Tree-Structured Synergy of Large Language Models and Bayesian Optimization for Efficient CASH

Beicheng Xu[†] **Weitong Qian**[†] **Lingching Tung**[†] **Yupeng Lu**[†] **Bin Cui**[†]
[†]School of CS & Key Lab of High Confidence Software Technologies (MOE) &
Beijing Key Laboratory of Software and Hardware Cooperative Artificial Intelligence Systems,
Peking University, Beijing, China.
{beichengxu,weitong.qian,lingchingtung}@stu.pku.edu.cn
{xinkelyp,cui.bin}@pku.edu.cn

Abstract

To lower the expertise barrier in machine learning, the AutoML community has focused on the CASH problem, which jointly automates algorithm selection and hyperparameter tuning. While traditional methods like Bayesian Optimization (BO) struggle with cold-start issues, Large Language Models (LLMs) can mitigate these through semantic priors. However, existing LLM-based optimizers generalize poorly to high-dimensional, structured CASH spaces. CASH requires coordinated exploration and exploitation both across candidate algorithms and within their algorithm-specific hyperparameter spaces. Yet existing approaches often rely on flat optimization histories that mix observations across algorithms and obscure configuration-refinement trajectories. In this paper, we propose LB-MCTS, a trajectory-structured optimization framework that uses a Monte Carlo Tree Search tree as a shared state for algorithm selection, hyperparameter refinement, and BO–LLM proposer synergy. Within this shared state, BO provides algorithm-specific surrogate modeling for quantitative search, while the LLM exploits path-aware selective memory to generate semantic proposals and reflections. As the surrogate model improves, a reliability-aware proposer policy adaptively shifts from LLM-driven to BO-driven proposals within a unified search trajectory. Experiments on 104 AMLB datasets demonstrate that LB-MCTS consistently outperforms BO-based, LLM-based, and hybrid baselines. Code: PKU-DAIR/open-box.

1 Introduction

In recent years, machine learning has advanced in diverse domains, such as computer vision [11, 20], and recommendation systems [44]. Despite these advancements, developing tailored solutions with strong performance remains a knowledge-intensive task, requiring careful selection of suitable ML algorithms [57] and hyperparameter tuning [36]. To lower barriers and streamline deployment, the AutoML community has introduced the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem [46] and proposed several methods [12, 15, 51] to automate this optimization process. Among them, the dominant approach is Bayesian Optimization (BO), which builds surrogate models of the performance landscape to guide iterative evaluations. However, by treating configurations as abstract numeric vectors, pure BO fails to incorporate prior knowledge, necessitating extensive evaluations to explore high-dimensional and conditional search spaces [7].

Recently, the reasoning and in-context learning (ICL) capabilities of large language models [43] have made them attractive for black-box optimization [19, 53]. LLMs can serve as context-aware "experts" with domain knowledge [45, 56], alleviating the limitations of BO. Early works treat the LLM as a standalone optimizer, iteratively prompting it to obtain candidates [53]. However, pure LLM optimizer suffers from inherent limitations and is unreliable in many practical scenarios [13].

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Because it does not fit a surrogate model (nor quantify uncertainty) and struggles to utilize the full optimization history with a limited context window. Therefore, attempts have been made to integrate LLMs with BO [1, 4], e.g., using LLMs to generate warm-starts or candidates for BO, or interleaving them with BO for alternating optimization. Such synergy has yielded improved performance.

However, existing LLM-based optimizers remain confined to unstructured and low-dimensional problems, leaving the highly structured and high-dimensional CASH problem unexplored. Tackling CASH with LLMs presents three critical challenges that current approaches fail to address: **C1**: **Efficient context utilization**. CASH features a hierarchical search space (i.e., specific hyperparameters active only for specific algorithms) where the history of one algorithm (e.g., SVM) provides negligible guidance for another (e.g., XGBoost). Yet, existing methods indiscriminately populate prompts with the best, recent, or entire optimization history. This approach introduces cross-algorithm noise, leading to low context utilization and high token costs. **C2**: **Exploration-exploitation trade-off**. Success in CASH demands a dual-level exploration-exploitation balance: selecting between distinct algorithms, and navigating between promising vs. unexplored regions within each algorithm's subspace. However, existing LLM-based methods lack an explicit trade-off mechanism, leading to premature convergence and local optima. **C3**: **Effective synergy with BO**. Existing hybrids often neglect BO's structured optimization trajectories, relying instead on a mere exchange of unstructured history. Furthermore, they either use static integration strategies without dynamic adaptation or rely on rigid thresholds to switch between LLM and BO, which generalizes poorly. These challenges share a common root: existing methods lack a structured state that models hierarchical CASH optimization trajectories and coherently integrates BO- and LLM-produced trajectories. Consequently, LLMs receive flat and noisy histories, and BO–LLM interaction reduces to heuristic alternation.

To address this limitation, we propose LB-MCTS, a trajectory-structured optimization framework for CASH. The key idea is to use an MCTS tree as a structured search state, where each path records a hierarchical optimization trajectory and supports BO–LLM proposer synergy through shared trajectory updates. The contributions are summarized as follows: (i) We formulate CASH as tree-structured search over optimization trajectories, enabling root-level algorithm selection and subtree-level exploration-exploitation within each algorithm's HPO space (**C2**). (ii) We realize BO–LLM proposer synergy by unifying BO- and LLM-produced optimization trajectories in the shared tree state and dynamically selecting the proposer for configuration proposal, achieving more sample-efficient optimization than either component alone (**C3**). (iii) We introduce Selective Tuning Memory (STM), which retrieves algorithm-specific and trajectory-relevant trials from the shared tree state, allowing the LLM to reason from focused, low-noise context (**C1**). (iv) Experiments on 104 AMLB datasets show that LB-MCTS achieves the best average validation rank of 2.16, outperforming the strongest baseline RB (3.49), as well as BO-based, LLM-based, and hybrid baselines.

2 Preliminary and Related Work

CASH Problem and Traditional Methods. Let $\mathcal{A} = \{A^1, \ldots, A^K\}$ be a set of K candidate algorithms, where each algorithm A^i is associated with a specific hyperparameter space Λ^i. Given a dataset partitioned into training and validation sets ($\mathcal{D}_{train}, \mathcal{D}_{val}$), CASH aims to jointly identify the optimal algorithm A^* and its configuration λ^* to maximize the validation metric $\mathcal{F}$:

$$(A^*,\ \boldsymbol{\lambda}^*) = \underset{A^i \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda^i}{\operatorname{argmax}} \mathcal{F}(A^i(\boldsymbol{\lambda}, \mathcal{D}_{train}), \mathcal{D}_{val}). \tag{1}$$

The CASH problem was first introduced by Auto-WEKA [46] and solved by Bayesian Optimization (BO) [42, 14, 2, 52], which iteratively fits a surrogate on evaluated configurations and maximizes an acquisition function to select the next trial. Many leading AutoML systems like Auto-sklearn [8], LightAutoML [47], and H2O [21] also adopt BO as the core solver for CASH. Another line of work improves BO by diversifying evaluated models [38, 33]. In addition, genetic programming [31] and the multi-armed bandits (MAB) based methods [22, 23] are also used to address the CASH problem.

LLM-based Optimizer. Recently, researchers have explored LLMs as standalone alternatives to traditional optimizers with their emergent reasoning capabilities. These approaches have been applied to diverse tasks, including chemistry [27], hyperparameter optimization [24], feature engineering [25, 50], and molecular optimization [30]. Typical methods like LLAMBO [24] and OPRO [53] use ICL to iteratively propose solutions. However, these methods are mainly validated on low-dimensional black-box tasks and often require frequent LLM calls, limiting scalability and reliability. Another

recent line integrates BO principles into pre-trained foundation models. FIBO [5] directly samples candidates from the posterior over the optimum, while ToSFiT [29] fine-tunes LLM with a Gaussian Process rewarding model. These methods remain largely generative-model-driven, approximating BO-style candidate generation rather than preserving an explicit numerical BO module, and thus still inherit limitations in numerical fidelity and uncertainty quantification.

In response to the limitations of pure LLM optimizers, hybrid approaches have emerged that combine LLMs and BO [35, 48]. BOPRO [1] optimizes prompt instructions with BO, whereas ADO-LLM [54] uses the LLM to generate BO candidates. SLLMBO [28] alternates between LLM-based exploitation and TPE [2] in a fixed 50:50 ratio. BORA [4] switches between LLM-based search and GP-based BO using a heuristic variance-threshold policy. Although these hybrid approaches improve upon pure LLM optimizers, they are still primarily restricted to low-dimensional settings ($D < 20$) with flat, unstructured parameter spaces. Naively extending them to the highly structured, high-dimensional CASH setting is ineffective due to the challenges **C1**–**C3** discussed in Section 1. Overall, effectively synergizing LLMs and BO for the CASH problem remains an open challenge.

Monte-Carlo Tree Search (MCTS). MCTS builds search trees for structured spaces by balancing exploration (trying new actions) and exploitation (refining promising actions) [18, 3, 9]. Each MCTS iteration consists of four phases: 1) **Selection:** Starting from the root, it traverses the tree to select the node for expansion. Modern variants (e.g., AlphaGo Zero [41]) employ PUCT [37] for selection. For a parent node s, the next child node s_i is selected from the set of children $\mathcal{C}(s)$ to maximize:

$$\arg\max_{s_i \in \mathcal{C}(s)} \left\{ Q(s_i) + c_{puct} \cdot P(s_i) \cdot \frac{\sqrt{N_s}}{1 + N_{s_i}} \right\}, \tag{2}$$

where $Q(s_i)$ is the mean value of s_i, $P(s_i)$ is the prior probability of visiting s_i, and $N_s(N_{s_i})$ denotes the visit count of the parent(child). The constant c_{puct} balances exploitation and exploration. 2) **Expansion:** Add a new child node with an untried action. 3) **Playout:** Conduct a simulation to yield a reward value. 4) **Backpropagation:** Propagate the reward back to update ancestral node statistics. Motivated by this, AlphaD3M [6] and MOSAIC [34] build upon MCTS to explore the pipeline search space. Recent work also combines LLMs with MCTS in sequential decision-making tasks, such as long-horizon robot task planning [55], text-based games [39], and code generation [26]. In contrast, LB-MCTS uses an MCTS tree as a shared CASH state, integrating algorithm selection, LLM-generated semantic proposals, and BO-based optimization into unified optimization trajectories.

3 Method

In this section, we present LB-MCTS, a trajectory-structured optimization framework that uses the **MCTS** search tree to model hierarchical CASH trajectories and synergize **L**LM and **B**O proposers. We first formulate the tree-structured search process, then describe algorithm selection, BO/LLM-based configuration proposal, and the reliability-aware proposer selection policy.

3.1 Tree Structured Search for CASH

We model CASH problem as a sequential decision-making process. As shown in Figure 1, the search space is organized into a tree structure containing three types of nodes:

- **CASH node** (root s_{root}): represents the initial state of the CASH problem. It encompasses all candidate ML algorithms and their respective hyperparameter spaces.
- **Algo nodes** (s_{algo}): immediate children of the root. Each represents the hyperparameter optimization problem for a specific algorithm A^i in the subspace Λ^i.
- **HP nodes** (s_{hp}): all descendants of an Algo node. Each HP node represents a concrete configuration $\boldsymbol{\lambda}_{s_{\text{hp}}}$ of the algorithm and its validation metric $y_{s_{\text{hp}}}$.

Each node s maintains three key statistics: the visit count N_s, the cumulative reward R_s, and the subtree best performance $y_s^{\max}$, i.e., the best validation metric observed so far in the subtree rooted at s. After that, we define actions as transitions between nodes, yielding three action types:

- **Algorithm Selection Action** ($s_{\text{root}} \xrightarrow{a_{\text{sel}}} s_{algo}$): select an algorithm $A^i \in \mathcal{A}$ to be optimized, constraining the search to its specific hyperparameter subspace.

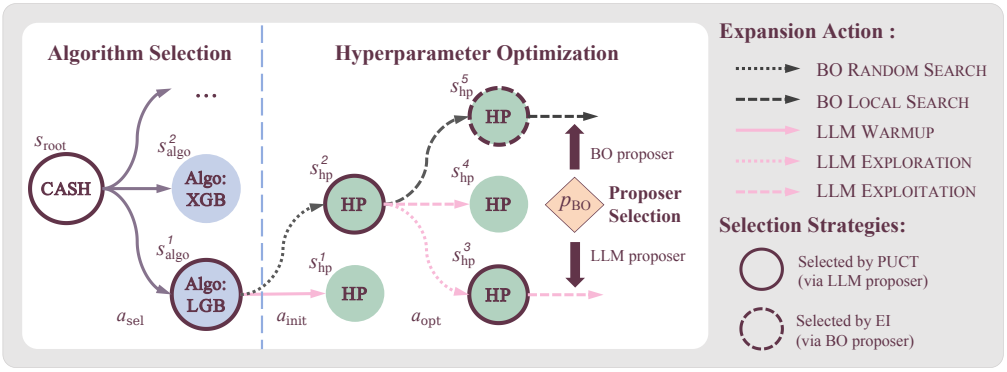


Figure 1: Tree-structured search of LB-MCTS; see Appendix A.3 for a concrete case.

- **Initialization Action** ($s_{\text{algo}} \xrightarrow{a_{\text{init}}} s_{\text{hp}}$): generates the initial configuration $\boldsymbol{\lambda}_0$ for the selected algorithm, serving as an initialization of an optimization trajectory.
- **Optimization Action** ($s_{\text{hp}} \xrightarrow{a_{\text{opt}}} s'_{\text{hp}}$): proposes a new configuration $\boldsymbol{\lambda}_{t+1}$ based on a basic configuration, aiming to refine the performance.

We define an **optimization trajectory** as a root-originated sequence of nodes and actions. *The trajectory structurally captures the process of choosing an algorithm, initializing a configuration, and iteratively refining its hyperparameter values.* CASH is thus reformulated as a **sequential decision-making process** to find the optimal trajectory that maximizes the validation performance of the final configuration. The process can be iteratively solved by MCTS introduced in Section 2. In our setting, Expansion represents executing an action (a_{init} or a_{opt}) to propose a configuration via LLM/BO. Playout evaluates the proposed configuration to obtain its validation metric y_{new}. Finally, Backpropagation propagates this value up the trajectory: for each ancestral node s, it computes a local relative-improvement reward $r_s = [y_{new} - y_s^{\max}]_+ / |y_s^{\max}|$ before the subtree-best update, where $[x]_+ = \max(x, 0)$. It then updates the visit count ($N_s \leftarrow N_s + 1$), the cumulative reward ($R_s \leftarrow R_s + r_s$), and the subtree best ($y_s^{\max} \leftarrow \max(y_s^{\max}, y_{new})$) to guide future iterations.

Surrogate model for each algorithm. We maintain a separate Gaussian Process (GP) [49] surrogate model f_i for each algorithm A^i. Importantly, f_i is trained on the local dataset $\mathcal{D}_i = \{(\boldsymbol{\lambda}_{s_{\text{hp}}^k}, y_{s_{\text{hp}}^k})\}_{k=1}^{N_i}$, comprising all N_i configurations and their performance collected within the subtree rooted at s^i_{algo}. This per-algorithm modeling exploits disjoint hyperparameter spaces to reduce dimensionality, enhancing precision compared to a single high-dimensional global surrogate [22]. We define the GP covariance function $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ as a mixed kernel, with a Matérn-5/2 kernel (k_ν) on continuous dimensions and a Hamming kernel (k_{cat}) on categorical dimensions:

$$k(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = k_\nu(\boldsymbol{\lambda}_{cont}, \boldsymbol{\lambda}'_{cont}) \times k_{cat}(\boldsymbol{\lambda}_{cat}, \boldsymbol{\lambda}'_{cat}), \tag{3}$$

3.2 Algorithm Selection

In each MCTS iteration, the Selection phase starts at s_{root} and chooses one of its K child Algo node, which directly corresponds to selecting an algorithm $A^i \in \mathcal{A}$ for CASH. Concretely, we utilize PUCT in Equation (2) and the estimated mean value of each Algo node s^i_{algo} is

$$Q(s^i_{\text{algo}}) = \tilde{y}^{\max}_{s^i_{\text{algo}}} \left(1 + R_{s^i_{\text{algo}}} / N_{s^i_{\text{algo}}}\right), \tag{4}$$

where $\tilde{y}^{\max}_{s^i_{\text{algo}}}$ is the normalized best validation performance in its subtree (min–max normalized using the global range of observed metric scores), and $R_{s^i_{\text{algo}}} / N_{s^i_{\text{algo}}}$ is the average local relative-improvement rate. This multiplicative form treats the mean value Q as an estimated performance potential, favoring algorithms with both high incumbent performance and continuous improvement tendency.

Algorithm Prior. We compute a PUCT prior score for each algorithm based on surrogate predictions. Specifically, for each algorithm A^i, we uniformly sample n_s configurations from Λ^i, predict their performance using the corresponding surrogate f_i, and compute the normalized mean predicted score $\bar{v}_i$. The prior distribution is then given by a softmax over these scores: $P(s^i_{\text{algo}}) = \frac{\exp(\bar{v}_i)}{\sum_{k=1}^{K} \exp(\bar{v}_k)}$.

