LLMSYS-HPOBENCH: HYPERPARAMETER OPTIMIZATION BENCHMARK SUITE FOR REAL-WORLD LLM SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) systems have been the frontier of AI in many application domains, leading to new challenges and opportunities for hyperparameter optimization (HPO) for the AutoML community. However, this type of system exhibits an unprecedented compound space of hyperparameter configuration from both the AI and non-AI components; rich and nonlinear implications from the fidelity factors; and diverse costs of measuring hyperparameter configurations, none of which have been fully captured in existing benchmarks. This paper presents the first (live) benchmark suite and datasets for HPO of real-world LLM systems, dubbed `LLMSYS-HPOBench`, covering data related to the inference objective values of hyperparameter configurations profiled from running the LLM systems. Currently, `LLMSYS-HPOBench` contains 364,450 hyperparameter configurations with a dimensionality of 12–23, 3–5 dimensions of fidelity factor leading to 932 settings, 3–9 inference objective metrics, and 2–10 cost metrics, together with generated logs from measuring the LLM systems. What we seek to advocate is not only a revalidation of the existing HPO algorithms over the frontier LLM systems, but also to provide an evolving platform for the AutoML community to explore new directions of research in this regard. The benchmark suite has been made available at: https://anonymous.4open.science/r/llmsys-hpobench-CD2B.

1 INTRODUCTION

The frontier of AI has evolved from machine/deep learning models into large language models (LLMs) that are often combined with planning, memory, retrieval, and tool usage (Barker et al., 2025), dubbed the system of LLM, or simply the LLM systems. Their real-world effectiveness and efficiency, however, are governed not only by core LLM capabilities, but also by the vast other system components that interact with the LLM, e.g., tools, database, and hardware containers. This leads to a multifaceted hyperparameter space, including the AI parts (e.g., LLM temperature and prompting choices) and non-AI parts (e.g., retrieval and tooling strategies)[1] (Agrawal et al., 2024). It has been shown that different hyperparameter configurations of LLM systems can lead to substantially different inference behaviors in terms of task quality, latency, robustness, and monetary cost (Barker et al., 2025). For example, in retrieval-augmented generation (RAG), increasing retrieval top-k items and enabling a stronger reranker can improve response quality, but often cause extra query time and token cost due to longer context and extra LLM calls (Barker et al., 2025).

The inherent black-box nature of LLM systems and their hyperparameter characteristics lead to a great opportunity for the AutoML community to conduct hyperparameter optimization (HPO), as shown in recent work (Brookes et al., 2025). However, despite the increasing importance of optimizing LLM systems' hyperparameters, we have yet to witness considerable research progress/innovation, primarily because the measurement of hyperparameter configurations over the LLM systems is extremely expensive and effort-heavy: measuring one hyperparameter configuration may require tens of minutes to hours and a non-trivial API/compute budget; scaling to thousands of configurations

[1]We treat the AI hyperparameters as those that are directly attached to the underlying LLM and can influence its behaviors; otherwise they are non-AI hyperparameters.

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.

can consume hundreds of GPU-hours (Zhou et al., 2025; Kim et al., 2025). This is in addition to the complexity of deploying real LLM systems.

The above is a corollary of lacking dedicated benchmark suite/datasets on configuration and HPO for LLM systems. Indeed, existing benchmarks/datasets for AutoML (Li et al., 2020; Eggensperger et al., 2021; Pfisterer et al., 2022; Bansal et al., 2022) and for computing systems (Menzies et al., 2026) cannot fulfill the needs of HPO for LLM systems due to several reasons:

- They primarily focus on either classic machine/deep learning models or computing systems, which fail to capture the unique structure, AI/non-AI hyperparameter interaction, and fidelity implication for LLM systems. For example, LLM temperature and backend scheduling policy can significantly, sparsely, and jointly affect both quality and latency in a LLM system, while many classic HPO benchmarks assume a model-centred pipeline (Ying et al., 2019; Pineda-Arango et al., 2021) or are systems only (Menzies et al., 2026).
- They focus on simple fidelity settings, while the tasks that LLM systems deal with are often much more complex due to the presence of natural language-based tasks. For example, the fidelity may vary with query difficulty, prompt length, and domain shift, which can non-monotonically influence the cost and objectives. In contrast, classic HPO benchmarks (Ying et al., 2019; Eggensperger et al., 2021) often instantiate fidelity settings through a small number of naturally ordered factors, such as dataset size and training epochs, resulting in a comparatively monotonic relationship between cost and fidelity level.
- The diverse costs of measuring every hyperparameter configuration, which can differ for LLM systems, are often limited in existing benchmarks. For example, the cost of LLM systems can include build time and the query tokens required to complete a set of tasks.

To bridge this gap, this paper presents `LLMSYS-HPOBench`, the first live benchmark suite with measured hyperparameter configurations and their inference results over real-world LLM systems, together with other relevant data, e.g., hardware and logs. At the time of writing this paper, `LLMSYS-HPOBench` contains data collected by running and profiling seven LLM systems that are of various categories: LLM agent, retrieval-augmented generation (RAG) with LLM, and LLM inference engine, leading to 364,450 hyperparameter configurations with a dimensionality of 12–23, 3–5 fidelity factors of 932 settings, 2–9 objective metrics and 2–10 cost metrics, together with hardware metrics and generated logs from measuring the LLM systems. What we advocate to provide the AutoML community is not merely an efficient suite to compare the better or worse among the HPO algorithms for LLM systems, but also paving the way toward new research opportunities and directions. It is worth noting that preparing and collecting `LLMSYS-HPOBench` requires great efforts and resources: by May 2026, the data available are the results of running on 10 servers with $\approx$ 2,277,600 CPU core-hours and $\approx$ 94,900 GPU-hours. Yet, `LLMSYS-HPOBench` is a live benchmark suite where new data/LLM systems are continually incorporated. `LLMSYS-HPOBench` can be accessed at: https://anonymous.4open.science/r/llmsys-hpobench-CD2B.

2 Preliminaries

2.1 Black-box Hyperparameter Optimization

Black-box HPO aims to identify a hyperparameter configuration that optimizes target performance objective(s) when the objective function can only be accessed through evaluation of the model. Formally, given a hyperparameter space $\boldsymbol{\mathcal{X}}$, the problem is defined as:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\boldsymbol{\mathcal{X}}} f(\boldsymbol{x}) \text{ or } \boldsymbol{x}^* = \arg\max_{\boldsymbol{x}\in\boldsymbol{\mathcal{X}}} f(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ denotes a concrete hyperparameter configuration, and $f(\boldsymbol{x})$ represents the performance objective of interest, e.g., error or runtime. In general, $\boldsymbol{\mathcal{X}}$ may comprise continuous, integer, ordinal, and categorical hyperparameters. This is a well-formulated problem that has been tackled by the AutoML community (Eggensperger et al., 2021; Hutter et al., 2011; Cowen-Rivers et al., 2022) and the configurable software/system community (Nair et al., 2020; Zhu et al., 2017). Such a formulation naturally applies to LLM systems, whose behavior is governed by a rich set of tunable hyperparameters, such as model selection, prompting strategies, and memory mechanisms.

HPO with multi-fidelity optimization improves search efficiency by allowing queries at cheaper, lower-fidelity settings. Instead of evaluating a configuration $\boldsymbol{x}$ only under the target setting, the algorithm may query $f(\boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{z} = (z_1, z_2, \ldots, z_m)$ denotes a fidelity setting instantiated from multiple fidelity factors (z_i), and each z_i controls one aspect of evaluation fidelity. Typical examples include dataset subsets (Klein et al., 2017; Hu et al., 2019), reduced training epochs (Li et al., 2017; Domhan et al., 2015), and feature subsets (Li et al., 2017). Similar cases can also be found under LLM systems, i.e., shorter prompt length, reduced dataset subset, or fewer document chunks. These lower-fidelity settings may reduce evaluation cost, albeit at the expense of approximation accuracy.

2.2 Related Work

In the following, we review two lines of benchmark efforts most relevant to this work.

HPO benchmarks for machine/deep learning models. In AutoML, benchmark development has progressed from queryable NAS resources to broader HPO. Early efforts such as `NAS-Bench-101` (Ying et al., 2019) and `NAS-Bench-201` (Dong & Yang, 2020) precomputed architecture evaluations to enable fast and reproducible NAS comparison across fixed search spaces and datasets. Subsequent benchmark datasets moved beyond architecture search: `HPO-B` (Pineda-Arango et al., 2021) organized a large-scale OpenML-based benchmark for black-box HPO, while `HPOBench` (Eggensperger et al., 2021) provided a unified suite with particular emphasis on multi-fidelity HPO, including raw, tabular, and surrogate variants. This line was further enriched by `YAHPO Gym` (Pfisterer et al., 2022), which offers a large surrogate-based collection for multi-fidelity and multi-objective HPO, together with `JAHS-Bench-201` (Bansal et al., 2022) and `NAS-Bench-Suite` (Mehta et al., 2022), which targets joint architecture and hyperparameter search and explicitly supports cost-aware and multi-fidelity studies. `AMLB` (Gijsbers et al., 2024) provides a standardized benchmark for end-to-end evaluation of AutoML under tasks and budgets.

HPO benchmarks for software systems. Existing benchmark efforts for software system HPO are more heterogeneous and have been widely used (Chen et al., 2018; 2024; Gong & Chen, 2023; Chen & Li, 2024; 2021; Ye et al., 2025). Some benchmarks focus on a specific class of systems. For example, `CATBench` targets compiler autotuning and captures the complexities of compiler optimization (Tørring et al., 2025). Others collect data from a broader range of configurable systems, spanning databases, web services, cloud systems, and video encoders, including single workload/environment per system (Xiong & Chen, 2025; Weber et al., 2023; Ha & Zhang, 2019; Nair et al., 2020; Xiang et al., 2026; Gong et al., 2025), and multiple workloads/environments (Mühlbauer et al., 2023; Lesoil et al., 2023; Krishna et al., 2021; Cao et al., 2023; Gong & Chen, 2024). More broadly, `MOOT` (Menzies et al., 2026) curates a set of realistic multi-objective tasks spanning various domains.

How does `LLMSYS-HPOBench` differ. Unlike `LLMSYS-HPOBench`, the prior benchmarks do not target HPO of LLM systems, differing in several aspects (see Appendix B for more details):

Real-world LLM systems (C1). `LLMSYS-HPOBench` contains the real-world LLM systems hosted at the GitHub—a type of artefacts that synergizes both the AI and systems (non-AI).

Rich and complex fidelity (C2). For each subject system, `LLMSYS-HPOBench` defines a multi-factor fidelity space with up to 5 fidelity factors related to the tasks, whose combinations induce nearly a thousand executable fidelity settings with non-monotonic influences. This substantially expands the fidelity design space beyond prior multi-fidelity HPO benchmarks, which typically expose only one or two fidelity factors and predominantly adopt factors with a natural monotonic ordering (e.g., using more data incurs higher cost while yielding higher fidelity) (Falkner et al., 2018; Ying et al., 2019; Eggensperger et al., 2021), and the single fidelity system benchmarks (Menzies et al., 2026).

Heterogeneous configuration space (C3). Each system exposes a mixed and heterogeneous hyperparameter space, spanning AI components (e.g., model and inference settings) and non-AI components (e.g., scheduling and storage/runtime controls). In the current state of the benchmark, subject systems contain a dimension of 5–8 and 4–17 for AI and non-AI hyperparameters, respectively, leading to more than 364K hyperparameter configurations in total, considering the fidelity settings.

Many-objective metrics (C4). `LLMSYS-HPOBench` records up to nine objective metrics for each measured configuration, covering complementary aspects of LLM system behavior, such as quality

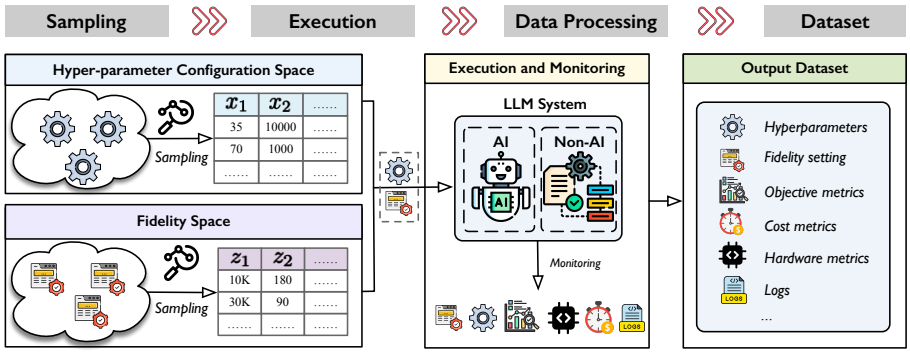


Figure 1: Workflow for constructing LLMSYS-HPOBench.

metrics (e.g., F1-score and success ratio) and performance metrics (e.g., throughput and time-to-first-token). This enables richer evaluation and supports trade-off analysis in HPO for LLM systems, and is competitive with the existing benchmarks with only two datasets have up to 12 objectives.

Multi-dimensional measurement cost (C5). LLMSYS-HPOBench records measurement cost up to 10 dimensions, including fine-grained benchmark duration, token usage, and the efforts to initialize the LLM system. This goes considerably beyond the others, which have merely 1–5 dimensions.

Hardware utilization metrics (C6). LLMSYS-HPOBench records hardware statistics for each evaluation/measurement, such as CPU/GPU and memory usage, supporting the analysis of execution efficiency and system behaviors. This is often missing in existing benchmarks.

