BOLT: A Benchmark to Democratize Black-box Optimization Research for Expensive LLM Tasks

Ruth Wan Theng Chew
National University of Singapore
ruthchew@nus.edu.sg

Zhiliang Chen
National University of Singapore
chenzhiliang@u.nus.edu

Apivich Hemachandra
National University of Singapore
apivich@u.nus.edu

Bryan Kian Hsiang Low
National University of Singapore
lowkh@comp.nus.edu.sg

Abstract

Optimization of LLM training and inference configurations, such as hyperparameters, data mixtures, and prompts, is critical to performance, but it is often approached heuristically in practice, leading to potentially suboptimal outcomes. By framing them as *noisy*, *expensive*, and *derivative-free* optimization problems, Bayesian optimization (BO) and other black-box optimization (BBO) methods offer a promising yet underexplored direction for principled, sample-efficient methods. However, LLM training and inference costs are prohibitively high for most of the BBO research community, and new methods are often only evaluated on synthetic test functions and small-scale datasets that fail to capture the challenges of modern LLM optimization problems. This impedes the development of BBO methods and makes it difficult to assess their effectiveness on modern LLM tasks. We introduce BOLT, the first LLM-centric benchmark that democratizes LLM research for the BBO community. BOLT is released at `https://github.com/chewwt/bolt`. BOLT covers broad and well-motivated LLM optimization problems, involving multi-fidelity, multi-objective, heteroscedastic noise, and high-dimensional search spaces. Each problem in BOLT is grounded in real experimental data and made fully reproducible and accessible through lightweight surrogate models fitted to the results of thousands of real LLM experiments. We benchmark BOLT against an extensive range of BO and BBO methods, showing that selected BO methods consistently outperform others across tasks and highlighting gaps in existing BBO methods on LLM tasks, underscoring the need to modernize benchmarks for the BBO community.

1 Introduction

Optimizing hyperparameters, data mixtures, and prompts for LLMs are critical decisions in any LLM pipeline and often have substantial computational cost. By framing them as black-box optimization (BBO) problems in the *noisy*, *expensive*, and *derivative-free* settings, Bayesian optimization (BO) and other BBO methods emerge as promising solutions to tackle these problems, capable of handling noisy observations and often requiring no gradients [24, 26]. Compared to heuristic approaches that are often inefficient and suboptimal [2], BO and BBO are principled and sample-efficient approaches for optimizing LLM configurations. However, researchers in the BBO community often lack sufficient computational resources to validate new methods (and ideas) on LLM-centric problems. As a result, many research endeavors in BBO are only validated by numerical or classical optimization benchmarks (e.g., BBOB [28], HPO-B [4]), making it difficult to reliably assess their effectiveness

Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning.

on modern LLM tasks. This impedes the development of BBO methods and leaves their potential untapped for LLM tasks.

We aim to democratize black-box optimization research for modern LLM problems by releasing BOLT, a **B**enchmark for **o**ptimization of expensive **L**LM **T**asks, the first black-box optimization benchmark grounded in real LLM objectives. The key goal of BOLT is to modernize the research efforts of the BBO community by allowing researchers to validate their methods on LLMs without requiring access to expensive computational resources. BOLT comprises three families of LLM optimization tasks: hyperparameter optimization (HPO), data mixture optimization (DMO), and prompt optimization (PO), all of which are active areas of research in LLMs [13, 70, 75]. Each task family consists of multiple problem instances, framed as derivative-free optimization problems that span key challenges in black-box optimization research, including multi-fidelity, multi-objective, heteroscedastic noise, and high-dimensional settings. We hope our benchmark serves as a modern, practical, and accessible validation process for future BBO research ideas.

BOLT lowers the barrier for BBO community to adopt LLM tasks as standard, reproducible benchmarks via emulators and tabular datasets of precomputed evaluations. In particular, BOLT consist of a suite of emulators, which are surrogate models trained on over 20k real LLM evaluations across all tasks, and require little compute and time to query. Each emulator is validated to ensure that conclusions drawn are consistent with the true objectives.

Our experiments show that BO methods consistently outperform standard HPO and evolutionary baselines across BOLT's tasks, while exposing structural and practical challenges that emerge under our LLM-based benchmarks. These results suggest that BO is a promising but underexplored direction for LLM optimization, and further underscores the importance of principled, accessible, and reproducible benchmarking to realize this potential.

In summary, our contributions are:

- **A suite of LLM tasks as BBO problems.** We introduce three task families grounded in real LLM objectives spanning hyperparameter (Sec. 4.3), data mixture (Sec. 4.4), and prompt optimization (Sec. 4.5). Optimization problems within each task family are designed to cover key challenges in black-box optimization research including multi-fidelity optimization, multi-objective trade-offs, and high-dimensional search spaces.
- **Emulators to democratize reproducible evaluation on LLM tasks.** Emulators trained on extensive real LLM experiments replace expensive LLM training and inference with cheap surrogate queries, providing access to validated LLM-grounded objectives and enabling reproducible comparisons without large-scale compute (Sec. 4.2).
- **Extensive empirical evaluation of BO and black-box baselines.** We benchmark a range of BO methods alongside HPO and evolutionary baselines, and provide insights into method selection and the structural and practical challenges of BO-based LLM optimization, highlighting their potential for LLM tasks (Sec. 5).

BOLT will be released as a Python library. The anonymized version of our code is available at `https://github.com/chewwt/bolt`, and emulator models and tabular data are available on the Huggingface Hub at `https://huggingface.co/collections/Glow-AI/bolt`. Sec. A provides usage examples.

2 Related Works

Black-box optimization benchmarks. BBO algorithms are most commonly evaluated on synthetic functions, such as Ackley, Branin-Hoo, and Hartmann-6 [9], with COCO [29] providing a broader suite including 24 noiseless BBOB functions [28] and noisy, large-scale, and bi-objective variants. While synthetic functions are convenient benchmarks as they are inexpensive to evaluate, analytically tractable, with known global optima, they may not adequately represent real-world BBO challenges [58].

Several real-world BBO benchmarks exist, each targeting specific problem settings. Liang et al. [49] introduced five datasets in the material science domain and Kumar et al. [42] introduced a suite of constrained optimization problems drawn from real-world applications, though closed-form equations are used rather than simulation or experimental results. EXPObench [10] provides four expensive

Table 1: Summary of BOLT optimization problems. **MF**: multi-fidelity. **MO**: multi-objective.

Family	Problem	Dim	Optimization challenges
Hyperparameters	HPO	7	mixed variables
	HPO-MF-Cont	8	mixed variables, MF (continuous)
	HPO-MF-Disc	8	mixed variables, MF (discrete)
Data mixtures	DMO	6	simplex constraint
	DMO-MO	6	simplex constraint, MO
	DMO-Het	6	simplex constraint, heteroscedastic noise
Prompts	PO-128	128	high-dim.
	PO-256	256	high-dim.
	PO-512	512	high-dim.
	PO-768	768	high-dim.

real-world problems, such as wind farm layout optimization, with mixed and conditional variables. Dreczkowski et al. [17] addresses mixed-variable and combinatorial BO including RNA inverse folding. Our benchmark complements these efforts with a focus on optimization problems arising in LLM development, a setting not addressed by existing BBO benchmarks.

Optimization benchmarks for deep learning tasks. Closer to our setting are benchmark suites targeting HPO in machine learning. HPOBench [18] and YAHPO Gym [61] provide surrogate- and tabular-based HPO problems over classical ML models, with support for multi-fidelity and multi-objective settings. HPO-B [4] is another large-scale black-box HPO benchmark derived from OpenML. These benchmarks have driven meaningful progress in HPO research. However, these benchmarks target classical or moderately-sized ML models and do not capture the optimization challenges that arise in LLM development, a gap our benchmark is designed to address.

BBO for LLM. Concurrently, a growing body of work has framed LLM optimization problems as BBO. In prompt optimization, Zhou et al. [81] and Yang et al. [77] first formulated it as a black-box problem and used the LLM itself as the optimizer. InstructZero [13] optimized prompts for black-box LLMs via BO with Gaussian process (GP) surrogates and instruction-coupled kernels. ZOPO [34] proposed NTK-based GPs targeting local optima, INSTINCT [51] used a neural surrogate over embeddings, and EASE [74] jointly optimized instructions and exemplars via neural bandits.

HPO for LLM fine-tuning presents another natural application for BO, as evaluation cost makes sample efficiency crucial. BO is well-established as an effective approach to HPO in machine learning [64, 66], and recent work extends this to LLM settings. Jang et al. [37] applied multi-objective BO to jointly optimize hyperparameters and fusion weights, while Seong-Eun et al. [63] injected LoRA domain knowledge into the search space to acheive strong configurations in as few as 30 iterations.

Beyond prompting and HPO, BO has been applied to data mixture optimization. Yen et al. [78] proposed a multi-fidelity framework that models uncertainty across mixtures and scales, while Chen et al. [14] used BO to re-weight post-training mixtures, achieving performance gains. These works indicate that BO is effective for a range of LLM optimization problems, with its sample efficiency and principled uncertainty quantification. Our benchmark complements these efforts by enabling accessible and reproducible comparison of BBO methods across these emerging problems.

3 Preliminaries

Black-box optimization seeks to optimize a black-box objective function $f : \mathcal{X} \to \mathbb{R}^d$, where d is the output dimension, $d = 1$ for single-objective problems and $d > 1$ if there are multiple objectives. Observed values are often noisy: $y = f(x) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma_{\mathrm{N}}^2)$. For instance, finding the hyperparameter that produces the best LLM performance is one such black-box optimization problem.

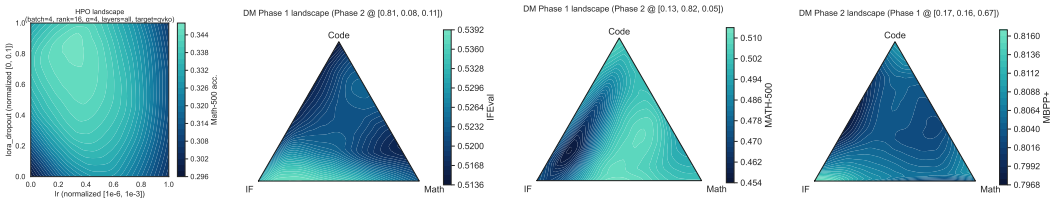


Figure 1: 2D slices of emulator landscapes

Table 2: Validation of objective function emulators using Spearman's rank correlation ρ_s between emulator predictions and true evaluations. HF: high fidelity. LF: low fidelity.

Emulator	**Problems**	**Objective**	ρ_{train} / ρ_{test}	n_{train} / n_{test}
HPO-8B	HPO, HPO-MF-Cont, HPO-MF-Disc (HF)	MATH-500	0.967 / 0.938	7404 / 785
HPO-4B	HPO-MF-Disc (LF)		0.916 / 0.905	7091 / 819
DMO-4B	DMO, DMO-MO, DMO-Het	IFEval	0.759 / 0.721	3014 / 160
		MATH-500	0.780 / 0.779	
		MBPP+	0.758 / 0.721	
DMO-Noise	DMO-Het	MATH-500 σ	0.945 / –	50 / –

4 BOLT

BOLT provides a suite of LLM problems for reproducible and accessible benchmarking of BO and other blackbox optimization methods, via emulators and tabular lookups backed by real LLM experiments. An overview of task families and optimization problems is in Table 1. There are 3 task families: **hyperparameter optimization** (HPO), **data mixture optimization** (DMO), and **prompt optimization** (PO). We provide detailed descriptions, problem formulation, and problem variants of HPO in Sec. 4.3, DMO in Sec. 4.4, and PO in Sec. 4.5.

