TUNE-R2: AMORTIZED SEARCH POLICIES FOR LLM-GUIDED HYPERPARAMETER OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter optimization (HPO) automates the selection of effective configurations in machine learning. Recent work uses large language models (LLMs) to guide this process through semantic priors and feedback from prior trials. However, most existing methods tightly couple *reasoning* with *acting*, repeatedly invoking the LLM over a growing trial history after each evaluation. This increases reasoning cost and can dilute decision-relevant signals with redundant context. In this paper, we introduce TUNE-R2, an **amortized search policy** framework that decouples LLM reasoning from individual objective evaluations. Instead of reasoning anew for every trial, TUNE-R2 induces a reusable search policy from a bounded search state and refines it into a queue of candidate configurations for multiple subsequent evaluations. A state-adaptive policy horizon determines how many evaluations reuse each policy before it is refreshed with new feedback, balancing reasoning amortization against feedback freshness. Across OpenML, FCNet, and NASBench-201, TUNE-R2 achieves the best overall regret, AOC, mean rank, and win rate among evaluated LLM-guided HPO methods while requiring 68%–93% fewer Tokens@Best. Additional experiments on LLM fine-tuning and kernel-SVM HPO further support the generality of the approach. These results demonstrate the effectiveness of amortizing LLM reasoning through reusable search policies.

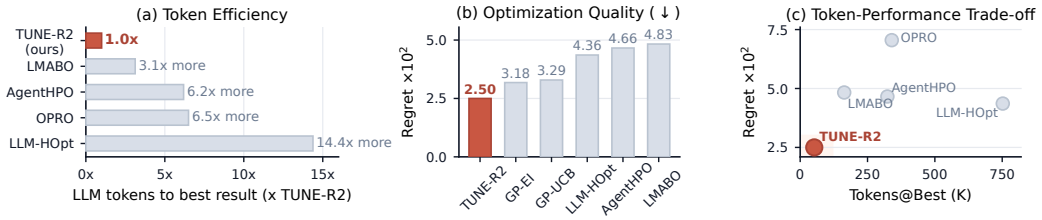

Figure 1: Overall HPO comparison results. (a) Token usage of LLM-based methods to reach the best incumbent, normalized by TUNE-R2. (b) Final optimization quality measured by Regret@30 across representative baselines. (c) Token–performance trade-off between Tokens@Best and Regret@30.

1 INTRODUCTION

Hyperparameter optimization (HPO) is a fundamental component of automated machine learning, seeking high-performing configurations under a limited budget of expensive black-box evaluations (Karmaker et al., 2021). Classical approaches, including random search, Bayesian optimization, and multi-fidelity methods, improve search efficiency through sampling strategies, surrogate models, acquisition functions, or resource-allocation rules (Bergstra & Bengio, 2012; Bergstra et al., 2011; Snoek et al., 2012; Li et al., 2018; Falkner et al., 2018). Recent work has explored large language models (LLMs) as a complementary mechanism for guiding HPO: by reasoning over the search space and feedback from previous trials, LLMs can exploit semantic priors to better inform subsequent search decisions (Tornede et al., 2024; Zhang et al., 2023; Liu et al., 2025; Ngo et al., 2026).

Nevertheless, existing LLM-guided HPO methods tightly and rigidly couple *reasoning* with *acting*. Under this per-trial paradigm, the LLM is synchronously queried after each individual objective

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

evaluation, repeatedly parsing an ever-expanding trajectory context to generate a single candidate configuration. This design introduces two critical limitations. First, repeatedly invoking the LLM increases inference latency and reasoning cost as optimization proceeds. Second, longer histories do not necessarily provide more useful guidance: LLMs can struggle to reliably exploit relevant information in long contexts and are susceptible to distraction from irrelevant evidence (Liu et al., 2024; Shi et al., 2023; Yang et al., 2025b). Our controlled analysis reveals a similar effect in HPO: appending a raw optimization history to an otherwise identical bounded search state degrades candidate quality, suggesting that additional historical context can actively hinder search decisions.

This motivates decoupling LLM reasoning from individual objective evaluations. Consecutive evaluations often share a common search intent, such as exploring a promising region, refining a subset of influential hyperparameters, or diversifying away from a plateau. Rather than reconstructing this intent after every trial, we treat each LLM reasoning cycle as a search-policy update whose guidance can be reused across multiple subsequent evaluations. We refer to this principle as **amortized search policies**. This decouples the frequency of LLM reasoning from the frequency of objective evaluation, introducing a fundamental trade-off between reasoning amortization and feedback freshness: longer policy reuse reduces reasoning overhead, whereas shorter reuse incorporates fresh feedback sooner.

Based on this principle, we introduce TUNE-R2, which realizes amortized search policies through a bounded-state, policy-reuse loop. At each policy refresh, TUNE-R2 first abstracts the accumulated trajectory into a bounded search state that summarizes the incumbent, recent progress, search regime, and prior reflection memory. The LLM then transforms this state into a structured reusable search policy that identifies influential hyperparameters, promising and discouraged regions, and the next search action. This policy is further refined into a short queue of candidate configurations, which are evaluated sequentially without additional LLM reasoning. A state-adaptive policy horizon controls how many evaluations reuse the current policy before the next refresh, allowing TUNE-R2 to amortize reasoning while remaining responsive to new feedback throughout the optimization process.

We evaluate TUNE-R2 across a diverse set of optimization settings, including tabular-model HPO, neural-network HPO, neural architecture search, LLM fine-tuning, and kernel-SVM HPO tasks. Across these settings, TUNE-R2 improves the optimization–reasoning efficiency trade-off over existing LLM-guided HPO methods, while remaining competitive with strong conventional HPO baselines across diverse optimization problem classes. Controlled analyses further isolate the proposed mechanism: bounded-state reasoning avoids the degradation caused by raw-history exposure, structured refinement improves candidate quality, and adaptive policy reuse outperforms fixed reasoning schedules by better balancing policy reuse with feedback freshness. These results support amortized search policies as a general design principle for reducing redundant LLM reasoning without sacrificing optimization quality. Our contributions are summarized as follows:

- We introduce **amortized search policies** for LLM-guided HPO, decoupling LLM reasoning from individual evaluations through reusable search policies.
- We develop TUNE-R2 with bounded-state policy induction, structured refinement, and adaptive policy reuse that balances reasoning amortization with feedback freshness.
- Extensive experiments show that TUNE-R2 achieves the best overall performance while using 68%–93% fewer Tokens@Best than existing LLM-guided HPO methods.

2 Related Work

Black-box and budget-aware hyperparameter optimization. Hyperparameter optimization is commonly formulated as black-box optimization, where each evaluation trains or validates a model under a chosen configuration. Random Search (Bergstra & Bengio, 2012) provides a strong model-free baseline, while Bayesian optimization improves sample efficiency by fitting a surrogate and selecting configurations through acquisition functions such as expected improvement (Jones et al., 1998), probability of improvement (Kushner, 1964), upper-confidence bounds (Srinivas et al., 2010), knowledge gradient (Frazier et al., 2008), and entropy search (Hennig & Schuler, 2012); modern libraries such as BoTorch (Balandat et al., 2020) provide scalable implementations. A complementary line reduces evaluation cost through adaptive resource allocation, including Hyperband (Li et al., 2018) and BOHB (Falkner et al., 2018), with benchmarks such as FCNet (Klein & Hutter, 2019) and HPOBench (Eggensperger et al., 2021). These methods optimize how objective evaluations

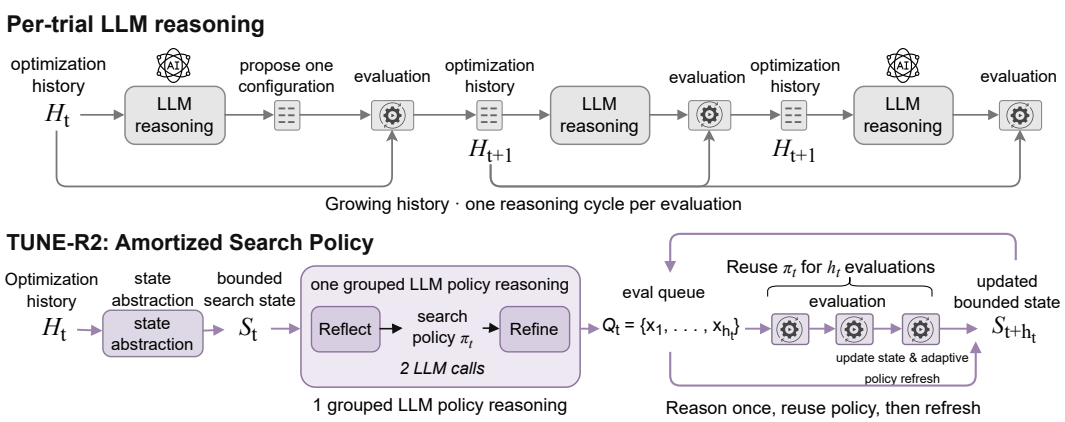

Figure 2: **Overview of TUNE-R2.** TUNE-R2 amortizes LLM reasoning by inducing a search policy from a bounded search state, refining it into an executable candidate queue, and adaptively controlling how many objective evaluations reuse the policy before incorporating new feedback.

are selected or allocated; TUNE-R2 instead focuses on the additional reasoning cost introduced by LLM-guided search and amortizes that reasoning across multiple evaluations in each cycle.