Execution logs (C7). LLMSYS-HPOBench records execution log for each evaluation, supporting inspection and debugging analysis. No current benchmarks contain such information.

2.3 Why AutoML needs HPO for LLM Systems?

Firstly, LLM systems have become a major form of modern AI applications (Du et al., 2024; Jin et al., 2026; Wang et al., 2024; Kim et al., 2024). While earlier AI research was primarily focused on conventional machine/deep learning models, many recent research directions of AI and real-world applications are centered around LLM systems. This makes HPO for LLM systems an important and timely problem for AutoML.

Secondly, unlike classic machine/deep learning models, the behaviors of LLM systems can be jointly shaped by many design choices, including interaction between AI and non-AI hyperparameters; exhibit complex implications from the fidelity settings; and involve diverse costs in measurement. All those create new challenges and opportunities for building more specialized AutoML algorithms.

Thirdly, AutoML's HPO algorithms are naturally well-positioned to address this challenge. Although further specializations are necessary, their current advances in black-box optimization, surrogate modeling, and transfer learning provide a strong methodological foundation for HPO of LLM systems.

What remains missing, however, is a benchmark suite that renders this problem systematically measurable, comparable, and reproducible. This is precisely the goal of LLMSYS-HPOBench.

3 LLMSYS-HPOBench: A Benchmark Suite of Datasets

3.1 Benchmark Dataset Construction Procedure

Figure 1 illustrates the construction pipeline of LLMSYS-HPOBench for LLM systems HPO. We start by selecting open-sourced real-world LLM systems from GitHub based on timeliness, stars, and completeness of the documentation. For each LLM system, we perform the following steps:

1. Hyperparameter configuration space definition. We define the hyperparameter configuration space by using all tunable hyperparameters for both the LLM and other system components from code interfaces, official documentation, and deployment settings.

Table 1: The LLM systems profiled so far.

LLM System	Domain	Task Dataset	#Obj.	#Cost	#HP	$\#\mathcal{C}$	$\#\mathcal{S}$	$\#\mathcal{F}$	$\#\mathcal{Z}$
LIGHTRAG (2024)	RAG	*HotpotQA* (2018)	9	10	23	1004-1025	23512	4	216
NAIVERAG (2024)	RAG	*Agriculture*, *Biography* (2025), *HotpotQA* (2018), *BioASQ* (2023)	8	7	12	1000	56220	4	108
HTMLRAG (2024)	RAG	*Default-HTML Tasks* (2025)	8	7	15	406-1000	27173	3	90
VLLM (2023)	Inference Engine	*ShareGPT* (2025)	5	2	16	1070	115685	4	216
SGLANG (2023)	Inference Engine	*ShareGPT* (2025)	5	2	16	190	50849	5	108
AUTOGPT (2023)	Agent	*AGBenchmark* (2023)	5	3	12	92	9011	3	112
OPENHANDS (2024)	Agent	*ProofWriter* (2021)	3	6	21	1000	82000	4	82

#Objective: No. of objectives; **#Cost:** No. of cost metrics; **#HP:** No. of hyperparameter; $\#\mathcal{C}$**:** No. of configurations per fidelity setting; $\#\mathcal{S}$**:** No. of total configuration samples so far; $\#\mathcal{F}$**:** Dimension of fidelity factors; $\#\mathcal{Z}$**:** No. of fidelity settings considered.

2. Fidelity space definition. The fidelity space comprises controllable fidelity factors tied to the standard/compatible task datasets (e.g., *BioASQ* (Krithara et al., 2023)) under which the LLM system runs. These factors affect measurement cost and objectives; each has a range/value set determined by system implementation, task design, and execution constraints, e.g., in LIGHTRAG, `context-count` is the number of contexts in the task's main retrieval corpus: larger values expand retrieval space and generally raise measurement cost.

3. Sampling strategy. We discretize each fidelity factor into executable levels, then compose fidelity settings from their combinations, and exclude constraint-violating or invalid ones. Since exhaustive measurement is impractical at this scale, we sample via Latin hypercube (McKay, 1992), uniform random, and distance-based (Kaltenecker et al., 2019) sampling to improve coverage per setting.

4. Execution and measurement pipeline. For each sampled configuration-fidelity pair, we deploy the LLM system on a cluster of servers, each node with an Intel Core i9 24 cores CPU, RTX 5090 GPU (32GB VRAM, CUDA 13.0), and 128GB RAM. Every LLM system is profiled on a dedicated server to mitigate hardware noise. The underlying LLMs used are varied, such as LLAMA-8B, DEEPSEEK-R1-14B, and QWEN3-14B etc.

5. Data validation and cleaning. The collected raw records are validated/cleaned to ensure consistency, completeness, and usability. This identifies failed or abnormal executions, filters invalid measurements, and standardizes the records into a unified format for downstream analysis/usage.

The details of LLM systems profiled so far are in Table 1, and more can be found in Appendix G.1.

3.2 ENCLOSED FOLDERS AND DATASETS FORMAT

In `LLMSYS-HPOBench`, the fidelity settings are organized as folders in the directory structure, as shown in Figure 2. Here, `moderate-req1-code_generation` is a setting with a total of three fidelity factors, where each item refers to the value of a factor separated by "-". This particular example means a fidelity setting of tasks with moderate difficulty under one request, and it belongs to a type of code generation task. There is a `.csv` file under each folder of the fidelity setting.

Every `.csv` file contains a standardized tabular format for storing measurements of configurations under a specific fidelity setting. Figure 3 presents an example record for a system under one specific fidelity setting. As can be seen, the columns are (more details can be found in Appendix G.3):

Hyperparameters. These columns record all tunable variables (with prefixes `cfg-ai-` and `cfg-` for AI and non-AI hyperparameters, respectively), with each column is one hyperparameter.

Objective metrics. These columns denote the target task objectives during inference, including quality (e.g., F1-score) and performance (e.g., latency), with a prefix `obj-`. The suffixes "+" and "−" indicate maximization and minimization, respectively. **Cost metrics.** These columns record measurement costs such as token usage and execution duration, indicated by a prefix `cost-`.

Hardware metrics. These columns record hardware statistics such as CPU/GPU utilization and memory usage, with `hw-file` linking to the raw record.

Logs. These columns expose execution traces such as error messages and diagnostic signals, with `log-file` linking to the raw record.

```
AutoGPT/
├── moderate-req1-code_generation/
│   ├── moderate-req1-code_generation.csv
│   ├── hw_file/
│   │   ├── hw-1.txt
│   │   └── ...
│   └── log_file/
│       ├── log-1.txt
│       └── ...
├── simple-req1-code_generation/
├── simple-req2-logic_puzzles/
└── ...
```

Figure 2: Example directory structure for the LLM system AUTOGPT.

cfg-ai-max_seqs	...	obj-throughput+	...	cost-duration(s)	...	hw-file	log-file
1024	...	145.2	...	84.7	...	hw-1.txt	log-1.txt
512	...	176.4	...	78.6	...	hw-2.txt	log-2.txt
256	...	156.2	...	83.4	...	hw-3.txt	log-3.txt
2048	...	198.4	...	67.5	...	hw-4.txt	log-4.txt
1024	...	242.5	...	56.8	...	hw-5.txt	log-5.txt
128	...	256.2	...	89.5	...	hw-6.txt	log-6.txt
1024	...	264.4	...	58.7	...	hw-7.txt	log-7.txt
64	...	175.3	...	74.3	...	hw-8.txt	log-8.txt
1024	...	146.6	...	65.8	...	hw-9.txt	log-9.txt
64	...	185.9	...	88.3	...	hw-10.txt	log-10.txt
1024	...	225.4	...	84.2	...	hw-11.txt	log-11.txt
512	...	245.7	...	82.6	...	hw-12.txt	log-12.txt
256	...	263.3	...	88.4	...	hw-13.txt	log-13.txt
1024	...	274.2	...	44.9	...	hw-14.txt	log-14.txt
512	...	178.8	...	64.7	...	hw-15.txt	log-15.txt
256	...	185.4	...	53.3	...	hw-16.txt	log-16.txt
1024	...	135.4	...	65.7	...	hw-17.txt	log-17.txt
32	...	195.3	...	66.8	...	hw-18.txt	log-18.txt
64	...	147.2	...	67.7	...	hw-19.txt	log-19.txt
1024	...	297.2	...	86.7	...	hw-20.txt	log-20.txt
...	...	...	...	...	...	...	...
512	...	265.4	...	85.6	...	hw-n.txt	log-n.txt

Figure 3: An exampled data of `moderate-req1-code_generation.csv`.

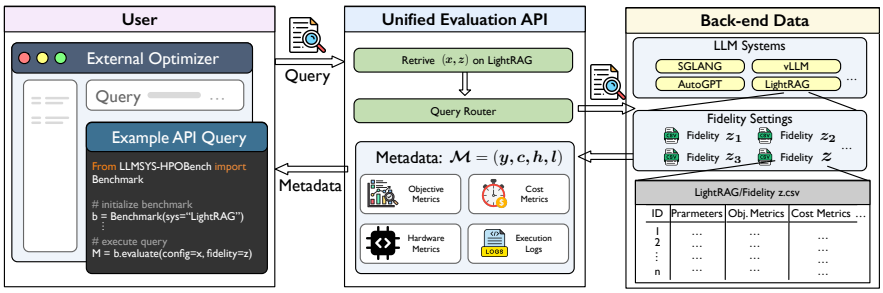


Figure 5: Overview of the `LLMSYS-HPOBench` workflow.

3.3 API ACCESS OF LLMSYS-HPOBENCH

Conveniently, `LLMSYS-HPOBench` is encapsulated as a unified benchmark abstraction with a standardized API. Figure 4 shows how a user selects a hyperparameter configuration $\boldsymbol{x}$ and fidelity setting $\boldsymbol{z}$, obtaining structured metadata $\mathcal{M}$, including the fields in Figure 3. As shown in Figure 5, the interface connects external HPO algorithms and analysis tools with the back-end tabular data. Given $\boldsymbol{x}$ and $\boldsymbol{z}$, a query is routed to the corresponding benchmark record, and the API returns $\mathcal{M}$, directly supporting a wide range of downstream tasks. When a queried hyperparameter configuration is absent from the datasets, `LLMSYS-HPOBench` uses a surrogate model to estimate its objective value and cost. By default, we adopt RANDOM FOREST, as prior evidence shows that it can better reflect outcomes from direct benchmark execution (Eggensperger et al., 2015). Yet, users may specify alternatives through the open interface (e.g., nearest-neighbor). For fair comparison, the adopted surrogate should be kept consistent across optimizers.

```python
from llmsys_hpobench import Benchmark

xbench = Benchmark(system="target_system")
X = xbench.get_config_space()
Z = xbench.get_fidelity_space()
z, spent, budget = Z.sample(), 0, 3600
while spent < budget:
    x = X.sample()
    M = xbench.evaluate(config=x, fidelity=z)
    perf, cost, log = M["perf"], M["cost"], M
        ["log"]
    spent += cost
```

Figure 4: Unified API of `LLMSYS-HPOBench`.

4 HOW TO USE LLMSYS-HPOBENCH? A CASE STUDY

We run an experimental case study that uses `LLMSYS-HPOBench` to assess how optimizers perform on single-/multi-objective LLM system cases and examine how the cost implies the informativeness of a fidelity setting. All studies use the full dataset, where RANDOM FOREST (RF) is used as a surrogate for estimating those unmeasured configurations, if needed. We repeat the experiment with 30 seeds under system-specific budgets that ensure good convergence (using the recorded time/token

Table 2: Final quality/performance under target-fidelity over 30 runs (mean±std; higher is better). Parenthesized numbers are Scott–Knott tiers, e.g., "(1)" is the best statistically indistinguishable tier.

System	HB	BOHB	SMAC	HEBO
NAIVERAG	.6861 ± .0351 (2)	.6783 ± .0344 (2)	.7044 ± .0181 (1)	.6854 ± .0403 (2)
LIGHTRAG	.0664 ± .0046 (3)	.0683 ± .0058 (2)	.0691 ± .0034 (2)	.0729 ± .0040 (1)
VLLM	7.1268 ± .0619 (3)	7.4956 ± .1239 (2)	7.6415 ± .1400 (1)	7.4239 ± .1546 (2)
HTML-RAG	.5810 ± .0140 (1)	.5787 ± .0143 (1)	.5810 ± .0100 (1)	.5862 ± .0109 (1)
OPENHANDS	.9335 ± .0154 (2)	.9384 ± .0206 (2)	.9555 ± .0114 (1)	.9572 ± .0148 (1)
SGLANG	53.178 ± 80.339 (2)	30.166 ± 65.442 (2)	59.087 ± 82.711 (2)	99.133 ± 86.717 (1)
AUTOGPT	.0809 ± .0045 (1)	.0825 ± .0044 (1)	.0823 ± .0028 (1)	.0828 ± .0031 (1)

costs) in Tables 6 of Appendices D.1 and D.2. The target fidelity, objective(s), and cost metric for each system are chosen by that protocol. More details are in Appendices D and F; additional experiments of protocol sensitivity and landscape analysis are reported in Appendices E and F.2.