4.1 Design Principles

BOLT is designed around three principles that distinguish it from existing optimization benchmarks.

Grounded in real LLM development workflows. Each task family in BOLT corresponds to critical decisions in the modern LLM pipeline. Objective landscapes are obtained via real LLM experiments, ensuring that algorithm comparisons on BOLT will be meaningful for practical LLM research.

Structured coverage of BO challenges. BOLT provides controlled variation of challenges most relevant to modern BO research: mixed continuous-discrete-categorical search spaces, multi-fidelity observations with both continuous and discrete fidelity parameters, multi-objective trade-offs, heteroscedastic noise, and high dimensions.

Reproducibility through emulators. Running BBO experiments with LLM training or inference in the loop is prohibitively expensive and often irreproducible due to under-specified configurations, making it hard to reliably compare methods. BOLT addresses both issues through deterministic *emulators*, which are surrogate models fitted on real experimental runs to reproduce the objective landscape at negligible query cost. Every blackbox function in BOLT is backed by an emulator or tabular data, so BBO methods can be evaluated without access to LLM infrastructure.

4.2 Emulators

The key purpose of our emulators is to *democratize* BBO research on modern LLM problems. Currently, validating BBO methods on LLM tasks requires large-scale compute that is inaccessible to most researchers. Our emulators remove this compute barrier, enabling BBO researchers to engage with modern LLM problems against validated surrogate objective landscapes at negligible cost. Using emulators in place of expensive oracles is also well-established in optimization benchmarking [30]; HPOBench and YAHPO Gym both offer surrogate models, and YAHPO Gym shows that surrogate-based benchmarks match real functions more closely than tabular-based ones.

We fit 2-layer MLPs for the optimization objective of HPO and DMO tasks, which map a given LLM training configuration (e.g., data mixture) to the observed performance. Data for the emulators were generated via random sampling and adaptive sampling strategies: one maximizing emulator gradient magnitudes (to focus samples in high-variation regions) and one maximizing predicted objectives (to ensure the emulator is anchored by observed high-performing configurations). Random samples provide broad input space coverage, while adaptive strategies concentrate effort where it matters.

Emulators are validated on a Sobol-sampled held-out test set to ensure uniform space-filling coverage. We report Spearman's rank correlation ρ as the validation metric, since candidate ranking matters more than absolute calibration in BO. Train and test ρ are close across all emulators (Table 2), indicating no overfitting. The more complex landscape of the DMO emulator in Fig. 1 and lower ρ compared to HPO, suggests that the DMO objective is intrinsically harder to optimize than HPO's.

For the heteroscedastic task in DMO, the noise model is a non-parametric kernel regression model fitted on 50 configurations, each with score standard deviation estimated from 5 repeated evaluations, with k-fold cross-validation. Additional details on emulator training, further analysis of emulators, including for multi-fidelity settings and noise emulators, are in Sec. B.

4.3 Hyperparameter optimization (HPO)

The choice of hyperparameters (e.g., learning rate, batch size) plays a significant role in determining the effectiveness of LLM training [46]. Our HPO benchmark consists of a moderate-dimensional search space that describes how the LLM performance varies w.r.t. the choice of LoRA fine-tuning hyperparameters (rank, alpha, target modules), learning rate, and batch size.

Our benchmark data in BOLT consists of datapoints from real LLM training runs with different training hyperparameters. Each training run fine-tunes a model on a fixed 100k-sample subset of OpenMathInstruct-2 [69] using LoRA under a given hyperparameter configuration. We then evaluated the fine-tuned model on MATH-500 [31, 50] using the `lm-eval` harness framework [25]. We have multiple observations for each training run via checkpoint evaluations throughout training, and used early stopping to increase coverage of the hyperparameter space within a fixed compute budget. In total, this yielded 7k+ observations per emulator across all runs and checkpoints. We trained 2 emulators, one for each model size (Qwen3-4B and Qwen3-8B [76]). More details are in Sec. C.1.

Problem formulation. The base problem search space is a 7 dimensional *mixed*-variable space, comprising continuous variables (learning rate, LoRA dropout), integer variables (batch size, LoRA rank, LoRA alpha, LoRA layers), and categorical variables (LoRA target modules). Full search space specification in Table 4. After fitting, the emulator is a 2-layer MLP and is deterministic.

Problem variants. There are three variants in this task family. *HPO* base problem is single-fidelity, single-objective at the full training budget, with a 7-dimensional search space. The other two are multi-fidelity variants with an additional fidelity parameter to indicate the observed LLM performance at a given scale (i.e., training tokens or model size), making them well-suited for multi-fidelity BO research [23]. *HPO with continuous token fidelity* (HPO-MF-Cont) adds a fidelity parameter $s \in [0, 1]$, normalized from the raw range $[10^5, 9.1 \times 10^6]$ that controls the number of training tokens seen. *HPO with discrete model-size fidelity* (HPO-MF-Disc) uses a 4B low-fidelity and 8B high-fidelity model. As a result, both multi-fidelity variants are 8-dimensional problems. Evaluating the function at a higher fidelity incurs more optimization budget, but yields more accurate observations.

4.4 Data mixture optimization (DMO)

Optimization over data mixtures can lead to performance gains for both pretraining [54, 75] and supervised fine-tuning (SFT)[14, 48]. Data curriculum adjusts mixture proportions across training stages, which further improves performance but increases search space complexity [57, 79]. There is also growing research interest in examining the optimization of LLM data mixtures over multiple, possibly competing, objectives. Our DMO problem settings span these research areas.

Datapoints were collected from real LLM training runs across different training data mixtures in both single and multi-objective settings. Each training data mixture contains 10k samples randomly drawn from three TULU-3 SFT data domains (instruction following, math, and code) [44] at a specified mixing proportion. We perform SFT on a pretrained Qwen3-4B model and evaluate on three objectives: IFEval [80], MATH-500 (4-shot), and MBPP+ [5, 53]. The DMO-4B emulator is

Table 3: Example prompt candidates and corresponding scores

Prompt	Score
Write the solution in a concise, technical style typical of advanced mathematics.	0.810
Break down the problem into smaller subproblems and solve each one sequentially.	0.716
Provide only the final answer without any explanation or intermediate steps.	0.374

trained on 3k+ datapoints, spanning diverse mixture configurations, and predicts all three objective scores given a data mixture curriculum. See Sec. C.2 for more training details.

Problem formulation. The search space is $\mathcal{X}_{\text{DMO}} \triangleq \left\{ x \in \mathbb{R}^6_{\geq 0} \mid \sum_{i=1}^{3} x_i = 1,\ \sum_{j=4}^{6} x_j = 1 \right\}$, where $x_{1:3}$ and $x_{4:6}$ are the mixture proportions over the three data sources at stage 1 (tokens 0–5M) and stage 2 (tokens 5M–10M), respectively. Full simplex-constrained space is specified in Table 5. Evaluations comprise IFEval, MATH-500, and MBPP+ scores and these are combined into different objectives for each of the three problem variants.

Problem variants. All three variants share $\mathcal{X}_{\text{DMO}}$ as the search space and a single multi-output emulator, differing only in objective definition and noise structure. *DMO* base problem has a single objective and defines the objective as the unweighted mean of the three scores. This serves as the entry point for BO on a simplex-constrained space. *Multi-objective DMO* (DMO-MO) defines the objective function $f_{\text{DMO}} : \mathcal{X}_{\text{DMO}} \to [0,1]^3$, returning the three scores directly and targeting Pareto-front methods such as multi-objective BO [15]. *DMO with heteroscedastic noise* (DMO-Het) has a single MATH-500 objective and adds $\varepsilon \sim \mathcal{N}(0, \sigma^2_{\text{DMO}}(x))$ to each output, where $\sigma_{\text{DMO}}(x)$ is the prediction from the noise model. This variant is a natural testbed for noise-aware methods [39, 45].

4.5 Prompt optimization (PO)

The right prompts can substantially improve performance at inference time [81]. However, the text space is discrete and unstructured, and its embedding space high-dimensional, making undirected search inefficient. Prompt selection at test time is also time-sensitive, and long-context inference is computationally costly. Our PO dataset and emulator makes it assessible for researchers to evaluate their methods on prompt optimization problems, without needing repeated, expensive inference.

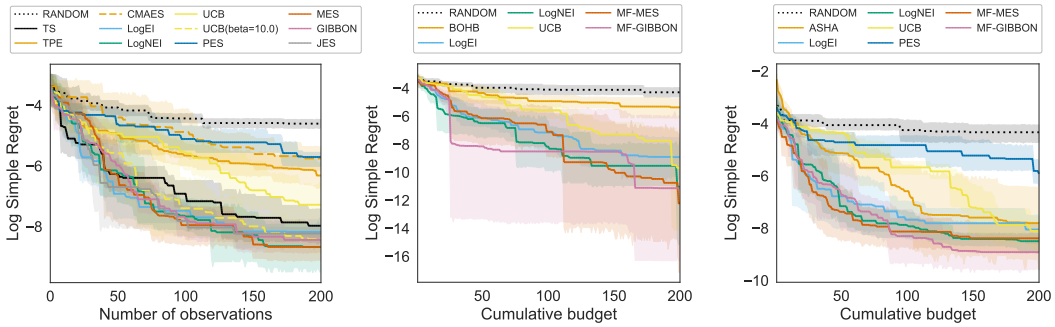
We pre-generated a corpus of $5{,}014$ candidate prompts for mathematical reasoning and embedded each prompt using EmbeddingGemma [62], which supports Matryoshka Representation Learning (MRL) [43] and can produce meaningful representations at varying dimensionalities. Each prompt candidate was evaluated on MATH-500 (0-shot), to maximize the influence of the prompt rather than the few-shot demonstrations. Some candidate prompts and their scores are in Table 3. Tabular data is used for this task rather than a learned emulator as interpolated embeddings cannot be easily decoded back to natural language prompts, and selecting from a pre-generated candidate pool is standard practice in prompt optimization [34, 51, 65, 81]. Further details on corpus generation and evaluation are in Sec. C.3.

Problem formulation. The search space is $\mathcal{X}_{\text{PO}} \triangleq \{z_i\}_{i=1}^{5014} \subset \mathbb{R}^d$, where each point z_i is the d-dim embedding of a candidate prompt i. The objective $f_{\text{PO}} : \mathcal{X}_{\text{PO}} \to [0,1]$ is the MATH-500 accuracy obtained under prompt i from Qwen3-14B model. Acquisition functions are optimized over the discrete candidate set.

Problem variants. *PO-128*, *PO-256*, *PO-512*, and *PO-768* correspond to MRL truncations at $d \in \{128, 256, 512\}$ and the full embedding dimension $d = 768$, respectively.