Large language models for optimization. Recent work uses LLMs as optimizers or decision modules in black-box search. OPRO (Yang et al., 2024) performs iterative optimization by prompting over previous solutions and scores, while LLM-HyperOpt (Zhang et al., 2023) directly proposes hyperparameters from the task description, search space, and trial history. LMABO (Ngo et al., 2026) uses LLMs to adapt acquisition behavior, and AgentHPO (Liu et al., 2025) formulates HPO as an agentic loop conditioned on accumulated feedback. These approaches demonstrate the value of semantic priors and history-conditioned reasoning, but typically couple LLM reasoning closely to individual optimization steps. TUNE-R2 instead introduces amortized search policies: it abstracts the trajectory into a bounded search state, induces reusable structured guidance, refines that guidance into an executable candidate queue, and adaptively controls how long the policy is reused before incorporating new feedback, while decoupling LLM reasoning from individual objective evaluations.

3 METHOD

Figure 2 provides an overview of TUNE-R2's amortized policy-reuse loop: (i) the accumulated history is abstracted into a bounded search state; (ii) policy guidance is induced from this state; (iii) the policy is refined into a candidate queue; and (iv) an adaptive policy horizon controls how many evaluations reuse the policy before new feedback triggers the next refresh. The remainder of this section formalizes the policy-reuse abstraction and details components of the refresh mechanism.

3.1 PROBLEM FORMULATION

Let $\mathcal{X}$ be the hyperparameter search space and $f : \mathcal{X} \to \mathbb{R}$ a black-box objective to maximize. At step t, the optimizer evaluates $x_t \in \mathcal{X}$, observes $y_t = f(x_t)$, and updates $\mathcal{H}_t = \{(x_i, y_i)\}_{i=1}^{t}$. Given a budget of T evaluations, the goal is to identify an effective configuration (Karmaker et al., 2021).

For LLM-guided HPO, objective evaluations and LLM reasoning operate on different timescales. Let K denote the number of reasoning cycles used over T evaluations, with cumulative reasoning cost

$$C_T^{\mathrm{LLM}} = \sum_{k=1}^{K} C_{\mathrm{LLM}}^{(k)}. \tag{1}$$

At a fixed evaluation budget T, we seek optimization quality while reducing C_T^{LLM}. Each reasoning cycle induces a search policy with an execution horizon h_k, where $\sum_{k=1}^{K} h_k = T$.

Let $\bar{h} = T/K$ denote the average policy horizon and $\bar{C}_{\mathrm{LLM}}$ the average reasoning cost per refresh. Then

$$C_T^{\mathrm{LLM}} = \frac{T}{\bar{h}} \bar{C}_{\mathrm{LLM}}. \tag{2}$$

Thus, longer policy lifetimes amortize each reasoning cycle over more objective evaluations, whereas shorter lifetimes incorporate new feedback sooner. The problem is therefore not only *what* to evaluate next, but also *how long* to reuse the current policy, exposing a trade-off between reasoning amortization and feedback freshness.

3.2 Amortized Search Policies

TUNE-R2 decouples LLM reasoning from individual evaluations through *policy refreshes*. Each refresh induces short-horizon search guidance from a bounded view of the current trajectory and realizes it as an executable candidate queue.

Let τ_k denote the first evaluation governed by the k-th policy. At each refresh boundary, the available history is abstracted as $S_k = \mathcal{A}(\mathcal{H}_{\tau_k - 1})$. Conditioned on this bounded state, the policy-refresh operator produces

$$(\pi_k, Q_k, h_k) = \mathcal{R}(S_k, \mathcal{X}), \tag{3}$$

where π_k denotes structured short-horizon search guidance, Q_k its executable candidate queue, and h_k its execution horizon.

Policy Execution. Writing $Q_k = (x_{k,1}, \ldots, x_{k,h_k})$, the queue is executed without additional LLM reasoning:

$$x_{\tau_k + j - 1} = x_{k,j}, \qquad j = 1, \ldots, h_k, \qquad \tau_{k+1} = \tau_k + h_k. \tag{4}$$

Objective feedback accumulates throughout execution and is incorporated at the next policy-refresh boundary. Thus, h_k determines the lifetime of the current policy: larger horizons amortize reasoning over more evaluations, whereas smaller horizons incorporate new feedback sooner. Per-trial reasoning is recovered as the special case $h_k = 1$.

3.3 Policy Refresh

The refresh operator $\mathcal{R}$ is realized through bounded-state construction, structured policy induction, adaptive horizon selection, and candidate refinement. Each refresh uses two LLM calls; horizon selection, queue construction, and state updates are performed locally.

Bounded Search State. The abstraction $\mathcal{A}$ distills the accumulated trajectory into $S_k = (\xi_k, b_k, \nu_k, \omega_k, m_k)$, where ξ_k summarizes the recent trajectory, b_k is the incumbent, ν_k contains progress indicators, ω_k denotes the current search regime, and m_k is bounded reflection memory. The state retains recent objective observations, improvement and plateau signals, influential hyperparameters, promising or discouraged regions, and the previous search action. The full history remains available internally for bookkeeping and state updates, but only S_k is exposed to the LLM. Since all fields have bounded capacity, the LLM-visible context does not grow with the optimization history.

Policy Induction. The first LLM call performs structured reflection over the bounded state:

$$(\pi_k, \hat{h}_k) = \mathrm{Reflect}_{\mathrm{LLM}}(S_k, \mathcal{X}), \tag{5}$$

where π_k identifies influential hyperparameters, promising and discouraged regions, diversification needs, and the next search action. The reflection also produces a recommended execution horizon $\hat{h}_k$. Rather than directly emitting the next configuration, this stage converts trajectory evidence into reusable search guidance.

Adaptive Policy Horizon. The reflection recommends an execution horizon conditioned on the current bounded search state. A valid recommendation is clipped to the admissible range $[h_{\min}, h_{\max}]$; if no valid recommendation is available, the controller falls back to a state-dependent prior based on the current search regime and plateau signal. We use $h_{\min} = 1$, $h_{\mathrm{mid}} = 3$, and $h_{\max} = 5$. This step requires no additional LLM call and allows policy lifetime to adapt to the evolving search state.

Candidate Refinement. Conditioned on the induced policy and selected horizon, the second LLM call generates an over-complete proposal pool:

$$U_k = \mathrm{Refine}_{\mathrm{LLM}}(S_k, \pi_k, h_k, \mathcal{X}). \tag{6}$$

Each proposal includes a relative policy-preference score and is then validated and deduplicated locally. The remaining valid candidates are ranked by combining these preference scores with a diversity-aware novelty bonus when diversification is requested, keeping queue construction local after proposal generation. The top h_k candidates form Q_k and are evaluated sequentially without further LLM reasoning, after which the accumulated feedback triggers the next policy refresh. The complete optimization procedure is provided in Algorithm 1 (Appendix A).

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate TUNE-R2 on **19 optimization tasks** spanning HPO, neural architecture search, and LLM fine-tuning. The main suite contains 17 tasks from three standard HPO/NAS benchmarks. OpenML (Vanschoren et al., 2014) includes 11 real-world tabular tasks spanning classification and regression. FCNet (Klein & Hutter, 2019) contains three neural HPO tasks over a tabular surrogate of $62{,}208$ configurations. NASBench-201 contains three architecture-search tasks on CIFAR-10, CIFAR-100, and ImageNet-16-120 over $15{,}625$ candidate CNN architectures (Dong & Yang, 2020). All main-suite tasks use a common budget of 30 objective evaluations. To assess generality, we include LoRA (Hu et al., 2021) fine-tuning of DeBERTa-v3-base (He et al., 2021) on MRPC (Dolan & Brockett, 2005), and kernel-SVM on MagicTelescope (Vanschoren et al., 2014). Details are in Appendix C.1.

Baselines. We compare TUNE-R2 against **11 baselines**, including four LLM-guided methods and seven conventional HPO methods. (i) LLM-based baselines include LMABO (Ngo et al., 2026), AgentHPO (Liu et al., 2025), LLM-HyperOpt (Zhang et al., 2023), and OPRO (Yang et al., 2024), covering LLM-based acquisition selection, agentic HPO, direct history-conditioned proposal generation, and general-purpose LLM optimization. (ii) Conventional baselines include Random Search (Bergstra & Bengio, 2012) and Gaussian-process Bayesian optimization with Expected Improvement (EI) (Jones et al., 1998), Probability of Improvement (PI) (Kushner, 1964), Upper Confidence Bound (UCB) (Srinivas et al., 2010), Lower Confidence Bound (LCB) (Srinivas et al., 2010), Knowledge Gradient (KG) (Frazier et al., 2008), and entropy-based acquisition (Entropy) (Hennig & Schuler, 2012). Implementation details and baseline configurations are provided in Appendix C.2.

Experimental Protocol and Metrics. For the 17-task main suite, we run three random seeds and report Regret@30, AOC@30, mean rank, and win rate as the main optimization metrics. Normalized regret is computed from the task-level best and worst objective values observed across compared methods, with direction adjusted for minimization and maximization tasks; AOC@30 averages normalized regret over the optimization trajectory, and mean rank is computed from Regret@30. For reasoning efficiency, we report Tokens@Best, the cumulative LLM tokens consumed before reaching the best incumbent of each run, together with Token Ratio normalized by TUNE-R2. Unless otherwise stated, LLM-guided methods use Qwen3-32B (Yang et al., 2025a); DeepSeek-V4.0-Flash and DeepSeek-V4.0-Pro (DeepSeek-AI, 2026) are used in robustness studies. Further details on evaluation protocols, metrics, backbone settings, and hardware are provided in Appendix D.

