SMALL LLMS WITH EXPERT BLOCKS ARE GOOD ENOUGH FOR HYPERPARAMTER TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyper-parameter Tuning (HPT) is a necessary step in machine learning (ML) pipelines but becomes computationally expensive and opaque with larger models. Recently, Large Language Models (LLMs) have been explored for HPT, yet most rely on models exceeding 100 billion parameters. We propose an Expert Block Framework for HPT using Small LLMs. At its core is the Trajectory Context Summarizer (TCS), a deterministic block that transforms raw training trajectories into structured context, enabling small LLMs to analyze optimization progress with reliability comparable to larger models. Using two locally-run LLMs (phi4:reasoning14B and qwen2.5-coder:32B) and a 10-trial budget, our TCS-enabled HPT pipeline achieves average performance within **0.9 percentage points** of GPT-4 across six diverse tasks.

1 INTRODUCTION

Hyperparameter tuning (HPT) is a critical step in achieving optimal model performance, but conventional approaches such as Bayesian Optimization and AutoML suffer from high computational cost, complex setup, and limited interpretability Akiba et al. (2019); Hutter et al. (2019a). Recent advances in Large Language Models (LLMs) have opened new possibilities for addressing these limitations by *asking LLMs to run HPT trials, and find best Hyperparamters within given time/computation budget*. Recent work has shown that LLMs can outperform traditional search methods by leveraging chain-of-thought reasoning and extended optimization horizons Wei et al. (2023), while frameworks such as AgentHPO Liu et al. (2025) and OptiMindTune Madiraju & Madiraju (2025) demonstrate the potential of multi-agent and structured reasoning approaches for HPT. However, these methods rely on large, resource-intensive models such as GPT-4, Gemini 1.5, raising questions of cost, accessibility, and scalability.

In this work we attempt to answer the question—*Can an HPT-specific deterministic expert block help small LLMs perform at par with large LLMs for the HPT problem?* To answer this question, we introduce a Trajectory Context Summarizer (TCS) which deterministically ingests trial artifacts (trial hyperparameters, per-epoch results for the current run, and aggregated history of prior runs) and emits a compact, machine- and human-readable state report (current status, latest experiment summary, per-parameter history, and comparative effect of recent changes). This structured report reduces token use and noisy signals, so small LLMs can reason about optimization progress and propose reliable hyperparameter updates.

2 RELATED WORKS

Hyperparameters are configuration settings defined before training a machine learning model that control its learning process and performance (e.g., learning rate, number of layers, batch size). As the performance of machine learning models is sensitive to these parameters, hyperparameter tuning (HPT) is performed to obtain suitable hyperparameters. Standard HPT methods includes grid search, random search, Bayesian optimization (BO), and end-to-end AutoML pipelines, but these approaches become computationally heavy when model evaluations are expensive and the pipeline is opaque or rigid Shahriari et al. (2015); Hutter et al. (2019a). Below we briefly review these methods to contextualize proposed approach for HPT.

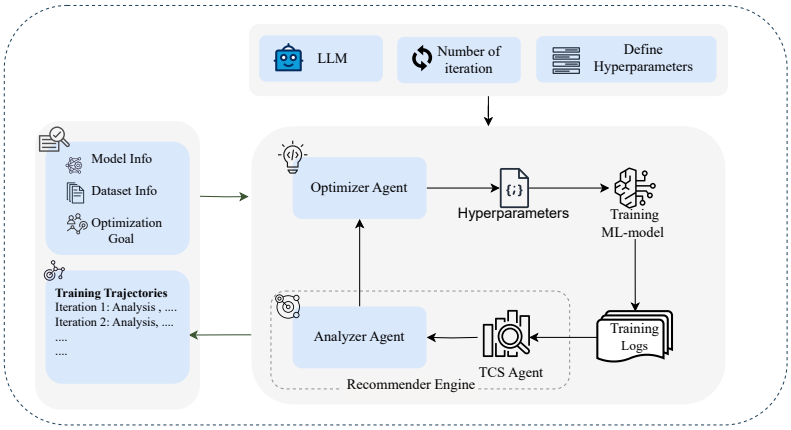


Figure 1: Flowchart of the proposed HPO pipeline with modular architecture.

2.1 Bayesian Optimization

Bayesian optimization (BO) is a black-box optimization method which does not require gradients of the model with respect to control variables Onorato (2024). The idea of BO is to construct a function surrogate of model's performance over hyperparameter space and use an acquisition strategy to sequentially select hyperparameters which lead to improved model performance. This intelligent selection strategy deliver strong sample efficiency relative to uninformed search, but often incurring high wall-clock cost for compute-intensive models Shahriari et al. (2015). Among BO variants, the Tree-structured Parzen Estimator replaces GP surrogates with density estimation over "good" and "bad" observations and is widely adopted in practical frameworks for HPT Watanabe (2023). Further, BO-based HPT methods typically do not assume task-specific prior over the hyperparameter space, thus re-exploration of well-understood regions lead to redundant evaluations, specifically when historical configurations are available for common model–dataset pairs Shahriari et al. (2015).