5 Experiments

We benchmark a wide range of BO methods on BOLT, as well as some common black-box optimization methods. All experiments have noisy observations with σ_N set to 0.001, except the hetereoscedastic noise problem DMO-Het, where noise is input-dependent and thus requires a per-point noise model $\sigma_{\text{DMO}}(x)$. For single-objective problems, we plot the logarithm of simple regret, $\log(f^* - \max_{t' \leq t} f(x_{t'}))$, where $x_{t'}$ is the observed point at iteration t'. For multi-objective problems, we plot the logarithm of hypervolume difference, $\log(\text{HV}(\text{PF}_t) - \text{HV}^*)$, where PF_t is the



(a) HPO (b) HPO-MF-Cont (c) HPO-MF-Disc

Figure 2: Results on BOLT's HPO problems. MF variants are plotted against the cumulative budget, where each observation cost range from 0.1 to 1, depending on the fidelity chosen.

noiseless Pareto frontier across all observations at iteration t. These are common evaluation metrics found in optimization literature [15, 67]. The optimal values $f*$ and HV^* are found empirically via exhaustive evaluation for tabular data, or gradient ascent of 200 best points found via grid evaluation for MLP emulators. Each method has 10 initial observations. All GP-based methods are implemented in BoTorch [6], using an RBF kernel with a dimensionality-scaled lengthscale prior [36]. Results are averaged over 5 runs and 95% confidence intervals are shown as shaded areas, with initial observations excluded in the plots. Additional experimental details and discussion on evaluation metrics are in Sec. D.

5.1 HPO experiments and analysis

We compare a range of acquisition functions including Thompson sampling (TS), expected improvement (EI, NEI), upper confidence bound (UCB), knowledge-gradients (KG), entropy-search methods (PES, MES, GIBBON), as well as common black-box HPO baselines (TPE, CMA-ES BOHB, ASHA). For multi-fidelity problems, we adapt acquisition functions to be cost-aware [23] and show the results w.r.t. the cumulative budget. The cost for each multi-fidelity evaluation was set to $0.9 \cdot \text{fid}(x) + 0.1$. More implementation details are in Sec. D.1 and results are shown in Fig. 2.

GP-based BO methods generally outperform non-GP-based HPO baselines. LogNEI, (MF-)MES, and (MF-)GIBBON are the strongest performers across all three settings over 200 iterations. We note that non-GP baselines have substantially faster wall-clock times (Table 9), although this overhead is negligible compared to the much more costly, repeated LLM trainings with different hyperparameters at every iteration.

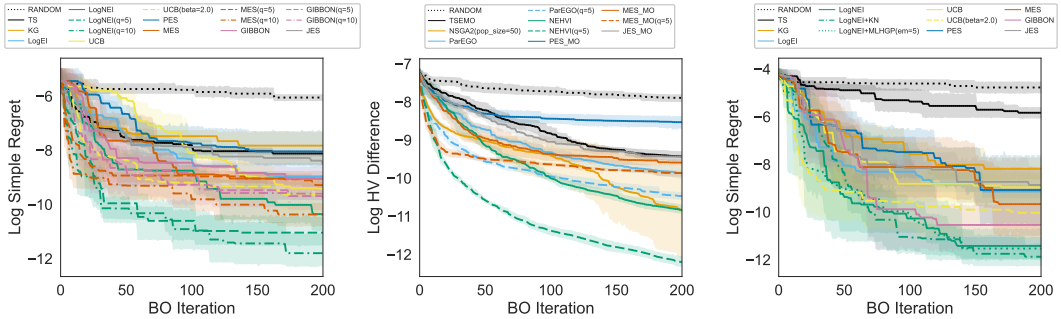
Evaluation budget is an important consideration for acquisition function selection. In the standard HPO setting (Fig. 2a), TS outperforms other methods under 25 iterations but is subsequently overtaken. UCB faces a similar budget dependency through its β parameter that balances exploration and exploitation, which we analyze in Sec. D.2.

Cost-scale sensitivity is a practical liability of MF methods. Multi-fidelity acquisition functions incorporate a cost utility to discount evaluations at lower fidelities, but there is no principled way to set this. We select cost scales by comparing early-stopped runs, though this choice can significantly affect performance and fidelity allocation across methods, affecting the transferability to new problems. Analysis of acquisitions functions at different cost scales on both MF problems are in Sec. D.3.

We include a breakdown of fidelity queries per method for both multi-fidelity problems in Sec. D.4.

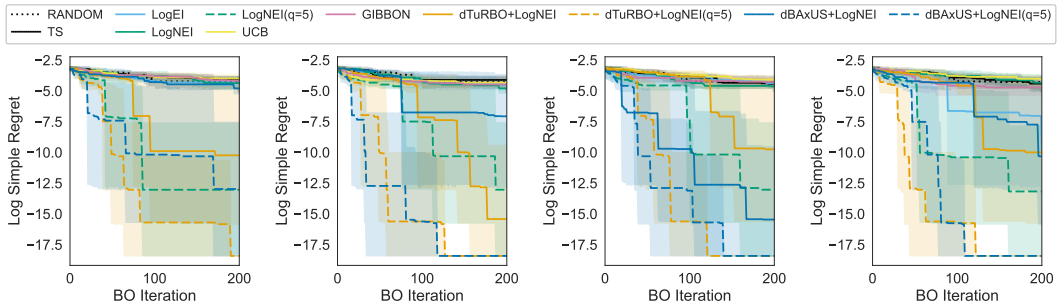
5.2 DMO experiments and analysis

Since LLM finetuning and evaluation are often conducted in parallel, we evaluate both single-observation and batched acquisition strategies, plotting performance against the number of optimization iterations. Batched candidates are selected via sequential greedy conditioning [6]. Batch sizes and population (for genetic algorithms) are noted where they differ from a default of one.



(a) DMO (b) DMO-MO (c) DMO-Het

Figure 3: Results on BoLT's DMO problems. ($q = 5$) indicates a batch size of 5, and NSGA2 has 50 observations in each iteration. All other methods use only 1 observation per iteration.



(a) PO-128 (b) PO-256 (c) PO-512 (d) PO-768

Figure 4: Results on BoLT's PO problems. ($q = 5$) indicates a batch size of 5. Note that dTURBO and dBAxUS are our adapted methods for a discretized search space.

We compare against a range of acquisition functions, including multi-objective variants and an evolutionary method (NSGA2). Since entropy-based methods (MES, PES, JES) rely on a candidate set to approximate the optimal-value/optimal-point distribution, we replace the standard box-uniform grid with Dirichlet-sampled points to respect the simplex constraint on each mixture group. For DMO-Het, we additionally include noise-aware MLHGP that learns a noise model with expectation maximization (EM) [39] and a GP with known ground-truth noise to serve as an oracle upper bound. Additional implementation details are in Sec. D.5 and results are shown in Fig. 3.

NEI-based methods dominate overall. LogNEI performs best on single-objective problems, while NEHVI($q = 5$) leads on multi-objective DMO-MO problem. Notably, NEHVI with a single observation per iteration matches NSGA2(pop_size=50) despite using $50\times$ fewer function evaluations per iteration, and with substantially tighter confidence intervals, indicating more consistent performance.

Batch BO accelerates convergence. Since batch methods collect more observations per iteration, it is expected and observed that they outperform single-observation BO over the same iterations. Larger batch sizes generally accelerate early convergence, though gains can diminish over time, for instance, MES($q = 5$) plateaus after 50 iterations. The effect of batch size is acquisition-function dependent: GIBBON shows little difference between $q = 5$ and $q = 10$, whereas LogNEI and MES exhibit more pronounced gains, possibly reflecting differences in the acquisition landscape across methods.

Noise-aware methods yield modest improvements under heteroscedastic noise. When noise is known (LogNEI+KN), performance surpasses standard homoscedastic LogNEI in 200 iterations, confirming that correct noise modelling is beneficial. LogNEI+MLHGP performed marginally better than LogNEI, suggesting the EM-learned noise model captures sufficient noise structure.

5.3 PO experiments and analysis

Beyond benchmarking acquisition functions, we evaluate trust-region and subspace-reduction methods, specifically TuRBO [20] and BAxUS [59]. Since PO problems operate over a discrete candidate set, we introduce discretized variants of these methods, which we term *dTuRBO* and *dBAxUS*. In these variants, the trust-region hyperrectangle is replaced by the k nearest neighbors of the current best solution in embedding space, where k is doubled upon success and halved upon failure. Consequently, Sobol candidate generation and sparse perturbation are no longer required. We omit SAASBO [19] due to its prohibitive computational cost, and MSR [60] as it is inapplicable to discretized search spaces. Finally, for acquisition functions that support batch selection, we include results with batch size 5, reflecting the parallel evaluation setting common in practice. More implementation details, including on dTuRBO and dBAxUS, are provided in Sec. D.6. Results are shown in Fig. 4.

Standard acquisition functions fail to scale. LogEI, LogNEI, UCB, TS, and GIBBON largely plateau between -3 and -5 log simple regret within the first 50 iterations, with negligible improvement thereafter, making them indistinguishable from random search. A notable exception is LogEI at PO-768, which descends to approximately -7, suggesting it benefits from the richer expressivity of the full embedding dimensionality, though it remains far behind trust-region methods.

Batch evaluation partially compensates for the absence of trust-region structure. Log simple regret of LogNEI($q = 5$) reaches approximately -12.5 for all PO problems without any local search constraints, far exceeding its sequential counterpart and showing that parallel acquisition provides meaningful gains independent of trust-region structures.

Local search methods lead to substantial performance gains. Across all problem dimensions, dTuRBO+LogNEI and dBAxUS+LogNEI reach log simple regret of approximately -10 to -15, demonstrating that the k-NN trust region adaptation perserves the effectiveness of local search in discretized search spaces. Combining local search with batch evaluation yields further gains, with dTuRBO+LogNEI($q = 5$) and dBAxUS+LogNEI($q = 5$) achieving the lowest regret and approaching the optimal solution within 200 iterations, across almost all PO problems.

Subspace reduction may be unnecessary for LLM embedding spaces. At PO-128, dBAxUS performs notably worse than dTuRBO, as PO-128 embeddings require 74.2% of principal components to explain 95% of variance (Fig. 5), violating the low-dimensional active subspace assumption underlying BAxUS [59]. At higher dimensionalities, where variance concentrates in fewer principal components, dBAxUS recovers but results are mixed for single-observation runs. With batch size 5, dBAxUS matches dTuRBO but converges more slowly, likely due to wasted evaluations in very low-dimensional subspaces before expansion. This suggests subspace reduction is most effective when the embedding space has genuine dimensional redundancy, a property that may not hold for LLM embeddings, as they tend to be broadly informative across dimensions.

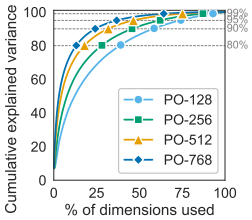


Figure 5: PCA explained variance for PO

6 Limitations

To collect sufficient data to fit reliable emulators for BOLT, we required practical measures such as smaller training token budgets, reduced maximum token limits, and, for HPO tasks, the exclusion of chain-of-thought reasoning. This may not represent the best achievable performance of the evaluated models under unconstrained inference. Nevertheless, the optimization challenges BOLT is designed to benchmark (mixed-variable search spaces, simplex-constraints, and more) are intrinsic to the structure of LLM tasks themselves and remain consistent across inference and training configurations. We also plan to update the benchmark with new LLM architectures in the future, in an effort to continue modernizing optimization benchmarks.