4.2 Overall Results

Overall Comparison. Table 1 summarizes optimization quality and reasoning efficiency under the common 30-evaluation budget across the 17-task main suite. TUNE-R2 achieves the strongest overall optimization performance, obtaining the lowest Regret@30 (2.50×10^{-2}) and AOC@30 (5.19×10^{-2}), the best mean rank (3.06), and the highest win rate (35.3%). Its per-task ranks range from 1 to 7, showing broad competitiveness across heterogeneous tabular HPO, neural HPO, and architecture-search tasks with substantially different search spaces and objective landscapes across benchmarks. Importantly, the gains extend beyond existing LLM-guided methods: TUNE-R2 also improves over all conventional HPO baselines in both final regret and AOC. This suggests that amortizing LLM reasoning does not trade away optimization quality for lower inference cost.

Table 1: **Performance and reasoning efficiency under a 30-evaluation budget.** Regret@30 and AOC@30 are scaled by 10^2; parentheses show Q1–Q3 for Regret@30 and min–max for Rank. Tokens@Best is in thousands; Ratio is relative to TUNE-R2. Lower is better except Win Rate.

Method	Regret@30↓	AOC@30↓	Rank↓	Tokens@Best(K)↓	Ratio↓	Win Rate(%)↑
Conventional HPO methods						
RS (Bergstra & Bengio, 2012)	3.53 (0.07–4.76)	5.61	5.53 (1–10)	–	–	13.3
GP-EI (Jones et al., 1998)	3.18 (0.07–3.82)	5.41	5.23 (1–11)	–	–	7.7
GP-UCB (Srinivas et al., 2010)	3.29 (0.08–4.76)	5.56	4.00 (1–8)	–	–	15.4
GP-LCB (Srinivas et al., 2010)	4.40 (0.11–5.56)	6.53	8.77 (6–12)	–	–	0.0
GP-PI (Kushner, 1964)	4.43 (0.12–4.76)	6.57	6.46 (1–11)	–	–	7.7
GP-Entropy (Hennig & Schuler, 2012)	3.44 (0.08–4.70)	5.47	5.69 (1–12)	–	–	15.4
GP-KG (Frazier et al., 2008)	3.21 (0.05–4.25)	5.59	4.60 (1–11)	–	–	13.3
LLM-guided HPO methods						
LMABO (Ngo et al., 2026)	4.83 (1.07–6.04)	7.81	7.12 (2–12)	163.4	3.1	0.0
AgentHPO (Liu et al., 2025)	4.66 (0.67–5.77)	5.99	7.35 (1–12)	323.7	6.2	5.9
LLM-HyperOpt (Zhang et al., 2023)	4.36 (0.16–4.96)	6.40	6.06 (1–12)	752.1	14.4	11.8
OPRO (Yang et al., 2024)	7.05 (1.81–9.28)	8.39	9.12 (3–12)	340.0	6.5	0.0
TUNE-R2	**2.50 (0.16–3.68)**	**5.19**	**3.06 (1–7)**	**52.3**	**1.0**	**35.3**

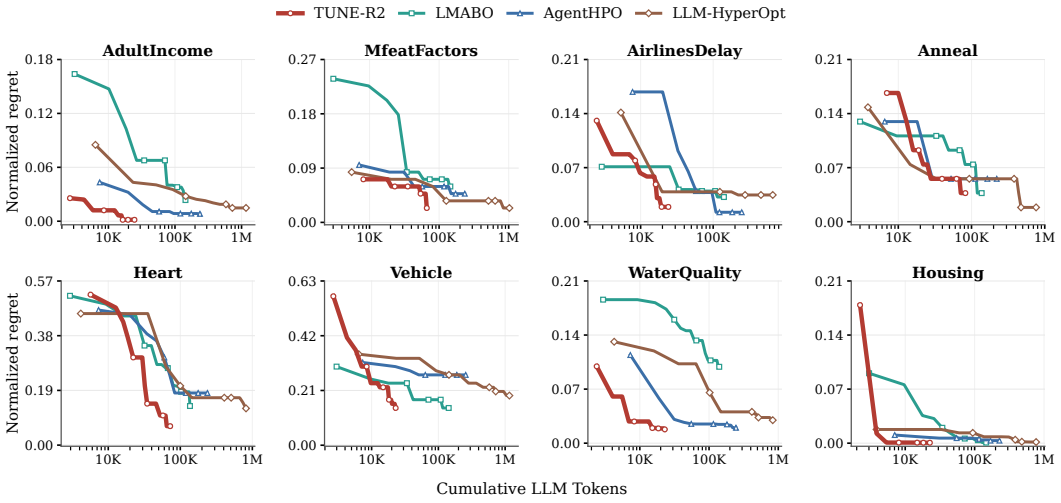

Figure 3: **Optimization quality versus cumulative reasoning cost on eight real-world tasks.** Normalized regret versus cumulative LLM tokens over the first 20 evaluations on a logarithmic x-axis; trajectories closer to the lower-left indicate better optimization quality with less LLM reasoning.

Reasoning Efficiency and Dynamics. TUNE-R2 achieves these gains with substantially less LLM reasoning, requiring only 52.3K Tokens@Best, compared with 163.4K for LMABO, 323.7K for AgentHPO, 340.0K for OPRO, and 752.1K for LLM-HyperOpt. This corresponds to $3.1\times$–$14.4\times$ lower token usage, or a 68%–93% reduction relative to the evaluated LLM baselines. Figure 3 further shows that this advantage persists throughout optimization rather than at the final incumbent. Across the eight real-world HPO tasks, TUNE-R2 reaches low-regret regions with substantially smaller cumulative reasoning budgets, with particularly rapid progress on AdultIncome, MfeatFactors, Heart, Vehicle, WaterQuality, and Housing. Although AirlinesDelay and Anneal exhibit less uniform trajectories, TUNE-R2 remains competitive under modest token budgets. Together, the endpoint and trajectory results show that state-adaptive policy reuse improves optimization progress per unit of LLM reasoning rather than merely reducing the number of LLM calls.

4.3 Understanding Amortized Policy Reuse

We analyze four mechanisms of amortized policy reuse in isolation: (1) bounded-state reasoning, (2) structured induction, (3) adaptive lifetime, and (4) candidate refinement.

Bounded-State Reasoning. We first test whether restricting the LLM to a bounded search state loses useful information from the optimization trajectory or instead removes redundant context. At

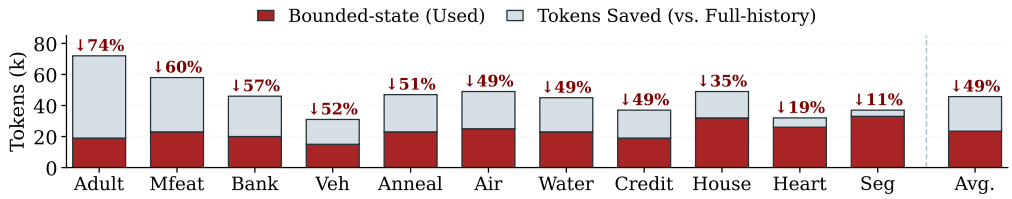

Figure 4: **Token savings from bounded-state reasoning.** Total LLM token usage with full-history and bounded-state reasoning across the 11 OpenML tasks.

matched checkpoints on FCNet and NASBench-201, we compare the default bounded state with an otherwise identical condition that exposes the full raw history, while keeping the search state, policy-refresh procedure, and evaluation protocol unchanged. As shown in Table 2, appending the raw history increases Queue Regret by 6.8% and reduces key-state reference density by 54.0%. Thus, providing more trajectory context does not necessarily improve policy induction; redundant historical observations can dilute the decision-relevant signals already summarized by the bounded state.

Figure 4 further shows the complementary efficiency effect across the 11 OpenML tasks. Replacing the growing history with the bounded state reduces total LLM usage from 473.3K to 244.5K tokens, corresponding to a 48.3% reduction overall. The reduction reaches 74% on AdultIncome and exceeds 50% on several tasks. These results show that bounded-state reasoning both concentrates decision-relevant information and prevents reasoning cost from growing with the optimization trajectory.

Table 2: **Effect of bounded-state reasoning.** Only history changes.

State Input	Queue Regret ↓	Ref. Density ↑
Bounded State	**0.3921**	**1.000**
Bounded + Full Hist.	0.4187	0.460

Table 3: **Effect of policy induction (11 tasks).** Lower rank is better; higher MRR, Rank-1, and W/T/L are better.

Method	Mean Rank ↓	MRR ↑	#Rank-1 ↑	W/T/L
w/o Induction	3.09	0.45	2	–
TUNE-R2	**2.18**	**0.68**	**6**	**5/6/0**

Structured Policy Induction. We next test whether converting the bounded state into structured search guidance improves subsequent decisions. Across 11 real-world HPO tasks, policy induction reduces mean rank from 3.09 to 2.18, improves MRR from 0.45 to 0.68, and increases rank-1 tasks from 2 to 6 (Table 3). It improves rank on five tasks, ties on six, and degrades none. These results support reflection as a useful policy-induction step rather than merely an additional LLM call.