4.1 SINGLE-OBJECTIVE HPO FOR LLM SYSTEMS

We compare `HB` (Li et al., 2017), `BOHB` (Falkner et al., 2018), `SMAC` (Hutter et al., 2011), and `HEBO` (Cowen-Rivers et al., 2022). `SMAC` and `HEBO` evaluate configurations only at the target fidelity, whereas `HB` and `BOHB` leverage low-fidelity to explore configuration space and promote promising configurations to the target setting. The resource schedules are specified in Appendix F.

Figure 6 compares the anytime trajectory of optimization for target-fidelity and Table 2 gives the corresponding final means, standard deviations, and statistical tiers. Scott–Knott places `SMAC` in the best tier on NAIVERAG and VLLM, `HEBO` on LIGHTRAG and SGLANG, and both optimizers on OPENHANDS. All four optimizers are statistically tied on HTML-RAG and AUTOGPT. The trajectories also show system-dependent convergence: SGLANG has a broad, bimodal throughput range, whereas the optimizers are more tightly clustered on HTML-RAG.

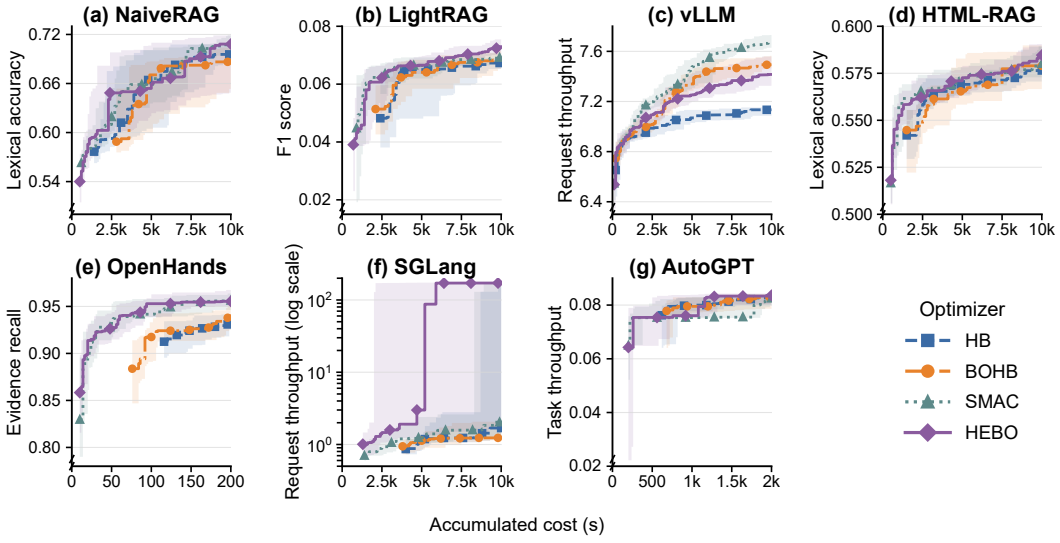


Figure 6: Optimization trajectory under the target-fidelity over 30 runs (median); bands show the interquartile range, and x-axes show evaluation cost. The SGLANG y-axis is log-transformed.

4.2 MULTI-OBJECTIVE HPO FOR LLM SYSTEMS

We evaluate `FlexiBO` (Iqbal et al., 2024), `ParEGO` (Knowles, 2006), `HVKG` (Daulton et al., 2023), `LITE` (Lustosa & Menzies, 2024), and `SWAY` (Chen et al., 2019). Each system has two objectives, and each optimizer is repeated 30 times. The objective pairs, target fidelities, budgets, and HV computation protocol are specified in Appendix D.2. Table 3 reports final hypervolume (HV). The largest mean changes by system: `ParEGO` leads on NAIVERAG, `FlexiBO` on LIGHTRAG, and

Table 3: Final HV values under target-fidelity over 30 runs (mean±std; higher is better). Formate is the same as Table 2.

System	FlexiBO	ParEGO	HVKG	LITE	SWAY
NAIVERAG	.8646 ± .05 (3)	.9223 ± .03 **(1)**	.8855 ± .05 (2)	.8913 ± .07 (2)	.8777 ± .04 (2)
LIGHTRAG	1.0000 ± .00 **(1)**	.8967 ± .14 (2)	.5501 ± .15 (4)	.7006 ± .18 (3)	.6218 ± .15 (4)
VLLM	.5557 ± .01 (2)	.5645 ± .01 (2)	.5460 ± .02 (3)	.5802 ± .01 **(1)**	.5282 ± .03 (4)
HTML-RAG	.3299 ± .08 (2)	.4045 ± .12 **(1)**	.3548 ± .12 (2)	.4144 ± .12 **(1)**	.3543 ± .11 (2)
OPENHANDS	1.0000 ± .00 **(1)**	1.0000 ± .00 **(1)**	1.0000 ± .00 **(1)**	1.0000 ± .00 **(1)**	.9333 ± .21 (2)
SGLANG	1.0000 ± .00 **(1)**	1.0000 ± .00 **(1)**	.9972 ± .01 **(1)**	.9999 ± .00 **(1)**	.5288 ± .49 (2)
AUTOGPT	.9857 ± .02 **(1)**	.9878 ± .01 **(1)**	.9650 ± .05 (2)	.9924 ± .01 **(1)**	.9509 ± .03 (3)

LITE on VLLM, HTML-RAG, and AUTOGPT. Several optimizers reach the same results on OPENHANDS and SGLANG. Indeed, the results might not identify the most cost-efficient optimizer earlier in the run; Appendix D.2 reports HV at 20% of the budget and cost–HV area under the curve.

4.3 RELATIONSHIPS BETWEEN COST AND FIDELITY LEVEL

We compare the cost of each fidelity with its Spearman correlation to the target fidelity, as shown in Figure 7. Overall, unlike standard HPO, we see that **increasing evaluation cost does not necessarily lead to a more faithful approximation of the target fidelity.** For several LLM systems, e.g., AUTOGPT, NAIVERAG, and VLLM, the correlation fluctuates substantially even as the cost increases. Notably, cheaper ones can achieve correlations comparable to or higher than more expensive ones—a similar trend for systems with relatively stable correlations. These reveal complex, non-monotonic cost–fidelity relationships, indicating that evaluation cost alone is not a reliable proxy for fidelity level. More details are in Table 14 of Appendix F.3.

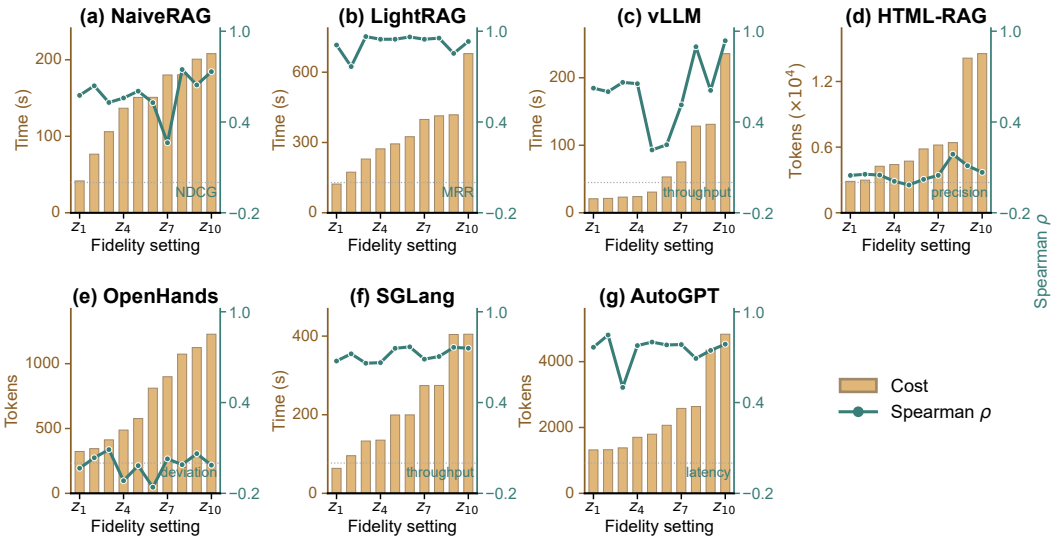


Figure 7: Evaluation cost and Spearman correlation with the target across ten sampled fidelity settings. Bars show cost; lines show Spearman correlation.

5 DISCUSSION

5.1 AUTOML RESEARCH OPPORTUNITIES WITH LLMSYS-HPOBENCH

Beyond serving as a benchmark for algorithm comparison, LLMSYS-HPOBench provides a reusable and convenient empirical resource for broader HPO research on LLM systems in the AutoML community. Below, we discuss these with respect to the characteristics of LLMSYS-HPOBench.

Short-term opportunities. Research directions that can be investigated in the near future include:

High-dimensional non-monotonic fidelity HPO (C2). As we have shown, the high dimensions and complexity of the fidelity factors in LLMSYS-HPOBench can impose non-monotonic and non-linear relationships to the evaluation cost and objective metrics, which might invalidate the existing multi-fidelity HPO algorithms, e.g., Hyperband (Li et al., 2017), BOHB (Falkner et al., 2018), and DEHB (Awad et al., 2021). All those raise the need for more specialized HPO algorithms.

AI and non-AI interaction-aware HPO (C3). The unique understanding of AI and non-AI hyperparameters in LLMSYS-HPOBench creates opportunities and knowledge to tailor HPO algorithm, e.g., in RAG, studies have shown that the AI and non-AI hyperparameters can be optimized sequentially with discriminative interactions for better quality (Chen & Chen, 2026).

Many-objective HPO (C4). Given the complexity of LLM systems, LLMSYS-HPOBench contains up to nine objective metrics to be simultaneously considered, which is often not the case for classic HPO scenarios. This additional high dimensionality, together with their diverse requirements (Wang & Chen, 2026a;b; Chen & Li, 2023), is known to be challenging (Li et al., 2022).

Cost- and hardware-sensitive HPO (C5–C6). LLMSYS-HPOBench's fine-grained per-configuration cost and hardware data allows HPO to incorporate these signals, which vary widely: e.g., RAG retrieval depth can double token costs and runtime (Barker et al., 2025), while batching or KV-cache choices can drastically shift GPU/memory use (Bambhaniya et al., 2026). Discriminating configurations by multi-dimensional cost/hardware measurements thus enables cost- and hardware-sensitive HPO, especially under multi-fidelity settings.

Intermediate-term opportunities. Research problems that require some effort to explore/define:

Multi-modal LLM-based HPO algorithm creation (C7). LLMSYS-HPOBench contains the execution logs from the LLM systems, which can be used as part of the modality of text prompt and optimization data to instruct a LLM for automatically creating new HPO algorithms. This is also an active topic for automated general algorithm creations (Liu et al., 2026).

Explainable HPO (C7). The log information in LLMSYS-HPOBench enables one to understand the internal working mechanism of the LLM system that goes beyond the individual LLM and/or the typical system components (e.g., a tool or vector database). This also provides opportunities for the explainability of HPO behaviors.

Long-term opportunities. Directions that have yet reached any established foundation are:

Empirical study via fitness landscape analysis (C1–C4). With the large number of cases/environments available in LLMSYS-HPOBench, empirical studies can be conducted to understand the domain knowledge of LLM systems via fitness landscape analysis (Pitzer & Affenzeller, 2012; Ye et al., 2026), stimulating new directions of landscape-driven HPO.

Industry and real-world uptakes (C1). The real-world property of LLMSYS-HPOBench ensures that any HPO algorithms built on it can be applied to the practice without many barriers.

5.2 LLMSYS-HPOBench as a Live Benchmark Suite

Given the easy-to-extend portal available in LLMSYS-HPOBench, it allows further extension by designs. As a live benchmark, it grows by incorporating both additional samples for existing systems and data for new LLM systems (see Appendix G.2). Further to our efforts, we also design a systematic protocol inviting the community's contributions of data/suggestions for LLMSYS-HPOBench.

6 Conclusion

In this paper, we present LLMSYS-HPOBench, the first open-sourced live benchmark for HPO of LLM systems. LLMSYS-HPOBench is unique in the sense that it contains a mix of hyperparameter configuration space of both AI (e.g., LLM) and non-AI components (e.g., memory or vector database) under varying fidelity factors that exhibit non-monotonic and nonlinear relationships to cost and inference objectives, together with measurement cost of individual configuration, hardware metrics, and logs. We release LLMSYS-HPOBench as a readily usable benchmark suite, hoping that it would play a non-trivial role in shaping the future research of AutoML in the LLM era.

AI USE STATEMENT

In this work, we did not use generative AI tools for any tasks with required disclosure. We used generative AI tools to identify relevant literature and to edit the paper to improve readability, which are tasks with recommended disclosure. We have reviewed all AI-assisted work. The identified literature was manually verified by the authors, and all AI-assisted edits to the paper were reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or sensitive personal data.

REPRODUCIBILITY STATEMENT

The benchmark suite, datasets, and source code are publicly available through our anonymous repository at `https://anonymous.4open.science/r/llmsys-hpobench-CD2B`. Section 3 describes the construction procedure, data organization, and unified API of `LLMSYS-HPOBench`, with further details of the benchmarked LLM systems and datasets provided in Appendix G. The experimental protocols, including evaluation settings, budgets, and optimizer configurations, are documented in Appendices D and F. Together, these materials provide the information and artifacts needed to reproduce the benchmark construction and the empirical studies presented in this work.

REFERENCES

Sharegpt conversation dataset. `https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered`, 2025. Accessed: 2026-05-05.

