fail, so separating productive nodes is critical	success exponent α	HotpotQA
Low ρ (e.g. <0.1)	failures are rare	success exponent α (little effect, keep default)	K-MSE, DROP
High score drift κ	early rankings are unreliable as bounds re-rank nodes	exploration type (use UCT) and score gate τ	HoVeR
Low κ	rankings stabilize early, safe to commit	exploration type (use PUCT)	HotpotQA
Low $\hat{\sigma}_A$ and low κ, or weak validator	aggressive exploitation over-commits to spurious gaps	shaping temperature T	K-MSE, DROP, LCB

Table 14: Further evaluation on an additional domain, GPU kernel optimization: 40 GPU-kernel tasks, each run on a single NVIDIA A100 40GB GPU, using ExTS's fixed default configuration (Table 10) with no tuning or diagnostics. All values are speedup factors ($\times$); L1, L2, and L3 are the difficulty levels (categories) defined by Gai et al. (2026). ExTS improves over MCTS (optimized) on every metric. Results are from a single run with Claude Opus 4.6, owing to the substantial compute cost of this domain. The reported speedups are the corrected results that exclude the invalid speedups admitted by the original validation of Gai et al. (2026).

Method	Geo. mean	Median	L1	L2	L3
MCTS (optimized)	4.15$\times$	3.09$\times$	3.68$\times$	5.04$\times$	3.69$\times$
ExTS	**4.36**$\times$	**3.81**$\times$	**3.76**$\times$	**5.25**$\times$	**4.30**$\times$

I Worked Example: The Virtual Child

Setup. The numbers are taken from a real default-config ExTS run on HotpotQA (seed 0, Qwen3-8B) and computed exactly as in Equations 7 and 8 and Algorithm 1. We reconstruct the iteration where the run's best candidate (score 68.67) is created: at iteration 104 the search descends root $\rightarrow v_1 \rightarrow$

$v_3 \to v_7$ and expands the leaf v_7. At that point the global score range is $[50.33, 65.0]$ (68.67 does not yet exist, being the child about to be created) and the defaults apply ($T = 0.3$, $\alpha = \sqrt{2}$, $C = 1.0$, PUCT with parent exponent 0.5). We show the virtual-child comparison at the pivotal node v_1, which has $n_v = 28$ visits, $n_v^+ = 4$ successful visits (success rate 0.143), score $s_v = 52.33$, and a single child v_3 ($n = 24$, $n^+ = 3$, score 55.33, rewards $\{62.33, 60.67, 65.0\}$).

Best real child. The rewards of v_3 normalize and shape (Eq. 3) to $\{0.528, 0.351, 1.000\}$ with mean 0.626; its success rate $3/24 = 0.125$ gives $(0.125)^{\sqrt{2}} = 0.053$, so the exploitation term is $0.053 \cdot 0.626 = 0.033$. With PUCT exploration $C \cdot \sqrt{n_v}/(1 + n_{\text{child}}) = \sqrt{28}/(1 + 24) = 0.212$, the total is $\text{ExUCT}(v_3) = 0.245$.

Virtual child. The fair-share visit count (Eq. 7) is $n_{\text{fair}} = 24/1 = 24$ and the sample size is $k = \text{round}(n_{\text{fair}} \cdot n_v^+/n_v) = \text{round}(24 \cdot 0.143) = 3$. The virtual child draws $k = 3$ scores from the parent's pool (reward history plus s_v), namely $\{55.33, 62.33, 60.67, 65.0, 52.33\}$ with shaped values $\{0.078, 0.528, 0.351, 1.000, 0.021\}$. Its exploration term is $\sqrt{28}/(1+24) = 0.212$, so in expectation it scores $(0.143)^{\sqrt{2}} \cdot 0.396 + 0.212 = 0.237$, just below the best real child at 0.245.

Decision and interpretation. The comparison is close, 0.237 versus 0.245, and turns on the shaped sample: widening wins only when the mean of the three draws exceeds about 0.52, which requires repeatedly drawing the two high pool entries (0.528 and 1.000). Enumerating all draws, this happens with probability 0.256, so ExTS deepens into v_3 about three quarters of the time, and the descent continues through v_3 to the leaf v_7 whose expansion produces the global best of 68.67. The virtual child's exploration bonus, computed at the honest fair-share count $n_{\text{fair}} = 24$, does not on average overcome a maturing real child that already carries an exploitation signal, so the mechanism favors deepening the productive chain. A fixed first-play-urgency constant cannot make this comparison, as it sees neither the parent's reward distribution nor the maturity of the existing children. The same close comparison recurs at the root (virtual 0.372 versus best real 0.381), while at v_3 itself widening is gated off (it is at the branching cap $M_0 = 3$ and below the widening visit threshold), so the search simply deepens into v_3's least-visited child v_7.

J Ethical Considerations and Broader Impact

ExTS is a general-purpose tree-search policy that improves budget efficiency for LLM-based agentic search. As a search algorithm, it could in principle accelerate any agent-based system, including potentially harmful ones. However, the method itself does not introduce new capabilities beyond what existing LLM agents already possess; it only improves how evaluation budget is allocated among candidates. All experiments use publicly available benchmarks and models accessed through standard APIs. The molecular elucidation task involves deducing known structures from spectra rather than de novo generation of novel compounds. We do not foresee significant dual-use risks specific to this work beyond those inherent to general LLM agent research.

K Artifact Documentation

Table 15 documents the artifacts used in this work, their licenses, and whether our use is consistent with intended purpose.

Table 15: Artifacts used, with licenses and intended-use consistency.

Artifact	Type	License	Consistent Use
GEPA	Framework	MIT	Yes
TreeQuest	Framework	Apache 2.0	Yes
AFlow	Framework	MIT	Yes
K-MSE	Framework	MIT	Yes
DSPy	Framework	MIT	Yes
vLLM	Inference engine	Apache 2.0	Yes
RDKit	Library	BSD-3-Clause	Yes
ColBERTv2	Model	MIT	Yes
Qwen3-8B	Model	Apache 2.0	Yes
HotpotQA	Dataset	CC BY-SA 4.0	Yes
HoVeR	Dataset	MIT	Yes
DROP	Dataset	CC BY-SA 4.0	Yes
LiveCodeBench	Benchmark	MIT	Yes
MolPuzzle	Dataset	MIT	Yes

All frameworks are used as search harnesses into which we integrate ExTS, consistent with their intended purpose as research tools. Datasets and benchmarks are used for evaluation as intended by their creators. Models are accessed via their supported APIs or inference engines.

L Use of AI Assistants

We used AI assistants to refine the grammar of the paper's text.