7 Conclusion and Future Work

Black-box optimization methods hold significant promise for automating and accelerating LLM pipeline optimization, but progress has been hampered by the lack of accessible and reproducible evaluations grounded in real LLM experiments. BOLT is critical to unlocking this progress, by

providing emulators and tabular datasets built on real LLM experiments, enabling fast, reproducible benchmarking without large-scale compute.

Our evaluation reveals that mature BBO methods, in particular, NEI- and GIBBON-based approaches, transfer effectively to LLM tasks, but strong performance required adaptation to LLM-specific structural challenges, including simplex-constrained and discretized search spaces. These challenges reflect the structure of real LLM pipeline optimization problems and highlight gaps for the BBO community to tackle.

BOLT is extensible to more optimization problems, for example contextual and constrained optimization, as new optimization settings can be constructed using our publicly available emulators and data. There are also many other LLM-centric tasks that fit the black-box optimization framework, such as decoding configuration and RLHF reward weighting, that can be future extensions of BOLT.

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] A. Albalak, Y. Elazar, S. M. Xie, S. Longpre, N. Lambert, X. Wang, N. Muennighoff, B. Hou, L. Pan, H. Jeong, et al. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827*, 2024.

[3] S. Ament, S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy. Unexpected improvements to expected improvement for Bayesian optimization. *Advances in Neural Information Processing Systems*, 36:20577–20612, 2023.

[4] S. P. Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. Hpo-b: A large-scale reproducible benchmark for black-box hpo based on openml. In *Thirty-fifth Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[5] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[6] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A framework for efficient monte-carlo bayesian optimization. *Advances in Neural Information Processing Systems*, 33:21524–21538, 2020.

[7] S. Belakaria, A. Deshwal, and J. R. Doppa. Max-value entropy search for multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[9] D. Bingham and S. Surjanovic. Virtual library of simulation experiments: Test functions and datasets, 2013. URL `https://www.sfu.ca/~ssurjano/optimization.html`.

[10] L. Bliek, A. Guijt, R. Karlsson, S. Verwer, and M. De Weerdt. Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *Applied Soft Computing*, 147:110744, 2023.

[11] E. Bradford, A. M. Schweidtmann, and A. Lapkin. Efficient multiobjective optimization employing Gaussian processes, spectral sampling and a genetic algorithm. *Journal of global optimization*, 71(2): 407–438, 2018.

[12] O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. *Advances in Neural Information Processing Systems*, 24, 2011.

[13] L. Chen, J. Chen, T. Goldstein, H. Huang, and T. Zhou. Instructzero: Efficient instruction optimization for black-box large language models. In *International Conference on Machine Learning*, 2024.

[14] Z. Chen, G. K. R. Lau, C.-S. Foo, and B. K. H. Low. DUET: Optimizing training data mixtures via feedback from unseen evaluation tasks. *arXiv:2502.00270*, 2025.

[15] S. Daulton, M. Balandat, and E. Bakshy. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. *Advances in Neural Information Processing Systems*, 34:2187–2200, 2021.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[17] K. Dreczkowski, A. Grosnit, and H. Bou Ammar. Framework and benchmarks for combinatorial and mixed-variable Bayesian optimization. *Advances in Neural Information Processing Systems Track on Datasets and Benchmarks*, 36:69464–69489, 2023.

[18] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[19] D. Eriksson and M. Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Uncertainty in artificial intelligence*, pages 493–503. PMLR, 2021.

[20] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[22] C. Fernando, D. S. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *International Conference on Machine Learning*, pages 13481–13544. PMLR, 2024.

[23] P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

[24] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[25] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[26] R. Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

[27] N. Hansen. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pages 75–102, 2006.

[28] N. Hansen, S. Finck, R. Ros, and A. Auger. *Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions*. PhD thesis, INRIA, 2009.

[29] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[30] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, M. Christensen, E. Liles, J. E. Hein, and A. Aspuru-Guzik. Olympus: a benchmarking framework for noisy optimization and experiment planning. *Machine Learning: Science and Technology*, 2(3):035021, 2021.

[31] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[32] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams. Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501. PMLR, 2016.

[33] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in Neural Information Processing Systems*, 27, 2014.

[34] W. Hu, Y. Shu, Z. Yu, Z. Wu, X. Lin, Z. Dai, S.-K. Ng, and B. K. H. Low. Localized zeroth-order prompt optimization. *Advances in Neural Information Processing Systems*, 37:86309–86345, 2024.

[35] C. Hvarfner, F. Hutter, and L. Nardi. Joint entropy search for maximally-informed Bayesian optimization. *Advances in Neural Information Processing Systems*, 35:11494–11506, 2022.

[36] C. Hvarfner, E. O. Hellsten, and L. Nardi. Vanilla Bayesian optimization performs great in high dimensions. In *International Conference on Machine Learning*, pages 20793–20817. PMLR, 2024.

[37] C. Jang, H. Lee, J. Kim, and J. Lee. Model fusion through Bayesian optimization in language model fine-tuning. *Advances in Neural Information Processing Systems*, 37:29878–29912, 2024.

[38] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[39] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. Most likely heteroscedastic gaussian process regression. In *International Conference on Machine learning*, pages 393–400, 2007.

[40] J. 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.

[41] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213, 2022.

[42] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, and S. Das. A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. *Swarm and evolutionary computation*, 56:100693, 2020.

[43] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[44] N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, X. Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025.

[45] M. Lázaro-Gredilla and M. K. Titsias. Variational heteroscedastic Gaussian process regression. In *International Conference on Machine Learning*, pages 841–848, 2011.

[46] H. Li, W. Zheng, Q. Wang, H. Zhang, Z. Wang, S. Xuyang, Y. Fan, Z. Ding, H. Wang, N. Ding, S. Zhou, X. Zhang, and D. Jiang. Predictable scale: Part i, step law – optimal hyperparameter scaling law in large language model pretraining. *arXiv:2503.04715*, 2025.

[47] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of machine learning and systems*, 2: 230–246, 2020.

[48] Y. Li, Z. Liu, and E. Xing. Data mixing optimization for supervised fine-tuning of large language models. In *International Conference on Machine Learning*, pages 35419–35437. PMLR, 2025.

[49] Q. Liang, A. E. Gongora, Z. Ren, A. Tiihonen, Z. Liu, S. Sun, J. R. Deneault, D. Bash, F. Mekki-Berrada, S. A. Khan, et al. Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains. *npj Computational Materials*, 7(1):188, 2021.

[50] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[51] X. Lin, Z. Wu, Z. Dai, W. Hu, Y. Shu, S.-K. Ng, P. Jaillet, and B. K. H. Low. Use your INSTINCT: Instruction optimization for llms using neural bandits coupled with transformers. In *International Conference on Machine Learning*, pages 30317–30345. PMLR, 2024.

[52] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. Smac3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[53] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

[54] Q. Liu, X. Zheng, N. Muennighoff, G. Zeng, L. Dou, T. Pang, J. Jiang, and M. Lin. Regmix: Data mixture as regression for language model pre-training. In *The Thirteenth International Conference on Learning Representations*, 2025.

[55] H. B. Moss, D. S. Leslie, J. Gonzalez, and P. Rayson. Gibbon: General-purpose information-based Bayesian optimisation. *Journal of Machine Learning Research*, 22(235):1–49, 2021.

[56] M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, and M. Onishi. Warm starting CMA-ES for hyperparameter optimization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 9188–9196, 2021.

[57] T. Olmo, A. Ettinger, A. Bertsch, B. Kuehl, D. Graham, D. Heineman, D. Groeneveld, F. Brahman, F. Timbers, H. Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[58] P. S. Palar, R. P. Liem, L. R. Zuhal, and K. Shimoyama. On the use of surrogate models in engineering design optimization and exploration: The key issues. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 1592–1602, 2019.

[59] L. Papenmeier, L. Nardi, and M. Poloczek. Increasing the scope as you learn: Adaptive Bayesian optimization in nested subspaces. *Advances in Neural Information Processing Systems*, 35:11586–11601, 2022.

[60] L. Papenmeier, M. Poloczek, and L. Nardi. Understanding high-dimensional Bayesian optimization. In *International Conference on Machine Learning*, pages 47902–47923. PMLR, 2025.

[61] F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. Yahpo gym-an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *International Conference on Automated Machine Learning*, pages 3–1. PMLR, 2022.

[62] H. Schechter Vera, S. Dua, B. Zhang, D. Salz, R. Mullins, S. Raghuram Panyam, S. Smoot, I. Naim, J. Zou, F. Chen, D. Cer, A. Lisak, M. Choi, L. Gonzalez, O. Sanseviero, G. Cameron, I. Ballantyne, K. Black, K. Chen, W. Wang, Z. Li, G. Martins, J. Lee, M. Sherwood, J. Ji, R. Wu, J. Zheng, J. Singh, A. Sharma, D. Sreepat, A. Jain, A. Elarabawy, A. Co, A. Doumanoglou, B. Samari, B. Hora, B. Potetz, D. Kim, E. Alfonseca, F. Moiseev, F. Han, F. Palma Gomez, G. Hernández Ábrego, H. Zhang, H. Hui, J. Han, K. Gill, K. Chen, K. Chen, M. Shanbhogue, M. Boratko, P. Suganthan, S. M. K. Duddu, S. Mariserla, S. Ariafar, S. Zhang, S. Zhang, S. Baumgartner, S. Goenka, S. Qiu, T. Dabral, T. Walker, V. Rao, W. Khawaja, W. Zhou, X. Ren, Y. Xia, Y. Chen, Y.-T. Chen, Z. Dong, Z. Ding, F. Visin, G. Liu, J. Zhang, K. Kenealy, M. Casbon, R. Kumar, T. Mesnard, Z. Gleicher, C. Brick, O. Lacombe, A. Roberts, Y. Sung, R. Hoffmann, T. Warkentin, A. Joulin, T. Duerig, and M. Seyedhosseini. Embeddinggemma: Powerful and lightweight text representations. *arXiv preprint arXiv:2509.20354*, 2025.

[63] B. Seong-Eun, L. Jung-Mok, K. Sung-Bin, and T.-H. Oh. Efficient hyper-parameter search for LoRA via language-aided Bayesian optimization. *arXiv preprint arXiv:2602.11171*, 2026.

[64] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[65] C. Shi, K. Yang, Z. Chen, J. Li, J. Yang, and C. Shen. Efficient prompt optimization through the lens of best arm identification. *Advances in Neural Information Processing Systems*, 37:99646–99685, 2024.

[66] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25, 2012.

[67] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, 2010.

[68] S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, and M. Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In *International Conference on Machine Learning*, pages 9334–9345. PMLR, 2020.

[69] S. Toshniwal, W. Du, I. Moshkov, B. Kisacanin, A. Ayrapetyan, and I. Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. In *The Thirteenth International Conference on Learning Representations*, 2023.

[70] C. Tribes, S. Benarroch-Lelong, P. Lu, and I. Kobyzev. Hyperparameter optimization for large language model instruction-tuning. In *AAAI Conference on Artificial Intelligence*, 2024.

[71] B. Tu, A. Gandy, N. Kantas, and B. Shafei. Joint entropy search for multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems*, 35:9922–9938, 2022.

[72] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 2609–2634, 2023.