3.3 Hyperparameter Optimization

Once the root selects an algorithm A^i, the task reduces to an algorithm-specific HPO subproblem: finding $\boldsymbol{\lambda} \in \Lambda^i$ with best performance. During subsequent phases, LB-MCTS integrates two complementary configuration proposers: BO for quantitative search, and LLM for semantic proposals.

3.3.1 BO proposer

The BO proposer directly generates a new node by selecting the best candidate from a sampled pool.

Candidate Generation. We employ two sampling strategies to generate a pool of candidate configurations $\mathcal{P}$. (i) Random sampling draws candidates randomly from the search space Λ^i. (ii) Local sampling treats every HP node s_{hp} in the subtree of s^i_{algo} as a basic configuration and samples candidates from its neighborhood (i.e., via small perturbations of configuration in s_{hp}).

Selection & Expansion. We then rank all candidates in $\mathcal{P}$ using the Expected Improvement (EI) [17] acquisition function based on the surrogate f_i, and select the candidate $\boldsymbol{\lambda}_{new}$ with the highest score. The expansion type depends on the source of $\boldsymbol{\lambda}_{new}$: If it originates from random sampling, it is treated as an initialization action (e.g., $s^1_{\text{algo}} \xrightarrow[\text{BO RANDOM SEARCH}]{a_{\text{init}}} s^2_{\text{hp}}$), expanding directly from the Algo node s^i_{algo}; If it originates from local sampling, it is treated as an optimization action (e.g., $s^2_{\text{hp}} \xrightarrow[\text{BO LOCAL SEARCH}]{a_{\text{opt}}} s^5_{\text{hp}}$), expanding as a child of the basic configuration node.

3.3.2 LLM proposer

Following the algorithm selection, the LLM proposer continues the Selection phase using PUCT until a leaf HP node is reached. It then expands by querying the LLM to generate a new configuration.

Selection. The LLM proposer extends the Selection by recursively applying PUCT (Equation (2) and Equation (4)) within the algorithm subtree. The traversal proceeds until it reaches a node that LLM has not fully expanded, e.g., s^3_{hp} in Figure 1. We assign a uniform prior $P(s_{\text{hp}})$ to all child nodes here.

Expansion via ICL. Given a selected leaf node s_{base} with a basic configuration $\boldsymbol{\lambda}_{\text{base}}$, the LLM proposer executes Expansion by querying the LLM to propose a new configuration through adjusting $\boldsymbol{\lambda}_{\text{base}}$. We construct a structured prompt comprising four key components: (i) the *task description* ψ (dataset description and optimization metric), (ii) the *Selective Tuning Memory* $\mathcal{M}_i(s_{\text{base}})$, containing informative historical trials from algorithm A^i, (iii) the basic configuration $\boldsymbol{\lambda}_{\text{base}}$, and (iv) an optimization directive $d \in \{\text{WARMUP}, \text{EXPLORATION}, \text{EXPLOITATION}\}$. Note that (ii) and (iii) are empty for an initialization expansion when s is an Algo node. Formally, the LLM generates a reasoning chain $\mathcal{R}$ followed by a new configuration $\boldsymbol{\lambda}_{\text{new}}$:

$$(\mathcal{R}, \boldsymbol{\lambda}_{\text{new}}) = \text{LLM}\big(\psi, \mathcal{M}_i(s_{\text{base}}), \boldsymbol{\lambda}_{\text{base}}, d\big). \tag{5}$$

The output $\boldsymbol{\lambda}_{\text{new}}$ is added as a new child node $s^{\text{new}}_{\text{hp}}$ of s_{base}. Crucially, the directive d explicitly governs the search mode: WARMUP provides a high-quality initial point when expanding an Algo node (e.g., $s^1_{\text{algo}} \xrightarrow[\text{LLM WARMUP}]{a_{\text{init}}} s^1_{\text{hp}}$). EXPLOITATION instructs the LLM to locally refine $\boldsymbol{\lambda}_{\text{base}}$ for immediate gains (e.g., $s^2_{\text{hp}} \xrightarrow[\text{LLM EXPLOITATION}]{a_{\text{opt}}} s^4_{\text{hp}}$); and EXPLORATION encourages bold adjustments to probe unknown regions (e.g., $s^2_{\text{hp}} \xrightarrow[\text{LLM EXPLORATION}]{a_{\text{opt}}} s^3_{\text{hp}}$). An Algo node is regarded as a leaf before it has generated $n_{\text{w}} = 3$ initial configurations via WARMUP. An HP node is considered fully expanded by the LLM only after both EXPLORATION and EXPLOITATION have been executed once; otherwise, it remains a leaf node. Detailed prompt template and example are shown in Appendix B.1 and B.4.

Reflection after playout. Upon generation, the new configuration $\boldsymbol{\lambda}_{\text{new}}$ is evaluated on the training/validation split to obtain the performance metric y_{new}. To bridge the gap between numerical rewards and the LLM's semantic reasoning, we subsequently incorporate a reflection mechanism inspired by Reflexion [40]. Excluding the WARMUP expansion of Algo nodes, we prompt LLM to analyze the evaluation result by comparing it against $\boldsymbol{\lambda}_{\text{base}}$ and configurations in memory $\mathcal{M}_i(s_{\text{base}})$:

$$\rho_{\text{new}} = \text{LLM}\big(\psi, \mathcal{M}_i(s_{\text{base}}), \boldsymbol{\lambda}_{\text{base}}, y_{\text{base}}, \boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}\big). \tag{6}$$

ρ_{new} summarizes what worked or failed and provides instructions for future expansion, which is stored in the new HP node $s^{\text{new}}_{\text{hp}}$. Prompt template and example are detailed in Appendix B.2 and B.4.

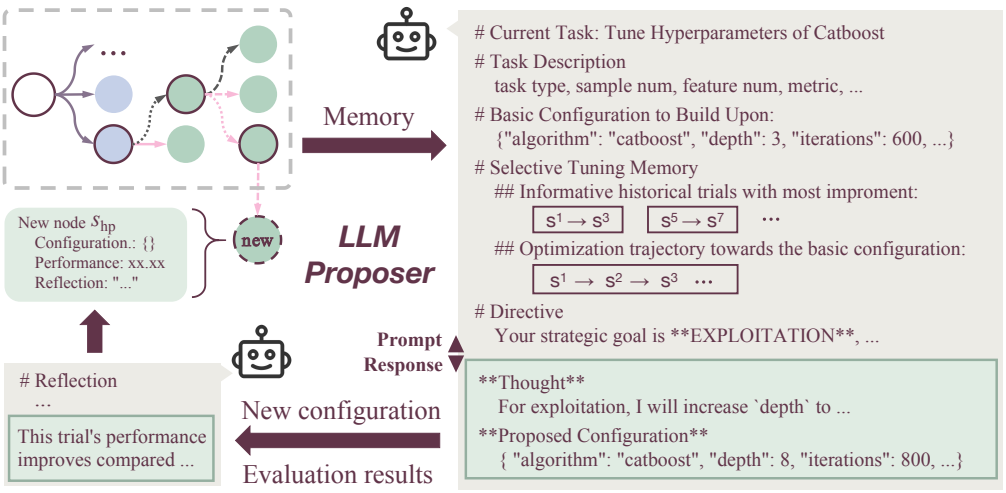


Figure 2: MCTS search of the LLM proposer.

Selective Tuning Memory. Efficiently utilizing optimization history is crucial for ICL. Therefore, we propose the Selective Tuning Memory (STM) mechanism. Given their disjoint search spaces, histories from different algorithms provide no mutual benefit. Thus, we first isolate the optimization history so that the memory $\mathcal{M}_i(s_{\text{base}})$ only contains trials from the same algorithm subtree as s_{base}. After that, we view each transition in the subtree (from a parent node s_p to a child s_c) as an "optimization attempt" $e = (\boldsymbol{\lambda}_p, y_p, \boldsymbol{\lambda}_c, y_c, \rho_c)$, capturing the modification from the parent configuration $\boldsymbol{\lambda}_p$ to the child $\boldsymbol{\lambda}_c$, their performances, and reflection ρ_c. For nodes expanded through BO and WARMUP lacking LLM reflection, reflections are programmatically generated via fixed rules (Appendix B.3).

Given an HP node with configuration $\boldsymbol{\lambda}_{\text{base}}$ to expand, our goal is to retrieve historical attempts that are both relevant (similar starting point) and instructive (high resulting performance y_c). The similarity between a historical attempt's starting configuration $\boldsymbol{\lambda}_p$ and $\boldsymbol{\lambda}_{\text{base}}$ is measured through the algorithm's GP covariance kernel $k(\boldsymbol{\lambda}_p, \boldsymbol{\lambda}_{\text{base}})$ in Equation (3), which jointly captures continuous-categorical similarity. For initialization attempts with an Algo node parent, we assign a similarity of $-\infty$. To balance relevance and performance, we compute the Pareto frontier over all historical attempts in the algorithm's subtree based on $(k(\cdot, \boldsymbol{\lambda}_{\text{base}}), y_c)$, selecting the non-dominated attempts as **Global Memory** $\mathcal{M}_{\text{global}}$. Complementing this, we include **Local Memory** $\mathcal{M}_{\text{local}}$, comprising the sequence of attempts along the ancestral path from the algorithm root to s_{base}. This preserves the optimization trajectory context. The final memory is the union $\mathcal{M}_i(s_{\text{base}}) = \mathcal{M}_{\text{global}} \cup \mathcal{M}_{\text{local}}$, providing both high-quality, relevant references and coherent trajectory.

LLM proposer workflow. Overall, the LLM proposer follows PUCT-based base-node selection, ICL-based expansion, evaluation, reflection, and backpropagation, as illustrated in Figure 2.

3.3.3 Dynamic Proposer Selection

To synergize the complementary strengths of semantic reasoning and quantitative modeling, LB-MCTS dynamically selects either the LLM or BO proposer at each HPO step. Our insight is that BO becomes increasingly reliable as data accumulates and the surrogate model matures, whereas LLMs may excel in the early stages with prior knowledge. Thus, for each algorithm A^i, we maintain a BO-selection probability p^i_{BO} based on the generalization capability of its surrogate f_i. Specifically, given the algorithm-specific dataset, we perform k-fold cross-validation, training f_i on training folds and computing the Kendall's rank correlation coefficient (τ^i) between predicted and true rankings on validation folds to quantify how well f_i can capture the relative order of configurations. This correlation is rescaled to $[0, 1]$ as a probability: $p^i_{\text{BO}} = \max(\epsilon, \frac{\tau^i+1}{2})$, where a small constant ϵ (e.g., 0.05) ensures the BO proposer remains active. We initialize the BO probability p^i_{BO} to ϵ and update it every k observations (e.g., $k = 5$). In each iteration, the BO proposer is selected with probability p^i_{BO} and the LLM proposer with probability $1 - p^i_{\text{BO}}$. This mechanism ensures a smooth transition from early LLM-driven search to BO-driven search as the surrogate model becomes accurate.

3.4 Algorithm Summary

The procedure of LB-MCTS is summarized in Appendix A.1. We run MCTS iteratively: 1) **Algorithm Selection**: The root selects an Algo node with PUCT. 2) **Proposer Selection**: For the chosen algorithm, we estimate the GP surrogate's quality to select the BO or LLM proposer. 3) **Proposal**: The BO proposer samples candidates and selects with EI; the LLM proposer extends PUCT to select a base node and prompts LLM to generate a new configuration. 4) **Playout & Reflection**: The configuration is evaluated, followed by a reflection, to add a new HP node. 5) **Backpropagation**: The result is propagated to update ancestral nodes' statistics. After the budget is exhausted, LB-MCTS returns the configuration with the best validation performance.

Difference with Previous Methods. Unlike prior LLM optimizers for flat black-box problems, LB-MCTS targets structured CASH. Unlike existing LLM–BO hybrids that pass flat histories to LLMs, LB-MCTS unifies BO- and LLM-produced trajectories in a shared tree state, enabling richer path-aware reasoning. Unlike prior LLM–MCTS methods, LB-MCTS uses the MCTS tree as a shared CASH state, integrating algorithm selection and LLM–BO synergistic HPO into unified trajectories.

Theorem 3.1 (**Global Convergence**). *If the BO-selection probability is lower bounded by $\epsilon > 0$, LB-MCTS converges to the global optimum almost surely as the number of iterations $T \to \infty$:*

$$\lim_{T\to\infty} \left(\mathcal{F}(\boldsymbol{\lambda}^*) - \max_{t=1}^{T} \mathcal{F}(\boldsymbol{\lambda}_t) \right) = 0, \tag{7}$$

Appendix A.2 provides the detailed proof, including the finite-time regret bound in Theorem A.5 and the conditional evaluation-budget comparison in Corollary A.6, showing that LB-MCTS inherits BO's convergence guarantee while leveraging LLMs for acceleration.

4 Experiment

4.1 Experiment Setup

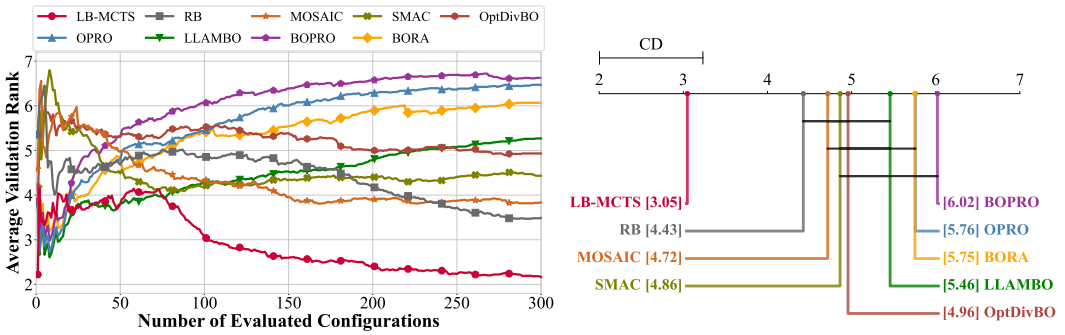
Baselines. We compare LB-MCTS with 8 baselines: — *Four BO-based CASH methods*: 1) **SMAC** [14]: standard BO; 2) **OptDivBO** [33]: a BO variant encouraging diversity; 3) Rising Bandit (**RB**) [22]: a multi-armed-bandit extension of BO for CASH; 4) **MOSAIC**: a BO variant with PUCT for algorithm selection. — *Two pure LLM Optimizer*: 5) **OPRO**, 6) **LLAMBO**, — *Two hybrid LLM-BO Methods*: 7) **BOPRO**, which optimizes LLM's prompt instructions with BO; and 8) **BORA**, which switches between LLM and BO via a variance heuristic.

CASH space, datasets, and metrics. We construct a hierarchical CASH search space comprising 8 machine learning algorithms (e.g., XGBoost) for both classification and regression tasks, resulting in high-dimensional search spaces with 45 and 46 hyperparameters, respectively (see Appendix C.1). For datasets, we utilize the AutoML Benchmark (AMLB) [10], which contains 71 classification tasks and 33 regression tasks across varying scales and domains. Following [8], we report Balanced Accuracy for classification and Mean Squared Error (MSE) for regression as performance metrics.

Basic settings. Each dataset is split into training, validation, and test sets with a 60/20/20 ratio. We report the best-observed validation metric during optimization and the final test metric. While it takes a different amount of time to evaluate the same configuration on different datasets, we use the evaluation iterations as the unit of budget, where each baseline evaluates 300 configurations. We implement the BO surrogate, the random and local sampling mechanism using OpenBox 0.8.1 [16], an open-source toolkit. For the algorithm prior, the number of sampled configurations n_s is set to 100. The exploration constant in PUCT is set to $c_{puct} = \sqrt{2}$. We employ `gpt-4o-mini-2024-07-18` [32], OpenAI's cost-efficient model, as the backbone for all LLM-based optimizers (ablation in Appendix C.2). More implementation details for other baselines are provided in Appendix C.3.

4.2 Comparison with Baselines

This section evaluates LB-MCTS on 104 AMLB datasets. Figure 3a displays the average rank of the *best-achieved validation performance* at each iteration, from which we draw four observations: (i) Current LLM-based baselines (BORA, LLAMBO, BOPRO, and OPRO) perform poorly on CASH. They are competitive only in the first $\sim$50 iterations; afterwards their average ranks steadily deteriorate, ending between 5.27 and 6.63—worse than other methods. This is because they treat



(a) Average validation rank during optimization. (b) CD plot of test rank with Nemenyi post-hoc test.

Figure 3: Average Performance of 9 methods across 104 datasets.

CASH as a flat joint space, mix histories across algorithms, and lack explicit exploration-exploitation control, getting stuck in local optima in the structured, high-dimensional search space. (ii) Among LLM-based optimizers, OPRO and BOPRO perform worst, as they rely on simplistic iterative prompting where BO (in BOPRO's case) is limited to superficial example selection. BORA (rank 6.07) gains slightly via LLM–BO switching, but its behavior is governed by fixed uncertainty thresholds that do not transfer well to CASH. LLAMBO attains the highest rank (5.27) but incurs prohibitive computational overhead. (iii) Among BO-based baselines, OptDivBO performs worst (rank 4.93) as it prioritizes ensemble diversity over individual model performance. MOSAIC improves on SMAC by decoupling algorithm selection from HPO. RB leads BO baselines by using local surrogates after algorithm selection instead of a global surrogate over the joint space. (iv) Among all methods, LB-MCTS significantly dominates the others. It consistently outperforms all baselines beyond around 70 iterations. While the second-best baseline (RB) ranks 3.49 finally, **the rank of LB-MCTS is 2.16**. Appendix C.4 further analyzes these results by task characteristics, showing that LB-MCTS remains robust across diverse tasks. To further assess generality beyond conventional ML algorithms, Appendix C.5 reports supplementary experiments on a broader search space including deep learning models and feature engineering, where LB-MCTS continues to achieve the best average rank.