2.2 AutoML

AutoML systems automate many parts of the machine learning pipeline. These include preprocessing, feature engineering, model selection, training, and HPT. They often rely on BO or grid search under standardized constraints. The goal is to produce deployable models with minimal manual intervention Hutter et al. (2019a). While accessible, this automation tend to behave as a black-box that limits interpretability and fine-grained control. Due to focus on general applicability, these may yield sub-optimal results compared to carefully engineered, domain-specific workflows under specialized resource or accuracy requirements Hutter et al. (2019b;a).

2.3 Large Language Models for HPT

Recent reviews of LLM reasoning and agentic pipelines Bubeck et al. (2023); Yao et al. (2022), highlight the potential of LLMs to act as iterative decision-makers. These LLM based agents can plan, observe feedback, and adapt. This has also motivated their use for HPT Liu et al. (2025); Zhang et al. (2023b). Recent AgentHPO work operationalizes this idea by having an LLM agent propose configurations, observe trial outcomes, and iterate, reporting improvements over baseline optimizers under comparable budgets across multiple tasks Liu et al. (2025); Zhang et al. (2023b). Complementarily, authors also show that iterative LLM-guided proposals can match or surpass traditional methods in accuracy per number of trials, strengthening evidence that LLMs can effectively steer HPT with limited evaluations Zhang et al. (2024; 2023a). A practical concern in LLM for HPT literature is reliance on *very large proprietary models* (e.g., GPT-class systems), which raises questions about computational efficiency relative to traditional methods that explore broad parameter spaces like BO or random search Liu et al. (2025); Zhang et al. (2024).

Recently, authors in Kochnev et al. (2025) provides direct evidence that a fine-tuned, smaller open-source LLM can achieve—or surpass—state-of-the-art methods such as Tree-structured Parzen Estimator Watanabe (2023), while accelerating the tuning process, aligning the efficacy narrative with a resource-efficiency perspective. Also, studies in software engineering demonstrate that the per-

formance of compact open-source language models (e.g., Llama-3.1-8B) on domain-specific tasks can be substantially enhanced without resorting to full fine-tuning. This improvement is achieved through the joint application of hyperparameter optimization and prompt engineering. To address the complexity of the search space, the approach employs the NSGA-II multi-objective optimization algorithm to efficiently prune candidate configurations, followed by a grid search procedure to identify Pareto-optimal solutions. The resulting configurations are evaluated with respect to both syntactic quality (e.g., fluency, grammaticality) and semantic quality (e.g., accuracy, contextual relevance) across multiple domains Mahammadli & Ertekin (2025); Madiraju & Madiraju (2025).

Positioning: In this work, our goal is to evaluate whether *small LLMs can perform hyperparameter tuning and achieve performance comparable to state-of-the-art Bayesian Optimization, AutoML, or large LLM-based methods*. To answer this question, we design an agentic pipeline that includes a deterministic expert block acting as a context summarizer for the HPT task. Our approach builds on recent findings that (a) LLMs can drive effective HPT Liu et al. (2025), and (b) small open-source models can match larger proprietary ones when supported with structured optimization and prompting Zhang et al. (2024); Mahammadli & Ertekin (2025). The proposed expert-block framework equips small LLMs with deterministic, task-aware context from training trajectories, allowing them to analyze optimization progress with reliability under a fixed trial budget. Unlike studies focused mainly on very large models or domain-specific generation, our method targets general HPT with small LLMs by introducing a Trajectory Context Summarizer (TCS) that transforms raw training dynamics into structured inputs, improving proposal quality which reduces the analysis load on small LLMs, used in the proposed HPT pipeline.

3 Proposed Methodology

The proposed idea hinges on the fact that small LLMs struggle with reliable reasoning, especially when it requires multiple stages of analysis and a larger context window. In the case of the HPT problem, the LLM agent must dissect the epoch–loss curve from previous iterations to suggest the next hyperparameter setting, while also analyzing all possible HPT iteration trajectories. This is analogous to an optimization solver having access to gradients. To address this, we propose a deterministic, HPT-specific Trajectory Context Summarizer (TCS), designed to help small LLMs analyze the most recent trajectory and retain what occurred in prior HPT iterations.

Figure 1 illustrates our proposed HPT framework and its key components. At its core, the framework is designed to simplify and humanize the hyperparameter tuning process. Unlike conventional systems that rely solely on rigid mathematical modeling, our approach allows researchers to describe their model, dataset, and optimization goals directly in natural language. With this flexibility—combined with full control over the choice of LLM and the number of iterations—the framework becomes both powerful and easy to use. The framework is organized into two modular, interacting components: the **Optimizer Agent** and the **Recommender Engine**. The Recommender Engine is central to the loop. It first restructures raw training logs into a reasoning-oriented format, enabling even smaller LLMs to operate with the depth and precision usually associated with larger models. This distilled information is then passed to an Analysis Agent, which guides the LLM to produce a structured, task-specific analysis by breaking the reasoning process into clear, manageable steps.

The refined insights are subsequently delivered to the Optimizer Agent, which generates improved hyperparameter candidates for the next iteration. This iterative cycle continues until the user-specified optimization rounds are completed. By combining interpretability, modular reasoning, and problem-specific guidance, the framework gives users not only transparency but also control—whether it's selecting the backbone LLM, setting iteration limits, or specifying detailed model and dataset information. The result is a human-centered, accessible, yet rigorous environment for hyperparameter optimization across diverse machine learning tasks.

Following subsection presents the details of the optimizer agent, including our motivation and design of the prompt. Later, we present recommender engine, along with details of proposed TCS agent and Analyzer Agent.