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. VIDUR: A large-scale simulation framework for LLM inference. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024. URL `https://proceedings.mlsys.org/paper_files/paper/2024/hash/b74a8de47d2b3c928360e0a011f48351-Abstract-Conference.html`.

AutoGPT Contributors. AutoGPT. `https://github.com/Significant-Gravitas/AutoGPT`, 2023. GitHub repository. Accessed: 2024.

Noor H. Awad, Neeratyoy Mallik, and Frank Hutter. DEHB: evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 2147–2153. ijcai.org, 2021. doi: 10.24963/IJCAI.2021/296. URL `https://doi.org/10.24963/ijcai.2021/296`.

Abhimanyu Rajeshkumar Bambhaniya, Hanjiang Wu, Suvinay Subramanian, Sudarshan Srinivasan, Souvik Kundu, Amir Yazdanbakhsh, Midhilesh Elavazhagan, Madhu Kumar, Minlan Yu, Arijit Raychowdhury, and Tushar Krishna. Mist: A co-design framework for heterogeneous, multi-stage llm inference, 2026. URL `https://arxiv.org/abs/2504.09775`.

Archit Bansal, Danny Stoll, Maciej Janowski, Arber Zela, and Frank Hutter. Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/fd78f2f65881c1c7ce47e26b040cf48f-Abstract-Datasets_and_Benchmarks.html`.

Matthew Barker, Andrew Bell, Evan Thomas, James Carr, Thomas Andrews, and Umang Bhatt. Faster, cheaper, better: Multi-objective hyperparameter optimization for LLM and RAG systems. *CoRR*, abs/2502.18635, 2025. doi: 10.48550/ARXIV.2502.18635. URL https://doi.org/10.48550/arXiv.2502.18635.

Paul Brookes, Vardan K. Voskanyan, Rafail Giavrimis, Matthew Truscott, Mina Ilieva, Chrystalla Pavlou, Andrei Staicu, Manal T. Adham, Will Evers-Hood, Jingzhi Gong, Kejia Zhang, Matvey Fedoseev, Vishal Sharma, Roman Bauer, Zheng Wang, Hema Nair, Wei Jie, Tianhua Xu, Aurora Constantin, Leslie Kanthan, and Michail Basios. Evolving excellence: Automated optimization of llm-based agents. *ArXiv*, abs/2512.09108, 2025. URL https://api.semanticscholar.org/CorpusID:283721557.

Rong Cao, Liang Bao, Chase Q. Wu, Panpan Zhangsun, Yufei Li, and Zhe Zhang. CM-CASL: comparison-based performance modeling of software systems via collaborative active and semisupervised learning. *J. Syst. Softw.*, 201:111686, 2023. doi: 10.1016/J.JSS.2023.111686. URL https://doi.org/10.1016/j.jss.2023.111686.

Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. "Sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering*, 45(6):597–614, 2019.

Pengzhou Chen and Tao Chen. Cds4rag: Cyclic dual-sequential hyperparameter optimization for rag. In *Proceedings of the 35th International Joint Conference on Artificial Intelligence, IJCAI 2026, Bremen, Germany, 15-21 August 2026*. ijcai.org, 2026.

Pengzhou Chen, Tao Chen, and Miqing Li. MMO: meta multi-objectivization for software configuration tuning. *IEEE Trans. Software Eng.*, 50(6):1478–1504, 2024. doi: 10.1109/TSE.2024.3388910. URL https://doi.org/10.1109/TSE.2024.3388910.

Tao Chen and Miqing Li. Multi-objectivizing software configuration tuning. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (eds.), *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pp. 453–465. ACM, 2021. doi: 10.1145/3468264.3468555. URL https://doi.org/10.1145/3468264.3468555.

Tao Chen and Miqing Li. Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives. *ACM Trans. Softw. Eng. Methodol.*, 32(3):68:1–68:41, 2023. doi: 10.1145/3571853. URL https://doi.org/10.1145/3571853.

Tao Chen and Miqing Li. Adapting multi-objectivized software configuration tuning. *Proc. ACM Softw. Eng.*, 1(FSE):539–561, 2024. doi: 10.1145/3643751. URL https://doi.org/10.1145/3643751.

Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. FEMOSAA: feature-guided and knee-driven multi-objective optimization for self-adaptive software. *ACM Trans. Softw. Eng. Methodol.*, 27(2):5:1–5:50, 2018. doi: 10.1145/3204459. URL https://doi.org/10.1145/3204459.

HtmlRAG Contributors. HtmlRAG. https://github.com/plageon/HtmlRAG, 2024. GitHub repository. Accessed: 2024.

Alexander Imani Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan-Rhys Griffiths, Alexandre Max Maraval, Jianye HAO, Jun Wang, Jan Peters, and Haitham Bou Ammar. HEBO: Pushing the limits of sample-efficient hyperparameter optimisation. In *First Conference on Automated Machine Learning (Journal Track)*, 2022. URL https://openreview.net/forum?id=SNqiALtOvyk.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Hypervolume knowledge gradient: A lookahead approach for multi-objective bayesian optimization with partial information. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 7167–7204, 2023.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Qiang Yang and Michael J. Wooldridge (eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 3460–3468. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/487.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Evaluating large language models in class-level code generation. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pp. 81:1–81:13. ACM, 2024. doi: 10.1145/3597503.3639219. URL https://doi.org/10.1145/3597503.3639219.

Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor H. Awad, Marius Lindauer, and Frank Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for HPO. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/93db85ed909c13838ff95ccfa94cebd9-Abstract-round2.html.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, pp. 1436–1445. PMLR, 2018. URL http://proceedings.mlr.press/v80/falkner18a.html.

Pieter Gijsbers, Marcos L. P. Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. AMLB: an automl benchmark. *J. Mach. Learn. Res.*, 25: 101:1–101:65, 2024. URL https://jmlr.org/papers/v25/22-0493.html.

Jingzhi Gong and Tao Chen. Predicting software performance with divide-and-learn. In Satish Chandra, Kelly Blincoe, and Paolo Tonella (eds.), *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pp. 858–870. ACM, 2023. doi: 10.1145/3611643.3616334. URL https://doi.org/10.1145/3611643.3616334.

Jingzhi Gong and Tao Chen. Predicting configuration performance in multiple environments with sequential meta-learning. *Proc. ACM Softw. Eng.*, 1(FSE):359–382, 2024. doi: 10.1145/3643743. URL https://doi.org/10.1145/3643743.

Jingzhi Gong, Tao Chen, and Rami Bahsoon. Dividable configuration performance learning. *IEEE Trans. Software Eng.*, 51(1):106–134, 2025. doi: 10.1109/TSE.2024.3491945. URL https://doi.org/10.1109/TSE.2024.3491945.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and fast retrieval-augmented generation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10746–10761, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.568. URL https://aclanthology.org/2025.findings-emnlp.568/.

Huong Ha and Hongyu Zhang. Deepperf: performance prediction for configurable software with deep sparse neural network. In Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (eds.), *Proceedings*

of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 1095–1106. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00113. URL https://doi.org/10.1109/ICSE.2019.00113.

HKUDS. LightRAG. https://github.com/HKUDS/LightRAG, 2024. GitHub repository. Accessed: 2024.

Yi-Qi Hu, Yang Yu, Wei-Wei Tu, Qiang Yang, Yuqiang Chen, and Wenyuan Dai. Multi-fidelity automatic hyper-parameter tuning via transfer series expansion. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 3846–3853. AAAI Press, 2019. doi: 10.1609/AAAI.V33I01.33013846. URL https://doi.org/10.1609/aaai.v33i01.33013846.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello (ed.), *Learning and Intelligent Optimization*, pp. 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25566-3.

Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. Flexibo: A decoupled cost-aware multi-objective optimization approach for deep neural networks (abstract reprint). In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 22700. AAAI Press, 2024. doi: 10.1609/AAAI.V38I20.30600. URL https://doi.org/10.1609/aaai.v38i20.30600.

Song Jin, Shuqi Li, Shukun Zhang, and Rui Yan. Finrpt: Dataset, evaluation system and llm-based multi-agent framework for equity research report generation. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 507–515. AAAI Press, 2026. doi: 10.1609/AAAI.V40I1.37014. URL https://doi.org/10.1609/aaai.v40i1.37014.

Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. Distance-based sampling of software configuration spaces. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 1084–1094. IEEE, 2019.

Jiin Kim, Byeong-Gon Shin, Jin-Won Chung, and Minsoo Rhu. The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective. *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–16, 2025. URL https://api.semanticscholar.org/CorpusID:279244186.

Woo Kyung Kim, Youngseok Lee, Jooyoung Kim, and Honguk Woo. Llm-based skill diffusion for zero-shot policy adaptation. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/0ccd800d59e94679246ec79d4b19587e-Abstract-Conference.html.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In Aarti Singh and Xiaojin (Jerry) Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, Proceedings of Machine Learning Research, pp. 528–536. PMLR, 2017. URL http://proceedings.mlr.press/v54/klein17a.html.

Joshua Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1): 50–66, 2006.

Rahul Krishna, Vivek Nair, Pooyan Jamshidi, and Tim Menzies. Whence to learn? transferring knowledge in configurable systems using BEETLE. *IEEE Trans. Software Eng.*, 47(12):2956–2972, 2021. doi: 10.1109/TSE.2020.2983927. URL https://doi.org/10.1109/TSE.2020.2983927.

Anastasia Krithara, Anastasios Nentidis, Konstantinos Bougiatiotis, and Georgios Paliouras. Bioasq-qa: A manually curated corpus for biomedical question answering. *Scientific Data*, 10:170, 2023. URL https://doi.org/10.1038/s41597-023-02068-4.

LangChain AI. NaiveRAG. https://github.com/langchain-ai/langchain, 2024. GitHub repository. Accessed: 2024.

Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. Input sensitivity on the performance of configurable systems an empirical study. *J. Syst. Softw.*, 201:111671, 2023. doi: 10.1016/J.JSS.2023.111671. URL https://doi.org/10.1016/j.jss.2023.111671.

Ke Li, Zilin Xiang, Tao Chen, Shuo Wang, and Kay Chen Tan. Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: an empirical study. In Gregg Rothermel and Doo-Hwan Bae (eds.), *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pp. 566–577. ACM, 2020. doi: 10.1145/3377811.3380360. URL https://doi.org/10.1145/3377811.3380360.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017. URL https://jmlr.org/papers/v18/16-558.html.

Miqing Li, Tao Chen, and Xin Yao. How to evaluate solutions in pareto-based search-based software engineering: A critical review and methodological guidance. *IEEE Trans. Software Eng.*, 48(5): 1771–1799, 2022. doi: 10.1109/TSE.2020.3036108. URL https://doi.org/10.1109/TSE.2020.3036108.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 37090–37098. AAAI Press, 2026. doi: 10.1609/AAAI.V40I43.41038. URL https://doi.org/10.1609/aaai.v40i43.41038.

Andre Lustosa and Tim Menzies. Learning from very little data: On the value of landscape analysis for predicting software project health. *ACM Transactions on Software Engineering and Methodology*, 33(3):58:1–58:22, 2024.

Michael D McKay. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceedings of the 24th conference on Winter simulation*, pp. 557–564, 1992.

Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. Nas-bench-suite: NAS evaluation is (now) surprisingly easy. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=0DLwqQLmqV.

Tim Menzies, Tao Chen, Yulong Ye, Kishan Kumar Ganguly, Amirali Rayegan, Srinath Srinivasan, and Andre Lustosa. MOOT: a repository of many multi-objective optimization tasks. *IEEE Mining Software Repositories (MSR) Conference*, 2026. doi: 10.48550/ARXIV.2511.16882. URL https://doi.org/10.48550/arXiv.2511.16882.

Stefan Mühlbauer, Florian Sattler, Christian Kaltenecker, Johannes Dorn, Sven Apel, and Norbert Siegmund. Analysing the impact of workloads on modeling the performance of configurable software systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2085–2097. IEEE, 2023.

Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. Finding faster configurations using FLASH. *IEEE Trans. Software Eng.*, 46(7):794–811, 2020. doi: 10.1109/TSE.2018.2870895. URL https://doi.org/10.1109/TSE.2018.2870895.

OpenHands Contributors. OpenHands. https://github.com/OpenHands/OpenHands, 2024. GitHub repository. Accessed: 2024.

Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. YAHPO gym - an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In Isabelle Guyon, Marius Lindauer, Mihaela van der Schaar, Frank Hutter, and Roman Garnett (eds.), *International Conference on Automated Machine Learning, AutoML 2022, 25-27 July 2022, Johns Hopkins University, Baltimore, MD, USA*, Proceedings of Machine Learning Research, pp. 3/1–39. PMLR, 2022. URL https://proceedings.mlr.press/v188/pfisterer22a.html.

Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/ec8956637a99787bd197eacd77acce5e-Abstract-round2.html.

Erik Pitzer and Michael Affenzeller. A comprehensive survey on fitness landscape analysis. *Recent advances in intelligent engineering systems*, pp. 161–191, 2012.

SGLang Contributors. SGLang. https://github.com/sgl-project/sglang, 2023. GitHub repository. Accessed: 2024.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-acl.317. URL https://aclanthology.org/2021.findings-acl.317/.

Jiejun Tan, Zhicheng Dou, Wen Wang, Mang Wang, Weipeng Chen, and Ji-Rong Wen. Htmlrag: HTML is better than plain text for modeling retrieved knowledge in RAG systems. In Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov (eds.), *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pp. 1733–1746. ACM, 2025. doi: 10.1145/3696410.3714546. URL https://doi.org/10.1145/3696410.3714546.