Test set performance. We further *evaluate the validation-best configurations on the test set* and illustrate the results in the Critical Difference (CD) diagram (Figure 3b). We observe that the ranks of methods on the test set may deviate from their validation ranks. The reason is that the distributions of the validation and test set are not exactly the same [15]. But overall, LB-MCTS remains statistically significantly superior to all baselines, attaining a **leading average rank of 3.05**. To further assess the practical utility, we evaluate the post-hoc ensemble performance of all methods in Appendix C.6.

The success of LB-MCTS stems from two main factors. First, LB-MCTS remedies key limitations of prior LLM-based optimizers: (i) The STM mechanism isolates algorithm-specific histories and combines global and local memory to enhance reasoning (validated in Appendix C.12). (ii) As shown in Section 4.3, MCTS enables a principled exploration–exploitation trade-off both across algorithms and within each algorithm's subtree. (iii) The reflection mechanism further strengthens STM by turning numeric feedback into reusable verbal guidance (Appendix C.11). Second, LB-MCTS achieves optimal synergy between LLM and BO proposers, outperforming either component alone (Section 4.4). Appendix C.7 provides analyses of both API cost and wall-clock overhead, showing that LB-MCTS achieves these gains with an economical cost of about $0.127 per task.

4.3 MCTS for Exploration-Exploitation Trade-off

In this section, we validate the ability of LB-MCTS to trade off exploration and exploitation at two levels: (1) Can it allocate optimization resources to superior algorithms? (2) Within each algorithm's subspace, can it effectively navigate the trade-off between promising and unexplored regions?

Resource allocation across algorithms. For each dataset, we first identified the optimal algorithm as the algorithm associated with the global best configuration found across all 9 methods and 300 iterations. Subsequently, for each CASH method, we compute the *ratio of its evaluated configurations that belong to the "optimal algorithm"*. Figure 4 visualizes the distribution of these allocation ratios

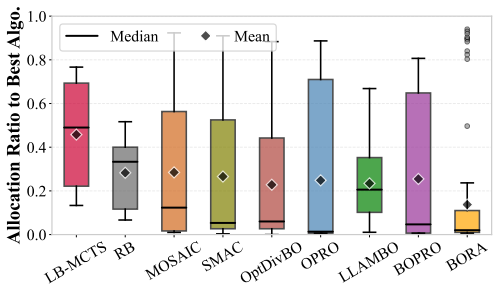


Figure 4: Resource allocation.

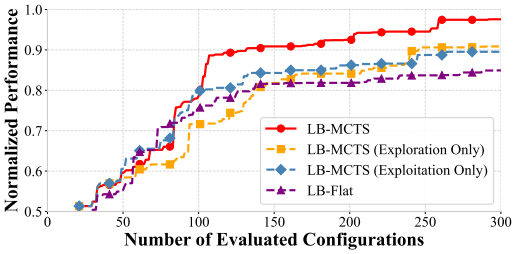


Figure 5: Impact of the tree-structured search.

for each method across the 104 datasets, from which we get two observations: (i) Superiority: LB-MCTS achieves the highest mean allocation ratios (0.43), whereas other baselines allocate 0.14 to 0.28 on average. This demonstrates its superior ability to correctly identify the most promising algorithm and concentrate optimization budget on it. (ii) Robustness: LB-MCTS exhibits the highest median (0.49), along with highest lower quartile (box bottom) and lower whisker, ensuring the optimal algorithm is rarely overlooked. In contrast, aggressive baselines (e.g., SMAC, OPRO, MOSAIC) exhibit near-zero medians and bottom whiskers, indicating frequent premature convergence to suboptimal algorithms; methods like RB and LLAMBO yield consistently low ratios due to conservative uniformity. In summary, LB-MCTS employs root-level PUCT to balance inter-algorithm trade-off, ensuring stable resource allocation to optimal algorithms across datasets.

Navigating within each algorithm's subspace. Effective HPO requires balancing exploiting promising regions and exploring unknown areas to avoid local optima. To achieve this, LB-MCTS explicitly prompts the LLM proposer to *expand the tree structure using two directives*: EXPLOITATION for local refinement and EXPLORATION for global search. We compare this design against two single directive variants—EXPLORATION only and EXPLOITATION only—expanded twice per node, and LB-Flat, which retains only the root and Algo Nodes, substituting subsequent tree search with unstructured history sharing between BO and LLM (an OPRO-style proposer). To conserve API costs, all ablation studies are conducted on 10 representative datasets from AMLB (details in Appendix C.8). Figure 5 plots the average *best-so-far validation performance* across the 10 datasets, where scores on each dataset are min-max normalized globally. The results show that relying solely on EXPLOITATION yields rapid initial gains but leads to premature stagnation, trapped in local optima. Conversely, using only EXPLORATION results in slower convergence due to insufficient refinement of promising regions. LB-MCTS outperforms both variants and LB-Flat, finally achieving **an average rank of 1.5**, compared to 2.3 and 2.4 for the two variants. Most importantly, the poor performance of LB-Flat (rank 3.4) underscores that unstructured history sharing—a common trait in existing methods—is insufficient for complex CASH problems without the hierarchical guidance of MCTS. Details of LB-Flat and further analysis are provided in Appendix C.9. In summary, the synergy of dual directives and tree-structured search enables LB-MCTS to effectively navigate the search space to avoid the local optima. Additional ablations in Appendix C.10 validate the choice of PUCT over alternative tree-search strategies and show that $c_{puct} = \sqrt{2}$ provides robust exploration.

4.4 Analysis of BO–LLM Collaboration

LB-MCTS maintains a BO-selection probability (P_{BO}) for each algorithm and samples the proposer at every HPO step. Figure 6 plots the average P_{BO} for each algorithm versus the number of evaluations over 104 datasets. Initially ($N < 25$), P_{BO} is low (< 0.3), reflecting reliance on semantic priors when data are scarce. As the number of evaluations grows, the BO surrogate becomes more reliable: around ~ 50 evaluations, P_{BO} rises to ≈ 0.5, marking a transition. After ~ 100 evaluations, it gradually stabilizes between 0.6 and 0.8, with modest variation across algorithms (likely reflects differences in landscape smoothness). On average, the LLM proposer is selected in 107 out of 300 iterations.

To quantify the benefit of this dynamic synergy, we perform an ablation study on 10 datasets against three variants: L-MCTS (pure LLM), B-MCTS (pure BO), and LB-MCTS(0.5) (fixed 50:50 probability). We also include LLAMBO, the strongest LLM-based baseline. Figure 7 tracks the average rank of the best validation performance. (i) L-MCTS dominates early (< 70 iterations), while B-MCTS prevails later (> 110 iterations), confirming LLMs excel at warm-starting while

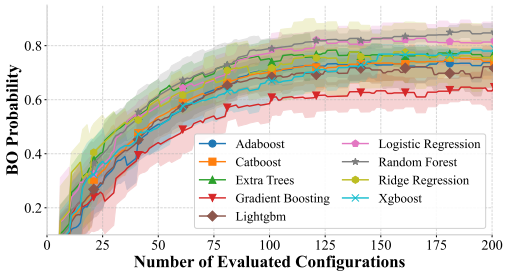


Figure 6: Average P_{BO} for each algorithm.

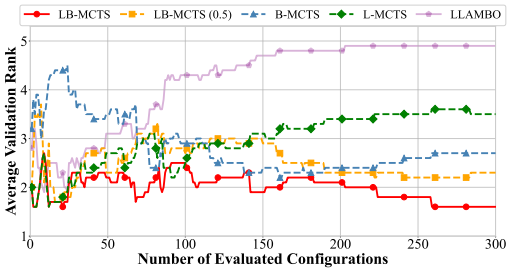


Figure 7: Average rank during optimization.

BO excels with sufficient data. (ii) Benefit of synergy: both LB-MCTS (0.5) and LB-MCTS outperform single-proposer variants. (iii) LB-MCTS achieves the best performance over the entire optimization, achieving a final **rank of 1.6** versus 2.3 for LB-MCTS (0.5), confirming the value of dynamic synergy. Notably, LB-MCTS outperforms L-MCTS even in the early phase, suggesting that occasional injections from the non-dominant proposer provide necessary diversity for performance gains. (iv) Finally, L-MCTS (rank 3.5) significantly outperforms LLAMBO (rank 4.9), reaffirming that our trajectory-aware MCTS enables more effective LLM reasoning than unstructured prompting.

5 Limitation and Future Work

Our method inherits the general challenge that validation feedback can be noisy in some scenarios, such as when the validation set is small or the data itself is highly noisy. In these cases, the resulting feedback signal may be less reliable and can occasionally mislead MCTS and the LLM proposer. While BO has built-in mechanisms to better tolerate noisy evaluations, the current LLM proposer does not explicitly model such noise. A promising direction for future work is therefore to develop noise-aware LLM tuning strategies, such as prompt designs or feedback processing schemes that are more robust to uncertain validation signals.

In addition, the effectiveness of our framework still depends to some extent on the capability of the underlying LLM. Although our experiments show that several relatively economical models, such as GPT-4o-mini and DeepSeek-V3, already perform well in our setting, weaker models may nevertheless provide less informative proposals and thus limit the overall gain. Future improvements could come from lightweight domain adaptation, fine-tuning, or the incorporation of external tools to further strengthen the tuning ability of smaller or less capable models.

From a broader-impact perspective, LB-MCTS may lower the expertise and cost barriers for applying AutoML by improving the efficiency of CASH optimization. However, as with other AutoML systems, the selected models may still inherit biases, privacy risks, or reliability issues from the underlying data and downstream application context. Therefore, when applying LB-MCTS to sensitive domains, users should conduct appropriate data governance, fairness checks, and domain-specific validation.

6 Conclusion

In this paper, we propose LB-MCTS, which synergizes LLMs and BO via tree search for the CASH problem. By organizing algorithm selection and hyperparameter refinement in a shared tree, LB-MCTS unifies the optimization trajectories produced by both proposers and enables them to build on accumulated search experience. The framework combines hierarchical exploration and exploitation, Selective Tuning Memory for algorithm-specific context retrieval, and a reliability-aware policy for adaptive proposer selection. Experiments on 104 AMLB datasets demonstrate that LB-MCTS outperforms BO-based, LLM-based, and hybrid baselines. Ablation studies further support the contributions of the search structure, memory mechanism, and BO–LLM collaboration.

Acknowledgments and Disclosure of Funding

This work is supported by the National Natural Science Foundation of China (U23B2048, U22B2037), the Beijing Municipal Natural Science Foundation (Grant No. L2603008), and the "Fundamental Research Funds for the Central Universities, Peking University". Bin Cui is the corresponding author.

References

[1] Dhruv Agarwal, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. Searching for optimal solutions with llms via bayesian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[3] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[4] Abdoulatif Cissé, Xenophon Evangelopoulos, Vladimir V Gusev, and Andrew I Cooper. Language-based bayesian optimization research assistant (bora). *arXiv preprint arXiv:2501.16224*, 2025.

[5] Gustavo Sutter Pessurno de Carvalho, Mohammed Abdulrahman, Hao Wang, Sriram Ganapathi Subramanian, Marc St-Aubin, Sharon O'Sullivan, Lawrence Wan, Luis Ricardez-Sandoval, Pascal Poupart, and Agustinus Kristiadi. Simplifying bayesian optimization via in-context direct optimum sampling. *arXiv preprint arXiv:2505.23913*, 2025.

[6] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni DE PAULA LOURENCO, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Alphad3m: Machine learning pipeline synthesis. In *ICML AutoML Workshop*, 2021.

[7] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018.

[8] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022.

[9] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[10] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

[13] Beichen Huang, Xingyu Wu, Yu Zhou, Jibin Wu, Liang Feng, Ran Cheng, and Kay Chen Tan. Exploring the true potential: Evaluating the black-box optimization capability of large language models. *arXiv preprint arXiv:2404.06290*, 2024.

[14] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[15] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[16] Huaijun Jiang, Yu Shen, Yang Li, Beicheng Xu, Sixian Du, Wentao Zhang, Ce Zhang, and Bin Cui. Openbox: A python toolkit for generalized black-box optimization. *Journal of Machine Learning Research*, 25(120):1–11, 2024.

[17] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

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

[19] Nicholas Kroeger, Dan Ley, Satyapriya Krishna, Chirag Agarwal, and Himabindu Lakkaraju. In-context explainers: Harnessing llms for explaining black box models. *arXiv preprint arXiv:2310.05797*, 2023.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[21] Erin LeDell and Sebastien Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, page 24, 2020.

[22] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. Efficient automatic cash via rising bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4763–4771, 2020.

[23] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. An admm based framework for automl pipeline configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4892–4899, 2020.

[24] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2025.

[25] Wei Liu, Yang Gu, Xi Yan, Zihan Nan, Beicheng Xu, Keyao Ding, Bin Cui, and Wentao Zhang. Profilitable: Profiling-driven tabular data processing via agentic workflows. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, page 3306–3317, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3817982.

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

[27] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

[28] Kanan Mahammadli and Seyda Ertekin. Sequential large language model-based hyper-parameter optimization. *arXiv preprint arXiv:2410.20302*, 2024.

[29] Nicolas Menet, Aleksandar Terzić, Michael Hersche, Andreas Krause, and Abbas Rahimi. Thompson sampling via fine-tuning of llms. *arXiv preprint arXiv:2510.13328*, 2025.

[30] Tung Nguyen and Aditya Grover. Lico: Large language models for in-context molecular optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[31] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

[32] OpenAI. Gpt-4o-mini model card., 2024. URL `https://platform.openai.com/docs/models/gpt-4o-mini`.

[33] Pranav Poduval, Sanjay Kumar Patnala, Gaurav Oberoi, Nitish Srivasatava, and Siddhartha Asthana. Cash via optimal diversity for ensemble learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2411–2419, 2024.

[34] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. Automated machine learning with monte-carlo tree search. In *IJCAI-19-28th International Joint Conference on Artificial Intelligence*, pages 3296–3303. International Joint Conferences on Artificial Intelligence Organization, 2019.

[35] Bojana Ranković and Philippe Schwaller. Bochemian: Large language model embeddings for bayesian optimization of chemical reactions. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023.

[36] Yili Ren, Beicheng Xu, Huaijun Jiang, Xi Liu, Ping Zhang, Qian Liang, Yu Shen, and Bin Cui. Hyperrps: An efficient multi-fidelity hyper-parameter tuning framework for rock particle segmentation. *Tsinghua Science and Technology*, 2026. doi: 10.26599/TST.2025.9010074.

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

[38] Yu Shen, Yupeng Lu, Yang Li, Yaofeng Tu, Wentao Zhang, and Bin Cui. Divbo: diversity-aware cash for ensemble learning. *Advances in Neural Information Processing Systems*, 35:2958–2971, 2022.

[39] Z Shi, M Fang, and L Chen. Monte carlo planning with large language model for text-based game agents. In *13th International Conference on Learning Representations Iclr 2025*, 2025.

[40] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[41] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[42] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, volume 25, pages 2951–2959, 2012.

[43] Yuan-Feng Song, Yuan-Qin He, Xue-Fang Zhao, Han-Lin Gu, Di Jiang, Hai-Jun Yang, and Li-Xin Fan. A communication theory perspective on prompting engineering methods for large language models. *Journal of Computer Science and Technology*, 39(4):984–1004, 2024.

[44] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[45] Yu-Zhu Sun, He-Li Sun, Jian-Cong Ma, Peng Zhang, and Xiao-Yong Huang. Multimodal agent ai: A survey of recent advances and future directions. *Journal of Computer Science and Technology*, 40(4):1046–1063, 2025.

[46] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[47] Anton Vakhrushev, Alexander Ryzhkov, Maxim Savchenko, Dmitry Simakov, Rinchin Damdinov, and Alexander Tuzhilin. Lightautoml: Automl solution for a large financial services ecosystem. *arXiv preprint arXiv:2109.01528*, 2021.

[48] Niki van Stein, Diederick Vermetten, and Thomas Bäck. In-the-loop hyper-parameter optimization for llm-based automated design of heuristics. *ACM Transactions on Evolutionary Learning*, 2024.

[49] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[50] Beicheng Xu, Keyao Ding, Wei Liu, Yupeng Lu, and Bin Cui. Cofeh: Llm-driven feature engineering empowered by collaborative bayesian hyperparameter optimization. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, page 5766–5777, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3817664.

[51] Beicheng Xu, Wei Liu, Keyao Ding, Yupeng Lu, and Bin Cui. PSEO: Optimizing post-hoc stacking ensemble through hyperparameter tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(32):27188–27196, 2026. doi: 10.1609/aaai.v40i32.39934.