Adaptive Policy Lifetime. We vary how long each induced policy is reused before incorporating new feedback. Table 4 compares the adaptive horizon with fixed $h \in \{1, 3, 5\}$ under the same 30-evaluation budget. Adaptive reuse achieves the lowest final regret and AOC while reducing LLM calls by 70.7% and tokens by 64.4% relative to per-evaluation reasoning ($h = 1$). Although $h = 5$ is cheaper, it yields worse optimization quality, showing that the benefit comes from adapting policy lifetime rather than maximizing reuse. Qualitative trajectories are provided in Appendix G.5.

Table 4: **Effect of adaptive policy lifetime.** Lower is better for all metrics.

Policy Horizon	Regret ↓	AOC ↓	Calls ↓	Tokens ↓
Adaptive	**0.0208**	**0.0397**	17.6	14.1K
$h = 1$	0.0444	0.0838	60.0	39.6K
$h = 3$	0.0305	0.0518	20.0	15.5K
$h = 5$	0.0279	0.0469	**12.0**	**11.1K**

Table 5: **Effect of candidate refinement.** Lower Regret is better.

Queue Construction	Regret ↓
Direct first-h	0.4306
Validation only	0.4286
Full refinement	**0.4187**

Candidate Refinement. Finally, we isolate how policy guidance is converted into an executable queue. Holding the search state, guidance, proposals, and horizon fixed, Table 5 shows that full refinement reduces Queue Regret from 0.4306 to 0.4187, a 2.8% improvement over direct first-h. This shows that reliable reuse depends on selecting a valid, policy-consistent subset rather than merely generating multiple candidates before spending objective evaluations on the candidate queue.

4.4 Generality and Robustness

Beyond the Main HPO Suite. We first test whether TUNE-R2 transfers beyond the tabular HPO/NAS settings used in the main evaluation. Tables 6 and 7 evaluate two different HPO settings using the same task-agnostic prompting interface without task-specific prompt engineering. On DeBERTa-v3-base LoRA tuning, TUNE-R2 achieves the highest accuracy at all optimization stages, reaching 88.85% final accuracy while requiring only 32.0 minutes, outperforming both the GP-BO baselines and AgentHPO. On kernel-SVM tuning over MagicTelescope, AgentHPO slightly leads at three evaluations, while TUNE-R2 attains the highest AUC by seven evaluations (92.97%), the best final AUC (93.48%), and the shortest search time (3.54 minutes). These results suggest that amortized policy reuse transfers across different model families, hyperparameter spaces, and optimization objectives while retaining its optimization–efficiency advantage beyond the main benchmark suite.

Table 6: **LoRA tuning on MRPC.** DeBERTa-v3-base across optimization stages.

Method	Acc.@3	Acc.@7	Final	Time
GP-EI	87.75	87.81	88.24	57.5 m
GP-LCB	87.62	88.05	88.60	60.0 m
GP-UCB	87.75	87.87	88.66	40.0 m
BO Avg.	87.71	87.91	88.50	52.5 m
AgentHPO	87.25	87.50	87.50	43.01 m
TUNE-R2	**88.48**	**88.48**	**88.85**	**32.0 m**

Table 7: **Kernel-SVM on MagicTelescope.** AUC and search duration across baselines.

Method	AUC@3	AUC@7	Final	Time
GP-EI	91.03	91.96	92.89	21.04 m
GP-LCB	88.44	92.47	93.37	5.56 m
GP-UCB	91.03	91.96	92.96	11.54 m
BO Avg.	90.17	92.13	93.07	12.71 m
AgentHPO	**91.62**	92.28	93.13	9.78 m
TUNE-R2	91.48	**92.97**	**93.48**	**3.54 m**

Prompt-Interface Robustness. We test whether the gains of TUNE-R2 can be explained by differences in prompt interface rather than the optimization strategy itself. We harmonize AgentHPO with the same task description, search-space format, optimization direction, budget, JSON schema, and decoding settings used by TUNE-R2, while retaining AgentHPO's original history representation and single-step Creator–Executor loop. As shown in Table 8, harmonization produces small and task-dependent changes across five tasks, improving two while slightly degrading three. This suggests that differences in generic prompt interface alone do not account for the observed performance gap, supporting the role of the underlying search-policy design. Complete prompt templates, output schemas, and the prompt-control protocol are provided in Appendix E.

Table 8: **Cross-method prompt control.**

Task	Original	Harmonized	Δ (10^{-3})
AdultIncome	0.93042	0.93030	-0.12
NASBench-201	0.93627	0.94177	$+5.50$
WaterQuality	0.57304	0.57659	$+3.55$
HousingPrice	0.57108	0.57064	-0.44
AirlinesDelay	0.65398	0.65322	-0.76

Across LLM Backbones. We next examine whether the optimization–reasoning advantage of TUNE-R2 depends on the underlying LLM. Table 9 reports normalized Regret@30 and online LLM cost on four representative tasks from diverse domains: AdultIncome, Vehicle, FCNet protein structure, and NASBench-201 CIFAR-10. Across Qwen3-32B, DeepSeek-V4-Flash, and DeepSeek-V4-Pro, TUNE-R2 achieves the lowest regret and the lowest cost among the evaluated LLM-guided methods. Relative to TUNE-R2, competing methods incur $1.7\times$–$24.2\times$ higher cost under the same backbone. These results show that the optimization–reasoning advantage of amortized policy reuse persists across different reasoning models rather than relying on a particular LLM backbone.

4.5 When Does Policy Reuse Help?

Task-Dependent Behavior. To examine when amortized policy reuse helps, we compare tasks with the largest favorable and less-favorable gaps to the strongest competitor (Table 10). TUNE-R2 performs well on FCNet-Protein and NASBench-201 CIFAR-10, but is outperformed on Segment and WaterQuality. These cases suggest that policy reuse works best when search guidance remains informative across several evaluations, but weakens when fresher feedback matters more. This reflects the trade-off between reasoning amortization and feedback freshness.

Table 9: **Robustness across LLM backbones.** Average normalized Regret@30 and online LLM cost over four representative tasks. Cost Ratio is normalized by TUNE-R2 within each backbone.

Method	Qwen3-32B			DeepSeek-V4.0-Flash			DeepSeek-V4.0-Pro		
	Regret↓	Cost($)↓	Ratio↓	Regret↓	Cost($)↓	Ratio↓	Regret↓	Cost($)↓	Ratio↓
LMABO	0.7231	0.11	4.9×	0.6079	0.04	2.1×	0.6069	0.57	1.7×
AgentHPO	0.6739	0.22	9.8×	0.6006	0.14	7.8×	0.5971	1.98	5.8×
LLM-HyperOpt	0.7249	0.54	24.2×	0.6053	0.18	10.1×	0.6047	3.47	10.2×
OPRO	0.8797	0.23	11.3×	0.6147	0.17	8.4×	0.6104	2.11	6.2×
TUNE-R2	**0.6553**	**0.02**	**1.0×**	**0.5950**	**0.02**	**1.0×**	**0.5923**	**0.34**	**1.0×**

Table 10: **Representative favorable and less-favorable cases.** Comparison between TUNE-R2 and the top-performing baseline on original task metrics. Best results are **bolded**.

Regime	Task	TUNE-R2	Best Competitor	Competitor Score
Favorable	FCNet-Protein (↓)	**0.2424 ± 0.0084**	LMABO	0.2480 ± 0.0079
	NASBench-201 (↑)	**0.9425 ± 0.0017**	GP-PI	0.9404 ± 0.0032
Less favorable	Segment (↑)	0.9784 ± 0.0018	LLM-HyperOpt	**0.9827 ± 0.0000**
	WaterQuality (↑)	0.5790 ± 0.0052	GP-EI	**0.5820 ± 0.0027**

Dynamics against traditional acquisition functions. On NASBench-201, we further compare TUNE-R2 with traditional acquisition-based BO baselines under the same evaluation budget. Figure 5 shows this behavior across CIFAR-10, CIFAR-100, and ImageNet16-120. On CIFAR-10, TUNE-R2 drops below 0.02 normalized regret within the first five evaluations and reaches 0.0076 at trial 20, compared with 0.0134 for GP-EI and 0.0141 for GP-UCB. On CIFAR-100, TUNE-R2 reaches 0.0230 at trial 20, matching the best GP baseline while staying below most acquisition functions after the early search phase. On ImageNet16-120, TUNE-R2 attains the lowest trial-20 regret among the compared methods, reaching 0.2638 versus 0.2679 for GP-EI. These trends indicate that TUNE-R2 improves early-budget search decisions relative to fixed acquisition rules.

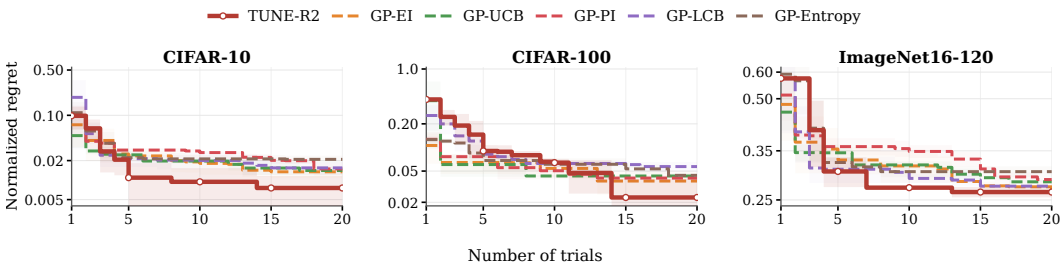

Figure 5: **Optimization dynamics against standard GP acquisition functions on NASBench-201.** Mean normalized regret with standard-error bands across runs; lower is better.