[73] Z. Wang and S. Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

[74] Z. Wu, X. Lin, Z. Dai, W. Hu, Y. Shu, S.-K. Ng, P. Jaillet, and B. K. H. Low. Prompt optimization with ease? efficient ordering-aware automated selection of exemplars. *Advances in Neural Information Processing Systems*, 37:122706–122740, 2024.

[75] S. M. Xie, H. Pham, X. Dong, N. Du, H. Liu, Y. Lu, P. S. Liang, Q. V. Le, T. Ma, and A. W. Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.

[76] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[77] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

[78] T. Yen, A. W. T. Siah, H. Chen, C. D. Guetta, T. Peng, and H. Namkoong. Data mixture optimization: A multi-fidelity multi-scale bayesian framework. In *Advances in Neural Information Processing Systems*, 2025.

[79] Y. Zhang, A. Mohamed, H. Abdine, G. Shang, and M. Vazirgiannis. Beyond random sampling: Efficient language model pretraining via curriculum learning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5776–5794, 2026.

[80] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[81] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2022.

A BOLT usage examples

```python
from bolt import HPO
import torch

# Instantiate a problem
problem = HPO(noise_std=0.001)

# BoTorch-compatible metadata
problem.bounds           # (2, 7) tensor of lower/upper bounds
problem.dim              # search-space dimensionality
problem.continuous_inds  # indices of continuous variables
problem.discrete_inds    # indices of integer variables
problem.categorical_inds # indices of categorical variables

# Sample random candidates and round integer/categorical variables
lb, ub = problem.bounds
X = lb + torch.rand(10, problem.dim) * (ub - lb)
X[:, problem.discrete_inds]    = X[:, problem.discrete_inds].round()
X[:, problem.categorical_inds] = X[:, problem.categorical_inds].round()
y = problem(X)  # (10, 1)
```

Listing 1: BOLT usage example.

B Additional emulator details and analysis

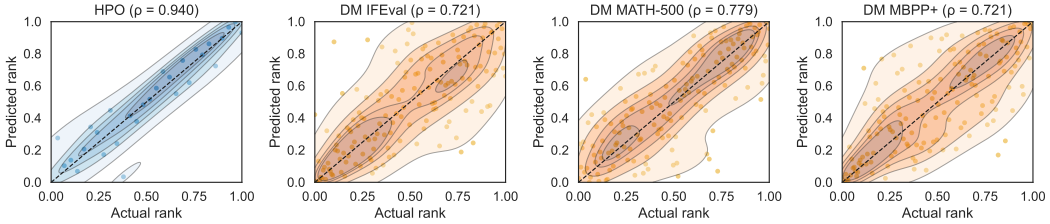


Figure 6: Real and predicted ranks on test set.

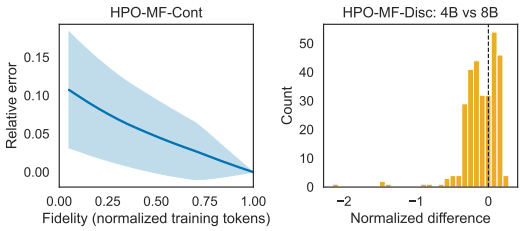


Figure 7: Differences in emulated objective at different fidelities

Emulator training. We chose 2-layer MLP for objective emulators to balance the trade off between model complexity and amount of training data needed, to prevent overfitting. Dropout and layer norm was also used during training for regularization.

Actual and predicted rank comparison of data points in the test set is shown Fig. 6, where predicted ranks largely align with actual ranks. For the DMO emulator, lower- and higher-ranked points are generally better aligned than intermediate ones, which could be due to mixture compositions in the intermediate range being more similar and harder to discriminate.

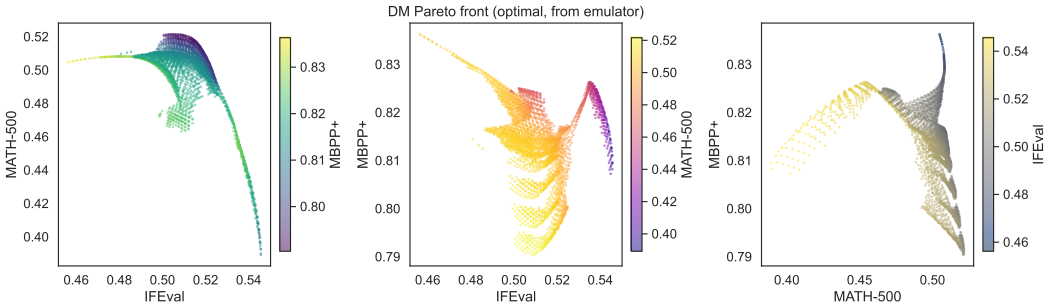


Figure 8: Estimated optimal Pareto front, as 2D slices of a 3D space

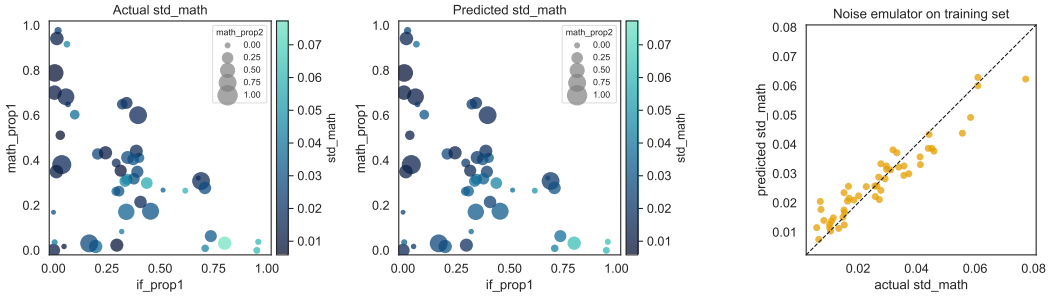


Figure 9: Noise emulator predictions for MATH-500 standard deviation. if_prop1 indicates mixture proportion of information following data at stage 1 of the curriculum, while math_prop1 and math_prop2 represent math data proportions at stage 1 and 2 respectively.

Multi-fidelity settings. We analyze the HPO emulators under multi-fidelity settings in Fig. 7. For continuous fidelity, as expected, error relative to the highest fidelity is largest at lower fidelities. For discrete fidelity, differences largely cluster around -0.6 to 0.2, indicating that the 4B model serves as a reliable proxy for the 8B model's performance.

Multi-objective setting. We visualize the optimal Pareto frontier of the DMO-4B emulator in Fig. 8, shown as three 2D slices of the joint IFEval–MATH-500–MBPP+ objective space. A clear trade-off is visible between IFEval and MATH-500, forming a concave frontier in that projection. MBPP+ varies more smoothly across the frontier, suggesting code performance is less in tension with the other two objectives. Overall, the frontier spans a diverse set of Pareto-optimal mixtures, reflecting meaningful flexibility across objective priorities.

For the noise model, hyperparameters such as kernel choice were selected via k-fold cross-validation with $k = 5$. The final model was trained on all 50 data samples using the selected hyperparameters: a Laplacian kernel with $\gamma = 0.1$. This model choice was motivated by the limited number of samples, as each data point requires multiple training runs and evaluations, making large-scale collection computationally expensive.

Visualization of the noise model predictions are shown in Fig. 9. The predicted noise distribution closely mirrors the actual, with higher noise concentrated at low if_prop1 (stage 1 information following proportion) and high math_prop1 (stage 1 math proportion) values. The predicted vs. actual plot on the training set also shows points tracking closely along the diagonal, confirming that the noise emulator fits the training data well. Some underestimation is visible at higher std_math values, which is expected given the limited number of high-noise samples in the 50-point training set.

C Details on data generation for emulators

C.1 Hyperparameter optimization

OpenMathInstruct-2 [69] was selected as the training data. We used only the answer for training, due to time and compute constraint. Long reasoning chains increase the training and evaluation time

substantially. Evaluation was similarly done by prompting for only an answer without reasoning, on MATH-500.

We fine-tune from Qwen3 Base models rather than the Instruct models for this task, for more variance in the evaluation scores. 2 H200 was used for each training run, and checkpoints are saved every 200 steps for evaluation.

HPO's mixed-variable search space is specified in Table 4.

Table 4: Mixed-variable search space for HPO.

Parameter	**Type**	**Range**
Learning rate	Cont.	$[10^{-6}, 10^{-3}]$ normalized to $[0, 1]$
Per device batch size (2^n)	Int.	$[2, 4]$ (effective batch size is $2\times$)
LoRA rank (2^n)	Int.	$[2, 5]$
LoRA alpha (2^n)	Int.	$[2, 5]$
LoRA dropout (2^n)	Cont.	$[0, 0.1]$ normalized to $[0, 1]$
LoRA layers	Int.	$[1, 30]$
LoRA target modules	Cat.	0: `(q_proj, v_proj)` 1: `(q_proj, v_proj, k_proj, o_proj)` 2: `(gate_proj, up_proj, down_proj)` 3: `(all-linear)`
[HPO-MF-Cont] Token fidelity	Cont.	$[10^5, 9.1 \times 10^6]$ normalized to $[0, 1]$
[HPO-MF-Disc] Model fidelity	Cat.	0: HPO-4B emulator 1: HPO-8B emulator

C.2 Data mixture optimization

We construct training mixtures from three datasets spanning instruction following, math, and coding: `tulu-3-sft-personas-instruction-following`, `tulu-3-sft-personas-math`, and `tulu-3-sft-personas-code` [44]. Each training run has two stages: stage 1 (tokens 0-5M) and stage 2 (tokens 5M-10M). At each stage, 10k samples are drawn from the three datasets following a mixing proportion. This roughly corresponds to 5M tokens or less for each 1 epoch. Training hyperparameters are: learning rate 10^{-4}, per device batch size 16 (effective batch size 32), lora rank 8, lora alpha 8, lora dropout 0.05, lora layers 30, and lora target modules `q_proj,v_proj`.

Evaluations were done with lm-eval [25]. We use the `inst_level_strict_acc` metric for IFEval, `math_verify` metric for MATH-500 (4-shot, minerva format), and `pass@1` for MBPP+.

Table 5: Simplex-constrained search space for DMO.

Parameter	**Type**	**Range**	**Constraint**
x_1	IF proportion (stage 1)	$[0, 1]$	$x_1 \geq 0,\ \sum_{i=1}^{3} x_i = 1$
x_2	Math proportion (stage 1)	$[0, 1]$	$x_2 \geq 0,\ \sum_{i=1}^{3} x_i = 1$
x_3	Code proportion (stage 1)	$[0, 1]$	$x_3 \geq 0,\ \sum_{i=1}^{3} x_i = 1$
x_4	IF proportion (stage 2)	$[0, 1]$	$x_4 \geq 0,\ \sum_{j=4}^{6} x_j = 1$
x_5	Math proportion (stage 2)	$[0, 1]$	$x_5 \geq 0,\ \sum_{j=4}^{6} x_j = 1$
x_6	Code proportion (stage 2)	$[0, 1]$	$x_6 \geq 0,\ \sum_{j=4}^{6} x_j = 1$
$\mathcal{X}_{\text{DMO}} \triangleq \{x \in \mathbb{R}^6_{\geq 0} \mid \sum_{i=1}^{3} x_i = 1,\ \sum_{j=4}^{6} x_j = 1\}$. Effective free parameters: 4.			