[52] Beicheng Xu, Lingching Tung, Yuchen Wang, Yupeng Lu, and Bin Cui. Mftune: An efficient multi-fidelity framework for spark sql configuration tuning. *Proc. VLDB Endow.*, 19(11): 3649–3662, September 2026. ISSN 2150-8097. doi: 10.14778/3836663.3836715.

[53] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

[54] Yuxuan Yin, Yu Wang, Boxun Xu, and Peng Li. Ado-llm: Analog design bayesian optimization with in-context learning of large language models. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.

[55] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in neural information processing systems*, 36:31967–31987, 2023.

[56] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. Db-gpt: Large language model meets database: X. zhou et al. *Data Science and Engineering*, 9(1):102–111, 2024.

[57] Tommaso Zoppi, Andrea Ceccarelli, and Andrea Bondavalli. A strategy for predicting the performance of supervised and unsupervised tabular data classifiers. *Data Science and Engineering*, 10(1):75–97, 2025.

A Method Details

A.1 Pseudocode of LB-MCTS

Algorithm 1 LB-MCTS: LB-MCTS for CASH

Input: dataset $\mathcal{D}$, algorithms $\mathcal{A} = \{A^1, \ldots, A^K\}$ with spaces $\{\Lambda^i\}$, evaluation budget T
Initialize root node s_{root} with Algo children $\{s^i_{\text{algo}}\}_{i=1}^K$
Initialize per-algorithm data $\mathcal{D}_i \leftarrow \emptyset$, surrogate f_i (GP), episode set $\mathcal{E}_i \leftarrow \emptyset$, $p^i_{\text{BO}} \leftarrow \epsilon$
Initialize node stats $N_s \leftarrow 0$, $R_s \leftarrow 0$, $y^{\max}_s \leftarrow -\infty$ for all s
$t \leftarrow 0$
while $t < T$ **do**
  // Algorithm selection (root PUCT)
  Compute $Q(s^i_{\text{algo}})$ and prior $P(s^i_{\text{algo}})$ for all i (Sec. 3.2)
  Select algorithm node s^i_{algo} by PUCT (Eq. 2); let A^i be the chosen algorithm
  // Dynamic proposer selection
  if $|\mathcal{D}_i| > 0$ **and** $|\mathcal{D}_i| \pmod 5 = 0$ **then**
    Estimate surrogate quality τ^i via k-fold CV on $\mathcal{D}_i$
    $p^i_{\text{BO}} \leftarrow \max(\epsilon, \frac{\tau^i+1}{2})$
  end if
  Sample $z \sim \text{Bernoulli}(p^i_{\text{BO}})$
  // HPO with LLM/BO
  if $z = 1$ **then**
    {use BO proposer}
    $(\boldsymbol{\lambda}_{\text{new}}, s_{\text{parent}}) \leftarrow \text{BOPROPOSE}(A^i, s^i_{\text{algo}}, f_i, \mathcal{D}_i)$
  else
    {use LLM proposer}
    $(\boldsymbol{\lambda}_{\text{base}}, s_{\text{base}}) \leftarrow \text{PUCTSELECTHP}(s^i_{\text{algo}})$ {HP-level PUCT with uniform prior}
    Build STM $\mathcal{M}_i(s_{\text{base}})$ from global episodes $\mathcal{E}_i$ and ancestors of s_{base}
    Choose directive $d \in \{\text{WARMUP}, \text{EXPLORATION}, \text{EXPLOITATION}\}$ for s_{base}
    $(\mathcal{R}, \boldsymbol{\lambda}_{\text{new}}) \leftarrow \text{LLM}\big(\psi, \mathcal{M}_i(s_{\text{base}}), \boldsymbol{\lambda}_{\text{base}}, d\big)$
    $s_{\text{parent}} \leftarrow s_{\text{base}}$
  end if
  // Playout and reflection
  Evaluate A^i with $\boldsymbol{\lambda}_{\text{new}}$ on $\mathcal{D}$ to obtain y_{new}
  Let $(\boldsymbol{\lambda}_{\text{base}}, y_{\text{base}})$ be the config/performance of s_{parent} (if any)
  if $z = 1$ **then**
    {BO: heuristic reflection}
    $\rho_{\text{new}} \leftarrow \text{SYNTHESIZESUMMARY}(\Delta\boldsymbol{\lambda}, \Delta y)$
  else
    {LLM proposer: verbal reflection}
    $\rho_{\text{new}} \leftarrow \text{LLM}\big(\psi, \mathcal{M}_i(s_{\text{parent}}), \boldsymbol{\lambda}_{\text{base}}, y_{\text{base}}, \boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}\big)$
  end if
  Create new HP node s_{new} with $(\boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}, \rho_{\text{new}})$ as child of s_{parent}
  Add $(\boldsymbol{\lambda}_{\text{base}}, y_{\text{base}}, \boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}, \rho_{\text{new}})$ to episode set $\mathcal{E}_i$
  Update dataset $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(\boldsymbol{\lambda}_{\text{new}}, y_{\text{new}})\}$ and refit GP f_i (periodically)
  // Backpropagation
  for each node s on path from s_{root} to s_{new} **do**
    $r_s \leftarrow [y_{\text{new}} - y^{\max}_s]_+ / |y^{\max}_s|$
    $N_s \leftarrow N_s + 1$; $R_s \leftarrow R_s + r_s$; $y^{\max}_s \leftarrow \max(y^{\max}_s, y_{\text{new}})$
  end for
  $t \leftarrow t + 1$
end while
Output: best $(A^*, \boldsymbol{\lambda}^*)$ observed so far (maximizing validation performance)

Algorithm 1 summarizes the procedure of LB-MCTS. The detailed workflow is as follows.

Initialization (Lines 2–5). Given a dataset $\mathcal{D}$ and a set of candidate algorithms $\mathcal{A} = \{A^1, \ldots, A^K\}$ with corresponding hyperparameter spaces $\{\Lambda^i\}_{i=1}^K$, we construct the search tree with a root CASH node s_{root} and K Algo children $\{s^i_{\text{algo}}\}_{i=1}^K$ (Lines 2–3). For each algorithm A^i, we maintain: (i) a local dataset $\mathcal{D}_i$ of evaluated configurations, (ii) a GP surrogate f_i over Λ^i, (iii) an episode set $\mathcal{E}_i$ storing optimization attempts (parent/child configurations, performances, and reflections).

While the budget T is not exhausted, LB-MCTS execute an MCTS loop over the CASH tree:

Algorithm selection at the root (Lines 7–9); Section 3.2. At each iteration, LB-MCTS first decides which algorithm to optimize by applying the PUCT rule (Equation (2)) to select an Algo node s^i_{algo}.

Dynamic proposer selection (Lines 10–15); Section 3.3.3. Given the chosen algorithm A^i, we decide whether the next configuration should be proposed by BO or LLM. We estimate the generalization quality of the surrogate f_i on the current local dataset via cross-validation, compute the BO selection probability, and sample $z \sim \text{Bernoulli}(p^i_{\text{BO}})$. When the surrogate is still inaccurate (small τ^i), the LLM proposer is chosen more often; as more data accumulate and the surrogate improves, BO is selected with higher probability. Specifically, p^i_{BO} is initialized to ϵ and updated independently for each algorithm A^i **whenever five new configurations have been added to its local history**.

Proposing a new configuration. (Lines 16–27) The configuration proposal depends on whether $z = 1$ (BO) or 0 (LLM).

- **BO proposer** (Line 17–19); Section 3.3.1. When $z = 1$, we call BOPROPOSE with the current algorithm A^i, its Algo node s^i_{algo}, the surrogate f_i, and data $\mathcal{D}_i$. This routine: 1) constructs a candidate pool by mixing *random search* candidates drawn uniformly from Λ^i (initialization actions), and *local search* candidates sampled in the neighborhoods of previously evaluated configurations; 2) scores all candidates with an acquisition function (we use EI under the GP f_i); 3) selects the candidate $\boldsymbol{\lambda}_{\text{new}}$ with the highest EI and identifies its parent node s_{parent} in the tree (the Algo node for random samples or the corresponding HP node for local-search samples).
- **LLM proposer** (Lines 20–26); Section 3.3.2. When $z = 0$, we use the LLM proposer. 1) First, we extend the MCTS Selection phase into the algorithm subtree by applying HP-level PUCT (Line 22). This yields a base HP node s_{base} and its configuration $\boldsymbol{\lambda}_{\text{base}}$. 2) Next, we build the Selective Tuning Memory $\mathcal{M}_i(s_{\text{base}})$ (Line 23), which combines: a *global memory* of Pareto-selected episodes and a *local memory* consisting of all HP nodes along the ancestral path from s^i_{algo} to s_{base}. 3) Then we choose a directive $d \in \{\text{WARMUP}, \text{EXPLORATION}, \text{EXPLOITATION}\}$ according to the expansion state of s_{base} (Line 24). 4) We then query the LLM to obtain a reasoning chain $\mathcal{R}$ and a proposed configuration $\boldsymbol{\lambda}_{\text{new}}$ (Line 25). In this case, the parent node is $s_{\text{parent}} = s_{\text{base}}$ (Line 26).

Playout and reflection (Lines 28–37). As described in Section 3.3.2, the newly proposed configuration $\boldsymbol{\lambda}_{\text{new}}$ is evaluated on $\mathcal{D}$ to obtain the validation performance y_{new} (Line 29). We then attach a reflection ρ_{new} to this trial, with different generation mechanisms depending on the proposer:

- **BO proposer** (Lines 32-33). We call a SYNTHESIZESUMMARY to generate a heuristic textual reflection from the parameter change $\Delta\boldsymbol{\lambda}$ and performance change Δy (Line 33). This yields short descriptions such as "*Increasing depth from 4 to 6 improved accuracy by 1.2%*".
- **LLM proposer** (Lines 35-36). We call the LLM to produce a semantic reflection, which explains why the modification helped or hurt and suggests how to adjust future configurations.

We then create a new HP node s_{new} as a child of s_{parent}, storing $(\boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}, \rho_{\text{new}})$. The episode $(\boldsymbol{\lambda}_{\text{base}}, y_{\text{base}}, \boldsymbol{\lambda}_{\text{new}}, y_{\text{new}}, \rho_{\text{new}})$ is added to the episode set $\mathcal{E}_i$, and the local dataset $\mathcal{D}_i$ is augmented with the new pair $(\boldsymbol{\lambda}_{\text{new}}, y_{\text{new}})$; the surrogate f_i is periodically refit on $\mathcal{D}_i$ (Lines 38–40).

Backpropagation (Lines 41–46). Finally, we propagate the result back through the tree to update node statistics.

Output (Line 49) After the evaluation budget T is exhausted, LB-MCTS returns the best combination $(A^i, \boldsymbol{\lambda})$ observed so far in terms of validation performance, providing a CASH solution obtained through the coordinated action of MCTS-guided selection, BO-based quantitative search, and LLM-based qualitative reasoning.

A.2 Finite-Time Regret Analysis

We provide a finite-time version of Theorem 3.1. Let A^* denote an algorithm branch that contains a global maximizer, and let $N_*(T)$ be the number of visits to this branch within a total budget T. Among these visits, let $M_*(T)$ be the number of times the BO proposer is selected. The global simple regret is

$$S_T^{\mathrm{Global}} = \mathcal{F}(\boldsymbol{\lambda}^*) - \max_{1 \le t \le T} \mathcal{F}(\boldsymbol{\lambda}_t).$$

Assumption A.1 (Finite-time root allocation)**.** Let $*$ be the index of the optimal branch A^*. Let $N_i(t)$ be the number of visits to algorithm branch A^i after t root selections, and denote by $Q_t(A^i)$ and $P_i(t)$ its root value and prior used at round t. For fixed gaps $\Delta_i > 0$, $i \neq *$, define the number of gap violations by

$$Z_i(T) = \sum_{t=1}^{T} \mathbf{1}\{Q_t(A^*) - Q_t(A^i) < \Delta_i\}.$$

For each budget T and $\rho \in (0, 1)$, assume deterministic nonnegative budgets $B_i(T, \rho)$ such that

$$\Pr(Z_i(T) \le B_i(T, \rho),\ \forall i \neq *) \ge 1 - \rho.$$

This condition allows early errors and misrankings caused by limited or noisy validation feedback, instead of requiring a fixed gap at every round. Without a concentration result for the root-value estimator, we do not specify a closed-form dependence of $B_i(T, \rho)$ on the noise level.

Under the PUCT selection rule in Eq. (2), if a suboptimal branch A^i is selected at round t while its gap condition holds, then

$$Q_t(A^i) + c_{puct} P_i(t) \frac{\sqrt{t-1}}{1 + N_i(t-1)} \ge Q_t(A^*) + c_{puct} P_*(t) \frac{\sqrt{t-1}}{1 + N_*(t-1)} \ge Q_t(A^*).$$

Here the visit counts are taken before the current selection. Combining the gap condition with this inequality and using $P_i(t) \le 1$ gives

$$1 + N_i(t-1) \le \frac{c_{puct} P_i(t) \sqrt{t-1}}{\Delta_i} \le \frac{c_{puct} \sqrt{T}}{\Delta_i}.$$

Since the count increases by one at each selection, at most $c_{puct}\sqrt{T}/\Delta_i$ selections of A^i can satisfy this inequality. All remaining selections of A^i occur on gap-violation rounds and number at most $Z_i(T)$. Consequently, on the assumed event, $N_i(T) \le c_{puct}\sqrt{T}/\Delta_i + B_i(T, \rho)$, and

$$N_*(T) = T - \sum_{i \neq *} N_i(T) \ge T - \sum_{i \neq *} \left[\frac{c_{puct}\sqrt{T}}{\Delta_i} + B_i(T, \rho) \right].$$

Thus larger violation budgets weaken the allocation guarantee; $B_i(T, \rho) = 0$ recovers the original gap-based bound.

Controlled misranking. Let $B_\Sigma(T, \rho) = \sum_{i \neq *} B_i(T, \rho)$. For fixed K, gaps, and ρ, if $B_\Sigma(T, \rho) = \mathcal{O}(\sqrt{T})$, then $N_*(T) = T - \mathcal{O}(\sqrt{T})$ on the above event, retaining the near-full allocation order with probability at least $1 - \rho$ at each horizon. If $B_\Sigma(T, \rho) = o(\sqrt{T})$, violations contribute only a lower-order penalty.

Persistent misranking. If $B_\Sigma(T, \rho) = \Theta(T)$, the gap-based bound can become vacuous. Nevertheless, for fixed K and $c_{puct} > 0$, if root values are uniformly bounded and root priors have a uniform positive lower bound, PUCT still guarantees $N_*(T) = \Omega(\sqrt{T})$, independently of the gap-violation budget. To see this, suppose otherwise: along some sequence of horizons $T \to \infty$, $N_*(T) = o(\sqrt{T})$. Throughout the last half of each such horizon, the exploration bonus of A^* would then diverge uniformly, since its visit count never exceeds $N_*(T)$. With bounded root values, any selected branch A^j would eventually need an exploration bonus greater than one, implying $1 + N_j(t-1) < c_{puct}\sqrt{T}$ because $P_j(t) \le 1$. Each branch can be selected only $\mathcal{O}(\sqrt{T})$ times under this count restriction. For finite K, this permits only $\mathcal{O}(K\sqrt{T})$ selections, contradicting the $\Theta(T)$ selections in the last half of the horizon. Hence persistent misranking can weaken the guaranteed allocation from near-full to square-root order, while the optimal branch still receives an unbounded number of visits under these root conditions.

Assumption A.2 (Finite-time BO regret)**.** Conditioned on m BO evaluations and q evaluated configurations proposed by the LLM in the optimal branch, denoted by $\mathcal{D}_q^{\mathrm{LLM}}$, the EI proposer is warm-started with this dataset and satisfies the branch-wise simple-regret rate of expected improvement [61]:

$$S_m^{\mathrm{BO|LLM}} \le C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) r_{\mathrm{EI}}(m),$$

where $C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) > 0$ is a data-dependent constant induced by the warm-start observations and

$$r_{\mathrm{EI}}(m) = \begin{cases} m^{-\nu/d_*}(\log m)^{\eta}, & 0 < \nu \le 1, \\ m^{-1/d_*}, & \nu > 1. \end{cases} \tag{8}$$

Here d_* is the dimension of the optimal branch, ν is the RKHS smoothness parameter of the GP kernel, and $\eta \ge 0$ denotes the logarithmic factor in the EI convergence rate. High-quality LLM proposals can reduce the finite-time constant by lowering the initial simple regret and providing informative posterior observations, while the EI exponent $r_{\mathrm{EI}}(m)$ remains unchanged.

Assumption A.3 (LLM-assisted proposal quality)**.** Motivated by empirical in-context scaling laws for LLMs [58, 68], we model the LLM-assisted regret after n observations in the optimal branch as a power-law decay with an irreducible floor:

$$S_n^{\mathrm{LLM}} \le \frac{C_{\mathrm{LLM}}}{n^{\alpha}} + \Delta_{\mathrm{inf}}, \tag{9}$$

where $C_{\mathrm{LLM}} > 0$, $\alpha > 0$, and $\Delta_{\mathrm{inf}} \ge 0$ is the irreducible floor of the LLM proposer. This assumption formalizes the finite-time benefit of semantic priors.