5 Conclusion

We introduced TUNE-R2, an amortized search policy framework for LLM-guided hyperparameter optimization. Rather than invoking the LLM after every objective evaluation, TUNE-R2 induces reusable search guidance from a bounded search state, refines it into an executable candidate queue, and adaptively controls how many evaluations reuse each policy before incorporating new feedback. Across OpenML, FCNet, and NASBench-201, TUNE-R2 achieves the best aggregate optimization performance in our evaluation while requiring 68%–93% fewer Tokens@Best than existing LLM-guided HPO baselines. Additional experiments on LoRA and kernel-SVM tuning, multiple LLM backbones, and controlled component analyses further support the generality of the approach and its underlying design choices. Overall, our results highlight a simple but important broader principle for LLM-guided optimization: reasoning need not be coupled one-to-one with actions, but can instead be amortized through reusable policies whose lifetimes adapt to the value of fresh feedback.

AI USE STATEMENT

LLMs are used in an auxiliary capacity to assist with literature search, grammar polishing, and code debugging.

ETHICS STATEMENT

This work studies hyperparameter optimization using public benchmarks and models and involves no human participants or private data. Practical deployment of LLM-guided optimization should account for the computational and monetary costs of model inference and objective evaluation.

REPRODUCIBILITY STATEMENT

We provide the complete optimization procedure in Appendix A, benchmark and baseline details in Appendix C, evaluation protocols and compute settings in Appendix D, and the full prompt templates and output schemas in Appendix E. Appendix G further provides a concrete end-to-end run illustrating the bounded search state, policy induction, candidate refinement, and run-level behavior.

REFERENCES

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL https://jmlr.org/papers/v13/bergstra12a.html.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011. URL https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.

Dean De Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro, 2026. Technical report and model card, accessed 2026-05-03.

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the third international workshop on paraphrasing (IWP2005)*, 2005.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1437–1446, 2018.

Peter I Frazier, Warren B Powell, and Savas Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(57):1809–1837, 2012. URL https://jmlr.csail.mit.edu/papers/v13/hennig12a.html.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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. doi: 10.1023/A:1008306431147.

Aditya Kadiwal. Water quality / drinking water potability dataset. https://www.kaggle.com/datasets/adityakadiwal/water-potability, 2020.

Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. Automl to date and beyond: Challenges and opportunities. *Acm computing surveys (csur)*, 54(8):1–36, 2021.

Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

Harold J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964. doi: 10.1115/1.3653121.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, 18(185):1–52, 2018.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Siyi Liu, Chen Gao, and Yong Li. Agenthpo: Large language model agent for hyper-parameter optimization. In *Conference on Parsimony and Learning*, pp. 1146–1169. PMLR, 2025.

Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

Giang Ngo, Dat Phan Trong, Dang Nguyen, Sunil Gupta, and Svetha Venkatesh. Adaptive acquisition selection for bayesian optimization with large language models. *arXiv preprint arXiv:2602.07904*, 2026.

OpenML. Airlines dataset. https://www.openml.org/search?type=data&id=42493, 2020. Source: Data Expo competition 2009.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International conference on machine learning*, pp. 31210–31227. PMLR, 2023.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015–1022, 2010. URL https://arxiv.org/abs/0912.3995.

Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, et al. Automl in the age of large language models: Current challenges, future opportunities and risks. *TRANSACTIONS ON MACHINE LEARNING RESEARCH*, pp. 1–31, 2024.

UCI Machine Learning Repository. Relative location of ct slices on axial axis. https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis, a.

UCI Machine Learning Repository. Multiple features dataset. https://archive.ics.uci.edu/dataset/72/multiple+features, b.

UCI Machine Learning Repository. Condition based maintenance of naval propulsion plants. https://archive.ics.uci.edu/ml/datasets/Condition+Based+Maintenance+of+Naval+Propulsion+Plants, c.

UCI Machine Learning Repository. Physicochemical properties of protein tertiary structure. https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure, d.

UCI Machine Learning Repository. Statlog heart dataset. https://archive.ics.uci.edu/dataset/145/statlog+heart, e.

UCI Machine Learning Repository. Statlog vehicle silhouettes dataset. https://archive.ics.uci.edu/dataset/149/statlog+vehicle+silhouettes, f.

UCI Machine Learning Repository. Annealing dataset. https://archive.ics.uci.edu/dataset/3/annealing, 1990a.

UCI Machine Learning Repository. Image segmentation dataset. https://archive.ics.uci.edu/ml/datasets/Image+Segmentation, 1990b.

UCI Machine Learning Repository. Statlog german credit data. https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data, 1994.

UCI Machine Learning Repository. Adult / census income dataset. https://archive.ics.uci.edu/dataset/2/adult, 1996. Also known as Census Income.

UCI Machine Learning Repository. Bank marketing dataset. https://archive.ics.uci.edu/dataset/222/bank+marketing, 2012.

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.

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

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*, 2024.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang, and Liangming Pan. How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13340–13358, 2025b.

Michael R Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

A PSEUDO-CODE OF TUNE-R2

Algorithm 1 TUNE-R2: Amortized Search Policies for LLM-Guided HPO

Require: Search space $\mathcal{X}$, objective f, evaluation budget T, horizon range $[h_{\min}, h_{\max}]$
Ensure: Best observed configuration $x^\star$

Convert f to the maximization convention if necessary, and denote the resulting objective by $\tilde{f}$
Initialize history $\mathcal{H} \leftarrow \emptyset$, reflection memory $m \leftarrow \emptyset$, and incumbent $(x^\star, y^\star) \leftarrow (\text{None}, -\infty)$
Initialize evaluation index $t \leftarrow 1$ and policy index $k \leftarrow 1$
while $t \leq T$ **do**
  {**Policy Refresh**}
  Construct bounded state $S_k \leftarrow \mathcal{A}(\mathcal{H}, x^\star, m)$
  $(\pi_k, \hat{h}_k) \leftarrow \text{Reflect}_{\text{LLM}}(S_k, \mathcal{X})$ {LLM call 1}
  Extract progress signals ν_k and search regime ω_k from S_k
  if $\hat{h}_k$ is valid **then**
    $h_k \leftarrow \Pi_{[h_{\min}, h_{\max}]}(\hat{h}_k)$
  else
    $h_k \leftarrow h_{\text{prior}}(\nu_k, \omega_k)$
  end if
  $h_k \leftarrow \min(h_k, T - t + 1)$ {respect remaining evaluation budget}
  $U_k \leftarrow \text{Refine}_{\text{LLM}}(S_k, \pi_k, h_k, \mathcal{X})$ {LLM call 2}
  Validate proposals in U_k against $\mathcal{X}$
  Remove invalid and duplicate configurations
  Rank valid candidates using proposal preference scores and diversity-aware novelty
  $Q_k \leftarrow \text{Top}_{h_k}(U_k)$
  Update bounded reflection memory m using π_k
  {**Policy Execution**}
  for $j = 1, \ldots, |Q_k|$ **do**
    Pop next configuration x_t from Q_k
    Evaluate $y_t \leftarrow \tilde{f}(x_t)$
    Update $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x_t, y_t)\}$
    if $y_t > y^\star$ **then**
      $(x^\star, y^\star) \leftarrow (x_t, y_t)$
    end if
    Update trajectory and progress statistics with (x_t, y_t)
    $t \leftarrow t + 1$
  end for
  $k \leftarrow k + 1$
end while
return $x^\star$

B EXISTING ASSETS AND LICENSES

Table 11: Existing assets used in this work.

Asset	Source/Citation	License/Terms
DeepSeek-V4	DeepSeek-AI (2026)	MIT License
Qwen3-32B	Yang et al. (2025a)	Apache License 2.0
OpenML datasets	Vanschoren et al. (2014)	Dataset-specific licenses reported by OpenML metadata
NAS-Bench-201	Dong & Yang (2020)	MIT License
FCNet tabular benchmarks	Klein & Hutter (2019)	BSD 3-Clause License

C Detail of Benchmarks and Baselines

C.1 Benchmarks

Real-World HPO Tasks. The real-world HPO suite consists of 11 supervised learning tasks: `AdultIncome`, `AirlinesDelay`, `Anneal`, `Bank`, `CreditG`, `Heart`, `HousingPrice`, `MfeatFactors`, `Segment`, `Vehicle`, and `WaterQuality`. Each task corresponds to hyperparameter optimization for a fixed tabular learning pipeline, where the optimizer selects model hyperparameters and receives the resulting validation performance. Unlike synthetic black-box functions, these tasks are derived from real datasets and therefore contain dataset-specific feature distributions, class imbalance, mixed categorical and numerical attributes, and non-smooth interactions between preprocessing choices and model hyperparameters. Table 12 summarizes the data source, scale, and prediction objective of each task.

FCNet Benchmarks. We further evaluate on three FCNet surrogate tasks: `FCNet_naval_propulsion`, `FCNet_protein_structure`, and `FCNet_slice_localization`. FCNet is a tabular benchmark for neural hyperparameter optimization in which each entry records the validation and test behavior of a two-hidden-layer fully connected neural network trained on a real UCI regression dataset (Klein & Hutter, 2019; Eggensperger et al., 2021). The benchmark is computationally lightweight because each query can be answered from precomputed tabular evaluations rather than by retraining a neural network from scratch. At the same time, it preserves realistic neural HPO structure through interactions among optimizer settings, learning-rate schedules, activation functions, hidden-layer widths, and dropout rates. The resulting configuration space contains $6 \times 4 \times 2 \times 2 \times 2 \times 6 \times 6 \times 3 \times 3 = 62{,}208$ candidate configurations. Table 13 lists the underlying regression datasets, and Table 14 summarizes the FCNet search space.