C.3 Prompt optimization

C.3.1 Prompt generation

Qwen3-14B was used to generate a set of prompts for the candidate set, using multiple meta-prompts to ensure a diverse set of generated prompts. Prompts are embedding using EmbeddingGemma [62].

Duplicate prompts were removed via exact text match and thresholding on embeddings distance. In addition to generated prompts, common prompts selected by prompt optimization papers in Table 6 were also added to the candidate set. Meta-prompts used to generate the rest of the prompts are shown in Table 7.

Table 6: Prompts from prior prompt optimization methods.

Method	**Prompt**
Zero-Shot CoT [41]	"Let's think step by step."
APE [81]	"Let's work this out in a step by step way to be sure we have the right answer."
OPRO [77]	"Take a deep breath and work through this problem step-by-step." "Break this down." "A little bit of arithmetic and a logical approach will help us quickly arrive at the solution to this problem." "Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer." "Let's work through this problem step-by-step:"
PromptBreeder [22]	"SOLUTION:"
ZOPO [34]	"Let's find the solution by using the given information."
INSTINCT [51]	"Let's think about it."
InstructZero [13]	"Let's use the instruction to solve the problem."
Plan-and-Solve+ [72]	"Let's first understand the problem, extract relevant variables and their corresponding numerals, and make a complete plan. Then, let's carry out the plan, calculate intermediate variables (pay attention to correct numerical calculation and commonsense), solve the problem step by step, and show the answer."

C.3.2 Prompt evaluation

We evaluate candidate instruction prompts on the MATH-500 benchmark [31] using the `lm-eval` library [25], on Qwen3-14B in non-thinking mode. The candidate instruction is placed in the system instructions. All evaluations use greedy decoding (temperature $T = 0$), 4-shot prompting, and `math_verify` as the scoring metric. We cap generation at 1024 tokens.

The best instruction prompt recovered by our BO search is: "Write the solution in a concise, technical style typical of advanced mathematics.", achieving 81.0% accuracy on MATH-500. Note that the optimal instruction is specific to our generation configuration, and may differ especially under a different token budget. Our cap of 1024 tokens is substantially smaller than those used in most frontier model papers, for example, the evaluation results in Qwen3 use a max output length of $32,768$ [76]. We chose a smaller budget to keep evaluation cost tractable across 5014 prompt candidates evaluations. Qwen3-14B in non-thinking mode is also able to produce concise, direct answers that fit comfortably within 1024 tokens.

D More experiment details

Monte Carlo (MC)-based implementations were used for LogNEI, PES, MES, GIBBON, JES, and KG acquisition functions, with 128 quasi-MC samples using Sobol sequences per iteration.

D.1 HPO experiments

For the single-fidelity HPO problem, we compare random sampling, Thompson sampling (TS) [12], log expected improvement (LogEI) [3, 38], log noisy expected improvement (LogNEI) [3], upper confidence bounds (UCB) [67], predictive entropy search (PES) [33], max-value entropy search (MES) [73], GIBBON [55], and joint entropy search (JES) [35]. Knowledge gradient (KG) [23] was excluded as it was prohibitively slow for mixed-variable search spaces. These methods were implemented using the BoTorch library [6].

To handle the mixed-variable space, we use a sum-and-product composite kernel comprising a Hamming-distance kernel for categorical variables and an RBF kernel for continuous and discrete

Table 7: Meta-prompt categories and generation prompts. All meta-prompts follow the template: *"Generate [N] diverse and distinct short instructions (1–4 sentences each) that [DESCRIPTION]. Each instruction should be phrased as a direct directive to the solver. Output one instruction per line with no numbering or extra formatting."* The table shows only the [DESCRIPTION] component and count N for each category.

Category	**Description**	**Count**
Reasoning Style	tell a math solver to use a specific *reasoning style* (e.g., step-by-step breakdown, chain-of-thought, working backwards, proof by contradiction, elimination of wrong choices, estimation first, drawing a diagram, considering special cases)	1000
Verbosity	tell a math solver to operate at a specific *level of detail or verbosity* (e.g., give only the final numerical answer, briefly sketch the key steps, show every algebraic manipulation, explain each line as if teaching a beginner, include intermediate checks)	1000
Persona	adopt a specific *voice or role* for solving math problems (e.g., address the reader as a student being guided, write as a peer collaborator, explain as a strict grader, narrate as if thinking aloud, use the tone of a competition preparation coach)	1000
Format	specify a *notation or output format* for math solutions (e.g., write all equations in LaTeX, box the final answer, use numbered lines for each step, present results as a table when possible, state assumptions explicitly before solving)	1000
Subject-Specific	guide a math solver to *identify and leverage the relevant mathematical structure* of a problem before solving (e.g., identify whether the problem is algebraic, geometric, or combinatorial, reframe the problem in the most natural mathematical language, check whether the problem reduces to a known result)	1000
Common Heuristics	are common, general-purpose *problem-solving heuristics* applicable to any math problem (e.g., think step-by-step, verify your answer at the end, re-read the problem carefully before starting, check your solution against special cases, estimate the answer before computing)	1000
Adversarial	are *unhelpful or counterproductive* for solving math problems (e.g., vague directives with no clear method, instructions to skip showing work, contradictory guidance, focus on irrelevant topics, overly restrictive rules that prevent flexible reasoning)	1000
CoT Triggers	serve as *zero-shot chain-of-thought triggers* — brief directives that prompt a solver to reason carefully before giving the final answer, as alternatives to "Let's think step by step" (e.g., "Take a deep breath and work through this carefully.", "Reason slowly and methodically.", "Walk through your logic before concluding.")	200
CoT Structured	impose an *explicit reasoning structure* on the solving process (e.g., plan your approach before computing, write out a scratchpad of intermediate values, number each logical sub-step, annotate each line with the rule or property used, decompose the problem into clearly labelled stages)	200
CoT Verification	ask the solver to *verify or sanity-check* during or after solving (e.g., substitute the answer back into the original equation, estimate the order of magnitude and check against it, consider a simple special case to test your formula, check each step for arithmetic or sign errors)	200
CoT Decomposition	guide the solver to *decompose the problem* into manageable parts before solving (e.g., identify all given quantities and unknowns, break into sub-problems and solve each in turn, reduce to a simpler analogous case first, separate into cases and handle each independently)	200
CoT Reflection	encourage *metacognitive self-monitoring* during solving — prompting the solver to reflect on their own reasoning process (e.g., pause after each step and ask whether it follows logically, flag uncertain steps and revisit them, backtrack if a contradiction is reached, ask "does this answer make intuitive sense?")	200

ordinal variables. Acquisition function optimization alternates between local search over discrete and categorical dimensions, and gradient-based optimization over continuous dimensions, with continuous relaxation and rounding applied to high-cardinality discrete variables (e.g., number of LoRA layers, with 30 discrete values) [6].

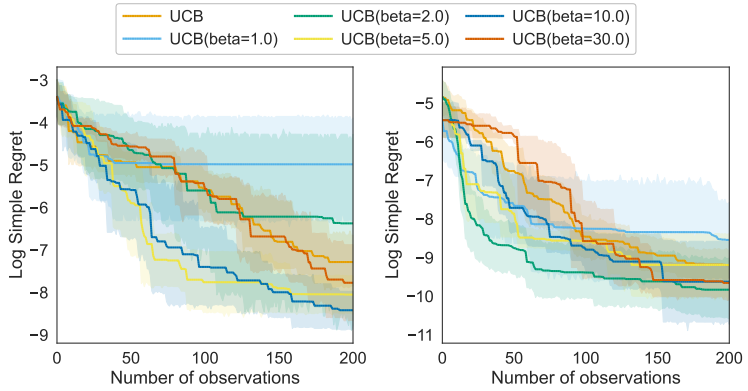
For multi-fidelity problems, we compare MF-MES [68], MF-GIBBON, as well as cost-scaled variants of single-fidelity acquisition functions (LogEI, LogNEI, UCB, and PES). In cost-scaled acquisition functions, acquisition values are divided by an affine cost model $c(x) = \alpha \cdot \mathrm{fid}(x) + 1$, where $\mathrm{fid}(x)$ is the fidelity parameter and α controls the cost sensitivity. For log-valued acquisition functions, $\log(c(x))$ is subtracted from the acquisition values instead. We set $\alpha = 0.005$ for UCB and PES, $\alpha = 0.01$ for EI, and $\alpha = 0.05$ for NEI, MF-MES, and MF-GIBBON. A sensitivity analysis over α is in Sec. D.3.

Finally, we include dedicated HPO baselines from popular HPO libraries: tree-structured Parzen estimator (TPE) [8] and covariance matrix adaptation evolution strategy (CMA-ES) [27, 56] from Optuna [1] for the single-fidelity setting problem, BOHB [21] from SMAC3 [52] for the continuous-fidelity setting, and asynchronous successive halving algorithm (ASHA) [47] from Optuna for the discrete-fidelity setting. Note that CMA-ES cannot handle categorical variables and will fall back to random sampling for the categorical LoRA target module variable.

D.2 Discussion on UCB's exploration parameter

Results comparing different β in UCB are shown in Fig. 10. While β_t following Srinivas et al. [67] often overexplore in practice, a fixed β allows direct control over the exploration-exploitation tradeoff and thus the convergence behavior. Smaller β values reduce exploration, enabling faster initial convergence but potentially at the cost of solution optimality, as seen for $\beta = 1.0$ in both HPO and DMO plots. Conversely, larger β values converge more slowly but tend to yield better final solutions. This suggests that β should be chosen with the compute budget in mind. To balance exploration and exploitation effectively, we selected $\beta = 10.0$ for HPO and $\beta = 2.0$ for DMO for comparison in main regret plots in Fig. 2a and Fig. 3a, where these values enable rapid convergence to a good solution.

These results also suggest that an adaptive β schedule could be beneficial in practice, starting with a smaller β to exploit promising regions early, then increasing it over iterations to ensure sufficient exploration for final solution quality. The schedule of Srinivas et al. [67] is designed for theoretical no-regret guarantees in the infinite-horizon setting and grows too aggressively for finite budgets. A schedule calibrated to the evaluation budget could offer a principled middle ground between rapid early convergence and high final solution quality, which we leave for future work.