3.1 THE OPTIMIZER AGENT

Optimizer Agent is a small LLM tasked with generating and refining hyperparameters to improve performance of ML model. The core of our approach is a specific prompting strategy that frames hyperparameter tuning as an iterative, data-driven reasoning problem. The prompt is dynamically constructed at each optimization step to provide the agent with a comprehensive snapshot of the training progress, enabling it to make informed, directive-based decisions.

3.1.1 PROMPT ARCHITECTURE FOR ITERATIVE REFINEMENT:

The agent's decision-making is guided by a multi-part prompt that acts as a form of **cognitive scaffolding**– integrating real-time feedback, expert analysis, and a constrained action space. Each component is designed to focus the LLM on the singular goal of optimizing the target metric. The prompt structure is composed of the following critical elements:

- **System-Level Persona and Goal:** We first establish the LLM's role as a machine learning expert. The system prompt explicitly defines the optimization objective, including the target metric (`metric_name`), the desired direction of improvement (`target_direction`), and a precise target value (`target_value`). This instruction grounds the agent in the specific task of HPTx.
- **Dynamic Performance Context:** The user prompt provides immediate and relevant data from the latest training run, including a restated **Optimization Goal** and a quantitative summary of **Current Performance** (e.g., the current metric value, gap to target, and performance trend). This creates a direct **feedback loop** from the training environment to the agent.
- **Guided Analytical Input:** A crucial element of the prompt is the **Latest Analysis** section. This provides a concise, high-level recommendation derived from the training logs. The agent is explicitly instructed to base its decision *strictly* on this analysis, which guides its reasoning and ensures its suggestions are hindged on analysis.
- **Constrained Search Space:** The prompt clearly delineates the **Hyperparameter Configuration**, separating parameters into fixed and optimizable sets. For each hyperparameter marked "to optimize," we provide a valid range or a discrete set of options. This strictly defines the agent's action space, ensuring all generated configurations are valid.
- **Structured Task and Response Format:** Finally, the agent is given a precise set of instructions under **Your Task** and is mandated to follow a **structured output** format. This consists of two parts: a brief textual **reasoning** that justifies its decision based on the provided analysis, and a machine-readable **hyperparameters** string (e.g., `learning_rate=0.005`).

Our main contribution is that this highly structured, feedback-driven prompt architecture can help the general-purpose LLM to focus on the specialized task of optimization. It effectively leverages the model's reasoning capabilities while imposing the necessary constraints to ensure its outputs are reliable i.e. avoid unexplainable sudden changes, and programmatically parsable for our automated HPT loop.

3.2 THE RECOMMENDER ENGINE

The core of our HPT pipeline's intelligence resides in the Recommender Engine, a component responsible for analyzing the results of a training iteration and producing a reasoned, expert-level recommendation for the next set of hyperparameters. This engine is specifically designed to overcome a common limitation of smaller LLMs: a tendency to hallucinate or produce superficial analysis when presented with large, unstructured contexts like raw training logs.

To mitigate this, the Recommender Engine employs a two-agent, modular architecture. The first agent, the **TCS (Trajectory Context Summarizer)**, distills the entire training history into a structured summary. The second, the **Analysis Agent**, then performs a guided analysis on this summary to generate a strategic recommendation. This output serves as the critical 'analysis' input for the **Optimizer Agent** described in the previous section.

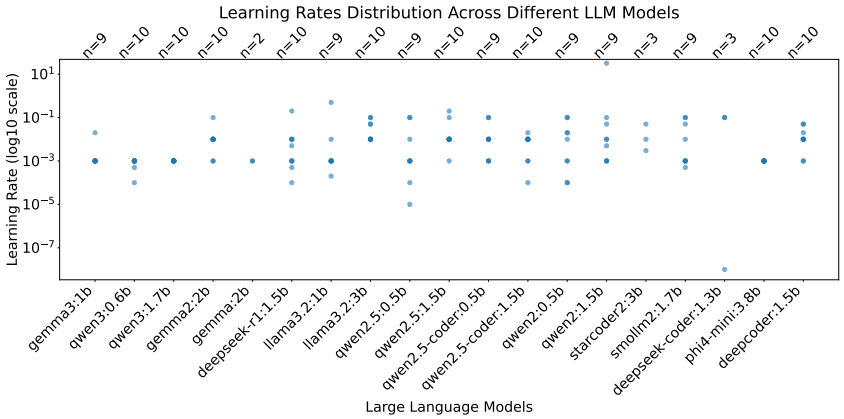


Figure 2: Variation in learning rates proposed by small LLMs over repeated trials with the same input prompt. Each dot represents one valid response. The value of n indicates the number of learning rates extracted after filtering invalid or irrelevant outputs. The wide spread illustrates the inconsistency of smaller models under identical conditions.

3.3 Agent 1: Trajectory Context Summarizer (TCS)

The motivation for TCS comes from the observation that small LLMs have limited reasoning capability, which leads to large variations in their predicted outputs, even when given the same input and prompt. We conducted a small experiment by asking for learning rates of a specific ML model, and as shown in Figure 2, small LLMs fluctuated considerably.