Jacob O Tørring, Carl Hvarfner, Luigi Nardi, and Magnus Själander. Catbench: A compiler autotuning benchmarking suite for black-box optimization. In *International Conference on Automated Machine Learning*, pp. 24–1. PMLR, 2025.

vLLM Contributors. vLLM. https://github.com/vllm-project/vllm, 2023. GitHub repository. Accessed: 2024.

Jiawei Wang, Renhe Jiang, Chuang Yang, Zengqing Wu, Makoto Onizuka, Ryosuke Shibasaki, Noboru Koshizuka, and Chuan Xiao. Large language models as urban residents: An LLM agent framework for personal mobility generation. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e142fd2b70f10db2543c64bca1417de8-Abstract-Conference.html.

Shihai Wang and Tao Chen. Light over heavy: Automated performance requirements quantification with linguistic inducement. In *48th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 2026a.

Shihai Wang and Tao Chen. Conjecture and inquiry: Quantifying software performance requirements via interactive retrieval-augmented preference elicitation. In *Findings: Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL, 2026b.

Max Weber, Christian Kaltenecker, Florian Sattler, Sven Apel, and Norbert Siegmund. Twins or false friends? A study on energy consumption and performance of configurable software. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pp. 2098–2110. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00177. URL https://doi.org/10.1109/ICSE48619.2023.00177.

Edward D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990.

Zezhen Xiang, Jingzhi Gong, and Tao Chen. Dually hierarchical drift adaptation for online configuration performance learning. In *48th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 2026.

Gangda Xiong and Tao Chen. Cotune: Co-evolutionary configuration tuning. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*, pp. 1490–1502. IEEE, 2025. doi: 10.1109/ASE63991.2025.00126. URL https://doi.org/10.1109/ASE63991.2025.00126.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Yulong Ye, Tao Chen, and Miqing Li. Distilled lifelong self-adaptation for configurable systems. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 1333–1345. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00094. URL https://doi.org/10.1109/ICSE55347.2025.00094.

Yulong Ye, Hongyuan Liang, Chao Jiang, Miqing Li, and Tao Chen. Revealing domain-spatiality patterns for configuration tuning: Domain knowledge meets fitness landscapes. *ACM Trans. Softw. Eng. Methodol.*, March 2026. ISSN 1049-331X. doi: 10.1145/3803859. URL https://doi.org/10.1145/3803859. Just Accepted.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, pp. 7105–7114. PMLR, 2019. URL http://proceedings.mlr.press/v97/ying19a.html.

Jin Zhou, Christian Belardi, Ruihan Wu, Travis Zhang, Carla Gomes, Wen Sun, and Kilian Weinberger. On speeding up language model evaluation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65092–65111, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a38cfba4c717e5a37b200294615e546e-Paper-Conference.pdf.

Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pp. 338–350, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350280. doi: 10.1145/3127479.3128605. URL https://doi.org/10.1145/3127479.3128605.

APPENDIX

A LIMITATIONS

LLMSYS-HPOBench is designed to provide a realistic and extensible benchmark for LLM systems HPO, but its current release inevitably reflects several construction trade-offs. The main limitations are summarized below.

- **LLM system coverage.** The seven systems were selected to cover major system types, widely adopted open-source projects, and non-trivial configuration, fidelity, objective, and cost spaces. The current release contains three RAG systems, two inference engines, and two agent frameworks, but does not yet cover multi-agent frameworks, orchestrators, standalone vector-search systems, multimodal systems, or the full range of model families and deployment settings. We therefore position it as a representative and extensible starting point rather than exhaustive coverage of the LLM deployment stack.
- **Fidelity space coverage.** To keep large-scale profiling tractable, the fidelity space is instantiated through a reduced, finite set of discretized settings. The selected levels balance resource constraints against the need to expose changes in system behavior across fidelity settings; finer-grained levels may reveal additional behavior and can be introduced in later versioned releases.
- **Hyperparameter space coverage.** The hyperparameter configurations in LLMSYS-HPOBench are constructed through sampling, which does not cover the full space. Yet, since exhaustive measurements are impractical for real-world LLM systems at scale, the ensembles of sampling methods used therein are common practice to ensure representativeness. Further, given the intuitive/easy-to-maintain structure in LLMSYS-HPOBench, more data can be easily appended in phases.
- **Measurement noises.** Time-related metrics would inevitably be subject to noise from the infrastructure and hardware. Yet, our data cleaning and curation process has ensured that the abnormal samples are removed, and that the measurements are conducted repeatably to preserve reliability.
- **Hardware generalization.** Timing and hardware metrics are specific to the profiling platform. All configurations of a system are measured under the same fixed hardware/software environment to preserve internally consistent comparisons. Measurements collected on new hardware are released as separate versioned profiles rather than being treated as interchangeable with the existing data.

Table 4: Comparing LLMSYS-HPOBench with existing HPO and system benchmarks.

Benchmark	Domain	#HP (AI)	#HP (non-AI)	# $\mathcal{F}$	# Obj.	# Cost	HW metrics?	Logs?
NAS-Bench-101 (2019)	NAS	26	0	1	3	1	✗	✗
NAS-Bench-201 (2020)	NAS	6	0	1	6	2	✗	✗
HPO-B (2021)	HP	1-53	0	✗	1	✗	✗	✗
HPOBench (2021)	HP/NAS	2-26	0	1-2	1-4	2	✗	✗
YAHPO Gym (2022)	HP	2-38	0	1	2-12	1-5	✗	✗
JAHS-Bench-201 (2022)	HP/NAS	10	0	4	10	4	✗	✗
Mühlbauer et al. (2023)	System	0	11-33	1	1	✗	✗	✗
Nair et al. (2020)	System	0	3-39	✗	1-2	✗	✗	✗
Krishna et al. (2021)	System	0	12-57	1	1	✗	✗	✗
Weber et al. (2023)	System	0	2-36	✗	2-4	1	✗	✗
MOOT* (2026)	System	0	3-39	1	1-3	✗	✗	✗
LLMSYS-HPOBench (ours)	LLM system	5-8	4-17	3-5	3-9	2-10	✓	✓

#**Objectives:** No. of objectives; #**HP:** No. of hyperparameters; #$\mathcal{F}$**:** Dimension of fidelity factors. #**Cost:** No. of cost metrics.

B Detailed Comparisons to Existing Benchmarks

Table 4 provides an aspect-wise comparison between LLMSYS-HPOBench and representative HPO/system benchmarks. The main differences can be summarized as follows:

- **Targeted systems/models.** Unlike conventional HPO benchmarks for ML/NAS and system benchmarks for traditional software systems, LLMSYS-HPOBench focuses on real-world end-to-end LLM systems, which is a synergy of both.
- **Hyperparameters.** Prior benchmarks typically expose either AI or non-AI hyperparameters only, but not both; LLMSYS-HPOBench jointly covers both, with heterogeneous spaces spanning 5–8 AI hyperparameters and 4–17 non-AI hyperparameters, respectively.
- **Fidelity factors.** Existing HPO benchmarks typically support one to two fidelity factors, often instantiated through resource variables such as epochs or data fraction. System benchmarks usually provide no explicit fidelity abstraction, or only a single workload-related factor. By comparison, LLMSYS-HPOBench exposes richer fidelity settings, with 3–5 dimensional fidelity spaces.
- **Objective metrics.** Existing HPO and system benchmarks often contain only a limited set of objective metrics, while LLMSYS-HPOBench records up to nine objective metrics, enabling an in-depth analysis of trade-offs in LLM systems HPO. Only two existing benchmarks consider higher dimensions of objectives than LLMSYS-HPOBench.
- **Cost metrics.** Prior benchmarks generally provide sparse or no explicit cost metrics. By comparison, LLMSYS-HPOBench records up to 10 cost metrics, which supports more systematic study of cost-aware optimization.
- **Hardware metrics.** Hardware metrics are rarely included in existing HPO or system benchmarks. LLMSYS-HPOBench explicitly records hardware metrics for each evaluation, providing an additional view of execution behavior beyond objective and cost metrics.
- **Logs.** Execution logs are typically unavailable in prior benchmarks. LLMSYS-HPOBench well records logs for each evaluation, supporting inspection and explainable analysis.

C Detailed Setup of the Case Study

The case study is conducted on a standard machine with an Intel Core i9-13900H CPU with 14 cores, 20 threads and 32 GB RAM. For an experiment run that requires at least 110 minutes for a HPO algorithm to run on the real LLM system, using LLMSYS-HPOBench merely needs a few seconds to several tens of seconds to complete. This significantly reduces the efforts required and hence expedite the research progress/discovery in the field.

The considered HPO algorithms in the case study are specified below. Note that for all algorithms, we use their default hyperparameter values.

- **Hyperband (HB) (Li et al., 2017):** HB is a multi-fidelity optimizer that improves search efficiency via adaptive resource allocation. It measures many configurations under low-fidelity settings with small budgets, prunes poorly performing candidates early, and progressively allocates more resources to promising configurations.
- **BOHB (Falkner et al., 2018):** BOHB combines Bayesian optimization (BO) with HB-style resource allocation for multi-fidelity optimization. It preserves the early-stopping mechanism of HB while using a probabilistic model to guide the sampling of promising configurations, thereby improving both sample efficiency and budget utilization.
- **SMAC** (Hutter et al., 2011): SMAC is a sequential model-based optimizer that uses a surrogate model, typically based on random forests, to estimate configuration performance. It selects new configurations by optimizing an acquisition function, making it well-suited for expensive and mixed-type hyperparameter optimization.
- **HEBO** (Cowen-Rivers et al., 2022): HEBO is a robust BO method for noisy and heterogeneous search spaces. It improves standard BO with input/output transformations, heteroscedastic uncertainty modeling, and evolutionary acquisition optimization, making it effective for mixed-type hyperparameter configuration spaces.

D Evaluation Protocol and Additional Results

D.1 Single-objective Evaluation Protocol

Table 6 summarizes the target objective, cost, and total budget for each system. The target fidelity shared by the single- and multi-objective evaluations is listed separately in Table 5. The experimental settings for single-objective comparison in Section 4.1 are summarized in Table 6.

Table 5: Target fidelity setting shared by the single- and multi-objective evaluations.

System	Target fidelity setting
NaiveRAG	`0.2-2-easy-agriculture`
LightRAG	`bridge-2-5-10`
vLLM	`15.0-2.0-32-200-r2`
HTML-RAG	`2-025-10`
OpenHands	`18-2-5-6`
SGLang	`inf-2.0-32-64-4096`
AutoGPT	`multi_stage-req4-code_generation`

Table 6: The protocol for single-objective comparison. A leading "+" denotes maximization.

System	Target objective	Cost	Budget (s)	Seeds
NaiveRAG	+lexical_ac	total_time	10,000	30
LightRAG	+f1_score	total_time	10,000	30
vLLM	+request_throughput	benchmark_duration_s	10,000	30
HTML-RAG	+lexical_ac	total_time	10,000	30
OpenHands	+evidence_recall	total_time_sec	200	30
SGLang	+request_throughput	duration	10,000	30
AutoGPT	+throughput_tasks_per_sec	total_duration	2,000	30

D.2 Multi-objective Evaluation Protocol and Anytime Results

Table 7 specifies the two objectives, evaluation-cost metric, and budget used for each multi-objective task. Both studies use the target fidelities in Table 5. A leading "+" denotes maximization and a leading "−" denotes minimization.

Table 7: System-specific protocol for multi-objective HPO.

System	Objective 1	Objective 2	Evaluation cost	Budget (s)
NaiveRAG	+lexical_ac	+ndcg	total_time	10,000
LightRAG	+f1_score	+ndcg	total_time	10,000
vLLM	+request_throughput	−p95_latency_ms	benchmark_duration_s	10,000
HTML-RAG	+lexical_ac	+answer_f1	total_time	10,000
OpenHands	+evidence_recall	+accuracy	total_time_sec	500
SGLang	+request_throughput	−mean_e2e_latency_ms	duration	10,000
AutoGPT	+throughput_tasks_per_sec	−p95_task_duration	total_duration	2,000

Each optimizer is evaluated over 30 seeds. For HV calculation, both objectives are oriented so that larger values are better and min–max normalized using the complete target-fidelity table; the reference point is $(0, 0)$.

To complement the final-budget comparison, we further examine optimizer performance under limited evaluation cost. Specifically, we report the best method at 20% of the total budget (HV@20%) to capture early-stage performance, together with the cost–HV area under the curve (AUC) to summarize performance throughout the tuning process. Table 8 presents these results alongside the best final-HV method.

The results reveal noticeable differences across evaluation stages. The method achieving the best final HV is often not the strongest early in the search; for example, such mismatches arise on LightRAG

and AUTOGPT. ParEGO achieves the best cost–HV AUC on six of the seven systems, indicating consistently strong performance over the optimization trajectory, although it is not always the final winner. These results suggest that full-budget performance alone can obscure important differences in how effectively optimizers utilize the available evaluation budget.

Table 8: Quality–cost trade-offs in multi-objective HPO. HV@20% measures early quality; AUC summarizes anytime cost efficiency.