NASBench-201 Benchmarks. Finally, we evaluate on NASBench-201 using the three standard image-classification datasets: `CIFAR-10`, `CIFAR-100`, and `ImageNet16-120`. NASBench-201 defines a cell-based architecture search space in which each candidate cell is a directed acyclic graph with four nodes. The six directed edges are independently assigned one of five candidate operations: `none`, `skip_connect`, `conv_1x1`, `conv_3x3`, or `avg_pool_3x3`. This gives $5^6 = 15{,}625$ possible architectures. Because the training, validation, and test performance of all architectures are precomputed, this benchmark enables controlled comparison of search trajectories without confounding differences from implementation details or training noise. Table 16 summarizes the three NASBench-201 tasks.

C.2 Baselines

LLM-based baselines. We include four online LLM-based HPO baselines. **LMABO** (Ngo et al., 2026) uses the LLM as an adaptive acquisition-function selector. We first fit the same GP surrogate as in the non-LLM BO baselines, then provide the LLM with a compact optimization-state summary, including the incumbent value, recent improvement trend, remaining budget, and posterior uncertainty statistics. The LLM selects one acquisition function from $\{\mathrm{EI}, \mathrm{PI}, \mathrm{UCB}, \mathrm{LCB}, \mathrm{KG}, \mathrm{Entropy}\}$, and the next configuration is obtained by optimizing the selected acquisition.

AgentHPO (Liu et al., 2025) is implemented as an agentic HPO loop. The agent reads the task description and trial history, summarizes promising and poor-performing regions, and proposes the next configuration. To ensure comparability, the agent is only allowed to select configurations within the predefined search space and cannot modify the training pipeline or evaluation protocol.

LLM-HyperOpt (Zhang et al., 2023) directly prompts the LLM to generate the next configuration from the dataset description, search space, objective, and previous trials. Unlike LMABO, it does not use an explicit surrogate model or hand-designed acquisition function.

OPRO adapts OPRO (Yang et al., 2024) to hyperparameter optimization. Each prompt lists previously evaluated configurations and their validation scores, and the LLM is asked to generate a new configuration expected to improve over the current best. This provides a generic prompt-optimization baseline without HPO-specific surrogate modeling.

Table 12: Real-world HPO tasks used in our benchmark suite. For classification tasks, the optimization objective is to maximize validation performance; for the regression task, the objective is to minimize validation error or equivalently maximize the chosen validation score.

Task	Data Domain	Dataset Statistics	Prediction Objective
`AdultIncome` (UCI Machine Learning Repository, 1996)	Census-income records	48,842 instances with 14 mixed demographic and employment attributes	Binary classification of whether annual income exceeds $50K
`AirlinesDelay` (OpenML, 2020)	Commercial flight records	539,383 flight instances with 8 scheduling and route-related attributes	Binary classification of whether a flight is delayed
`Anneal` (UCI Machine Learning Repository, 1990a)	Steel annealing process	898 steel-treatment records with mixed process and material attributes	Multi-class classification of annealing outcome class
`Bank` (Moro et al., 2014; UCI Machine Learning Repository, 2012)	Marketing campaign data	45,211 client records with 16 mixed demographic and financial attributes	Binary classification of whether client subscribes to term deposit
`CreditG` (UCI Machine Learning Repository, 1994)	Credit-risk assessment	1,000 credit applications with 20 mixed categorical/numerical attributes	Binary classification of good vs. bad credit risk
`Heart` (UCI Machine Learning Repository, e)	Clinical medical records	270 patient records with 13 clinical attributes	Binary classification of heart-disease presence
`HousingPrice` (De Cock, 2011)	Residential property sales	1,460 houses from Ames benchmark with 79 explanatory variables	Regression of final sale price
`MfeatFactors` (UCI Machine Learning Repository, b)	Handwritten digits	2,000 digit samples, 200 per class, with 216 correlation features	10-class digit classification
`Segment` (UCI Machine Learning Repository, 1990b)	Outdoor image segmentation	2,310 image regions represented by 19 numerical region descriptors	7-class image-region classification
`Vehicle` (UCI Machine Learning Repository, f)	Vehicle silhouette images	846 silhouettes represented by 18 shape descriptors	4-class vehicle-type classification
`WaterQuality` (Kadiwal, 2020)	Drinking-water quality	3,276 water samples with 9 physicochemical measurements	Binary classification of potable vs. non-potable water

Conventional HPO and BO baselines. Random Search (Bergstra & Bengio, 2012) samples each configuration independently from the task-specific search space. Continuous variables are sampled uniformly from their valid ranges unless a log scale is specified, and categorical variables are sampled uniformly from their candidate sets.

For GP-based BO baselines, we fit a Gaussian-process surrogate to $\mathcal{D}_t$ at each iteration. Let $\mu_t(\mathbf{x})$ and $\sigma_t(\mathbf{x})$ denote the posterior mean and standard deviation, and let $y_t^+ = \max_{i \le t} y_i$ be the incumbent best score. All GP baselines share the same input encoding, output normalization, kernel fitting, and acquisition-optimization routine, differing only in the acquisition function.

Table 13: FCNet surrogate tasks used in our benchmark suite. Each task evaluates the hyperparameters of a two-hidden-layer fully connected neural network on a real UCI regression dataset.

Task	Data Domain	Dataset Statistics	Objective
`naval_propulsion` (UCI Machine Learning Repository, c)	Naval-vessel gas-turbine simulation	11,934 simulator records with 16 numerical input features	Minimize validation MSE
`protein_structure` (UCI Machine Learning Repository, d)	Protein tertiary-structure prediction	45,730 protein-decoy records with 9 numerical input features	Minimize validation MSE
`slice_localization` (UCI Machine Learning Repository, a)	CT-slice localization	53,500 CT-image records from 74 patients, represented by 384 extracted image features	Minimize validation MSE

Table 14: Search space of the FCNet benchmark. Each configuration specifies the training hyperparameters and architecture choices of a two-hidden-layer fully connected network.

Hyperparameter	Categorical configuration space
Initial LR	{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1}
Batch size	{8, 16, 32, 64}
LR schedule	{`cosine`, `const`}
Activation, layer 1	{`relu`, `tanh`}
Activation, layer 2	{`relu`, `tanh`}
Layer 1 width	{16, 32, 64, 128, 256, 512}
Layer 2 width	{16, 32, 64, 128, 256, 512}
Dropout, layer 1	{0.0, 0.3, 0.6}
Dropout, layer 2	{0.0, 0.3, 0.6}

Table 15: Search space of NASBench-201. Each architecture is a four-node cell with six directed edges, where each edge independently selects one operation.

Component	Configuration
Cell topology	Directed acyclic graph with 4 nodes
Number of edges	6 directed edges
Candidate operations	`none`, `skip_connect`, `nor_conv_1x1`, `nor_conv_3x3`, or `avg_pool_3x3`
Search-space size	$5^6 = 15{,}625$ architectures
Evaluation metric	Validation accuracy or validation error

EI (Jones et al., 1998) maximizes the expected improvement over y_t^+. **PI** (Kushner, 1964) maximizes the probability of improving over y_t^+. **UCB** (Srinivas et al., 2010) uses $\alpha_{\mathrm{UCB}}(\mathbf{x}) = \mu_t(\mathbf{x}) + \kappa_t \sigma_t(\mathbf{x})$, while **LCB** uses the corresponding lower confidence bound under the objective direction. **KG** (Frazier et al., 2008) selects the point with the largest estimated one-step value of information. **Entropy** (Hennig & Schuler, 2012) selects the point expected to most reduce uncertainty about the optimizer location.

D Experimental Details

Evaluation Protocol. For the 17-task main suite, all methods use a common budget of 30 objective evaluations per task and three random seeds. Each method operates online, conditioning subsequent decisions on previously observed configuration–objective pairs. Additional generality experiments use task-specific budgets: 10 evaluations for DeBERTa-v3-base LoRA fine-tuning on MRPC and 20

Table 16: NASBench-201 tasks used in our benchmark suite. All tasks share the same architecture search space (4 nodes, 6 edges, 5 candidate operations, and 15,625 candidate cells) and evaluate on different image-classification datasets.

Task	Dataset Description & Partitioning	Objective
`CIFAR-10`	60,000 color images (32×32) across 10 classes (50,000 train / 10,000 test)	Maximize validation accuracy
`CIFAR-100`	60,000 color images (32×32) across 100 classes (500 train / 100 test per class)	Maximize validation accuracy
`ImageNet16-120`	Downsampled ImageNet variant with 16×16 images from 120 classes under standard split	Maximize validation accuracy

Table 17: **Prompt placeholders used by TUNE-R2.**

Placeholder	Meaning
`{objective_direction}`	Minimize or maximize
`{objective_name}`	Optimization objective
`{phase}`	Current search regime
`{h_base}`	Horizon proposed by the local prior
`{h_t}`	Validated policy horizon
`{plateau_detected}`	Whether recent progress indicates a plateau
`{search_space}`	Hyperparameter names, types, and valid ranges
`{history_summary}`	Bounded summary of recent search progress
`{best_trace}`	Recent incumbent trajectory
`{best_config}`	Current incumbent configuration
`{reflection_memory}`	Up to four recent structured reflections
`{n_generate}`	Size of the over-complete proposal pool

for kernel-SVM tuning on MagicTelescope. The same task-agnostic TUNE-R2 prompting interface is used across these tasks without task-specific prompt engineering.