Lemma A.4 (BO evaluations in the optimal branch)**.** *Suppose that whenever A^* is selected, the BO proposer is selected with conditional probability at least $\epsilon > 0$. Then*

$$\Pr\left(M_*(T) \ge \frac{\epsilon N_*(T)}{2} \mid N_*(T) \right) \ge 1 - \exp\left(-\frac{\epsilon N_*(T)}{8} \right).$$

Proof. Index the visits to A^* by $j = 1, \dots, N_*(T)$. The proposer-selection variable can be written as $Z_j = \mathbb{I}\{U_j \le p_j\}$, where U_j are independent uniform random variables and the adaptive probability p_j satisfies $p_j \ge \epsilon$. Thus Z_j stochastically dominates $\mathbb{I}\{U_j \le \epsilon\}$, and $M_*(T)$ dominates a binomial random variable with parameters $N_*(T)$ and ϵ. The multiplicative Chernoff bound gives

$$\Pr\left(M_*(T) < \frac{\epsilon N_*(T)}{2} \right) \le \exp\left(-\frac{\epsilon N_*(T)}{8} \right),$$

which proves the claim. □

Theorem A.5 (Finite-time global simple regret)**.** *Under Assumptions A.1–A.3, with probability at least*

$$1 - \exp\left(-\frac{\epsilon N_*(T)}{8} \right),$$

the global simple regret of LB-MCTS satisfies

$$S_T^{\mathrm{Global}} \le \min\left\{ \frac{C_{\mathrm{LLM}}}{N_*(T)^{\alpha}} + \Delta_{\mathrm{inf}},\ C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) r_{\mathrm{EI}}(M_*(T)) \right\}. \tag{10}$$

Proof. By Assumption A.1, $N_*(T) = \Omega(T)$. Conditioned on this allocation, Lemma A.4 implies that the event $\mathcal{E}_{\mathrm{count}} = \{M_*(T) \ge \epsilon N_*(T)/2\}$ holds with probability at least $1 - \exp(-\epsilon N_*(T)/8)$.

On $\mathcal{E}_{\mathrm{count}}$, Assumption A.2 gives

$$S_{M_*(T)}^{\mathrm{BO|LLM}} \le C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) r_{\mathrm{EI}}(M_*(T)).$$

Assumption A.3 gives

$$S_{N_*(T)}^{\mathrm{LLM}} \le \frac{C_{\mathrm{LLM}}}{N_*(T)^{\alpha}} + \Delta_{\mathrm{inf}}.$$

The best configuration returned by LB-MCTS is the best point observed over the entire global history. Therefore, its global simple regret is no larger than the regret achieved by either the BO-generated points or the LLM-assisted points inside the optimal branch:

$$S_T^{\mathrm{Global}} \le \min\left\{ S_{N_*(T)}^{\mathrm{LLM}}, S_{M_*(T)}^{\mathrm{BO|LLM}} \right\}.$$

Combining the two branch-wise bounds with the root-allocation and proposer-count events yields Eq. (10). □

Proof of Theorem 3.1. By Assumption A.1, $N_*(T) = \Omega(T)$ as $T \to \infty$. Since $N_*(T) = \Omega(T)$, the failure probabilities $\exp(-\epsilon N_*(T)/8)$ are summable. By Lemma A.4 and the Borel–Cantelli lemma, the event $M_*(T) \ge \epsilon N_*(T)/2$ fails only finitely often almost surely. Hence $M_*(T) \to \infty$ almost surely. The EI rate satisfies $r_{\mathrm{EI}}(M_*(T)) \to 0$. Assumption A.2 gives the warm-started BO bound

$$S_{M_*(T)}^{\mathrm{BO|LLM}} \le C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) r_{\mathrm{EI}}(M_*(T)).$$

The warm-start set affects only the data-dependent constant; along the optimization trajectory this constant is finite and remains bounded, so the BO term converges to zero. Therefore, the BO term in Eq. (10) converges to zero almost surely, and because the global simple regret is upper-bounded by the minimum in Eq. (10), the global simple regret also converges to zero almost surely. □

Corollary A.6 (Finite-time budgets under linear root allocation)**.** *Under Assumptions A.2 and A.3, consider the linear-allocation regime $N_*(T) = \Omega(T)$ for both LB-MCTS and the BO-only B-MCTS baseline, on their respective root-allocation events. Under Assumption A.1, controlled violations with $B_\Sigma(T, \rho) = \mathcal{O}(\sqrt{T})$ provide such a regime; the allocation condition must hold for each method separately. Assume the BO regret constants admit fixed uniform bounds C_{EI} for the baseline and $C_{\mathrm{EI}}^{\mathrm{LLM}}$ for the interleaved search, where $C_{\mathrm{EI}}(\mathcal{D}_q^{\mathrm{LLM}}) \le C_{\mathrm{EI}}^{\mathrm{LLM}}$ along the latter trajectory. For the assumed EI rate in Eq.* (8) *with $\nu > 1$,*

$$r_{\mathrm{EI}}(m) = m^{-1/d_*}.$$

For a target simple regret $\delta > 0$, a sufficient total evaluation budget for the BO-only baseline is, up to logarithmic factors,

$$T_{\mathrm{Base}}(\delta) = \widetilde{\mathcal{O}}\left(\left(\frac{C_{\mathrm{EI}}}{\delta} \right)^{d_*} \right).$$

Here the EI search in the optimal branch has dimension d_*, and linear root allocation converts its branch budget to a total budget of the same order. By Eq. (10)*, either the LLM term or the BO term falling below δ suffices to certify the target for LB-MCTS. For $\delta > \Delta_{\mathrm{inf}}$, this gives*

$$T_{\mathrm{LB}}(\delta) = \widetilde{\mathcal{O}}\left(\min\left\{ \left(\frac{C_{\mathrm{LLM}}}{\delta - \Delta_{\mathrm{inf}}} \right)^{1/\alpha}, \frac{1}{\epsilon} \left(\frac{C_{\mathrm{EI}}^{\mathrm{LLM}}}{\delta} \right)^{d_*} \right\} \right),$$

where root-allocation constants and fixed confidence factors are suppressed. These are sufficient high-probability budget bounds under the stated assumptions. For the BO count, coupling the first deterministic n branch visits with independent Bernoulli variables of mean ϵ bounds the joint event $\{M_*(T) < \epsilon n/2,\ N_*(T) \ge n\}$ by $\exp(-\epsilon n/8)$. This failure probability and those of the root-allocation events are combined by a union bound. When $\delta \le \Delta_{\mathrm{inf}}$, the LLM floor prevents the first term from certifying the target, but the BO term still gives

$$T_{\mathrm{LB}}(\delta) = \widetilde{\mathcal{O}}\left(\frac{1}{\epsilon} \left(\frac{C_{\mathrm{EI}}^{\mathrm{LLM}}}{\delta} \right)^{d_*} \right).$$

For an algebraic comparison, define the nominal ratio of the displayed leading budget expressions, setting aside the suppressed allocation and confidence factors:

$$R_{\mathrm{budget}}(\delta) := \begin{cases} \max\left\{ \left(\frac{C_{\mathrm{EI}}}{\delta} \right)^{d_*} \left(\frac{\delta - \Delta_{\mathrm{inf}}}{C_{\mathrm{LLM}}} \right)^{1/\alpha},\ \epsilon \left(\frac{C_{\mathrm{EI}}}{C_{\mathrm{EI}}^{\mathrm{LLM}}} \right)^{d_*} \right\}, & \delta > \Delta_{\mathrm{inf}}, \\ \epsilon \left(\frac{C_{\mathrm{EI}}}{C_{\mathrm{EI}}^{\mathrm{LLM}}} \right)^{d_*}, & \delta \le \Delta_{\mathrm{inf}}. \end{cases} \tag{11}$$

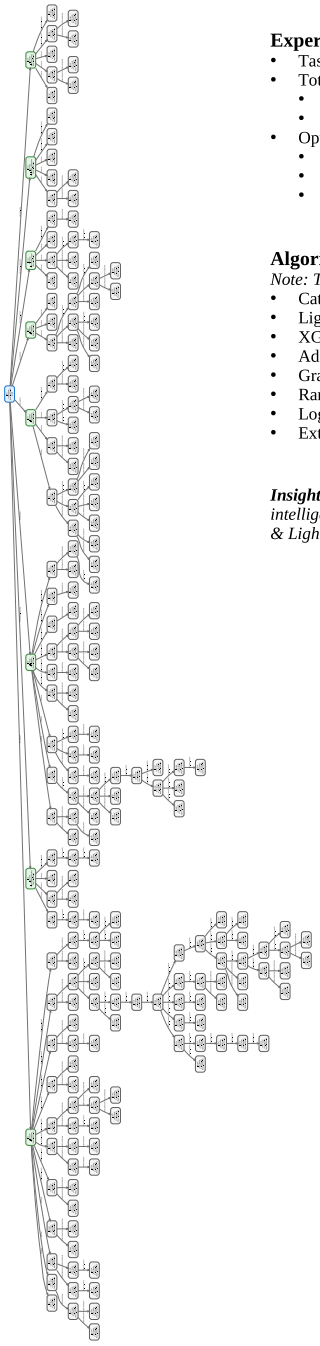
This ratio compares budget expressions; it is not a bound on the ratio of actual hitting times, which cannot be inferred by dividing two complexity upper bounds. Informative LLM observations may lower the BO regret constant, although $C_{\mathrm{EI}}^{\mathrm{LLM}} < C_{\mathrm{EI}}$ is not assumed. Under Assumption A.3, directly evaluated LLM proposals also provide a separate route to the target through Eq. (9)*.*

Other allocation regimes. The above total-budget orders apply whenever the corresponding linear-allocation conditions hold, including some cases with $B_\Sigma(T,\rho) = \Theta(T)$, such as $B_\Sigma(T,\rho) = 0.2T$. More generally, if both methods satisfy $N_*(T) = \Omega(T^\beta)$ for a common $0 < \beta \leq 1$, converting their branch budgets to total budgets raises the leading expressions to the power $1/\beta$. With allocation constants, logarithmic factors, and confidence factors suppressed as above, the corresponding nominal budget ratio is $[R_{\mathrm{budget}}(\delta)]^{1/\beta}$. Square-root allocation corresponds to $\beta = 1/2$, giving the nominal ratio $[R_{\mathrm{budget}}(\delta)]^2$. Thus, whenever $R_{\mathrm{budget}}(\delta) > 1$, a favorable nominal budget comparison is preserved across these allocation regimes.

Theoretical-Empirical Alignment. Empirically, LB-MCTS achieves an average $2.37\times$ acceleration over B-MCTS (which retains only the BO proposer) across 10 datasets in the BO–LLM collaboration ablation (Figure 7). This is an observed speedup, distinct from the nominal budget comparison above.

A.3 A Case of MCTS Result

While Figure 1 in the main paper shows an abstract schematic of the MCTS search structure, Figure 8 presents a concrete example of an actual search tree generated by LB-MCTS on the "Adult" dataset. The visualization reveals how BO and the LLM proposer collaboratively expand the tree: The first leaf under each Algo node serves as the initial configuration for that path, established via either an LLM warm-start or BO random search. Along a typical optimization path, BO-driven local search and LLM-driven exploratory/exploitative proposals alternate in refining configurations.



Experiment Statistics

- Task: Classification (Dataset: adult)
- Total Nodes: N=309
  - 1 CASH Node + 8 Algo Nodes
  - 300 HPO Nodes
- Optimization Strategy:
  - Warmup: 24 trials
  - LLM(98 trials): Exploration(56) & Exploitation(42)
  - BO(178 trials): Random(46) & Local Search(132)

Algorithm Counts

Note: The visualization selectively displays dense trajectories for clarity.

- CatBoost: 129
- LightGBM: 81
- XGBoost: 27
- AdaBoost: 22
- Gradient Boosting: 11
- Random Forest: 10
- Logistic Regression: 10
- Extra Trees: 10

***Insight**: The tree structure reveals a dynamic search pattern. The method intelligently distributes 70% of the budget between the two top performers(CatBoost & LightGBM) while automatically assigning fewer trials to other candidates.*

Figure 8: MCTS tree generated by LB-MCTS on the Adult dataset.

B Prompt Design and Case Studies

B.1 Prompt Design For Hyperparameter Tuning Phase

The hyperparameter tuning process follows a dual-prompt mechanism: a static **System Prompt** defining the agent's persona, and a dynamic **User Prompt** supplying the optimization context. The User Prompt is constructed using a template comprising four key components: (i) task description, (ii) dataset context, (iii) selective tuning memory, and (iv) output requirements. Crucially, the strategic directive within the task description is instantiated at runtime based on the MCTS state.

System Prompt:

You are an expert AutoML strategist. Your current role is **Hyperparameter Tuner** in a Combined Algorithm Selection and Hyperparameter optimization (CASH) task.
An algorithm has been selected. Your goal is to adjust a basic hyperparameter configuration on a monte carlo tree search node to find the best hyperparameter configuration for it.

Template: User Prompt For Hyperparameter Tuning Phase

Task Description:

Your ultimate goal is to solve the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem: finding the best possible algorithm and its optimal hyperparameters for a classification task on the provided dataset.
Algorithm '`[ALGORITHM]`' has been selected. Your immediate objective is to propose a hyperparameter configuration for it.

The proposed configuration must strictly adhere to the defined search space:

- `parameter_1: [TYPE], [RANGE/CHOICES]`
- `parameter_2: [TYPE], [RANGE/CHOICES]`
- `...`

The Basic Configuration to Build Upon:
This configuration is derived from previous trials and represents a promising starting point:
`{ "parameter_1": [VALUE], "parameter_2": [VALUE], ...}`

Your current strategic goal is `[MODE]. [INSTRUCTION_TEXT]`
(Note: This directive is dynamically instantiated based on the MCTS state. See "Strategy Variants" below.)

Dataset Context:

The target task is a `[TASK_TYPE]` problem evaluated on the provided dataset. The primary optimization metric is `[METRIC]`. The dataset consists of `[N_SAMPLES]` samples and `[N_FEATURES]` features...

Selective Tuning Memory:

This component integrates historical trajectories and high-performing trials to inform the current optimization step.

Optimization Trajectory Context:
`[Sequence: Root → ... → Ancestral Node → Current Basic Node]`

Informative Historical Trials:
`[List of Top-k High-Performing & Similar Historical Trials]`

Required Output Format:
You must provide your reasoning followed by the final configuration in the following structure.

Thought: ...
Action: `{ "parameter_1": [VALUE], ...}`

Optimization Directives : Strategy Variants

The placeholders `[MODE]` and `[INSTRUCTION_TEXT]` in the template above are dynamically instantiated based on the MCTS search state.

Mode: WARMUP
Instruction: Propose a promising configuration **from scratch**. Focus on identifying configurations that are **distinct from previous attempts** while still demonstrating strong performance. Analyze historical data to **balance novelty and effectiveness**.

Mode: EXPLORATION
Instruction: Modify the basic configuration to **explore new areas** of the hyperparameter space. Focus on making changes that **introduce diversity** and help uncover potentially high-performing configurations. Make **bold adjustments** to discover new possibilities.

Mode: EXPLOITATION
Instruction: **Refine and optimize** the basic configuration to achieve peak performance. Focus on making **minor, incremental adjustments** to the basic configuration, guiding it towards a more optimal setup using insights from successful trials.

B.2 Prompt Design For Reflection Phase

Similar to the tuning phase, the reflection phase is governed by a dedicated System Prompt that establishes the 'Performance Analyst' persona. The User Prompt is structured to feed execution results back to the LLM, enabling it to perform quantitative attribution analysis and synthesize reusable insights.

System Prompt

You are an expert AutoML strategist. Your current role is **Performance Analyst** in a Combined Algorithm Selection and Hyperparameter optimization (CASH) task.
Your goal is to analyze the results of a hyperparameter tuning trial. You must provide a quantitative, structured, and globally useful reflection that explains the relationship between parameter changes and performance outcomes to guide future trials.

Template: User Prompt For Reflection Phase

Task Description:

Your ultimate goal is to solve the Combined Algorithm Selection and Hyperparameter optimization(CASH) problem: finding the best possible algorithm and its optimal hyperparameters for a classification task on the provided dataset.
A hyperparameter configuration for algorithm `'[ALGORITHM]'` has just been evaluated. Your immediate objective is to **Reflect on Trial Performance**...

The Trial to Analyze:
This section details the configuration changes and the resulting performance shift relative to the basic configuration:
Basic Configuration `{ "parameter": [VALUE], ...}`
Basic Performance `[BASIC_PERF]`
Original Rationale `[AGENT_THOUGHT_FROM_TUNING_PHASE]`
Resulting Configuration `{ "parameter": [VALUE], ...}`
Resulting Performance `[NEW_PERF]` (`[CHANGE_%]` relative to basic)
Global Ranking `[RANK]` **out of** `[TOTAL_TRIALS]` trials

Analytical Reflection Requirements:
Your analysis must be "quantitative, structured, and globally useful":

- In Reflection:
  1. Triage & Compare: Position this trial among historical data...
  2. Attribute & Synthesize: Explain which parameter deviation led to the outcome...
- In Reflection Summary: Distill into a high-confidence summary of global lessons. (Only this summary will be stored in memory).

Dataset Context:

The target task is a `[TASK_TYPE]` problem evaluated on the provided dataset. The primary optimization metric is `[METRIC]`. The dataset consists of `[N_SAMPLES]` samples and `[N_FEATURES]` features...