System	Best HV@20%	Best cost–HV AUC	Best final HV	Pattern
NAIVERAG	ParEGO (.7237)	ParEGO (.7949)	ParEGO (.9223)	Consistent winner
LIGHTRAG	LITE (.3991)	ParEGO (.6521)	FlexiBO (1.0000)	Early–final mismatch
VLLM	ParEGO (.5590)	LITE (.5572)	LITE (.5802)	Small method gap
HTML-RAG	ParEGO/HVKG (.2529)	ParEGO (.3131)	LITE (.4144)	Close competition
OPENHANDS	FlexiBO (1.0000)	ParEGO (.9078)	Four methods (1.0000)	Ceiling behavior
SGLANG	Four methods (.2668)	ParEGO (.6122)	FlexiBO/ParEGO (1.0000)	SWAY weaker at final
AUTOGPT	SWAY (.8210)	ParEGO (.8758)	LITE (.9924)	Early–final mismatch

E ROBUSTNESS ANALYSIS TO SAMPLING DENSITY

To examine whether benchmark sparsity affects algorithm evaluation, we treat the collected dataset at 100% density as the reference and construct coverage-preserving subsets at 20%, 40%, 60%, and 80% density. We then evaluate all compared algorithms on each density for 30 independent runs. This analysis simulates progressively sparser benchmark construction while controlling the underlying configuration space, allowing us to assess how sampling density influences comparative conclusions. Table 9 reports the detailed performance results across densities.

Table 10 further compares the algorithm rankings at each reduced density against those obtained at 100% density. Overall, the rankings remain largely consistent: the median Spearman correlation is 0.9, and 26 of the 28 reduced-density comparisons achieve $\rho \geq 0.7$. Nevertheless, the mean-best method changes in 12 of 28 comparisons, indicating that identifying the single best-performing algorithm is more sensitive to sampling density than recovering the overall ranking. These results suggest that sparse sampling generally preserves broad comparative trends, while fine-grained conclusions about the top-performing method warrant greater caution.

Table 9: Final performance across sampling densities over 30 seeds (mean±std). Parenthesized numbers are Scott–Knott tiers.

System	Density (p)	HB	BOHB	SMAC	HEBO
NAIVERAG	20%	.7048 ± .0135 (3)	.7172 ± .0198 (2)	.7357 ± .0106 (1)	.7045 ± .0165 (3)
	40%	.7082 ± .0099 (3)	.7205 ± .0259 (2)	.7325 ± .0129 (1)	.7051 ± .0218 (3)
	60%	.7124 ± .0071 (3)	.7215 ± .0197 (2)	.7317 ± .0082 (1)	.7110 ± .0217 (3)
	80%	.7109 ± .0074 (2)	.7315 ± .0099 (1)	.7295 ± .0067 (1)	.7102 ± .0237 (2)
	100%	.7080 ± .0091 (3)	.7226 ± .0158 (2)	.7305 ± .0082 (1)	.7138 ± .0314 (3)
LIGHTRAG	20%	.0692 ± .0053 (3)	.0695 ± .0044 (3)	.0847 ± .0079 (1)	.0755 ± .0078 (2)
	40%	.0707 ± .0022 (4)	.0711 ± .0037 (4)	.0809 ± .0081 (1)	.0752 ± .0065 (2)
	60%	.0702 ± .0025 (3)	.0698 ± .0042 (3)	.0760 ± .0068 (1)	.0723 ± .0035 (2)
	80%	.0705 ± .0025 (2)	.0704 ± .0048 (2)	.0754 ± .0054 (1)	.0716 ± .0031 (2)
	100%	.0715 ± .0041 (4)	.0716 ± .0036 (4)	.0790 ± .0066 (2)	.0828 ± .0059 (1)
VLLM	20%	7.0047 ± .0547 (4)	7.1275 ± .0640 (3)	7.2803 ± .0448 (1)	7.2438 ± .0538 (2)
	40%	7.1539 ± .0289 (4)	7.2198 ± .0480 (3)	7.3041 ± .0290 (1)	7.2911 ± .0164 (2)
	60%	7.1460 ± .0872 (4)	7.5723 ± .0998 (3)	7.8313 ± .0403 (1)	7.7191 ± .0927 (2)
	80%	7.2112 ± .0674 (4)	7.6194 ± .0797 (3)	7.8273 ± .0442 (1)	7.7107 ± .1052 (2)
	100%	7.2006 ± .0722 (4)	7.5722 ± .0920 (3)	7.7668 ± .0900 (1)	7.6542 ± .1161 (2)
HTML-RAG	20%	.5890 ± .0077 (3)	.5984 ± .0091 (2)	.6155 ± .0102 (1)	.6103 ± .0121 (1)
	40%	.5891 ± .0115 (2)	.5934 ± .0124 (2)	.6118 ± .0113 (1)	.6091 ± .0112 (1)
	60%	.5929 ± .0091 (3)	.5963 ± .0131 (3)	.6253 ± .0118 (1)	.6160 ± .0118 (2)
	80%	.5921 ± .0086 (2)	.5939 ± .0108 (2)	.6102 ± .0133 (1)	.6134 ± .0110 (1)
	100%	.5879 ± .0120 (2)	.5925 ± .0114 (2)	.6100 ± .0121 (1)	.6082 ± .0109 (1)
OPENHANDS	20%	.9821 ± .0075 (3)	.9845 ± .0074 (3)	.9961 ± .0032 (1)	.9966 ± .0028 (1)
	40%	.9724 ± .0119 (4)	.9778 ± .0086 (3)	.9919 ± .0055 (1)	.9941 ± .0051 (1)
	60%	.9573 ± .0119 (3)	.9603 ± .0121 (3)	.9779 ± .0095 (1)	.9820 ± .0071 (1)
	80%	.9631 ± .0127 (3)	.9665 ± .0126 (3)	.9853 ± .0084 (1)	.9860 ± .0050 (1)
	100%	.9540 ± .0143 (3)	.9600 ± .0138 (3)	.9816 ± .0098 (1)	.9795 ± .0089 (1)
SGLANG	20%	91.626 ± 57.728 (4)	87.724 ± 58.561 (4)	141.647 ± 7.273 (1)	131.602 ± 21.775 (2)
	40%	99.481 ± 79.234 (3)	99.460 ± 78.257 (3)	165.511 ± 3.291 (1)	157.555 ± 29.134 (1)
	60%	95.157 ± 80.807 (3)	100.871 ± 79.844 (3)	168.227 ± 2.768 (1)	160.098 ± 28.859 (1)
	80%	98.880 ± 85.791 (3)	104.661 ± 84.730 (3)	174.039 ± 1.522 (1)	167.072 ± 30.918 (1)
	100%	99.949 ± 87.060 (2)	105.767 ± 86.258 (2)	176.002 ± .890 (1)	163.115 ± 43.973 (1)
AUTOGPT	20%	.0826 ± .0035 (2)	.0856 ± .0028 (1)	.0847 ± .0026 (1)	.0854 ± .0022 (1)
	40%	.0909 ± .0054 (2)	.0926 ± .0028 (2)	.0940 ± .0044 (1)	.0949 ± .0012 (1)
	60%	.0913 ± .0045 (2)	.0913 ± .0040 (2)	.0942 ± .0024 (1)	.0941 ± .0030 (1)
	80%	.0912 ± .0039 (2)	.0911 ± .0031 (2)	.0941 ± .0022 (1)	.0944 ± .0016 (1)
	100%	.0888 ± .0050 (3)	.0880 ± .0046 (3)	.0944 ± .0032 (1)	.0946 ± .0028 (1)

Table 10: Rank stability at reduced densities relative to 100% density.

System	$\rho_{20,100}$	$\rho_{40,100}$	$\rho_{60,100}$	$\rho_{80,100}$	Mean-best method, 20%→100%
NAIVERAG	.70	.70	.70	.50	SMAC→SMAC→SMAC→BOHB→SMAC
LIGHTRAG	.90	.90	.80	.80	SMAC→SMAC→SMAC→SMAC→HEBO
VLLM	.90	1.00	.90	1.00	SMAC→SMAC→SMAC→SMAC→SMAC
HTML-RAG	1.00	1.00	.90	.90	SMAC→SMAC→SMAC→HEBO→SMAC
OPENHANDS	.90	.90	.90	.90	HEBO→HEBO→HEBO→HEBO→SMAC
SGLANG	.90	.90	1.00	1.00	SMAC→SMAC→SMAC→SMAC→SMAC
AUTOGPT	.00	.90	.80	1.00	BOHB→HEBO→SMAC→HEBO→HEBO

F MULTI-FIDELITY SCHEDULE AND LANDSCAPE DIAGNOSTICS

F.1 HB/BOHB RESOURCE SCHEDULE

Although `LLMSYS-HPOBench` exposes multidimensional fidelity factors, standard `HB` and `BOHB` require a single ordered resource variable. For each system, we therefore select one system-specific fidelity factor and hold the remaining factors fixed. We use $\eta = 2$ and $R = 4$, giving geometric resource levels $(1, 2, 4)$. Table 11 specifies the successive-halving brackets, and Table 12 maps the abstract resource levels to the concrete fidelity files and their measured costs.

The schedule is geometric in the selected resource variable, but not necessarily in actual evaluation cost or informativeness. This distinction is important for interpreting multi-fidelity results: the fidelity-to-cost and fidelity-to-target relationships are empirical properties of each LLM system rather than assumptions imposed by the benchmark.

Table 11: Resource allocation in the three Hyperband brackets. Each entry is $n_i@r_i$: n_i configurations evaluated with r_i resource units.

Stage	$s = 2$	$s = 1$	$s = 0$
1	4@1	3@2	3@4
2	2@2	1@4	–
3	1@4	–	–

Table 12: Mapping from resource levels to concrete fidelity settings and measured evaluation cost (mean±std seconds). Geometric resource levels do not necessarily imply geometric runtime costs.

System	Level	Fidelity setting	Cost (s)
NAIVERAG	1	`0.5-0-easy-cs`	81.31 ± 67.89
	2	`0.5-1-easy-cs`	144.11 ± 87.83
	4	`0.5-2-easy-cs`	222.04 ± 131.22
LIGHTRAG	1	`bridge-0-0-2`	117.53 ± 186.81
	2	`bridge-1-2-5`	461.81 ± 213.95
	4	`bridge-2-5-10`	392.44 ± 264.10
VLLM	1	`15.0-2.0-32-50-r1`	12.39 ± 1.55
	2	`15.0-2.0-32-100-r1`	18.14 ± 2.39
	4	`15.0-2.0-32-200-r1`	29.57 ± 4.25
HTML-RAG	1	`2-01-10`	120.92 ± 56.59
	2	`2-02-10`	218.61 ± 104.27
	4	`2-025-10`	297.96 ± 95.92
OPENHANDS	1	`10-4-5-6`	6.00 ± 9.94
	2	`14-4-5-6`	9.49 ± 13.82
	4	`20-4-5-6`	6.01 ± 3.58
SGLANG	1	`inf-2.0-32-64-1024`	270.10 ± 131.34
	2	`inf-2.0-32-64-2048`	401.90 ± 156.42
	4	`inf-2.0-32-64-4096`	647.56 ± 351.87
AUTOGPT	1	`complex-req1-code_generation`	27.78 ± 27.25
	2	`complex-req2-code_generation`	39.22 ± 17.26
	4	`complex-req4-code_generation`	95.75 ± 47.60

F.2 CONFIGURATION-SPACE AUTOCORRELATION

We compute autocorrelation (Weinberger, 1990) to quantify the ruggedness of the LLM systems (one target fidelity and objective per system). Higher autocorrelation indicates a smoother landscape, whereas values near zero indicate that the objective values for neighboring configurations vary dramatically. Table 13 shows that most systems have autocorrelation below .30, although the magnitude is system-dependent.

Table 13: Configuration-space autocorrelation for the seven LLM systems.

System	Autocorrelation
NAIVERAG	.285
LIGHTRAG	−.006
VLLM	.150
HTML-RAG	.109
OPENHANDS	−.022
SGLANG	.396
AUTOGPT	.244

F.3 RELATIONSHIP BETWEEN EVALUATION COST AND FIDELITY LEVEL

For each system, we select a high-cost target fidelity and compare it with ten sampled fidelity settings. Figure 7 in the main text illustrates their cost–correlation relationship. Table 14 reports the mean evaluation cost of each setting and its Spearman correlation with the target-fidelity ranking.

Table 14: Evaluation cost and Spearman correlation with the target fidelity for ten sampled fidelity settings. Higher-cost fidelities do not consistently yield stronger rank correlation.