The primary function of the Trajectory Context Summarizer (TCS) is to serve as a deterministic data preprocessor, analyzer and context aggregator. After each training run, it collects and organizes all training related information into a comprehensive, structured report as shown in Figure 3. This ensures that the subsequent analysis is grounded in factual data rather than abstract interpretation. The generated summary is a compact snapshot containing following key components:

- **Current Situation:** A high-level overview of the optimization goal, the current metric value, the gap to the user defined target, and the performance trend (e.g., *IMPROVING*, *STAGNATING*).
- **Latest Experiment:** The specific hyperparameters used in the most recent iteration and the resulting performance metrics.
- **Detailed Hyperparameter Analysis:** A parameter-by-parameter breakdown of the entire experiment history. For each hyperparameter, it lists its valid range, its current value, and its historical performance, while flagging unexplored regions.
- **Previous Experiment Comparison:** A differential analysis between the latest and previous runs, highlighting the hyperparameter changes and their impact on the target metric.

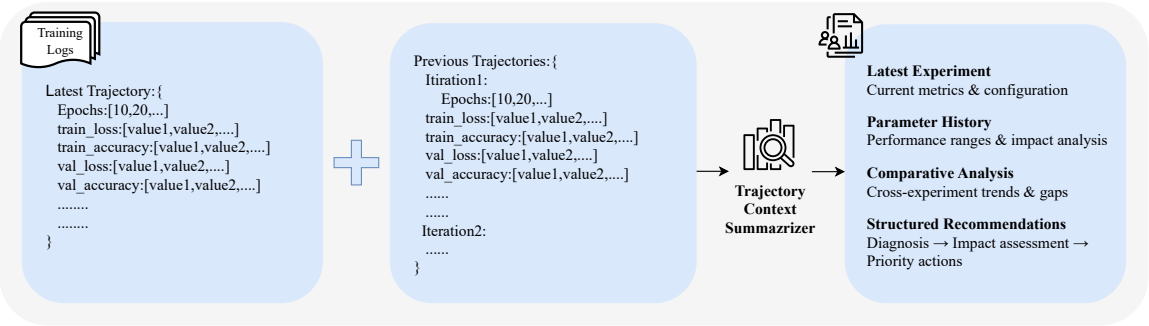


Figure 3: Conceptual depiction of input and output of the proposed Trajectory Context Summarizer.

3.4 AGENT 2: ANALYSIS AGENT

The Analysis Agent is a small LLM tasked with performing a root-cause analysis of HPT task based on completed iterations and formulating the next step. Its performance comes not from a single complex prompt, but from a step-by-step support method, often called cognitive scaffolding. In our case, this means using two instructional templates that guide the LLM's reasoning in a structured way.

First, a system-level directive establishes the agent's persona as an expert optimizer and provides it with a mandatory reasoning approach. This framework compels the agent to follow a specific diagnostic sequence:

1. Identify the primary problem like overfitting or underfitting etc.
2. Assess which hyperparameter categories (e.g., regularization, model capacity) which are most likely to address the problem identified in step 1.
3. Select a *single most impactful* hyperparameter to adjust.
4. Provide clear cause-and-effect reasoning for the choice of hyperparameter in step 3.

Second, the user-facing directive takes the structured summary from system level directive and embeds it within a rigorous analytical framework. This framework requires the agent to populate a six-part response:

1. **Problem Diagnosis:** A concise statement of the core issue.
2. **Hyperparameter Impact Assessment:** A theoretical evaluation of how each available parameter could influence the metric.
3. **Primary Action:** The selection of the single most critical hyperparameter to modify.
4. **Specific Recommendation:** The exact new value or direction of change for the chosen parameter.
5. **Reasoning:** A detailed justification for the recommendation over other alternatives.
6. **Expected Outcome:** A prediction of the anticipated performance improvement.

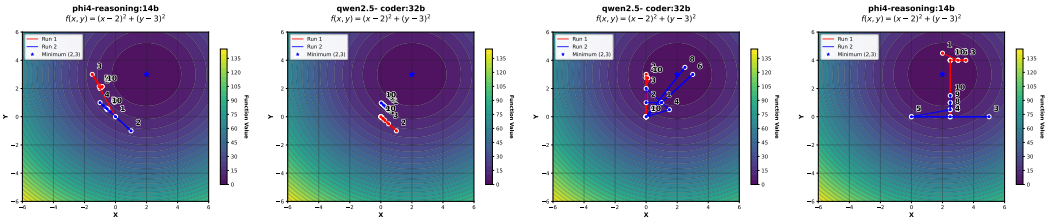
This two-part approach—first giving a factual summary and then guiding a step-by-step reasoning process—helps the small LLM break down the complex task of hyperparameter tuning into smaller, manageable steps. Our idea is that asking small LLMs to do this analysis in two different ways can reduce the randomness in their responses.

3.5 COMPLETE TCS-ENABLED HPT LOOP

The proposed HPT cycle has following flow. Each iteration t begins with the **Recommender Engine** providing a strategic report, A_{t-1}, based on the historical log. This analysis directs the **Optimizer Agent** to generate a new set of hyperparameters, Θ_t, and a corresponding justification, J_t. The system then executes a training run with Θ_t to yield performance results, R_t. The pipeline for the iteration is completed as the **Recommender Engine** takes over: **TCS** first processes the results (R_t) to create a structured summary, S_t. This summary is then passed to the **Analysis Agent**, which produces the final strategic report, A_t. This report, A_t, then serves as the primary directive to kickstart the next optimization cycle. This continuous loop of analysis, generation, and execution creates a powerful feedback mechanism that systematically steers the search toward Θ^*. Details of the same are also given in Algorithm 1.