Metrics. For the main suite, we report Regret@30, AOC@30, mean rank, and win rate. Normalized regret is computed from the task-level best and worst objective values across compared methods, with direction adjusted for minimization and maximization tasks; AOC@30 averages normalized regret over the 30-trial trajectory. For reasoning efficiency, Tokens@Best measures cumulative LLM tokens consumed before reaching the best incumbent of a run, and Token Ratio is normalized by TUNE-R2. Early-budget behavior is additionally reported over the first 20 trials in the token-efficiency and NASBench-201 analyses.

Backbones and Compute. Unless otherwise stated, LLM-guided methods use Qwen3-32B (Yang et al., 2025a); DeepSeek-V4-Flash and DeepSeek-V4-Pro (DeepSeek-AI, 2026) are used for robustness studies. Experiments were conducted on NVIDIA RTX A6000 GPUs and Intel Xeon Platinum 8358 CPUs. Objective-evaluation and optimizer-side reasoning costs are reported separately.

E Prompt Templates and Interface

Prompt Interface. TUNE-R2 uses a fixed, task-agnostic prompt interface across all optimization tasks. Task-specific information—including the objective, search space, and observed feedback—is instantiated automatically from the bounded search state. Each policy refresh uses exactly two LLM calls: the first induces structured search guidance and recommends a policy horizon, while the second generates an over-complete pool of candidate configurations conditioned on that guidance. Horizon validation, candidate filtering, and evaluation-queue construction are performed locally and require no additional LLM calls.

Table 17 summarizes the placeholders shared by the two LLM calls. The policy-specific fields `important_parameters`, `promising_regions`, `discouraged_regions`, `diversification`, and `action` are populated from the JSON-parsed output of the first call.

E.1 Policy Induction Prompt

The first LLM call maps the bounded search state to reusable structured search guidance. Rather than directly proposing the next configuration, it identifies which hyperparameters and regions should guide the subsequent evaluations and recommends how long this policy should remain active.

System Prompt

```
You are analyzing a hyperparameter search trajectory.

Return ONLY valid JSON with keys:
important_parameters,
promising_regions,
discouraged_regions,
diversification,
action,
recommended_horizon,
rationale.

diversification must be one of
\{low, moderate, high\}.

action must be one of
\{explore, balance, exploit, diversify\}.

recommended_horizon must be an integer.
Response must be under 400 tokens.
```

User Prompt

```
Objective: {objective_direction} {objective_name}
Current phase: {phase}
Base horizon from local prior: {h_base}
Plateau detected: {plateau_detected}
Best trace (recent): {best_trace}
Current best config: {best_config}

Search space:
{search_space}

Recent trials (compressed):
{history_summary}

Previous reflection memory:
{reflection_memory}

Analyze which hyperparameters matter most now, what regions look
    promising
or discouraged, whether the search should diversify or exploit next,
    and
what policy horizon should be used for the next refinement batch.
```

Output Schema

```
{
  "important_parameters": ["<hyperparameter>", "..."],
```

```
  "promising_regions": ["<promising region>", "..."],
  "discouraged_regions": ["<discouraged region>", "..."],
  "diversification": "low | moderate | high",
  "action": "explore | balance | exploit | diversify",
  "recommended_horizon": 3,
  "rationale": "<brief reason>"
}
```

The response is JSON-parsed, with missing fields supplied by heuristic defaults. The recommended horizon is converted to an integer when possible and subsequently validated by the local horizon controller. The resulting structured guidance conditions candidate refinement and is retained in the bounded reflection memory.

E.2 Local Policy-Horizon Selection

Selecting the final policy horizon does not require an additional LLM call. Let $\widehat{h}_k$ denote the horizon recommended by the policy-induction call. If the recommendation is valid, we use

$$h_k = \mathrm{clip}\left(\widehat{h}_k, h_{\min}, h_{\max}\right). \tag{7}$$

Otherwise, TUNE-R2 falls back to a local state-dependent prior determined by the current search regime and recent progress signals. We use $h_{\min} = 1$, $h_{\mathrm{mid}} = 3$, and $h_{\max} = 5$ throughout the experiments. The resulting h_k specifies how many objective evaluations reuse the current search policy before the next policy refresh.

E.3 Candidate Refinement Prompt

The second LLM call realizes the induced policy as an over-complete proposal pool. It conditions jointly on the bounded search state, the structured policy guidance, and the selected horizon, while explicitly encouraging diversity among candidate configurations.

System Prompt

```
You are the refinement module of a black-box hyperparameter optimizer.

Return one JSON object and no additional text.

Return this schema:
{
  "proposals": [
    {
      "config": {"parameter": "valid value"},
      "score": 0.0,
      "reason": "brief reason"
    }
  ]
}

The score is a relative preference score, not an objective prediction.
```

User Prompt

```
Use the bounded state and reflection below to propose exactly
{n_generate} candidate configurations.

Search space:
{search_space}
```

```
Bounded decision state:
{bounded_state}

Reflection guidance:
{reflection}

Return this schema:
{
  "proposals": [
    {
      "config": {"parameter": "valid value"},
      "score": 0.0,
      "reason": "brief reason"
    }
  ]
}

The score is a relative preference score, not an objective prediction.
Include some diversity when the reflection requests exploration
or diversification.
```

Output Schema

```
{
  "proposals": [
    {
      "config": {
        "<hyperparameter_1>": "<value>",
        "<hyperparameter_2>": "<value>"
      },
      "score": 0.0,
      "reason": "<brief reason>"
    }
  ]
}
```

The score is a relative preference under the reflected search policy, rather than a prediction of the black-box objective.

E.4 Local Queue Construction

Candidate generation is followed entirely by local operations. The over-complete pool U_k is processed by search-space validation, deduplication, and diversity-aware ranking using the proposal preference scores returned by the refinement call. The top h_k valid candidates form the executable queue Q_k.

$$U_k \to \text{validation} \to \text{deduplication} \to \text{diversity-aware ranking} \to Q_k. \tag{8}$$

The configurations in Q_k are then evaluated sequentially without further LLM reasoning. Their objective feedback is accumulated into the search state and incorporated when the next policy refresh is triggered.

The complete refresh workflow is therefore

$$S_k \to \text{Reflect}_{\text{LLM}} \to (\pi_k, \widehat{h}_k) \to h_k \to \text{Refine}_{\text{LLM}} \to U_k \to Q_k, \tag{9}$$

where only the Reflect and Refine operations invoke the LLM.

E.5 Cross-Method Prompt Control

The prompt-control experiment in Table 8 tests whether differences in generic prompt interface can account for the performance gap between TUNE-R2 and AgentHPO. Specifically, we harmonize AgentHPO with the task description, search-space representation, optimization direction, evaluation

budget, JSON output format, and decoding settings used by TUNE-R2. We retain AgentHPO's original history representation and single-step Creator–Executor workflow so that the intervention changes the prompt interface without replacing its optimization strategy.

Across the five evaluated tasks, prompt harmonization produces small and non-systematic changes: performance improves on two tasks and slightly degrades on three. This control therefore does not support an explanation based solely on generic prompt formatting or omitted task information.

For the remaining LLM-based baselines, we follow the prompt templates and protocols provided by their corresponding papers or released implementations. TUNE-R2 uses the same task-agnostic prompt templates across all evaluated tasks and does not introduce task-specific prompt engineering.

F Supplementary Results

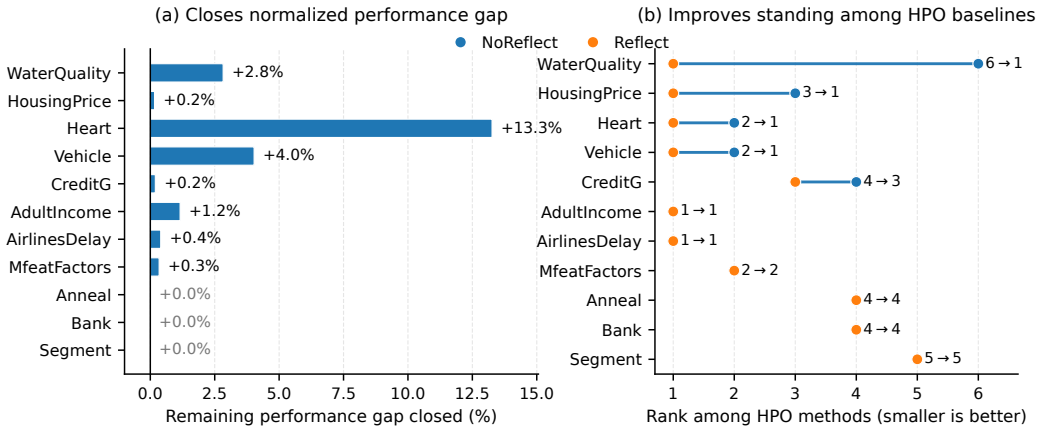

Figure 6: Per-task effect of reflection in TUNE-R2 across 11 real-world tasks. Panel (a) shows the normalized performance gain, computed as the fraction of the remaining gap to perfect performance closed by reflection. Panel (b) shows the rank movement among HPO baselines, where smaller ranks are better. Reflection improves rank on 5 out of 11 tasks and ties on the remaining 6, with no rank degradation. The largest improvement is observed on WaterQuality, where TUNE-R2 improves from rank 6 to rank 1.