System	Quantity	z_1	z_2	z_3	z_4	z_5
AUTOGPT	Cost (tokens)	1319.430	1324.766	1381.013	1701.848	1795.827
	ρ (latency)	.766	.846	.500	.777	.800
HTML-RAG	Cost (tokens)	2871.490	2982.244	4251.838	4405.999	4720.336
	ρ (precision)	.047	.055	.049	.008	−.016
LIGHTRAG	Cost (time)	122.624	173.230	229.330	272.859	294.052
	ρ (MRR)	.909	.766	.965	.947	.947
NAIVERAG	Cost (time)	41.641	76.644	105.988	136.682	150.846
	ρ (NDCG)	.576	.639	.529	.559	.604
OPENHANDS	Cost (tokens)	322.766	344.824	412.426	488.091	576.698
	ρ (deviation)	−.034	.035	.089	−.116	−.017
SGLANG	Cost (time)	63.560	95.787	133.076	135.348	199.361
	ρ (throughput)	.675	.722	.660	.664	.758
VLLM	Cost (time)	20.752	21.298	23.216	23.981	30.624
	ρ (throughput)	.624	.600	.662	.653	.215

System	Quantity	z_6	z_7	z_8	z_9	z_{10}
AUTOGPT	Cost (tokens)	2067.234	2580.033	2637.435	4347.051	4833.337
	ρ (latency)	.781	.783	.692	.745	.786
HTML-RAG	Cost (tokens)	5833.296	6193.095	6413.310	14135.468	14545.924
	ρ (precision)	.021	.047	.187	.111	.067
LIGHTRAG	Cost (time)	324.685	398.371	413.693	418.269	679.216
	ρ (MRR)	.962	.947	.954	.853	.932
NAIVERAG	Cost (time)	150.961	180.151	180.369	200.819	208.055
	ρ (NDCG)	.528	.263	.747	.645	.733
OPENHANDS	Cost (tokens)	811.050	899.249	1073.878	1124.231	1227.228
	ρ (deviation)	−.160	.028	−.011	.062	−.014
SGLANG	Cost (time)	199.635	274.154	274.486	404.146	404.856
	ρ (throughput)	.768	.687	.704	.765	.759
VLLM	Cost (time)	53.089	75.311	128.415	131.062	235.752
	ρ (throughput)	.250	.513	.897	.609	.937

G MORE DETAILS OF LLMSYS-HPOBENCH

G.1 LLM SYSTEMS

Here, we list a brief description of the LLM systems in LLMSYS-HPOBench. However, it is worth noting that since LLMSYS-HPOBench is a live benchmark, more LLM systems will be added progressively.

- **LIGHTRAG:** LIGHTRAG is a graph-enhanced retrieval-augmented generation framework that integrates knowledge graph-based indexing with a dual-level retrieval paradigm. It enables efficient and context-aware information retrieval by jointly leveraging entity-level and global semantic search, while supporting incremental updates for low-cost and adaptive knowledge integration.
- **NAIVERAG:** NAIVERAG is a modular retrieval-augmented generation pipeline that follows the standard chunk-retrieve-generate paradigm. It constructs configurable vector indexes over chunked corpora and performs top-k similarity retrieval to provide evidence for downstream answer generation.
- **HTMLRAG:** HTMLRAG is an HTML-focused retrieval-augmented generation framework tailored to semi-structured web documents. It first compresses noisy DOM content through a two-stage pruning process, combining embedding-based filtering with generation-model-based refinement, and then retrieves task-relevant evidence for downstream question answering.
- **VLLM:** VLLM is a high-throughput LLM inference engine designed around efficient batching and KV-cache management, providing an OpenAI-compatible serving API. It targets low-latency, high-concurrency generation with configurable scheduling and memory policies.
- **SGLANG:** SGLANG is a system for structured LLM serving and orchestration that emphasizes scheduling and memory optimization for long-context and multi-request workloads. It supports fine-grained control of cache, prefill, and parallelism to balance throughput and latency.
- **AUTOGPT:** AUTOGPT is an autonomous agent framework that couples LLM reasoning with tool use, memory, and execution loops. It is a representative of agentic AI systems that solve multi-step tasks through iterative planning and action.
- **OPENHANDS:** OPENHANDS is an open-source, model-agnostic platform for autonomous software development agents. It supports code-related task execution through sandboxed runtimes, tool use, and SDK/API interfaces, making it a representative agentic coding system in LLMSYS-HPOBench.

G.2 MAINTENANCE AND DISSEMINATION

Under what license the LLMSYS-HPOBench is released? LLMSYS-HPOBench is released under the GNU General Public License v3.0.

Who is maintaining the benchmark? The maintenance and consolidation of LLMSYS-HPOBench will be completed by the authors' research group.

How are the updates conducted? We maintain immutable benchmark releases with frozen data and default evaluation protocols. New systems, measurements, and hardware profiles are introduced only through a new release rather than by modifying archived releases. The current plan is to publish a uniquely versioned release on the open-source platform every three months as new data become available. Reported results should identify both the benchmark release and protocol version so that comparisons remain reproducible over time.

How will the datasets in the LLMSYS-HPOBench be distributed? The entire volume of data in LLMSYS-HPOBench can potentially become very large (i.e., currently the size of all file is approximately 38.5GB); however, we have designed a multiple-level chunking mechanism that can

ensure users only need to download the parts that one really needs. In all cases, the API would remain functional. For example, the datasets are split into different LLM systems at different fidelity settings. The hardware and log files are also separated from the `.csv` data—if one is only interested in testing the objective values of hyperparameter configurations, s/he could only need to download the `.csv` files.

How to ask questions or raise issues? One can use the issue tracking service of our repository, which is a very typical feature of popular hosting platforms like GitHub.

How to contribute to `LLMSYS-HPOBench`? We have provided a comprehensive, intuitive, yet straightforward step-by-step guideline on how to contribute new data for `LLMSYS-HPOBench`:

1. **Open a tracking issue.** The contributor first describes the target LLM system, its family (RAG pipeline, inference engine, or agentic system), the intended contribution type (new system, new fidelity settings, or additional measurements), and the expected hardware/software requirements.
2. **Create the system manual.** Add a manual under the corresponding documentation directory, i.e., `RAG/manuals/`, `Engine/manuals/`, or `Agent/manuals/`. The manual should define the AI and non-AI hyperparameters, their types and value ranges, implementation references, default settings, and any deployment assumptions.
3. **Implement the benchmark interface.** Add scripts, samplers, clients, or workload generators under the matching code directory. Each contribution should expose the same conceptual interface as existing entries in `LLMSYS-HPOBench`: a configuration space, a fidelity space, a runnable evaluation procedure, and machine-readable metadata for the returned objectives, costs, hardware traces, and logs.
4. **Specify fidelities and sampling policy.** Document all fidelity dimensions, their allowed values, the Cartesian product or sampling strategy, and their expected impact on cost/resources. If the full Cartesian product is too expensive, the contributor should justify the reduced sampling plan and include a small pre-experiment showing that the selected fidelities remain informative.
5. **Generate and organize measurements.** Store raw outputs, processed `.csv` summaries, cost logs, error logs, hardware traces, and visualizations in the standardized data layout. Every measured configuration should have a stable identifier so that objectives, costs, hardware metrics, and execution logs can be joined unambiguously.
6. **Record the environment.** Add a reproducibility note with hardware specifications, operating system, Python/CUDA/library versions, launch commands, random seeds, container or virtual-environment details, and known failure modes.
7. **Run validation before review.** Execute a smoke test on a small configuration–fidelity subset, check that the processed files match the metadata schema of `LLMSYS-HPOBench`, verify that failed or partial runs are explicitly marked, and confirm that the provided scripts can reproduce at least one sample end-to-end.
8. **Submit a pull request.** The pull request should link the tracking issue, summarize the added system/fidelity/data volume, list the validation result, and identify any limitations. Benchmark maintainers then check the manual, benchmark interface, data schema, and reproducibility artifacts before merging.

More details are available in the anonymous project repository: https://anonymous.4open.science/r/llmsys-hpobench-CD2B.

G.3 Full Metadata

We showcase the details of metadata on hyperparameter space in Tables 15–21; fidelity space in Table 22; the objective metrics in Table 23; the cost metrics in Table 24; the hardware information in Table 25; and an example of the logging trace file in Figure 8.

Table 15: Details of hyperparameters of LIGHTRAG.

Configuration Option	Component	Type	Default	Range
llm_model_name	AI	Categorical	llama3.1:8b	{llama3.1:8b, qwen3:8b, deepseek-r1:14b, gemma3:12b, qwen3:14b}
llm_model_temperature	AI	Float	0.7	[0.2, 1.2]
llm_model_top_p	AI	Float	0.9	[0.0, 1.0]
llm_model_top_k	AI	Integer	40	[5, 80]
llm_model_repeat_penalty	AI	Float	1.1	[0.9, 1.5]
llm_model_num_ctx	AI	Integer	32768	[20480, 40480]
max_token_for_text_unit	non-AI	Integer	4000	[2000, 6000]
max_token_for_global_context	non-AI	Integer	4000	[2000, 6000]
max_token_for_local_context	non-AI	Integer	4000	[2000, 6000]
llm_model_max_token_size	non-AI	Integer	8192	[8192, 32768]
chunk_token_size	non-AI	Integer	1200	[800, 2000]
chunk_overlap_token_size	non-AI	Integer	120	[50, 200]
entity_extract_max_gleaning	non-AI	Integer	1	[1, 4]
embedding_batch_num	non-AI	Integer	32	[16, 64]
embedding_func_max_async	non-AI	Integer	16	[8, 32]
force_llm_summary_on_merge	non-AI	Integer	6	[3, 15]
llm_model_max_async	non-AI	Integer	5	[3, 10]
max_parallel_insert	non-AI	Integer	5	[3, 15]
summary_to_max_tokens	non-AI	Integer	500	[250, 1000]
mode	non-AI	Categorical	global	{local, global, hybrid, naive}
response_type	non-AI	Categorical	Multiple Paragraphs	{Multiple Paragraphs, Single Paragraph, Bullet Points}
top_k	non-AI	Integer	60	[30, 100]
cosine_threshold	non-AI	Float	0.2	[0, 0.6]

Table 16: Details of hyperparameters of NAIVERAG.

Configuration Option	Component	Type	Default	Range
embedding_temperature	AI	Float	0.8	[0, 1.0]
embedding_num_ctx	AI	Integer	1024	[512, 2047]
embedding_top_k	AI	Integer	50	[10, 99]
embedding_repeat_penalty	AI	Float	1.1	[0.9, 1.5]
chat_temperature	AI	Float	0.8	[0, 1.0]
chat_num_ctx	AI	Integer	4096	[512, 8191]
chat_top_k	AI	Integer	50	[10, 99]
chat_repeat_penalty	AI	Float	1.1	[0.9, 1.5]
database_type	non-AI	Categorical	1	{1, 2, 3}
chunk_size	non-AI	Integer	512	[256, 2047]
chunk_overlap	non-AI	Integer	64	[32, 255]
retriever_k	non-AI	Integer	5	[1, 9]

Table 17: Details of hyperparameters of HTMLRAG.

Configuration Option	Component	Type	Default	Range
embedding_temperature	AI	Float	0.8	[0.6, 1.0]
embedding_num_ctx	AI	Integer	2048	[1024, 4095]
embedding_top_k	AI	Integer	20	[0, 49]
embedding_repeat_penalty	AI	Float	1.1	[0.9, 1.3]
chat_temperature	AI	Float	0.4	[0.2, 0.6]
chat_num_ctx	AI	Integer	4096	[2048, 8191]
chat_top_k	AI	Integer	40	[20, 59]
chat_repeat_penalty	AI	Float	1.2	[1.0, 1.4]
chunk_size	non-AI	Integer	512	[256, 1023]
chunk_overlap	non-AI	Integer	64	[32, 127]
retriever_k	non-AI	Integer	5	[3, 9]
max_node_words_embed	non-AI	Integer	64	[32, 127]
max_context_window_embed	non-AI	Integer	512	[256, 1023]
max_node_words_gen	non-AI	Integer	32	[16, 63]
max_context_window_gen	non-AI	Integer	512	[256, 1023]

Table 18: Details of hyperparameters of vLLM.

Configuration Option	Component	Type	Default	Range
model	AI	String	Llama-3.1-8B	model/repo id
draft_model	AI	String	None	model/repo id
num_speculative_tokens	AI	Integer	5	[3, 10]
top_k	AI	Integer	-1	[10, 100]
min_p	AI	Float	0.0	[0.8, 0.95]
repetition_penalty	AI	Float	1.0	[1.1, 1.3]
length_penalty	AI	Float	1.0	[0.8, 1.2]
best_of	AI	Integer/None	None	{None} or [2, 5]
tensor_parallel_size	non-AI	Integer	1	[1, #GPUs]
max_num_seqs	non-AI	Integer	256	[64, 8192]
max_num_batched_tokens	non-AI	Integer	2048	[64, 8192]
block_size	non-AI	Categorical	16	{8, 16, 32}
scheduler_delay_factor	non-AI	Float	0.0	[0, 2]
enable_chunked_prefill	non-AI	Boolean	False	{True, False}
enable_prefix_caching	non-AI	Boolean	True	{True, False}
disable_custom_all_reduce	non-AI	Boolean	False	{True, False}
use_v2_block_manager	non-AI	Boolean	True	{True, False}

Table 19: Details of hyperparameters of SGLang.

Configuration Option	Component	Type	Default	Range
top_k	AI	Integer	-1	[10, 100]
min_p	AI	Float	0.0	[0.8, 0.95]
repetition_penalty	AI	Float	1.0	[1.1, 1.3]
length_penalty	AI	Float	1.0	[0.8, 1.2]
best_of	AI	Integer/None	None	{None} or [2, 5]
context_length	non-AI	Integer	model max	[1, model max]
mem_fraction_static	non-AI	Float	0.8	(0, 1]
max_total_tokens	non-AI	Integer	auto	[1, +inf)
chunked_prefill_size	non-AI	Integer	-1	{-1} or [1, +inf)
schedule_policy	non-AI	Categorical	fcfs	{fcfs, priority}
dtype	non-AI	Categorical	auto	{auto, float16, bfloat16, float32}
quantization	non-AI	Categorical	none	{none, fp8, int8, int4}
kv_cache_dtype	non-AI	Categorical	auto	{auto, fp8_e5m2, fp8_e4m3}
speculative_algorithm	non-AI	Categorical	none	{none, eagle, draft}
speculative_num_steps	non-AI	Integer	3	[1, +inf)
enable_metrics	non-AI	Boolean	True	{True, False}
log_level	non-AI	Categorical	info	{debug, info, warning}

Table 20: Details of hyperparameters of AutoGPT.