4 RESULTS AND DISCUSSIONS

The proposed methodology was applied to six distinct tasks spanning five key domains: Computer Vision (CV), Natural Language Processing (NLP), Recommender Systems (RecSys), Tabular Learning, and Graph Neural Networks (GNN). The task selection was deliberately curated to include both foundational benchmarks (e.g., CIFAR-10, Cora) and contemporary, real-world problems to ensure



(a) Phi-4 Reasoning (14B) without TCS (b) Qwen2.5-Coder (32B) without TCS (c) Qwen2.5-Coder (32B) with TCS (d) Phi-4 Reasoning (14B) with TCS

Figure 4: Optimization trajectories for different LLMs with and without the Trajectory Context Summarizer (TCS). The left two plots show runs without TCS, where both models fail to converge efficiently. The right two plots show runs with TCS, where structured summaries enable both models to converge reliably to the global minimum.

a robust assessment of our framework's capabilities. A detailed summary of the domains, applications, datasets, target models, and primary evaluation metrics is presented in Table 3 in Appendix. To demonstrate the effectiveness of the proposed TCS-enabled HPT tool, we present a comparative analysis with the study in Liu et al. (2025). This choice aligns with our goal of evaluating whether the proposed expert system block can help reduce the performance gap between small LLM-based HPT pipelines and proprietary large LLM-based tools. Some important implementation details and language models used are discussed below.

The proposed framework leverages the reasoning capabilities of small, locally-run language models deployed via the **Ollama** framework Ollama Inc. (2025). The two primary models used for the Optimizer and Recommender agents were `phi4:reasoning14b` and `qwen2.5-coder:32b`. This approach ensures reproducibility and control over the experimental environment. To ensure the reliability and stability of our results, we conducted **5 independent HPT runs** for each task. Each run consisted of **10 iterations (trials)**. The final reported performance is the mean of the best performance achieved across these 5 independent runs. All experiments are done on a system with Intel Xeon W7-2495X processor having 256GB RAM and Nvidia RTX 4500Ada graphics card.

Below we first discuss inconsistency in small LLM responses, then analyses the effectiveness of TCS using a function optimization problem and then discuss HPT results as a comparative analysis with results from Liu et al. (2025).

4.1 Inconsistency in Responses of Small LLMs

We tested the reasoning ability of small LLMs by repeatedly giving them the same prompt. Each model was asked ten times to propose the best learning rate for improving accuracy given a set of training trajectories. As shown in Figure 2, the responses of small LLMs were highly inconsistent. Models with fewer parameters produced scattered answers and suggested very different learning rates across trials. In some cases, the models ignored the instructions and generated irrelevant outputs that could not be plotted. When the model size increased, the variability in responses decreased. Larger models converged to a smaller set of learning rate values and showed more reproducibility across repeated prompts. This result highlights a key limitation of small LLMs: their lack of consistency makes them unreliable for tasks like hyperparameter selection. Increasing model size reduces this issue and leads to more stable reasoning.

Table 1: Performance comparison across different tasks. All values are mean $\pm$ std%.

Task	Random	Bayesian	GPT-3.5	GPT-4	Human	Proposed Method	Without TCS
Tabular Regression	56.49 $\pm$0.40%	56.85 $\pm$0.05%	56.78 $\pm$0.36%	58.01 $\pm$0.11%	56.9	56.82 $\pm$0.52%	56.02 $\pm$0.52%
Node Classification	78.80 $\pm$0.81%	74.96 $\pm$2.70%	81.13 $\pm$0.22%	81.38 $\pm$0.22%	81.5	80.87 $\pm$0.19%	75.48 $\pm$1.52%
Image Classification (CIFAR-10)	81.63 $\pm$3.88%	79.87 $\pm$4.19%	83.87 $\pm$1.18%	85.18 $\pm$0.52%	85.05	84.75 $\pm$0.99%	81.17 $\pm$3.91%
Image Classification (Butterfly)	78.99 $\pm$2.32%	63.57 $\pm$8.67%	84.79 $\pm$1.01%	85.92 $\pm$0.57%	82.74	87.00 $\pm$2.15%	69.70 $\pm$3.55%
Text Classification (SST-2)	90.28 $\pm$0.76%	90.28 $\pm$0.45%	90.34 $\pm$0.79%	91.32 $\pm$0.11%	90.71	89.58 $\pm$0.43%	87.48 $\pm$18.31%
Click Through Rate (CTR)	81.94 $\pm$0.24%	81.67 $\pm$0.92%	82.14 $\pm$0.06%	82.09 $\pm$0.05%	82.19	82.03 $\pm$0.02%	80.66 $\pm$0.01%

4.2 Effect of TCS: Study using Function Optimization

Before presenting large-scale benchmarks, we first study a simple minimization problem $(x-2)^2 + (y-3)^2$ to test whether the Trajectory Context Summarizer (TCS) improves optimization. This

task serves as a controlled setting where the impact of structured versus unstructured optimization signals can be clearly observed.