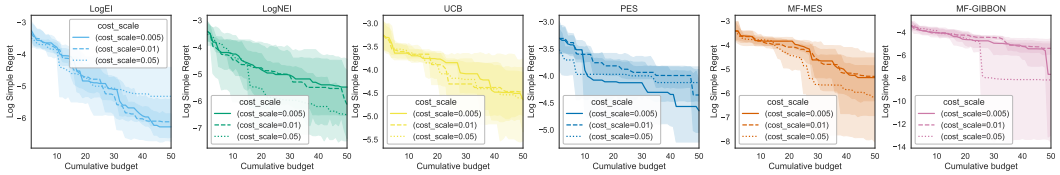


(a) HPO (b) DMO

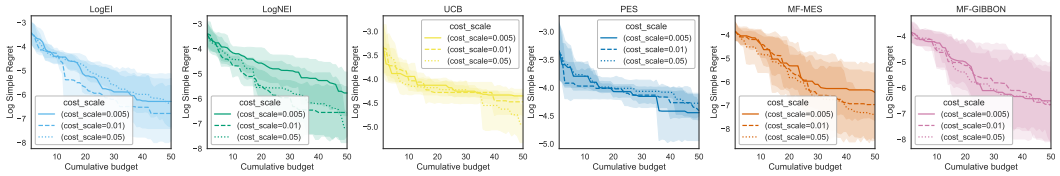
Figure 10: Comparison of β for UCB on HPO and DMO base problems. "UCB" refers to using scheduled β_t from Srinivas et al. [67]

D.3 Cost scale sensitivity

In cost-scaled acquisition functions, acquisition values are divided by an affine cost model $c(x) = \alpha \cdot \mathrm{fid}(x) + 1$, where $\mathrm{fid}(x)$ is the fidelity parameter and α controls the cost sensitivity. Larger α penalizes high-fidelity queries more heavily in their acquisition scores, encouraging more low-fidelity exploration. We evaluate sensitivity to the cost scale parameter α, comparing $\alpha \in \{0.005, 0.01, 0.05\}$ on both multi-fidelity HPO variants (continuous token-based fidelity and discrete model-size fidelity) in Fig. 11. The degree of sensitivity varies across acquisition functions and problems. On the discrete-fidelity problem (HPO-MF-Disc), methods are generally less sensitive to cost scale, likely because the binary fidelity choice limits the ways in which cost weighting can reshape query patterns. On the continuous-fidelity problem (HPO-MF-Cont), LogNEI, MF-MES, and MF-GIBBON exhibit the greatest sensitivity, with certain cost scales yielding substantially better performance over all 50 iterations. MF-GIBBON in particular shows large spread across α values. While LogEI, UCB, and PES are comparatively robust across cost scales on both problems, they also achieve worse final



(a) HPO-MF-Cont



(b) HPO-MF-Disc

Figure 11: Performance on MF HPO problems on different cost scale.

performance. This suggests that cost scale sensitivity and strong performance may go hand-in-hand: methods that respond to cost scale can exploit it to achieve better results, making cost scale an important parameter that requires tuning.

D.4 Fidelity proportions over iterations

Figure 12 shows how each method allocates queries across fidelity levels over the course of optimization. Recall the acquisition functions are all cost scaled, with $\alpha = 0.005$ for UCB and PES, $\alpha = 0.01$ for EI, and $\alpha = 0.05$ for NEI, MF-MES, and MF-GIBBON.

For the continuous-fidelity setting (HPO-MF-Cont), BOHB concentrates mass at a small number of discrete fidelity levels determined by its bracket schedule, as expected from its Hyperband-based design. All GP-based methods, including MF-MES and MF-GIBBON, rapidly saturate near fidelity=1 after a brief initial exploration phase, with only a sparse cloud of low-fidelity evaluations in the early queries. Interestingly, LogEI, LogNEI, and UCB each exhibit a secondary cluster of observations near the lowest fidelity, indicating a bimodal allocation pattern in which intermediate fidelities are largely avoided. This does not appear to harm performance, as LogNEI achieves competitive regret with MF-MES on this task (Figure 2b).

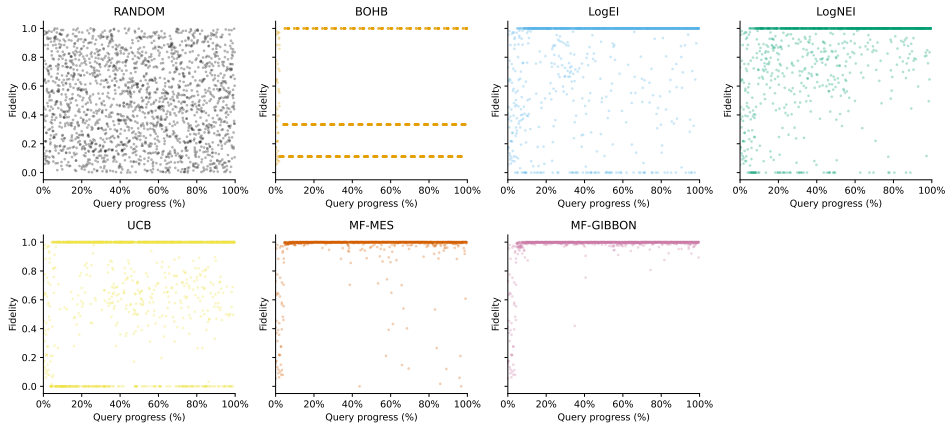
For the discrete-fidelity setting (HPO-MF-Disc), acquisition-based methods generally shift toward high-fidelity evaluations as optimization progresses, suggesting a tendency to exploit the high-fidelity region despite its greater cost. The notable exception is MF-MES, which begins with nearly all high-fidelity queries and progressively reallocates toward the cheap surrogate (fid=0) as the budget is consumed. The empirical performance of MF-MES on this task, achieving the third-lowest regret behind MF-GIBBON and LogNEI (Figure 2c), suggests that this allocation is not detrimental.

Given that methods with vastly different fidelity allocation strategies can perform well, these results suggest a complex relationship between fidelity allocation and final optimization quality, one that likely depends on the specific configurations evaluated at each fidelity level rather than the allocation pattern alone.

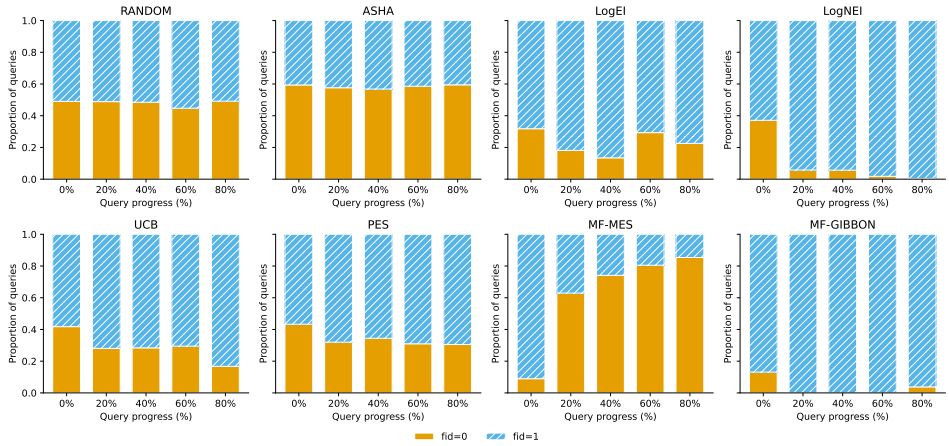
D.5 DMO experiments

For the single-objective problems, we compare random sampling, TS, LogEI, LogNEI, KG, UCB, PES, MES, GIBBON, and JES. For the multi-objective problem, we benchmarked TSEMO [11], NSGA2 [16], ParEGO [40], NEHVI [15], PESMO [32] MESMO [7], and JESMO [71].

Information-theoretic acquisition functions (PES, MES, GIBBON, JES, and their multi-objective variants) require approximations of the global optimum or Pareto-optimal set. Since standard box-constrained sampling cannot satisfy the simplex equality constraints of the data-mixture search space, we instead draw feasible candidates via symmetric Dirichlet sampling and select approximate optima by Thompson sampling and taking the argmax of each posterior draw over the candidate set. For JES, optimal inputs and values are drawn from the same posterior sample to ensure consistency.



(a) HPO-MF-Cont



(b) HPO-MF-Disc

Figure 12: Fidelity proportions of observations on MF HPO problems.

For multi-objective entropy-search methods, we draw multiple joint posterior samples over feasible candidates, extract the non-dominated set from each sample as an approximate Pareto front.

The reference HV point HV^* was set per objective as 10% below the observed minimum, with slack scaled to each objective's range. Minima were computed from surrogate predictions over a dense regular grid on the mixture-ratio simplex.

D.6 PO experiments

For PO experiments, we compare random sampling, TS, LogEI, LogNEI, UCB, and GIBBON, as well as batched, dTuRBO, and dBAxUS variants with LogNEI. dTuRBO and dBAxUS are our discretized adaptations of TuRBO and BAxUS.

TuRBO searches within trust-regions for scalability, expanding and contracting trust-region hyperrectangles upon successive improvements and failures respectively, while BAxUS searches in reduced-dimensional subspaces via random projection and applies trust-region filtering within that subspace. TuRBO and BAxUS are originally designed for continuous spaces, particularly in their trust-region methodology.

We adapted TuRBO and BAxUS to discretized search spaces by representing the trust regions as the k nearest neighbors of the current best solution, where k is doubled after τ_{succ} consecutive improvements and halved after τ_{fail} consecutive non-improvements. The trust region is initialized with $k_{\text{init}} = \lfloor N/2 \rfloor$ candidates, where $N = 5014$ is the total prompt pool size. Both methods use $\tau_{\text{succ}} = 3$, dTuRBO uses a fixed $\tau_{\text{fail}} = 10$, while for dBAxUS, τ_{fail} is set dynamically per subspace

expansion stage, calibrated so that the number of failures required to fully contract the trust region matches the per-stage evaluation budget, following the continuous-space contraction rate in BAxUS.

We show how selected enhancements (batch size, dTuRBO, dBAxUS) affect the simple regret at iteration 200 in Table 8.

Table 8: Delta in log simple regret at iteration 200 vs LogNEI baseline (mean $\pm$ std over 5 seeds). Negative is better as it indicates lower log regret.

Enhancement	**PO128**	**PO256**	**PO512**	**PO768**
`q=5`	-8.67 ± 7.75	-8.23 ± 7.61	-8.40 ± 7.54	-8.85 ± 7.67
`dTuRBO`	-5.87 ± 7.55	-10.61 ± 7.10	-5.12 ± 7.78	-5.67 ± 7.67
`dTuRBO+q=5`	$\mathbf{-14.05 \pm 0.63}^{\dagger}$	$\mathbf{-13.61 \pm 0.46}^{\dagger}$	$\mathbf{-13.78 \pm 0.41}^{\dagger}$	$\mathbf{-14.09 \pm 0.87}^{\dagger}$
`dBAxUS`	-0.43 ± 1.09	-2.27 ± 6.57	-10.82 ± 6.72	-5.99 ± 7.38
`dBAxUS+q=5`	-8.61 ± 6.88	$\mathbf{-13.61 \pm 0.46}^{\dagger}$	$\mathbf{-13.78 \pm 0.41}^{\dagger}$	$\mathbf{-14.09 \pm 0.87}^{\dagger}$

† All 5 seeds hit the log-regret floor (≈ -18.42); deltas coincide when methods fully floor out.

D.7 On evaluation metric

In Sec. 5, for single-objective problems, we evaluate performance using simple regret, which is defined as $f^* - \max_{t' \leq t} f(x_{t'})$. While simple regret provides a clean picture of optimization progress over the budget t, it requires access to the noiseless f in their $\max$ term, which is computable on benchmarks but unavailable in practice.

Another popular evaluation metric is inference regret, defined as $f^* - f(\hat{x}_t)$, where $\hat{x}_t \triangleq \arg\max_{x \in \mathcal{X}} \mu_t(x)$ is the point that maximizes the posterior mean at iteration t. Inference regret is commonly used in entropy search-based methods [73], and it utilizes the surrogate model in estimating the optimal point. We additionally report the *per-step* variant of simple regret $f^* - f(\tilde{x}_t)$, where $\tilde{x}_t \triangleq \arg\max_{t' \leq t} y_{t'}$ is the best point by noisy observations. Both of these metrics are not guaranteed to be monotonically non-increasing, but matches practical evaluation conditions more closely.