Configuration Option	Component	Type	Default	Range
fast_llm	AI	Categorical	Qwen2.5-1.5B	{none, Qwen2.5-1.5B, Qwen2.5-3B}
smart_llm	AI	Categorical	Qwen2.5-7B	{Qwen2.5-7B}
big_brain	AI	Boolean	True	{True, False}
send_token_limit	AI	Integer/None	None	{None, 2048}
llm_temperature	AI	Float	0.0	{0.0, 0.7}
llm_max_output_tokens	AI	Integer	512	{512, 2048}
use_functions_api	AI	Boolean	False	{True, False}
enabled_components	non-AI	Categorical	all enabled	{FM, FM+CE, FM+CE+WS, FM+CE+WS+CTX}
allow_fs_access	non-AI	Boolean	False	{True, False}
full_message_count	non-AI	Integer	4	{2, 4, 8}
shell_command_control	non-AI	Categorical	allowlist	{allowlist, denylist}
cycle_budget	non-AI	Integer/None	3	{3, 10, 25}

Table 21: Details of hyperparameters of OPENHANDS.

Configuration Option	Component	Type	Default	Range
max_message_chars	AI	Integer	10000	{10000, 30000, 60000, 120000}
max_input_tokens	AI	Integer	16384	{16384, 32768, 65536, 131072, 200000}
max_output_tokens	AI	Integer	256	{256, 512, 1024, 2048, 4096}
temperature	AI	Float	0.0	{0.0, 0.2, 0.5, 0.8, 1.0}
top_p	AI	Float	0.7	{0.7, 0.85, 0.95, 1.0}
disable_vision	AI	Boolean	True	{True, False}
caching_prompt	non-AI	Boolean	True	{True, False}
timeout_llm	non-AI	Integer	30	{30, 60, 120, 300}
num_retries	non-AI	Integer	0	{0, 2, 5, 8}
retry_min_wait	non-AI	Integer	0	{0, 3, 5, 15}
retry_max_wait	non-AI	Integer	30	{30, 60, 120, 300}
retry_multiplier	non-AI	Float	1.0	{1.0, 1.5, 2.0, 3.0}
native_tool_calling	non-AI	Boolean	True	{True, False}
function_calling	non-AI	Boolean	False	{False, True}
enable_browsing	non-AI	Boolean	False	{False, True}
enable_history_truncation	non-AI	Boolean	True	{True, False}
max_iterations	non-AI	Integer	1	{1, 3, 5, 10}
max_budget_per_task	non-AI	Float	0.0	{0.0, 0.005, 0.01, 0.05}
sandbox_timeout	non-AI	Integer	60	{60, 120, 300}
use_host_network	non-AI	Boolean	False	{False, True}
enable_auto_lint	non-AI	Boolean	False	{False, True}

Table 22: Details of fidelity factors. $\mathcal{Z}$ denotes the number of tested fidelity settings. The naming of each fidelity setting for the folders follow the same order as presented here, where the the value of each factor is separated by "−".

LLM System	Fidelity Factors	Values	$\mathcal{Z}$
LIGHTRAG	question-type context-length supporting-facts-count context-count	$\{bridge, comparison\}$ $\{400, 800, 1200, 1600\}$ $\{4, 8, 10\}$ $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$	216
NAIVERAG	question-ratio corpus-scale question-difficulty dataset-category	$\{0.2, 0.5, 0.8\}$ $\{0, 1, 2\}$ $\{easy, midian, hard\}$ $\{agriculture, art, biography, cs\}$	108
HTMLRAG	data-count html-ratio question-ratio	$\{1, 2, 3, 4, 5, 6\}$ $\{0.1, 0.15, 0.2, 0.25, 0.3\}$ $\{0.5, 0.75, 1.0\}$	90
VLLM	request_rate burstiness max_concurrency num_prompts repeat_count	$\{1, 5, 10\}$ $\{0.5, 1.0, 2.0\}$ $\{4, 8, 16, 32\}$ $\{50, 100, 200\}$ $\{1, 2\}$	216
SGLANG	request_rate burstiness max_concurrency gsp_num_groups gsp_system_prompt_len	$\{5.0, 15.0, inf\}$ $\{1.0, 2.0\}$ $\{16, 32\}$ $\{16, 32, 64\}$ $\{1024, 2048, 4096\}$	108
AUTOGPT	task_type requests_count workload_category	$\{simple, moderate, complex, multi_stage\}$ $\{1, 2, 3, 4\}$ $\{math_reasoning, code_generation, logic_puzzles, data_analysis,$ $memory_retrieval, instruction_adherence, text_classification\}$	112
OPENHANDS	facts-count rules-count proof-depth support-count	$\{7, 8, 9, 10, ..., 20\}$ $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\{1, 2, 5\}$ $\{1, 3, 6\}$	82

Table 23: Details of the inference objectives.

LLM System	Objectives	Description
LightRAG	`mrr`	Mean reciprocal rank.
	`ndcg`	Ranking quality of retrieved contexts.
	`context_similarity`	Semantic similarity between query and retrieved contexts.
	`f1`	Token-level F1 score measuring overlap between prediction and ground truth.
	`best_match_position`	Rank position of the first relevant document in the retrieved list.
	`relevant_docs_count`	Number of retrieved documents exceeding the relevance similarity threshold.
	`precision`	Token-level precision: proportion of predicted tokens that are correct.
	`recall`	Token-level recall: proportion of ground truth tokens that are recovered.
	`test_time`	Time spent evaluating test questions
NaiveRAG	`mrr`	Mean reciprocal rank.
	`ndcg`	Ranking quality of retrieved contexts.
	`context_similarity`	Semantic similarity between query and retrieved contexts.
	`lexical_ac`	Lexical answer correctness of generated responses.
	`answer_precision`	Precision of generated answers.
	`answer_llmaaj`	LLM-as-a-judge correctness score.
	`avg_similarity`	Average semantic similarity score.
	`test_time`	Time spent evaluating test questions.
HtmlRAG	`mrr`	Mean reciprocal rank.
	`ndcg`	Ranking quality of retrieved contexts.
	`context_similarity`	Semantic similarity between query and retrieved contexts.
	`lexical_ac`	Lexical answer correctness of generated responses.
	`answer_precision`	Precision of generated answers.
	`answer_llmaaj`	LLM-as-a-judge correctness score.
	`avg_similarity`	Average semantic similarity score.
	`test_time`	Time spent evaluating test questions.
vLLM	`throughput`	Tokens processed per second.
	`ttft`	Time to first token.
	`tpot`	Token processing time per generated token.
	`normalized_latency`	End-to-end latency normalized by output length.
	`success_ratio`	Fraction of successful requests.
SGLang	`throughput`	Tokens processed per second.
	`ttft`	Time to first token.
	`tpot`	Token processing time per generated token.
	`normalized_latency`	End-to-end latency normalized by output length.
	`success_ratio`	Fraction of successful requests.
AutoGPT	`success_rate`	Fraction of tasks completed successfully.
	`timeout_rate`	Aggregate timeout fraction.
	`step_timeout_rate`	Fraction of tasks with per-step timeout.
	`avg_steps_per_wall_timeout`	Average steps before a wall timeout.
	`instruction_adherence`	Degree of constraint and directive compliance.
OpenHands	`accuracy`	final answer matches the gold answer.
	`evidence_recall`	Gold evidence participation rate measures the fraction of gold supporting evidence used in the model's reasoning.
	`proof_step_deviation:`	Proof-step difference measures the absolute gap between the generated and gold proof lengths.

Table 24: Details of the cost of hyperparameter configuration measurements.

LLM System	Cost	Description
LightRAG	total_time	Total pipeline execution time.
	insert_time	Time spent on document insertion (indexing phase).
	insert_input	Number of input tokens consumed during insertion.
	insert_output	Number of output tokens generated during insertion.
	insert_total	Total tokens (input + output) used in insertion.
	insert_calls	Number of API calls made during insertion.
	query_input	Number of input tokens consumed during query processing.
	query_output	Number of output tokens generated during query processing.
	query_total	Total tokens (input + output) used in query processing.
	query_calls	Number of API calls made during query processing.
NaiveRAG	total_time	Total pipeline execution time.
	build_time	Time spent building the retrieval index or database.
	chunk_time	Time spent chunking input documents.
	total_tokens	Total number of consumed tokens.
	embedding_tokens	Number of tokens consumed by the embedding model.
	llm_tokens	Number of tokens consumed by the generation model.
	avg_tokens_per_question	Average token usage per question.
HtmlRAG	total_time	Total pipeline execution time.
	build_time	Time spent building the retrieval index or database.
	chunk_time	Time spent chunking input documents.
	total_tokens	Total number of consumed tokens.
	embedding_tokens	Number of tokens consumed by the embedding model.
	llm_tokens	Number of tokens consumed by the generation model.
	avg_tokens_per_question	Average token usage per question.
vLLM	process_cpu_seconds_stats	Total CPU time consumed by the vLLM process.
	benchmark_duration_s	End-to-end benchmark duration.
SGLang	process_cpu_seconds_stats	Total CPU time consumed by the SGLang process.
	benchmark_duration_s	End-to-end benchmark duration.
AutoGPT	prompt_tokens	Input tokens consumed.
	completion_tokens	Output tokens consumed.
	total_duration_s	Total wall-clock time for all tasks.
OpenHands	total_time	Total wall-clock time consumed by the task.
	cput_time	Total CPU time consumed during task execution.
	total_tokens	Total number of tokens consumed, including input and output tokens.
	input_tokens	Number of tokens used as model input.
	output_tokens	Number of tokens generated by the model.
	peak_memory_usage	Peak memory usage during task execution.

Table 25: Details of the hardware metrics used for all LLM systems.

Hardware	Description
CPU Utilization	Percentage of CPU resources currently in use.
RAM Utilization	Percentage of system memory currently occupied.
GPU Core Utilization	Percentage of GPU compute cores actively in use.
VRAM Utilization	Percentage of GPU memory currently occupied.
GPU Temperature	Current temperature of the GPU in degrees Celsius.
GPU Power Consumption	Current power usage of the GPU in watts.

```
============================================================
2026-04-30T00:00:30+08:00 AutoGPT Agent Protocol launch
  cwd : <workspace>/AutoGPT/classic/original_autogpt
  command : poetry run serve
  log path : <benchmark-log-dir>/autogpt_server.log
============================================================

--- AutoGPT server ---
<python-env>/site-packages/pydantic/_internal/_fields.py:160:
  UserWarning: Field "model_base_url_map" has conflict with protected namespace "model_".
2026-04-30 00:00:32,264 INFO HTTP Request: GET http://localhost:8080/v1/models "HTTP/1.1
     200 OK"
2026-04-30 00:00:32,409 WARNING Frontend not found. <workspace>/AutoGPT/classic/frontend/
     build/web does not exist. The frontend will not be available.
2026-04-30 00:00:32,409 INFO AutoGPT server starting on http://localhost:8000
2026-04-30 00:00:56,436 INFO Configure google_api_key and custom_search_engine_id to use
     Google API search.
2026-04-30 00:00:57,828 INFO HTTP Request: POST http://localhost:8080/v1/chat/completions
     "HTTP/1.1 200 OK"
2026-04-30 00:00:57,831 WARNING Parsing attempt #1 failed: InvalidAgentResponseError:
     Assistant did not use a tool
2026-04-30 00:00:59,384 INFO HTTP Request: POST http://localhost:8080/v1/chat/completions
     "HTTP/1.1 200 OK"
2026-04-30 00:00:59,386 WARNING Parsing attempt #2 failed: InvalidAgentResponseError:
     Assistant did not use a tool
...
2026-04-30 00:01:00,814 INFO HTTP Request: POST http://localhost:8080/v1/chat/completions
     "HTTP/1.1 200 OK"
2026-04-30 00:01:00,817 WARNING Parsing attempt #3 failed: InvalidAgentResponseError:
     Assistant did not use a tool
2026-04-30 00:01:02,145 INFO HTTP Request: POST http://localhost:8080/v1/chat/completions
     "HTTP/1.1 200 OK"

--- vLLM API server ---
(APIServer pid=<pid>) INFO: Started server process [<pid>]
(APIServer pid=<pid>) INFO: Application startup complete.
(APIServer pid=<pid>) INFO: 127.0.0.1:<port> - "GET /v1/models HTTP/1.1" 200 OK
(APIServer pid=<pid>) INFO 04-30 00:00:57 [loggers.py:259] Engine 000: Avg prompt
     throughput: 38.3 tokens/s, Avg generation throughput: 11.9 tokens/s, Running: 1 reqs,
     Waiting: 0 reqs, GPU KV cache usage: 0.3%, Prefix cache hit rate: 0.0%
(APIServer pid=<pid>) INFO: 127.0.0.1:<port> - "POST /v1/chat/completions HTTP/1.1" 200 OK
...
(APIServer pid=<pid>) INFO 04-30 00:01:07 [loggers.py:259] Engine 000: Avg prompt
     throughput: 20.0 tokens/s, Avg generation throughput: 96.0 tokens/s, Running: 1 reqs,
     Waiting: 0 reqs, GPU KV cache usage: 0.3%, Prefix cache hit rate: 82.6%
```

Figure 8: Sanitized logging trace of the LLM system AUTOGPT.