In the baseline setup, raw training logs are passed directly to the analysis agent without TCS. Figure 4 shows that this leads to unstable optimization across different models. The Phi-3.5 model oscillates around suboptimal points and does not show consistent progress. The Qwen2.5-Coder model, despite having more parameters, also fails to converge and follows a random-walk pattern. Even GPT-3.5, used as a commercial reference, does not reach the optimum under this setup. These outcomes suggest that without structured preprocessing, LLMs—small or large—struggle to interpret training trajectories, retain memory across trials, or systematically guide the search.

When TCS is introduced, the optimization behavior changes significantly. As shown in Figure 4, both models now display clear and directed trajectories toward the global minimum. The Qwen2.5-Coder model systematically explores the space, avoiding previously tested regions and moving closer to the optimum in each step. The Phi-3.5 reasoning model, though smaller, achieves faster convergence with fewer iterations. This demonstrates that reasoning-focused LLMs, when supplied with structured trajectory context, can outperform larger models in efficiency.

The improvement can be explained by the role of TCS in restructuring the optimization history. Instead of presenting the LLMs with long, noisy logs, TCS condenses the information into a state-like summary. This allows the analysis agent to: maintain optimization memory across iterations, compare new results against past trials, Identify unexplored hyperparameter ranges and Propose next steps based on structured evidence rather than raw signals. This experiment shows that the presence of TCS is the deciding factor in whether small LLMs can perform reliable hyperparameter tuning. Without TCS, even large or commercial models fail to converge. With TCS, smaller open-source models produce stable and efficient optimization trajectories. These findings validate our design choice and motivate the use of TCS in the full benchmarking experiments that follow.

4.3 Results for HPT Task

As explained before, our target is to see if TCS can elevate small LLM up to large LLM level performance for HPT task. So, for a fair and consistent comparison, all benchmark values for baselines results are taken directly from the results reported in the original **AgentHPO** paper Liu et al. (2025). Our evaluation compares proposed TCS-Enabled HPT against two categories of methods:

1. **Traditional HPT Methods:** We measure our framework's efficiency and performance against two widely-recognized traditional methods: **Random Search**, which remains a formidable baseline Bergstra & Bengio (2012); Eggensperger et al. (2022), and **Bayesian Optimization**, a prominent model-based approach.

2. **State-of-the-Art LLM Optimizers:** To situate our work at the forefront of this field, we directly compare our small-model approach against the performance of leading large, proprietary models: **GPT-3.5** and **GPT-4**.

For the task of HPT, we first conducted experiments using small LLMs (up to 3.8B parameters). These models exhibited a lack of consistency in their responses, making it difficult to extract hyperparameters reliably. A major limitation was hallucination, where the models often generated incomplete or misleading instructions. From these observations, it became evident that increasing model size improved response stability, with larger models producing more consistent outputs for identical inputs.

Based on these findings, we subsequently tested moderately larger models and observed that certain ones provided stable and reproducible results. In particular, `Qwen2.5-32B` Hui et al. (2024) and `Phi-4-14B` Abdin et al. (2024) were evaluated in depth. To assess their effectiveness, we developed a two-agent pipeline consisting of an Optimizer Agent and a Recommender Engine, which iteratively refined hyperparameters using training trajectories. Two pipeline variants were tested: one incorporating a TCS to preprocess trajectories, and one without it (raw training trajectories passed directly to the analyzer).

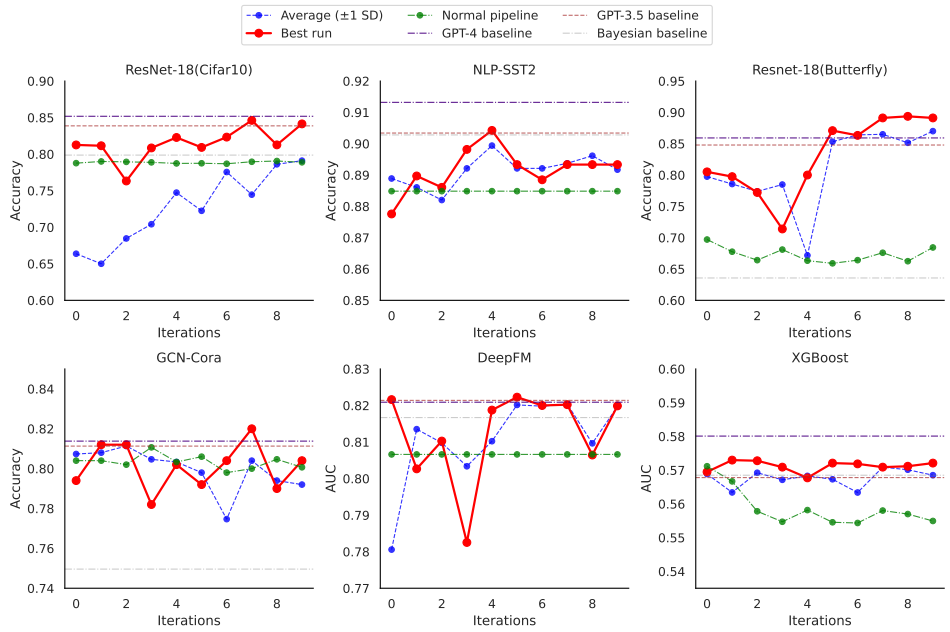


Figure 5: **Phi4:reasoning14b-Instruct Performance.** Results are averaged over 5 independent runs. Our AS-HPO pipeline (blue line with shaded SD) significantly outperforms our Normal Pipeline (green line, an ablation without the Summarizer) and the GPT-3.5 baseline, while performing competitively with the GPT-4 baseline.