Selective Tuning Memory:

`[Identical to the Tuning Phase Input: Trajectory Context + Historical Trials]`

Required Output Format:

Reflection:
(Your structured analysis addressing the two points above...)
Reflection Summary:
(A compact, single paragraph summary of global, reusable and high-confidence conclusions...)

B.3 Reflection Design for BO and LLM WARMUP

While the LLM proposer follows each recommendation with a dedicated reflection stage, the BO proposer lacks an inherent mechanism for semantic reasoning. Similarly, LLM WARMUP trials are excluded from LLM-based reflection because the sparse early-stage history provide insufficient observations and lack a reference "basic configuration" for meaningful comparison. To bridge this gap and maintain a consistent search history, we employ fixed programmatic rules to automatically summarize these trials.

Reflection of BO LOCAL SEARCH. The BO LOCAL SEARCH generates candidates by sampling from the neighborhood of a basic configuration via small perturbations. Since this process is

inherently incremental, its reflection is designed to explicitly capture these fine-grained modifications by concatenating two programmatic components:

1. **Parameter Change Description**: Lists all hyperparameters modified from the parent configuration, formatted as "Changed parameters: `{param_name}: {old} -> {new}`".
2. **Performance Change Description**: Quantifies the performance shift using the computed delta, formatted as "Performance {improved|declined} from `{parent_perf}` to `{current_perf}`".

Example Case: Consider a LOCAL SEARCH trial on the `cmc` classification dataset. The optimization transition is defined by the following parent and child nodes:

- Parent Node:
  - Performance: `0.5846677`
  - Configuration: `{"algorithm": "SAMME", "learning_rate": 4.049957e-02, "max_depth": 5, "n_estimators": 217}`
- Child Node:
  - Performance: `0.59120365`
  - Configuration: `{"algorithm": "SAMME", "learning_rate": 4.000687e-02, "max_depth": 5, "n_estimators": 231}`

Based on this transition, the system automatically generates the following textual reflection, which serves as part of the context for future LLM iterations:

System Generated Reflection

```
"Bayesian Local search performed; Changed parameters: learning_rate:
4.049957e-02 -> 4.000687e-02, n_estimators: 217 -> 231. Performance
improved from 0.5846677 to 0.59120365."
```

Special Cases: BO RANDOM SEARCH and LLM WARMUP:

For WARMUP trials and BO RANDOM SEARCH trials (where the parent node is an `HPORootNode`), the reflection string is set to "Warmup configuration." Moreover, instead of a performance change description, the system appends "Initial performance: `{current_perf}`" to indicate this is the first trial of an optimization trajectory.

Notably, if WARMUP or BO RANDOM SEARCH trials achieve exceptionally high performance and are selected by the Pareto frontier, they may appear in the `Informative Historical Trials` section of the prompt. In such cases, only the Child configuration and Reflection are displayed (no Parent), since the parent is an `HPORootNode`.

B.4 Prompt Example: Execution Trace

This section presents a concrete execution trace from the `cmc` classification dataset experiment.It displays the raw input prompts and the corresponding model outputs during a **Hyperparameter Tuning** step and its subsequent **Reflection** step.

User Prompt For Hyperparameter Tuning

Task Description:

Your ultimate goal is to solve the Combined Algorithm Selection and Hyperparameter optimization(CASH) problem: finding the best possible algorithm and its optimal hyperparameters for a classification task on the provided dataset.
Algorithm "adaboost" has been selected. Your immediate goal is to propose a hyperparameter configuration for it.
The proposed configuration must strictly adhere to the defined search space:

- algotirhm: Categorical, {SAMME.R, SAMME}
- learning_rate: Float, Range=[1e-02, 2e+00] (Log-Scale)
- max_depth: Integer, Range=[2, 8]

- n_estimators: Integer, Range=[50,500]

The Basic Configuration to Build Upon:
This configuration is derived from previous trials and represents a promising starting point:
{"algorithm": "SAMME", "learning_rate": 4.5e-02, "max_depth": 5, "n_estimators": 350}

Your current strategic goal is **Exploitation**. Refine and optimize the basic configuration to achieve peak performance. Focus on making minor, incremental adjustments to the basic configuration, guiding it towards a more optimal setup using insights from successful trials.

Dataset Context:

The target task is a **3-class classification** problem evaluated on the provided dataset. The primary optimization metric is **balanced accuracy**, where a higher value indicates superior performance. The dataset consists of **1,178 samples** and **9 total features**, partitioned into 9 numeric and 0 categorical features. The target classes are distributed as **42.7%** for Class 1, **34.7%** for Class 3, **22.6%** for Class 2.

Selective Tuning Memory:

This component integrates historical trajectories and high-performing trials to inform the current optimization step.

Optimization Trajectory Context:
This sequence preserves the ancestral path from the algorithm root, providing trajectory awareness:

...

Ancestral Node { "performance": 0.58031504, "configuration": {"algorithm": "SAMME", "learning_rate": 3.9e-02, "max_depth": 5, "n_estimators": 265}, "reflection": "..." }

Current Basic Node { "performance": 0.57471745, "configuration": {"algorithm": "SAMME", "learning_rate": 4.5e-02, "max_depth": 5, "n_estimators": 350}, "reflection": "Performance degradation (-0.97%) indicates overfitting caused by simultaneously increasing learning_rate (0.039->0.045) and n_estimators (265->350). Future tuning must constrain the learning rate when scaling up model complexity."}

Informative Historical Trials:
The following trials represent top-performing historical attempts that are most similar to the current configuration:

Historical Trial A(Proposed by BO Local Search):

- **Parent**: { "performance": 0.5846677, "configuration": {"algorithm": "SAMME", "learning_rate": 4.049957e-02, "max_depth": 5, "n_estimators": 217} }
- **Child**: { "performance": 0.59120365, "configuration": {"algorithm": "SAMME", "learning_rate": 4.000687e-02, "max_depth": 5, "n_estimators": 231} }
- **Reflection**: "Decreasing learning_rate (0.04049957->0.04000687) and increasing n_estimators (217->231) improved performance by 0.0065 (+1.12%)."

Historical Trial B(Proposed by LLM):

- **Parent**: { "performance": 0.58264467, "configuration": {"algorithm": "SAMME", "learning_rate": 4.1953e-02, "max_depth": 5, "n_estimators": 233} }
- **Child**: { "performance": 0.58918061, "configuration": {"algorithm": "SAMME", "learning_rate": 3.8e-02, "max_depth": 5, "n_estimators": 280} }

- **Reflection**: `"Reducing the learning rate to 0.038 successfully mitigated oscillation, creating the stability required to profitably scale n_estimators from 233 to 280. This confirms that higher model capacity demands a stricter learning rate constraint."`

...

Required Output Format:

You must provide your reasoning followed by the final configuration in the following structure:

Thought:
(Provide a detailed rationale. Analyze the **Optimization Trajectory Context** and the **Informative Historical Attempts** and connect your insights with the **Dataset Summary** and the current **Directive** to justify your parameter adjustments.)

Action:
Output the configuration as a single, valid JSON block.

```
{ "parameter_1": value_1, "parameter_2": value_2, ... }
```

ASSISTANT Response (Tuning)

Thought:
The current directive is **EXPLOITATION**, which requires refining the basic configuration based on historical insights.
Analyzing the *Optimization Trajectory Context*, the current basic node (LR 0.045, Est 350) suffered a performance drop due to overfitting. The reflection explicitly warns that the learning rate must be constrained when scaling up estimators.
Looking at *Informative Historical Trials*, specifically "Historical Trial B", we see a clear success pattern where a lower learning rate of 0.038 enabled the effective scaling of estimators to 280. This aligns perfectly with the trajectory's lesson.
Therefore, to exploit this finding and optimize performance: 1. I will revert the `learning_rate` to the proven stable value of **0.038**. 2. I will set `n_estimators` to **300**. This is a precise adjustment that sits between the under-fitted 265 and the over-fitted 350, slightly pushing the boundary of the successful Trial B (280) to maximize model capacity within the safe learning rate regime. 3. I will maintain the `algorithm` as "SAMME" and `max_depth` at 5, as these are established optima.

Action:

```
{
    "algorithm": "SAMME",
    "learning_rate": 0.038,
    "max_depth": 5,
    "n_estimators": 300
}
```

User Prompt For Reflection

Task Description:

Your ultimate goal is to solve the Combined Algorithm Selection and Hyperparameter optimization(CASH) problem: finding the best possible algorithm and its optimal hyperparameters for a classification task on the provided dataset.
A hyperparameter configuration for algorithm **"adaboost"** has just been evaluated. Your immediate objective is to Reflect on Trial Performance. You must analyze the results of the recent trial to synthesize quantitative, structured, and reusable lessons for future tuning.

The Trial to Analyze:

This section details the configuration changes and the resulting performance shift relative to the basic configuration:

Basic Configuration `{"algorithm": "SAMME", "learning_rate": 4.5e-02, "max_depth": 5, "n_estimators": 350}`

Basic Performance `0.57471745`

Original Rationale "The current directive is EXPLOITATION... [omitted detailed analysis] ... "

Resulting Configuration `{"algorithm": "SAMME", "learning_rate": 3.8e-02, "max_depth": 5, "n_estimators": 300}`

Resulting Performance `0.59244859` (improved by 3.09% relative to the basic configuration)

Global Ranking 1 out of 219 trials of 'adaboost'

Analytical Reflection Requirements:

Your analysis must be **"quantitative, structured, and globally useful"**:

- **In Reflection**:
  1. **Triage & Compare**: Position this trial among historical data using numerical comparisons (absolute differences and percentages).
  2. **Attribute & Synthesize**: Explain which parameter deviation led to the outcome.
     - Highlight **clearly important** parameters, **value ranges** that look promising/risky, and any **interactions** inferred from numbers.
     - Make reasoning **reusable** to guide future tuning (not just describing this single trial).
     - Only include conclusions you are **confident about** based on quantitative evidence.
- **In Reflection Summary**: Distill into a high-confidence summary of global lessons.
  - Include only insights supported by multiple numerical observations.
  - Do **not** invent rules for noisy data; do **not** suggest explicit "next steps" or new configurations.

Dataset Context:

The target task is a **3-class classification** problem evaluated on the provided dataset. The primary optimization metric is **balanced accuracy**, where a higher value indicates superior performance. The dataset consists of **1,178 samples** and **9 total features**, partitioned into 9 numeric and 0 categorical features. The target classes are distributed as **42.7%** for Class '1', **34.7%** for Class '3', **22.6%** for Class '2'.

Selective Tuning Memory:

... (For brevity, the Selective Tuning Memory content is omitted as it remains identical to the Tuning Phase input) ...

Required Output Format:

Reflection:

(Your structured analysis addressing the two points above, focused on quantitative comparison and attribution of the outcome.)

Reflection Summary:

(A compact, single paragraph summary of global, reusable and high-confidence conclusions derived from this analysis.)

ASSISTANT Response (Reflection)

Reflection:

1. **Triage & Compare**: The trial achieved a score of **0.5924**, improving by **3.09%** over the basic configuration (0.5747). Ranking **1st out of 219**, this confirms the configuration is in the optimal region.

2. **Attribute & Synthesize**: The primary driver of success was reducing the `learning_rate` from 0.045 to **0.038**. As hypothesized in the rationale, the previous failure at LR 0.045 was due to overfitting when combined with high estimators. By constraining the LR to 0.038, the

Table 1: Search space for ML algorithms. We distinguish categorical (cat) hyperparameters from numerical (cont) ones.

Type of Classifier	#λ	cat	cont
AdaBoost	4	1	3
Random Forest	5	2	3
Extra Trees	5	2	3
Gradient Boosting	7	1	6
Logistic Regression	4	2	2
LightGBM	7	-	7
Catboost	4	-	4
Xgboost	9	-	9
Total (8 algos)	45	8	37

Type of Regressor	#λ	cat	cont
AdaBoost	4	1	3
Random Forest	5	2	3
Extra Trees	5	2	3
Gradient Boosting	7	1	6
Ridge	4	1	3
LightGBM	7	-	7
Catboost	5	1	4
Xgboost	9	-	9
Total (8 algos)	46	8	38

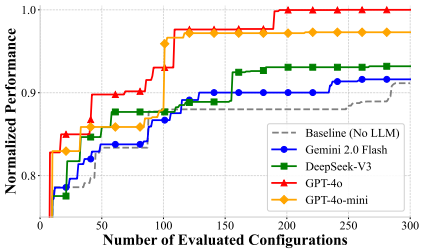


Figure 9: Average performance on 2 datasets.

Table 2: Ablation study of different LLM backbones on the `ada` (classification) and `quake` (regression) datasets. Cost is reported in USD. The best performance for each dataset is bolded, and the second-best results are underlined.

LLM Backbone	ada		quake	
	Bal Acc. ↑	Cost ($)	MSE ↓	Cost ($)
No LLM (Pure BO)	0.8086	0.00	0.03475	0.00
Gemini 2.0 Flash	0.8078	0.11	0.03462	0.06
GPT-4o-mini	<u>0.8126</u>	0.14	<u>0.03458</u>	0.11
DeepSeek V3	0.8102	0.23	0.03472	0.14
GPT-4o	**0.8142**	3.81	**0.03449**	2.85

model successfully accommodated the `n_estimators` (300), validating the interaction pattern observed in Historical Trial B. The stability of `max_depth=5` further supported this gain.

Reflection Summary:
For 'adaboost' on the provided dataset, optimal performance is achieved by **constraining the learning rate to ≈ 0.038** when scaling **n_estimators** (280-300). The interaction is critical: higher capacities require stricter learning rate constraints to prevent overfitting. **max_depth** should be maintained at 5.

C Results and Analysis

C.1 Search Space

We evaluate LB-MCTS across both classification and regression tasks using a search space adapted from the VolcanoML AutoML system [67]. Specifically, we refined the candidate pool by pruning less common models, ultimately retaining eight representative algorithms for each task type to ensure both diversity and efficiency (see Table 1). These selections span major model families, including tree ensembles (e.g., Random Forest, Extra Trees), boosting frameworks (e.g., XGBoost, LightGBM), and linear models (e.g., Logistic Regression, Ridge).

For each algorithm, we construct a comprehensive hyperparameter search space involving a mix of continuous and categorical parameters. As shown in the table, the classification setting involves a total of 45 hyperparameters (37 continuous, 8 categorical) across the 8 algorithms, while the regression setting comprises 46 hyperparameters (38 continuous, 8 categorical). This formulation constitutes a high-dimensional, hierarchical CASH problem, where the optimizer must first select an algorithm and then navigate its specific conditional hyperparameter subspace.

C.2 Ablation on Different LLM Backbones

This appendix further studies how the choice of LLM backbone affects LB-MCTS. We compare four LLMs (GPT-4o [69], GPT-4o-mini [32], DeepSeek-V3 [63], and Gemini-2.0-Flash [65]) and a No LLM variant that removes the LLM proposer and uses only the BO proposer. Due to the substantial

API costs associated with premium models (for instance, GPT-4o can incur expenses of several USD per individual task), experiments are run on one classification task (ada) and one regression task (quake) to cover both metric types while maintaining a manageable budget.

Figure 9 reports the normalized best-so-far validation performance versus the number of evaluations. The most capable (and most expensive) GPT-4o consistently achieves the highest curve, with GPT-4o-mini close behind. Importantly, *all* LLM-based variants outperform the pure-BO baseline, confirming that our integration framework can effectively exploit LLM guidance to complement Bayesian Optimization (BO) regardless of the specific backbone.

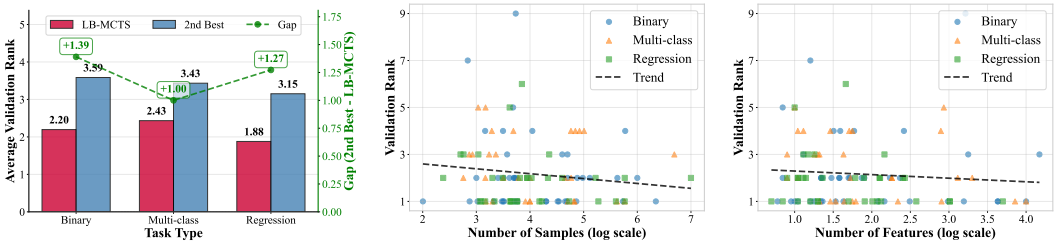
Table 2 summarizes the final performance and cost. GPT-4o represents the performance upper bound, achieving the best results on both datasets (0.8142 Balanced Accuracy and 0.03449 MSE). However, it is also the most expensive, costing significantly more than other backbones. Conversely, Gemini 2.0 Flash is the most affordable option but yields the lowest performance among the LLM variants. GPT-4o-mini emerges as a highly cost-effective choice. It secures the second-best performance ranking while maintaining a low cost (e.g., 0.14 for the "ada" task compared to GPT-4o's 3.81). Its performance-to-price ratio makes it a practical recommendation for budget-constrained large-scale CASH tasks.

To sum up, the results indicate that the "ceiling" of our proposed method is directly correlated with the reasoning capabilities of the underlying LLM. As the field of LLM research continues to evolve—leading to models with enhanced reasoning capabilities and reduced inference costs—the efficacy and accessibility of our framework are expected to scale proportionally, making it increasingly viable for a wider range of applications.