Per-task effect of reflection. Figure 6 provides a per-task view of how reflection affects TUNE-R2. Panel (a) reports the normalized performance gain, measured as the fraction of the remaining gap to perfect performance closed by reflection. Panel (b) visualizes the corresponding rank movement among HPO baselines, where smaller ranks are better. Reflection improves the relative rank on 5 out of 11 tasks and ties on the remaining 6, without degrading any task. The largest improvement appears on WaterQuality, where TUNE-R2 moves from rank 6 to rank 1, while HousingPrice improves from rank 3 to rank 1. These per-task trends support the aggregate results in Table 3, showing that reflection provides consistent rank-level benefits rather than isolated gains on a single benchmark.

G AdultIncomeTask Run Schema of the LLM-visible Search State

This appendix provides a concrete run example of the LLM-visible search state used by TUNE-R2 on AdultIncomeTask. The run uses 30 trials with seed 1 and optimizes validation ROC-AUC. The optimizer keeps the full evaluation history internally, but only exposes a compact search state to the LLM during reflection and refinement.

Table 18: AdultIncomeTask run setting.

Item	Value
Task	`AdultIncomeTask`
Objective	ROC-AUC
Direction	Maximize
Method	TUNE-R2
Trial budget	30
Seed	1
Reasoning horizons	$h_{\max} = 5,\ h_{\mathrm{mid}} = 3,\ h_{\min} = 1$
Final best value	0.930851
Total LLM calls	16
Total tokens	23,636

Table 19: AdultIncomeTask search space.

Hyperparameter	Type	Range
`n_estimators`	integer	$[100, 500]$
`max_depth`	integer	$[3, 11]$
`learning_rate`	float	$[0.001, 1.0]$
`subsample`	float	$[0.5, 1.0]$
`colsample_bytree`	float	$[0.5, 1.0]$
`min_child_weight`	integer	$[1, 10]$
`gamma`	float	$[0.0, 0.5]$
`reg_alpha`	float	$[0.0, 1.0]$
`reg_lambda`	float	$[0.0, 1.0]$
`scale_pos_weight`	float	$[1.0, 10.0]$

G.1 AdultIncomeTask Bounded Search State

We provide a concrete snapshot of the bounded search state used by TUNE-R2 on AdultIncomeTask. The snapshot is taken after 24 objective evaluations in the 30-trial run described in Table 18. Although the complete evaluation history is retained internally for bookkeeping and state updates, the LLM observes only the fixed-format state $S_k = (\xi_k, b_k, \nu_k, \omega_k, m_k)$ summarized in Table 20. Thus, increasing the number of completed evaluations does not require replaying the full raw trajectory to the LLM.

Table 20: **Concrete bounded search state on AdultIncomeTask.** The snapshot is taken after 24 evaluations and contains only the information exposed to the LLM at the next policy refresh.

State Component	**Content**
Trajectory sketch ξ_k	$n = 24$; recent ROC-AUC $= [0.926, 0.917, 0.925]$; trend $= -0.0008$/trial; best $= 0.9305$
Incumbent b_k	best ROC-AUC $= 0.9305$; `n_estimators`$= 450$, `max_depth`$= 6$, `learning_rate`$= 0.028$, `subsample`$= 0.85$, `colsample_bytree`$= 0.55$, `min_child_weight`$= 5$, `gamma`$= 0.05$, `reg_alpha`$= 0.9$, `reg_lambda`$= 0.9$, `scale_pos_weight`$= 1.5$
Progress signals ν_k	trend $= -0.0008$/trial; plateau = true
Search regime ω_k	phase = late exploitation; direction = maximize
Reflection memory m_k	important hyperparameters = [`learning_rate`, `n_estimators`, `max_depth`]; promising = higher `n_estimators`, moderate `max_depth` (5–8), lower `learning_rate` (0.01–0.1); discouraged = very low `n_estimators`, extreme `max_depth`, very high `learning_rate`; previous action = exploit; previous policy horizon = 5

This example illustrates the role of bounded-state reasoning in TUNE-R2. The raw history is compressed into a fixed set of trajectory, incumbent, progress, regime, and memory fields before each policy refresh. The resulting state preserves decision-relevant information for policy induction while preventing the LLM-visible trajectory representation from growing with the optimization history.

G.2 AdultIncomeTask Reflection Output

Given the bounded search state in Table 21, the reflection module returns structured policy guidance rather than directly emitting a single configuration.

This reflection illustrates how TUNE-R2 uses the LLM as a trajectory-level reasoning module. The LLM summarizes which regions should be exploited or avoided and provides control signals for refinement, instead of simply emitting one isolated configuration.

Table 21: Canonical reflection output from TUNE-R2 on AdultIncomeTask.

Field	Value
Important parameters	learning_rate, n_estimators, max_depth
Promising regions	higher n_estimators; moderate max_depth (5–8); lower learning_rate (0.01–0.1)
Discouraged regions	very low n_estimators; extreme max_depth; very high learning_rate
Diversification	`low`
Action	`exploit`
Recommended horizon	5
Rationale	The current best configuration shows a plateau and recent trials indicate diminishing returns. The most impactful hyperparameters remain learning_rate, n_estimators, and max_depth. Continuing exploitation within a moderate horizon allows refinement without overcommitting to unpromising regions.

G.3 AdultIncomeTask Refinement Output

The refinement step expands the reflection into an over-complete lookahead proposal set and then selects a smaller evaluation queue. In the AdultIncomeTask run, a typical late-stage refinement requests 17 mutually diverse proposals and keeps 5 candidates in the queue. Table 22 shows representative queued candidates.

Table 22: **AdultIncomeTask refinement output.** The proposal set contains 17 candidates; TUNE-R2 keeps 5 after validation, deduplication, and diversity-aware ranking (top 3 shown for brevity).

Rank	Candidate Configuration
1	`n_estimators`=450, `max_depth`=6, `learning_rate`=0.02, `subsample`=0.8, `colsample_bytree`=0.9, `min_child_weight`=4, `gamma`=0.1, `reg_alpha`=0.3, `reg_lambda`=0.7, `scale_pos_weight`=3.0
2	`n_estimators`=480, `max_depth`=7, `learning_rate`=0.05, `subsample`=0.9, `colsample_bytree`=0.7, `min_child_weight`=6, `gamma`=0.2, `reg_alpha`=0.5, `reg_lambda`=0.5, `scale_pos_weight`=5.0
3	`n_estimators`=420, `max_depth`=5, `learning_rate`=0.01, `subsample`=1.0, `colsample_bytree`=0.8, `min_child_weight`=4, `gamma`=0.02, `reg_alpha`=0.15, `reg_lambda`=0.85, `scale_pos_weight`=1.5

The proposal set is intentionally larger than the number of configurations evaluated immediately. This gives TUNE-R2 room to remove invalid or duplicate proposals and rank the remaining candidates using their proposal preference scores, with an additional novelty bonus when diversification is requested. The selected queue is then evaluated one trial at a time before the next reflection is triggered.

G.4 AdultIncomeTask Run-level Behavior

Table 23 summarizes the overall behavior of TUNE-R2 on AdultIncomeTask. The run performs 8 reflection cycles and 8 refinement cycles over 30 trials. Each cycle updates the compact search state and produces a lookahead evaluation queue.

G.5 Qualitative Policy-Horizon Dynamics

Figure 7 provides trajectory-level examples complementing the aggregate horizon ablation in Table 4. Fixed $h = 1$ incorporates feedback after every evaluation but incurs substantially more reasoning, whereas fixed $h = 5$ amortizes reasoning more aggressively at the cost of slower feedback incorporation. The adaptive horizon adjusts this reuse interval according to the current search state.

Table 23: Run-level behavior of TUNE-R2 on AdultIncomeTask.

Metric	Value
Trials	30
Best ROC-AUC	0.930851
Runtime	148.14 seconds
Reflection steps	8
Refinement steps	8
Total LLM calls	16
Total tokens	23,636
Fallbacks	0
Typical proposal set size	17
Typical queued candidates	5

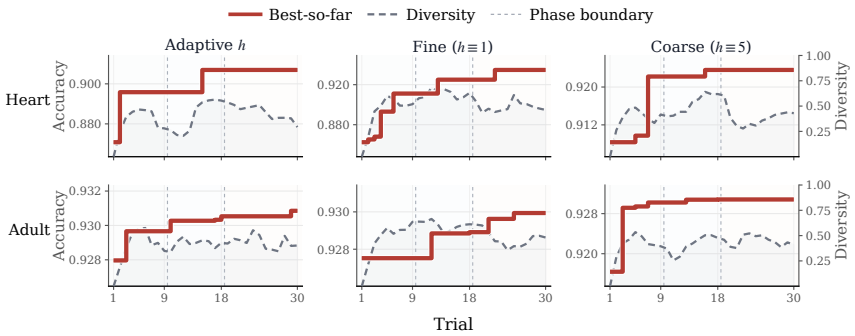

Figure 7: **Qualitative policy-horizon dynamics.** Incumbent performance and candidate diversity under adaptive policy reuse, per-evaluation refresh ($h = 1$), and five-evaluation reuse ($h = 5$).