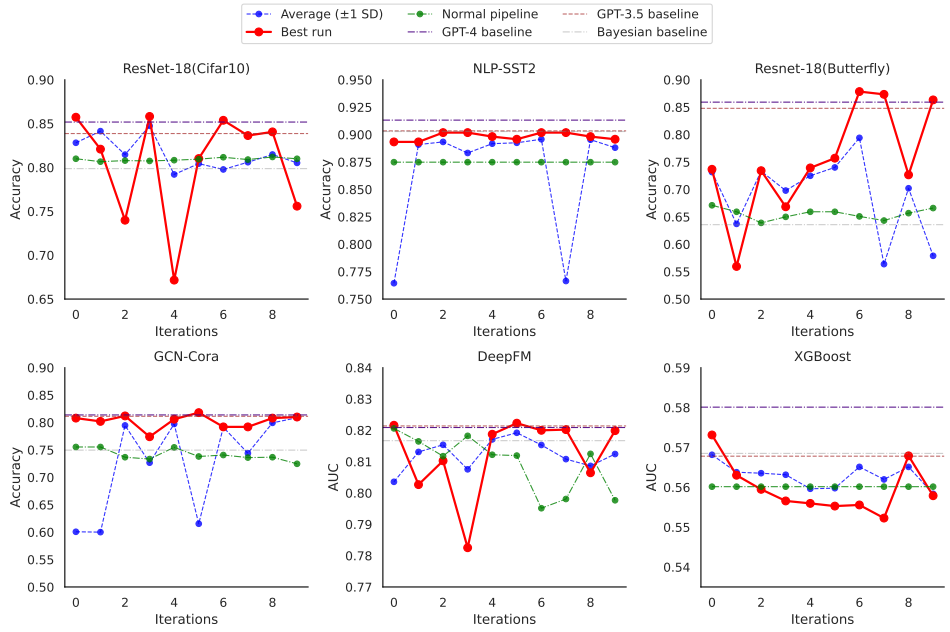


Figure 6: **Qwen2.5 Coder-32B Performance.** Results are averaged over 5 independent runs. The AS-HPO pipeline (blue line) demonstrates strong performance, surpassing the GPT-3.5 and Bayesian baselines and showing competitive results against GPT-4, validating our approach with a different model architecture.

4.4 Semantic Energy Analysis

To better understand the stability and convergence of the optimization process, we analyzed the *Semantic Energy* Ma et al. (2025) variation of the agents throughout the pipeline iterations. Semantic

energy serves as a proxy for the ambiguity and "stress" within the model's reasoning process; lower energy states typically correlate with higher confidence and more stable solutions.

We observed distinct trends when comparing the TCS framework against the non-TCS baseline. As shown in Table 2, the non-TCS framework exhibited consistently higher mean energy scores across all agents. This high-energy state suggests that the baseline struggles to converge to a stable optimal solution, explaining its comparatively poor performance. In contrast, the TCS pipeline demonstrated consistently lower energy scores. This reduction indicates that a deterministic reasoning mechanism can mitigate the uncertainties that arise when the reasoning process is delegated entirely to the LLM. Our results confirm that TCS successfully guides the LLM toward a more stable, lower-energy state

Table 2: Comparison of Mean Semantic Energy (E) per Agent across Pipelines. Lower scores indicate higher stability and better optimization alignment.

Agent	**Qwen2.5 Coder**		**Phi-4 Reasoning**	
	Baseline	**w/ TCS**	**Baseline**	**w/ TCS**
Optimizer	-14.85	**-15.23**	19.46	**-15.32**
Analyzer	-13.98	**-13.02**	-12.83	**-34.34**

5 Conclusion and Future Work

This work introduced a trajectory–summarization-based framework for hyperparameter tuning, designed to enable small, open-source LLMs to operate effectively under constrained trial budgets. By integrating the Trajectory Context Summarizer (TCS), we showed that small LLMs can deliver results competitive with Bayesian Optimization, AutoML approaches, and even larger commercial LLMs, while avoiding the instability observed in unstructured pipelines. Our analysis across diverse tasks—including vision, text, tabular, graph, and recommendation domains—demonstrates that structured summarization and guided reasoning are essential for reliable performance in hyperparameter optimization. In future work, we aim to extend the framework in three directions. First, by incorporating richer forms of summarization, such as uncertainty-aware or probabilistic representations of trajectories. Second, by exploring the integration of reinforcement learning signals into the agent loop, enabling adaptive exploration–exploitation balancing. Finally, we plan to evaluate the method on large-scale, industrial datasets and under distributed multi-agent settings, where communication efficiency and robustness to partial information will become critical. Together, these directions will further clarify the potential of small LLMs as practical and resource-efficient optimizers.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Takuya Akiba, Shotaro Sano, et al. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631. ACM, 2019. doi: 10.1145/3292500.3330701.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Sébastien Bubeck, Varun Chandrasekaran, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Katharina Eggensperger, Philipp Müller, et al. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo, 2022. URL https://arxiv.org/abs/2109.06716.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, et al. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.). *AutoML: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, Cham, 2019a. ISBN 978-3-030-05317-8. doi: 10.1007/978-3-030-05318-5.