C.3 Implementation Details of Baselines

We adhered to the open-source versions or the methodologies outlined in the original papers for all other baseline implementations. For consistency, we implement the BO components of all baselines with Openbox 0.8.1 [16], an open-source toolkit designed for black-box optimization. We employ the same LLM backbone, GPT-4o-mini [32], for all LLM-based optimizers to eliminate differences due to model strength. Since prior LLM-based methods typically use only a few initial configurations (e.g., $\sim$5 in LLAMBO and BORA) on much simpler tasks, we standardize and strengthen warm-starting for our harder CASH setting. Accordingly, to ensure fairness, all LLM-based baselines use the same number of warm-start configurations as our method (3 LLM-generated initial configurations per algorithm). Meanwhile, pure BO-based baselines follow their standard initialization protocols, as they are natively designed for the CASH setting. All the experiments are conducted on a machine with 24 'AMD EPYC 7702P' CPU cores and 80G memory.

- **SMAC [14].** We implement SMAC as a standard BO-based CASH optimizer using a random-forest surrogate. The optimizer treats CASH as a structured configuration space with conditional hyperparameters, and iteratively proposes one configuration per evaluation.
- **OptDivBO [33].** OptDivBO enhances standard BO by encouraging diversity in the proposed models through ensemble selection. Following the original paper, we fix the diversity trade-off parameter τ at 0.2 and set the ensemble size for post-hoc selection to 25.
- **Rising Bandit(RB) [22].** RB addresses the CASH problem by treating algorithm selection as a multi-armed bandit problem. If the upper bound of the reward of an arm k (algorithm A^k) is less than or equal to the lower bound of another arm in the candidate set, then the arm k will be eliminated from the candidate set. The parameter C for computing the smooth growth rate is set to 7.
- **MOSAIC [34].** MOSAIC uses a tree policy for algorithm selection and BO for HPO. Algorithm selection is guided by a UCB-style criterion with exploration constant $C_{\text{ucb}} = 1.3$, and tree expansion is controlled by progressive widening with coefficient $PW = 0.6$. For the BO component, MOSAIC fits a *single global* random-forest surrogate over the entire CASH space using all observed trials (rather than per-algorithm surrogates), and proposes configurations under the selected algorithm accordingly.
- **OPRO [53].** OPRO treats LLM as a standalone optimizer. In each iteration, we prompt the LLM with the task description and a truncated history containing only the top-20 best-performing trials found so far. The LLM is instructed to identify patterns from these high-quality examples and propose a single new configuration for evaluation.



(a) Performance by task type. (b) Performance by sample size. (c) Performance by feature count.

Figure 10: Performance vs. task characteristics across 104 datasets.

- **LLAMBO [24].** LLAMBO replaces traditional BO components with LLM. Following the official implementation, we include the entire optimization history in the prompt (subject to context limits). In each round, the LLM is tasked to generate candidates that improve upon the current best score. The method generates 20 candidate configurations per iteration. It then randomly shuffles the order of historical examples 10 times to aggregate the predictions. Consequently, each optimization round necessitates a total of 30 API calls.
- **BOPRO [1].** BOPRO uses BO to optimize the prompt instructions themselves. For the in-context learning component, the prompt includes a set of 8 historical examples. In each iteration, the BO-selected instruction template guides the LLM to reason over these examples and generate a new hyperparameter configuration.
- **BORA [4].** BORA dynamically switches between an LLM and a GP-BO proposer based on posterior variance. When the LLM is active, we follow the original paper's setting by including the full optimization history in the prompt to maximize context awareness.

C.4 Impact of Task Characteristics

We further analyze how the performance of LB-MCTS correlates with task characteristics by examining the final ranks after 300 iterations across all 104 datasets. Figure 10 visualizes the relationship between the validation rank and three key factors: task type, sample size, and feature dimensionality.

Impact of Task Type. Figure 10a compares the average rank of LB-MCTS against the second-best baseline across Binary Classification, Multi-class Classification, and Regression tasks. LB-MCTS consistently outperforms the runner-up with a significant rank advantage across Binary ($+1.39$), Multi-class ($+1.00$), and Regression ($+1.27$) tasks. Notably, it achieves the best performance on Regression tasks (Rank 1.88). This may be attributed to the continuous nature of the MSE metric used in regression, which provides smoother gradient-like feedback for the LLM's reflection mechanism compared to the discrete nature of Balanced Accuracy in classification.

Impact of Problem Complexity. Figure 10b and Figure 10c plot the validation rank against the $\log_{10}$-scale sample size and feature count, respectively, with fitted trend lines. We observe a clear downward trend in rank (indicating better performance) as task complexity increases. To quantify this, we partition the datasets into three equal-sized groups (Small/Medium/Large) based on sample size or feature count respectively.

- **Sample Size:** The average rank progressively improves from 2.31 on small datasets ($n < 10^{3.54}$) to 2.24 on medium datasets ($10^{3.54} \leq n < 10^{4.53}$), and reaches **1.94** on large datasets ($n \geq 10^{4.53}$).
- **Feature Count:** Similarly, the rank improves from 2.37 on low-dimensional tasks ($p < 10^{1.33}$) to 2.15 on medium-dimensional tasks ($10^{1.33} \leq p < 10^{2.05}$), and achieves **1.97** on high-dimensional tasks ($p \geq 10^{2.05}$).

Overall, across all 104 datasets, LB-MCTS demonstrates robust dominance, achieving Rank 1 on 43 tasks and Rank ≤ 3 on 87 tasks. These results suggest that LB-MCTS is particularly adept at handling complex, large-scale optimization problems where traditional methods often struggle to scale.

Table 3: Search space for DL models. The same space is used for classification and regression tasks.

Type of Model	#λ	cat	cont
MLP	5	3	2
ResNet-Tabular	7	4	3
FT-Transformer	9	6	3
TabNet	10	3	7
Total (4 algos)	31	16	15

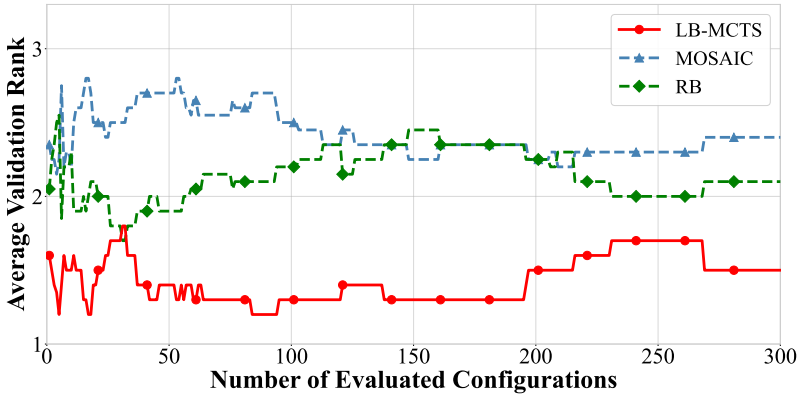


Figure 11: Average validation rank over 10 datasets on the deep learning pipeline scenario.

C.5 Supplementary Experiments on Deep Learning and Feature Engineering

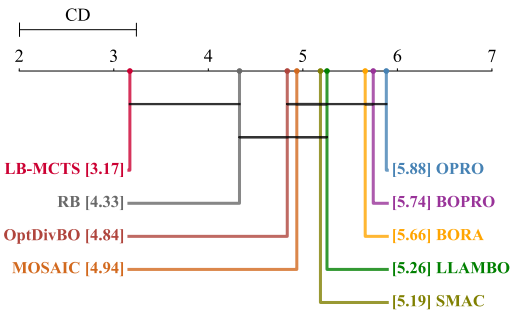
Furthermore, we conducted supplementary experiments on a more diverse search space including both deep learning (DL) models and feature engineering (FE). Specifically, we considered four DL models, MLP, ResNet-Tabular and FT-Transformer [66], and TabNet [59], forming a 31-dimensional model hyperparameter space (Table 3), together with a 10-dimensional FE space. In each model subtree, LB-MCTS jointly optimizes the model hyperparameters together with the 10-dimensional FE space. This search space is shared by classification and regression tasks, and jointly optimizing FE with each model is natural since effective feature engineering is often model-dependent.

Results across 10 representative datasets in Figure 11 show that LB-MCTS achieves the best average rank of 1.5, outperforming the top two baselines from our main experiments. This confirms that our framework consistently excels at searching within structured spaces, regardless of the underlying model type.

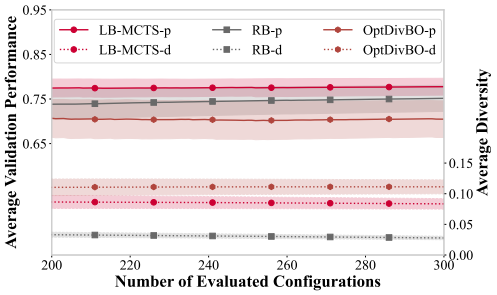
C.6 Post-hoc Ensemble Performance

In practical applications, an ensemble of promising configurations is sometimes preferred over a single best model, as ensembling can potentially enhance overall performance [64, 71]. To evaluate this auxiliary aspect, we perform post-hoc ensembling using the pool of 300 models identified during the optimization process (without retraining). Among different ensemble strategies (e.g., Bagging, Boosting, Stacking), we adopt the ensemble selection [62], which works empirically well with AutoML as shown in previous studies [67] and [38, 33]. In short, ensemble selection starts from an empty ensemble and iteratively adds models from the pool with replacement to maximize the ensemble validation performance (with uniform weights).

Figure 12a presents the *test performance after ensemble selection* on 104 datasets. Two points emerge: (i) Notably, OptDivBO rises from the lowest-ranking BO baseline (in single-model settings) to the second-best position. This validates that effective ensembling requires not just strong individual learners but also diversity among them [60, 73], which aligns precisely with OptDivBO's motivation of considering model diversity. (ii) LB-MCTS maintains its leading position, achieving **an average rank of 3.17**—a significant margin over the second-best performing method (4.33). This is attributed to its explicit balance of inter- and intra-algorithm exploration and exploitation; LB-MCTS not only



(a) CD plot of ensemble test performance.



(b) Average validation performance and diversity on "ada".

Figure 12: Post-hoc ensemble performance of 9 methods across 104 datasets.

identifies high-performing individual models but also ensures broad coverage, effectively achieving the trade-off between individual model performance and diversity among models.

To support this interpretation, Fig. 12b reports, on the "ada" dataset, how (i) the *average validation performance* (solid lines) of all evaluated models and (ii) the *average inter-model diversity* (dashed lines) evolve over the course of optimization for the three methods with the best average ensemble performance. Following [72] and [38], we quantify the diversity between two models $(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j)$ as the average Euclidean distance between their predicted class-probability vectors on the validation set:

$$\mathrm{Div}(x_i, x_j) = \frac{\sqrt{2}}{2} \cdot \frac{1}{|D_{\mathrm{val}}|} \sum_{s \in D_{\mathrm{val}}} \left\| T_{\boldsymbol{\lambda}_i}(s) - T_{\boldsymbol{\lambda}_j}(s) \right\|_2 ,$$

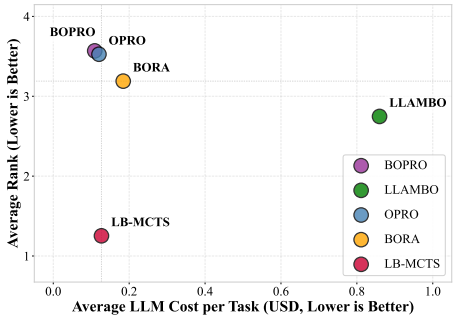
where $T_x(s)$ denotes the class-probability prediction of model x on validation example s. We can observe from the figure that OptDivBO yields the highest average diversity among the three models, but its average validation performance is substantially worse than the other two methods, resulting in an extreme "diverse-but-weak" pool. RB shows the opposite behavior: it achieves the second-best average validation performance but produces the least diverse model set. In contrast, LB-MCTS attains the best average validation performance while maintaining high diversity (second only to OptDivBO), providing a more balanced and thus more effective pool for post-hoc ensemble selection, which aligns with its strongest ensemble results. We note that this balance arises as a by-product of LB-MCTS 's exploration–exploitation design; explicitly optimizing for ensemble performance is left for future work.

C.7 LLM Cost Comparison

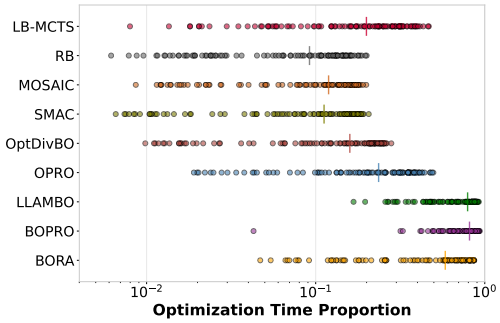
To comprehensively evaluate the practicality of LB-MCTS, we analyze the cost from two perspectives: the monetary cost of LLM API usage and the temporal overhead of the optimizer. Figure 13 summarizes the results across 104 datasets.

Monetary Cost vs. Performance. We estimate API cost using model pricing at the time of experiments, accounting for input and output tokens. Figure 13a illustrates the final average validation rank (across 104 datasets after 300 rounds) versus the average LLM API cost per task (in USD). The results highlight several key insights: (i) LB-MCTS occupies the ideal bottom-left quadrant of the plot, achieving a state-of-the-art **average rank of 1.25** with an **economical cost of approximately $0.127 per task**. Compared to other baselines, LB-MCTS provides a significant performance leap without a proportional increase in expenditure. (ii) While LLAMBO attains the second-best rank (2.73), it is by far the most expensive method, costing $0.860 per task—nearly 6.8× that of LB-MCTS. This high cost stems from its high-volume API calls per iteration, which do not translate into superior rankings compared to our tree-search approach. (iii) Methods like BOPRO, OPRO, and BORA cluster in the low-cost region ($0.11–$0.18) but suffer from poor optimization efficacy, with average ranks exceeding 3.20. Although these methods are budget-friendly, their inability to effectively navigate complex search spaces limits their utility.

Optimizer Time Overhead. Figure 13b reports the ratio of time spent on optimization (proposal generation) to the total runtime (optimization + model evaluation) over 300 iterations. Ideally, the



(a) Cost-performance trade-off of LLM-based methods.



(b) Proportion of time spent on optimization.

Figure 13: Comprehensive Cost and Performance Analysis.

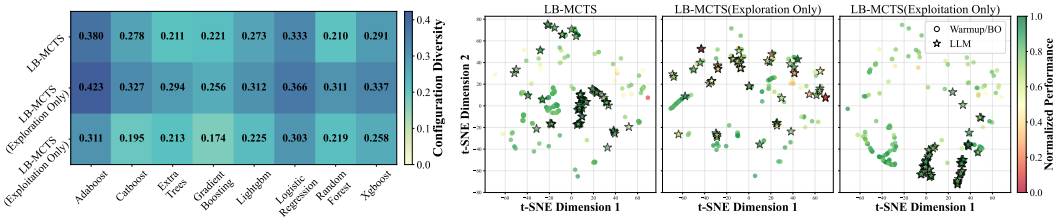
Table 4: Overview of the 10 AMLB datasets selected for ablation studies.

Task Type	Dataset Name (OpenML ID)	Samples (n)	Features (p)	Classes (C)
Binary Classification	ada (190411)	4,147	49	2
	blood-transfusion (359955)	748	5	2
	sylvine (359972)	5,124	21	2
	qsar-biodeg (359956)	1,055	41	2
Multi-class Classification	cmc (359959)	1,473	9	3
	GesturePhaseSegmentation (359970)	9,873	32	5
	connect-4 (359977)	67,557	42	3
Regression	house_prices_nominal (359951)	1,460	79	-
	pol (359946)	15,000	48	-
	quake (359930)	2,178	3	-

optimizer should be lightweight, directing the majority of the time budget toward model training and evaluation. The results highlight several key insights: (i) LB-MCTS maintains a low overhead with a median ratio of 0.20, meaning $\approx 80\%$ of the time is productively spent on evaluation. This is significantly more efficient than other LLM-based approaches like OPRO (0.2356) and BORA (0.5841). The most extreme examples are LLAMBO (0.80), which suffers from excessive API calls, and BOPRO (0.81), which is slowed down by embedding model computations for all configurations. (ii) Scalability to Complex Tasks: Intuitively, as the dataset scale increases, model evaluation time grows, naturally reducing the optimizer's relative overhead. Indeed, for large-scale tasks, LB-MCTS's overhead ratio drops as low as 0.008. Crucially, this aligns perfectly with our finding in Appendix C.4 that LB-MCTS's performance advantage is most pronounced on large, complex datasets, making it highly suitable for real-world, high-stakes optimization scenarios. (iii) From a wall-clock perspective, in our GPT-4o-mini benchmark runs, the average per-iteration time costs are 13.78s for LLM inference and 2.56s for BO computation, while the black-box evaluation (model training and validation) averages 63.02s. The LLM overhead is therefore worthwhile because better proposals reduce the number of expensive evaluations needed: LB-MCTS achieves average end-to-end wall-clock speedups of 2.0$\times$ over RB and 2.2$\times$ over MOSAIC, the second- and third-ranked baselines, respectively. (iv) Finally, we note that as LLM inference speeds continue to improve, the temporal overhead of LB-MCTS will further decrease, enhancing its applicability to time-sensitive tasks.