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

Roman Kochnev, Arash Torabi Goodarzi, et al. Optuna vs code llama: Are llms a new paradigm for hyperparameter tuning?, 2025. URL https://arxiv.org/abs/2504.06006.

Siyi Liu, Chen Gao, and Yong Li. Agenthpo: Large language model agent for hyper-parameter optimization. In *Conference on Parsimony and Learning*, volume 280, pp. 1146–1169. PMLR, 24–27 Mar 2025. URL https://proceedings.mlr.press/v280/liu25c.html.

Huan Ma, Jiadong Pan, Jing Liu, Yan Chen, Joey Tianyi Zhou, Guangyu Wang, Qinghua Hu, Hua Wu, Changqing Zhang, and Haifeng Wang. Semantic energy: Detecting llm hallucination beyond entropy, 2025. URL https://arxiv.org/abs/2508.14496.

Meher Bhaskar Madiraju and Meher Sai Preetam Madiraju. Optimindtune: A multi-agent framework for intelligent hyperparameter optimization, 2025. URL https://arxiv.org/abs/2505.19205.

Kanan Mahammadli and Seyda Ertekin. Sequential large language model-based hyper-parameter optimization, 2025. URL https://arxiv.org/abs/2410.20302.

Ollama Inc. Ollama: Local framework for large language models. https://ollama.com, 2025.

Gabriele Onorato. Bayesian optimization for hyperparameters tuning in neural networks, 2024. URL https://arxiv.org/abs/2410.21886.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023. URL https://arxiv.org/abs/2304.11127.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2022.

Lei Zhang et al. Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979*, 2023a.

Michael R. Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization, 2024. URL https://arxiv.org/abs/2312.04528.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. Automl-gpt: Automatic machine learning with gpt. *arXiv preprint arXiv:2305.02499*, 2023b.

Algorithm 1 TCS-Enabled HPT

Input: LLM_O: Optimizer agent model; LLM_A: Analysis agent model; M: model architecture; D: dataset;
$\mathcal{H}_{\text{space}}$: hyperparameter search space; G: optimization goal; T: maximum iterations.
Output: Θ^*: optimal hyperparameter configuration; $\mathcal{L}$: complete experimental logs.

Initialization:
Optimizer $\leftarrow$ INITIALIZEOPTIMIZER$(LLM_O, \mathcal{H}_{\text{space}}, G)$
Recommender $\leftarrow$ INITIALIZERECOMMENDER(LLM_A, G)
$\mathcal{L} \leftarrow \emptyset$ `// Initialize experimental logs`
$A_0 \leftarrow$ RECOMMENDER.BOOTSTRAPANALYSIS$(\mathcal{H}_{\text{space}})$

Main Loop:
for $t \leftarrow 1$ **to** T **do**
  $(\Theta_t, J_t) \leftarrow$ OPTIMIZER.GENERATE(A_{t-1})
  $R_t \leftarrow$ EXECUTETRAINING(M, D, Θ_t)
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\Theta_t, J_t, R_t)\}$
  $S_t \leftarrow$ RECOMMENDER.TCS.SUMMARIZE$(\mathcal{L})$
  $A_t \leftarrow$ RECOMMENDER.ANALYSISAGENT.ANALYZE(S_t)
end

Finalization:
$\Theta^* \leftarrow$ FINDBESTCONFIGURATION$(\mathcal{L}, G)$
return $(\Theta^*, \mathcal{L})$

Table 3: Domain-Specific Model Summary

Field	Application	Data Source	Best Model	Hyperparameters [Range]	Measure
Computer Vision	Image Classification	CIFAR-10, Butterfly	ResNet-18	lr: [10^{-5}, 10^{-1}]; optimizer: [adam, sgd]; epochs: [25, 200]; weight_decay: [10^{-6}, 10^{-1}]; dropout: [0, 0.5]; momentum: [0.5, 1]; batch_size: [32, 64, 128, 256]	Accuracy
Natural Language Processing	Text Classification	SST-2	DistilBERT	lr: [10^{-6}, 10^{-2}]; epochs: [1, 4]; dropout/attn/seq: [0, 0.5]; batch_size: [8, 16, 32, 64, 128]; activation: [gelu, relu, silu]; weight_decay: [10^{-6}, 0.1]	Accuracy
Recommender Systems	Click-Through Rate	MovieLens	DeepFM	lr: [10^{-5}, 10^{-1}]; embed_size: [8, 16, 32, 64]; optimizer: [adam, sgd]; reg_weight: [10^{-6}, 10^{-1}]; dropout: [0, 0.5]; batch_size: [256–2048]; mlp_hidden: [32–512]; mlp_layers: [1–4]	AUC
Tabular Learning	Regression	House Price	XGBoost	lr: [10^{-3}, 1]; max_depth: [3–11]; min_child_weight: [1–10]; subsample: [0.5–1]; colsample_bytree: [0.5–1]; n_estimators: [100–500]; gamma: [0–0.5]; reg_alpha/reg_lambda: [0–1]	R^2
Graph Neural Networks	Node Classification	Cora	GCN	layers: [1–5]; lr: [10^{-6}, 10^{-1}]; optimizer: [adam, sgd]; epochs: [1–200]; hidden_size: [8–64]; activation: [relu, elu, silu]; weight_decay: [10^{-6}, 10^{-1}]; dropout: [0–0.5]	Accuracy