C.8 Representative Dataset Selection for Ablation Study

Conducting hyperparameter optimization (HPO) with Large Language Models (LLMs) incurs substantial computational overhead and API costs. To balance experimental rigor with resource constraints, we selected a representative subset of **10 datasets** from the AutoML Benchmark (AMLB) suite for our ablation studies. As summarized in Table 4, these datasets were carefully chosen with three criteria: (i) Diverse task types: Including Binary Classification (4), Multi-class Classification (3), and Regression (3). This distribution is aligned with that of the original AMLB benchmark. (ii) Varying data scale: Sample sizes (n) ranging from 748 to 67,557. (iii) Varying feature dimensionality: Feature



(a) Average diversity heatmap across 10 datasets.

(b) Distribution visualization of evaluated configurations with t-SNE.

Figure 14: Visualizations for the Exploration-Exploitation Ablation Study. (a) compares the hyperparameter diversity, showing LB-MCTS achieves a balance between the extremes. (b) visualization of dominant algorithm "Adaboost" on "qsar-biodeg"

counts (p) ranging from 4 to 80. As all the datasets are collected from OpenML [70], we provide the OpenML ID as the identification of the dataset.

To quantify the representativeness of this subset, we compare the nine methods in the main experiment (Figure 3a) at 100, 200, and 300 evaluations. At each checkpoint, we rank the methods on each dataset by their best-so-far validation performance and average these ranks over the original 10-dataset subset and over the full 104-dataset benchmark, respectively. We then compute the Spearman and Kendall correlations between the resulting two nine-method average-rank vectors.

Table 5: Rank correlations between the nine-method average-rank vectors on the original 10-dataset subset and the full 104-dataset benchmark.

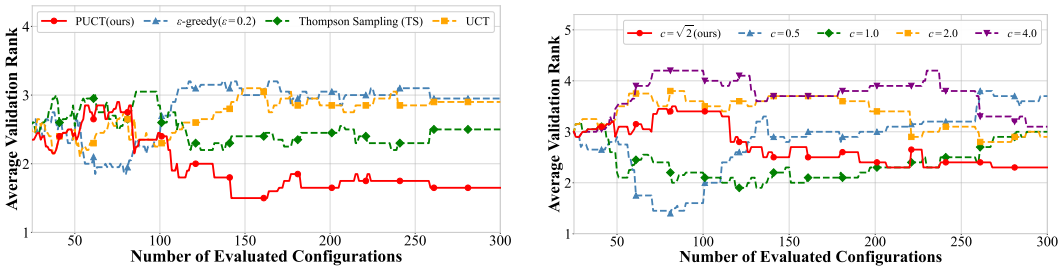
Evaluations	**Spearman ρ**	**Kendall τ**
100	0.783	0.611
200	0.850	0.722
300	0.971	0.930

Table 5 shows that the ranking agreement is already substantial at 100 evaluations and increases with the evaluation budget, becoming near-perfect at 300 evaluations. These results indicate that the original 10 datasets capture the relative performance trends of the methods in the main comparison, supporting their use for the ablation studies.

C.9 Ablation Study: Exploration and Exploitation Strategy

Implementation details of LB-Flat. To isolate the contribution of the hierarchical tree structure and trajectory-aware reasoning, we implement LB-Flat, a variant that collapses the MCTS into a two-level structure (Root and Algo Nodes). It treats all evaluated configurations and performance within an algorithm's subspace as an unstructured history. The BO proposer remains unaffected, as it still suggests candidates via random sampling and local refinement of historical observations. Similarly, the Proposer Selection mechanism is unchanged, continuing to calculate based on the BO surrogate model's ability to fit the existing observations for each algorithm. In contrast, the LLM proposer is significantly modified: it no longer employs PUCT rules to select basic node configurations for refinement via exploration or exploitation directives, but prompts the LLM to directly recommend the next configuration. Due to the absence of optimization trajectories, Global and Local memories are discarded; instead, the top-20 historical observations are used as memory (following the OPRO approach). Finally, the reflection prompt is adjusted by removing all information related to basic configurations.

Tuning behaviors of EXPLORATION and EXPLOITATION. This appendix extends the ablation study on LB-MCTS's EXPLORATION and EXPLOITATION strategies by investigating their impact on the distribution of proposed configurations. We first analyze the *diversity of proposed configurations*. Unlike our previous analysis of ensemble diversity, which considered the predictive differences between models on the validation set, here we quantify diversity directly at the level of the proposed hyperparameter configurations themselves. This allows for a more direct behavioral analysis



(a) Tree-search strategy.

(b) Exploration weight c_{puct}.

Figure 15: Ablations of the PUCT-based tree search.

of how different strategies shape the search space. Specifically, the diversity score is calculated independently for each algorithm. For a given algorithm A^i with a hyperparameter space Λ^i, the overall diversity score $D(A^i)$ is defined as the mean of the normalized diversity scores of all its hyperparameters: $D(A^i) = \frac{1}{|\Lambda^i|} \sum_{p \in \Lambda^i} S(p)$, where $S(p)$ denotes the normalized diversity score for a specific parameter p. The calculation of $S(p)$ depends on the parameter type (numeric or categorical).

- **Numeric Parameters.** For continuous or integer parameters (e.g., learning rate, max depth), we normalize sampled values to $x' \in [0, 1]$ using linear or log-domain min-max scaling as appropriate. The diversity score $S_{num}(p)$ is then defined as the standard deviation of these normalized samples: $S_{num}(p) = \text{std}(\{x'_i\})$.
- **Categorical Parameters.** We measure categorical diversity using normalized Shannon entropy: $S_{cat}(p) = \frac{-\sum_{k=1}^{K} q_k \ln(q_k)}{\ln(K)}$, where q_k is the empirical probability of category k among K possibilities. This ensures $S_{cat}(p) \in [0, 1]$, with 1 representing a perfectly uniform distribution (maximum diversity).

Figure 14a presents the heatmap of diversity scores by algorithm across all 10 datasets. Note that diversity is quantified per algorithm rather than across algorithms, preventing bias from different hyperparameter dimensionalities. We can observe that the EXPLORATION Only strategy (middle row) exhibits the highest diversity across almost all algorithms, indicating a broad but potentially inefficient search that mimics random sampling. The EXPLOITATION Only strategy (bottom row) shows the lowest diversity, suggesting rapid convergence to a narrow region of the hyperparameter space, carrying the risk of getting trapped in local optima. LB-MCTS (top row) maintains an intermediate diversity level. This confirms that our method effectively balances the trade-off, exploring sufficient configuration space before focusing resources on promising regions. To further illustrate how these diversity patterns arise, Figure Figure 14b visualizes the evaluated configurations for the AdaBoost algorithm on the "qsar-biodeg" dataset using t-SNE, with LLM-proposed points marked as stars and warmup/BO proposals as circles. Under the EXPLORATION Only strategy (middle), LLM proposals are widely scattered over the space, with many mediocre or poor configurations. Conversely, the EXPLOITATION Only case (right) shows LLM proposals tightly clustered in a single area; while the performance is acceptable, the search fails to expand beyond this local neighborhood, leaving the majority of the space unexplored. In contrast, LB-MCTS (left) generates a distribution that forms distinct, high-performing clusters (dark green stars) while retaining separation between them. This demonstrates that LB-MCTS successfully guides the search to exploit promising modes deeply while periodically exploring new regions to escape stagnation. Collectively, these analyses provide a mechanistic explanation for the accelerated and superior convergence of LB-MCTS observed in Figure 5.

C.10 Ablation of PUCT-Based Tree Search

In this appendix, we conduct ablation studies on 10 representative datasets to first evaluate whether PUCT is an appropriate tree-search policy for LB-MCTS, and then analyze the sensitivity of its exploration weight c_{puct}.

Choice of tree-search policy. We adopt PUCT because it combines empirical value estimates with prior-guided exploration, making it well suited for structured CASH search. As shown in Figure 15a,

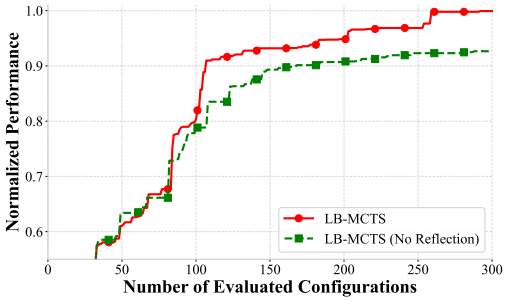


Figure 16: Ablation study on Reflection over 10 datasets.

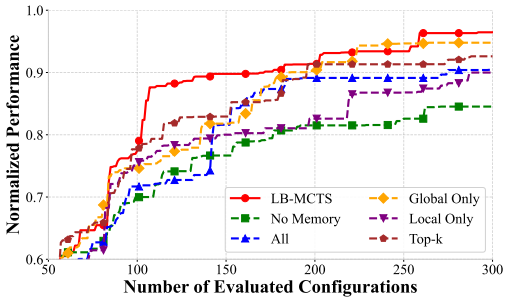


Figure 17: Ablation study on STM over 10 datasets.

PUCT achieves the best average rank (1.65), outperforming Thompson Sampling (2.50), UCT (2.90), and ϵ-greedy (2.95). The advantage comes from PUCT's ability to bias exploration toward globally promising branches while still preserving sufficient search diversity. In contrast, UCT ignores prior information and may spend evaluations on clearly weak branches. ϵ-greedy tends to over-commit to the current best branch, with exploration occurring only through random perturbations, making it vulnerable to local optima. Thompson Sampling provides stochastic exploration, but lacks the explicit prior-weighted control used by PUCT, leading to less stable search in our setting.

Exploration-weight sensitivity. We further evaluate the sensitivity to the exploration weight c_{puct}. As shown in Figure 15b, small values lead to over-exploitation and premature convergence, whereas large values over-emphasize exploration and slow down refinement. The results show that intermediate values are generally stable, and our default choice $c_{puct} = \sqrt{2}$ maintains strong performance throughout the search process.

C.11 Ablation Study: Reflection Mechanism

To assess the contribution of the LLM's self-reflection capability, we conducted an ablation study by deactivating the reflection module (denoted as No Reflection) while keeping all other components unchanged. We aggregated the *normalized best-achieved validation performance* across the 10 selected datasets to analyze the global trend. As observed in Figure 16, (i) LB-MCTS consistently maintains a higher normalized score throughout the optimization process. (ii) The performance gap becomes more pronounced in the later stages (e.g., after 110 iterations). Finally, LB-MCTS achieves a superior **average validation rank of 1.1**, compared to 1.7 for the No Reflection baseline. This indicates that while both methods start similarly, the reflection mechanism effectively enables the framework to learn from historical failures and successes. It acts as a trajectory-level state summarizer to improve planning in the long run rather than a reactive proposal.

C.12 Ablation Study: Selective Tuning Memory

In this appendix, we validate the effectiveness of our Selective Tuning Memory (STM) mechanism. Unlike prior methods that mix history from different algorithms, a key prerequisite of our framework is the isolation of optimization history by algorithm to prevent cross-algorithm noise. Building on this cleaner, algorithm-specific history, we adapt and thereby strengthen retrieval strategies from existing LLM-based optimizers for fair comparison: (1) All: including the entire history of the current algorithm, mimicking the strategy of BORA and LLAMBO; (2) Top-k: retrieving only the best-performing trials, mimicking OPRO (here we set $k = 20$ to match the context length of OPRO). Next, we ablate the components of our STM design: (3) Global Only: retaining only the Pareto-selected global memory; (4) Local Only: retaining only the ancestral trajectory. Note that *all variants operate on per-algorithm histories (no cross-algorithm mixing)*, so differences come purely from the retrieval strategy within each algorithm. Finally, (5) we include a lower bound baseline, No Memory, which provides zero historical trials.

Figure 17 plots the normalized performance averaged over 10 datasets (detailed in Appendix C.8). Five observations can be concluded: (i) LB-MCTS (full STM) outperforms all variants, achieving the **best final average rank of 1.80**. (ii) We observe that strategies relying solely on high-performing samples—Global Only (Rank 2.30) and Top-k (Rank 2.80)—perform reasonably well but are sub-

optimal. This indicates that while knowing "where the best solutions are" is useful, lacking the context of "how we got here" (trajectory) limits the LLM's reasoning ability. In particular, trajectory information may help the LLM (a) avoid repeatedly moving back into regions that have already been explored and found unpromising, and (b) infer trends between parameter changes and performance changes—analogous to a coarse, history-based gradient signal—both of which are absent when only Global Memory are shown. (iii) Global Only outperforms Top-k, confirming that retrieving trials based on the Pareto frontier of similarity and performance provides more relevant guidance to the current expansion task than simply selecting based on raw performance alone. (iv) All (Rank 3.60) exposes the entire history and therefore includes the trajectory, but suffers from severe context pollution—many mediocre or redundant trials dilute the useful signal and consume the prompt budget. (v) Finally, Only Local (Rank 4.30) and No Memory (Rank 5.40) yield the poorest results, underscoring that without a set of high-quality global references, the search struggles to efficiently locate optimal regions.

Appendix References

[58] Cem Anil, Esin Durmus, Nina Panickssery, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Meg Tong, Jesse Mu, Daniel Ford, et al. Many-shot jailbreaking. *Advances in Neural Information Processing Systems*, 37:129696–129742, 2024.

[59] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[60] Yijun Bian and Huanhuan Chen. When does diversity help generalization in classification ensembles? *IEEE Transactions on Cybernetics*, 52(9):9059–9075, 2021.

[61] Adam D Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(10), 2011.

[62] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[63] DeepSeek-AI. DeepSeek-V3 technical report, 2024. URL `https://arxiv.org/abs/2412.19437`.

[64] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

[65] Google DeepMind. Gemini 2.0 flash, 2025. URL `https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash`.

[66] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34: 18932–18943, 2021.

[67] Yang Li, Yu Shen, Wentao Zhang, Ce Zhang, and Bin Cui. Volcanoml: speeding up end-to-end automl via scalable search space decomposition. *The VLDB Journal*, 32(2):389–413, 2023.

[68] Toni JB Liu, Nicolas Boullé, Raphaël Sarfati, and Christopher Earls. Llms learn governing principles of dynamical systems, revealing an in-context neural scaling law. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 15097–15117, 2024.

[69] OpenAI. Gpt-4o model card., 2024. URL `https://platform.openai.com/docs/models/gpt-4o`.

[70] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[71] Beicheng Xu, Wei Liu, Keyao Ding, Yupeng Lu, and Bin Cui. PSEO: Optimizing post-hoc stacking ensemble through hyperparameter tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(32):27188–27196, 2026. doi: 10.1609/aaai.v40i32.39934.

[72] Wentao Zhang, Jiawei Jiang, Yingxia Shao, and Bin Cui. Efficient diversity-driven ensemble for deep neural networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 73–84. IEEE, 2020.

[73] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 summarize the scope and contributions of LB-MCTS, including tree-structured LLM–BO collaboration, STM, dynamic proposer selection, and evaluation on 104 AMLB datasets. These claims are supported by Sections 3.1, 3.2 and 3.3.3 and the experimental results in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Section 5, including noisy validation feedback and dependence on the underlying LLM, and analyze API cost and wall-clock overhead in Appendix C.7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The global convergence theorem is stated in Theorem 3.1, and Appendix A.2 provides the full assumptions, finite-time regret analysis, acceleration characterization, and proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experimental setup in Section 4 specifies the baselines, datasets, metrics, data splits, evaluation budget, LLM backbone, and key hyperparameters. Further reproducibility details are provided in Appendix C.1 (search space), Appendix C.3 (baseline implementations and compute environment), and Appendix B.1 (LLM prompt templates).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the code as supplementary material with instructions for reproducing the main experiments. The code is also available at `https://github.com/PKU-DAIR/open-box`. The datasets are publicly available through AMLB/OpenML.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup is described in Section 4, including datasets, metrics, data splits, baselines, budget, and key hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We assess statistical significance for the main benchmark using a Critical Difference diagram with the Nemenyi post-hoc test over 104 AMLB datasets, as shown in Figure 3b. A similar CD analysis for post-hoc ensemble performance is provided in Appendix C.6.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the compute environment in Appendix C.3 and the evaluation budget in Section 4. Appendix C.7 further analyzes API cost and wall-clock overhead.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses public benchmark datasets and does not involve human subjects, private data collection, or deployment in high-risk settings. To the best of our knowledge, the work conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss broader impacts in Section 5.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release pretrained generative models, scraped datasets, or other assets with high misuse risk.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We properly cite the existing assets used in this work, including AMLB/OpenML datasets, prior AutoML systems and baselines, and LLM backbones. Public datasets are accessed through AMLB/OpenML under their original licenses and terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release the implementation of LB-MCTS as supplementary material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing, user studies, human-subject experiments, or human data collection.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human-subject experiments, crowdsourcing, user studies, or human data collection.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLM usage is a core component of the proposed method. We describe the LLM proposer in Section 3.3.2, and the LLM backbone used in experiments in Section 4. Prompt designs for hyperparameter tuning and reflection are provided in Appendix B.1 and Appendix B.2.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.