Inference regret plots are Fig. 13 and per-step simple regret plots are in Fig. 14. With inference regret, methods are less differentiable in some problems like HPO, suggesting that they may have similar posterior mean despite different observations. Nevertheless, insights drawn from both plots are generally similar to the main simple regret plots in Sec. 5: LogNEI, MF-MES, and MF-GIBBON performs well for multi-fidelity HPO-MF-Disc and DMO experiments, methods using noise models and information perform better in heteroscedastic noise settings, and batched and dTuRBO methods are important for high-dimesional PO problems.

D.8 Wall-clock time for experiments

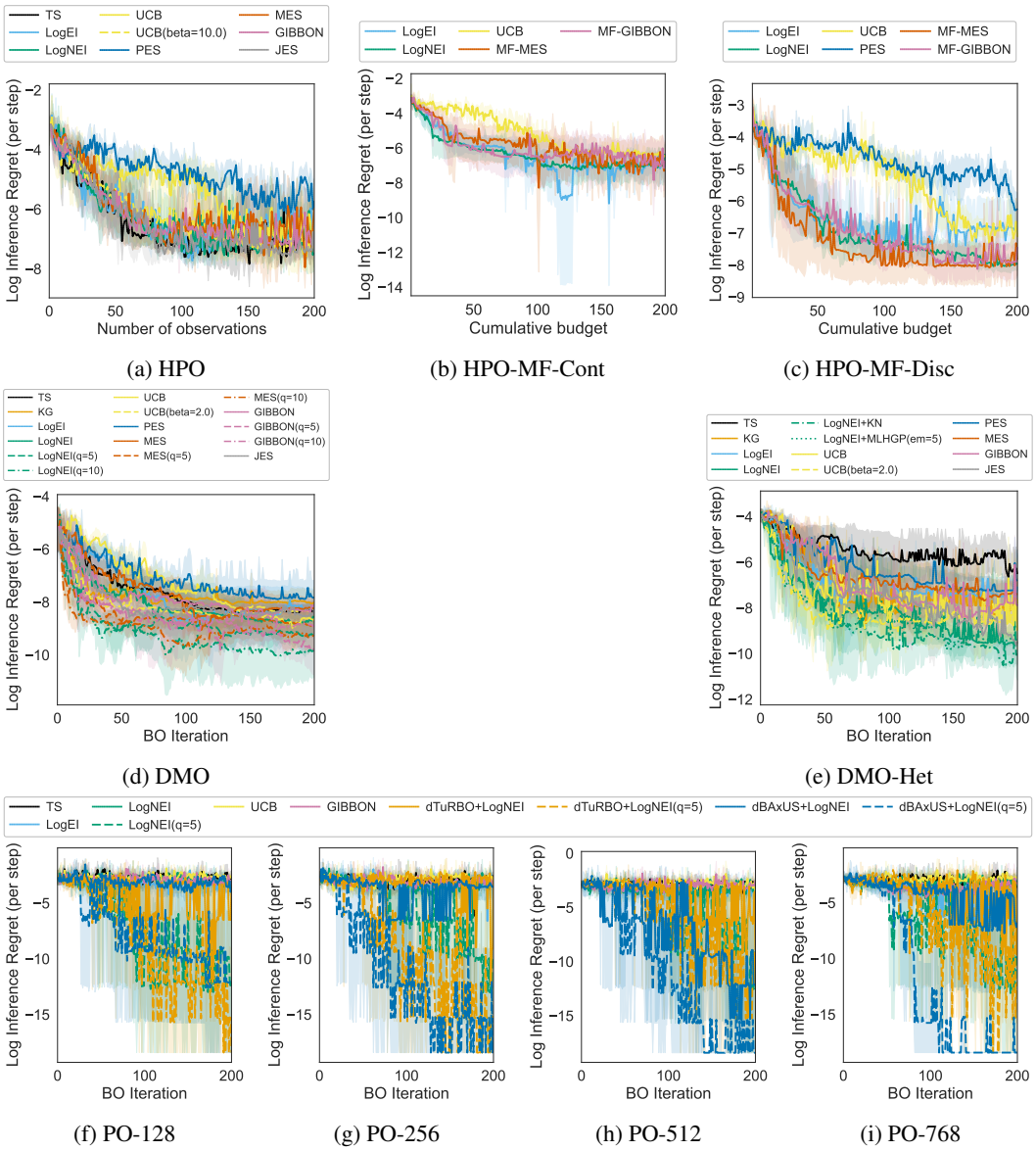
Wall-clock runtimes are reported in Table 9 and Table 10. Note that these runtimes include emulator query time, which is negligible. Most experiments ran on L40 GPUs, with a subset using H100 or H200 GPUs to accommodate the higher VRAM demands of the GPU-accelerated implementation, which scales with the number of observations and input dimensionality (e.g., PO-768 has $d = 768$ which is memory intensive).

E Compute usage

We detail the compute used for generation of BOLT's data, and BO experiments in Table 11.

F Broader Impact

Our work introduces BOLT, a benchmark to spur development of BBO methods for LLM configuration optimization, and does not directly affect LLM model outputs where societal impacts are more pronounced. By enabling more efficient LLM pipeline optimization, BOLT could indirectly

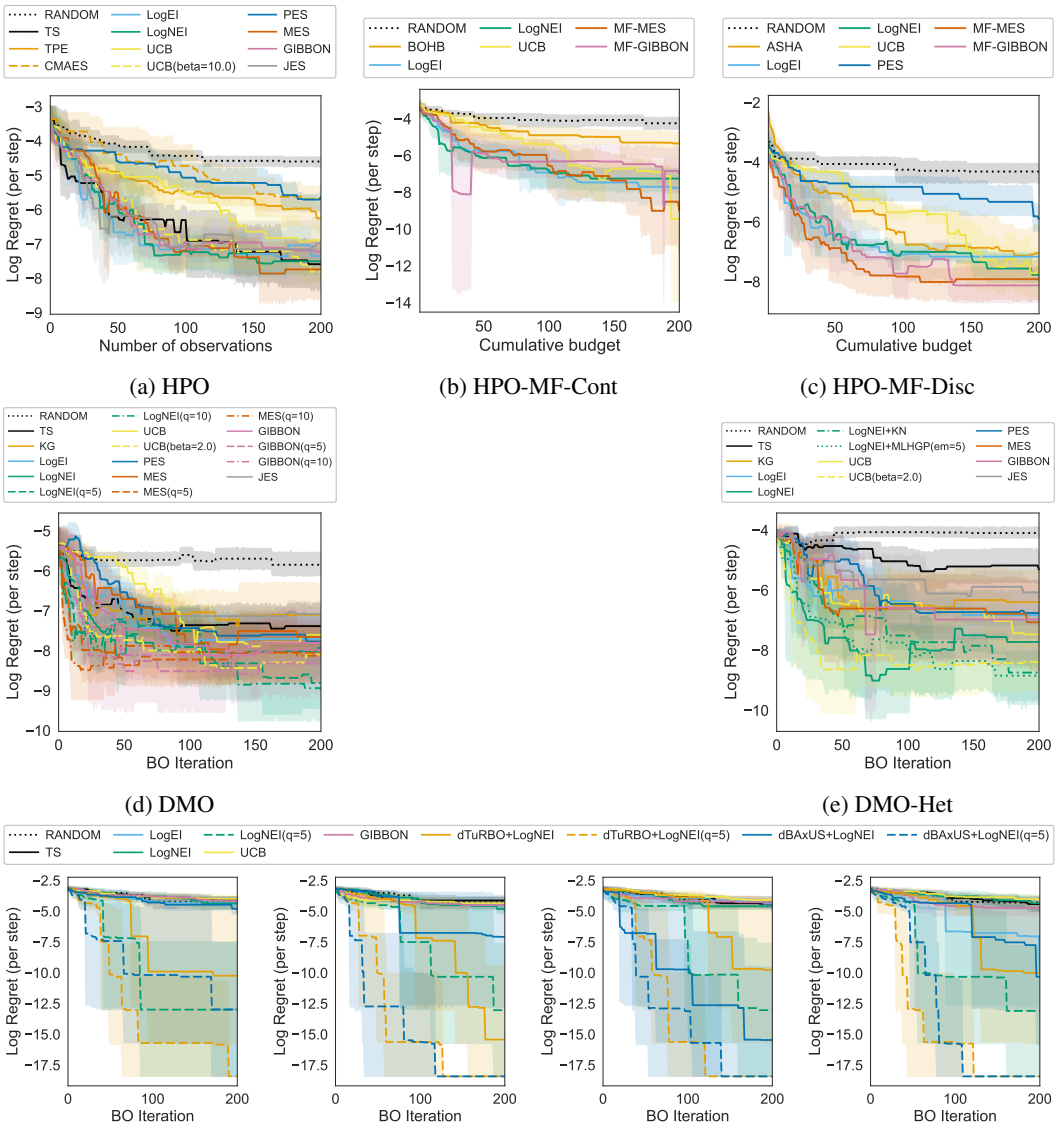


(a) HPO (b) HPO-MF-Cont (c) HPO-MF-Disc

(d) DMO (e) DMO-Het

(f) PO-128 (g) PO-256 (h) PO-512 (i) PO-768

Figure 13: Results on BOLT's HPO problems using inference regret. Random is not included as random sampling does not maintain a posterior mean.



(a) HPO (b) HPO-MF-Cont (c) HPO-MF-Disc

(d) DMO (e) DMO-Het

(f) PO-128 (g) PO-256 (h) PO-512 (i) PO-768

Figure 14: Results on BOLT's HPO problems using per-step simple regret.

Table 9: Mean wall-clock time per BO step — HPO and DMO.

(a) **HPO**

Method	Time / step
HPO	
RANDOM	0.00186 s
TS	10.8 s
TPE	0.0113 s
LogEI	13.5 s
LogNEI	12.9 s
UCB	12.5 s
UCB(beta=10.0)	12.8 s
PES	16.3 s
MES	17 s
GIBBON	14.6 s
JES	17.1 s
HPO-MF-Cont	
RANDOM	0.00654 s
BOHB	0.665 s
LogEI	17.6 s
LogNEI	15.8 s
UCB	22 s
MF-MES	15.8 s
MF-GIBBON	13.6 s
HPO-MF-Disc	
RANDOM	0.00525 s
ASHA	0.0175 s
LogEI	16.4 s
LogNEI	12.6 s
UCB	24.5 s
PES	41.7 s*
MF-MES	29.4 s
MF-GIBBON	18.5 s

(b) **DMO**

Method	Time / step
DMO	
RANDOM	0.00509 s
TS	2.02 s
KG	21 s
LogEI	3.7 s
LogNEI	5.53 s
LogNEI(q=5)	17.1 s
LogNEI(q=10)	1.8 min
UCB	3.1 s
UCB(beta=2.0)	3.15 s
PES	7.03 s
MES	3.31 s
MES(q=5)	7.83 s
MES(q=10)	14.5 s
GIBBON	3.17 s
GIBBON(q=5)	21.4 s
GIBBON(q=10)	52.8 s
JES	4.1 s
DMO-MO	
RANDOM	0.0246 s
TSEMO	2.35 s
NSGA2(pop_size=50)	8.78 s
ParEGO	4.65 s
ParEGO(q=5)	19.3 s*
NEHVI	1.35 min
NEHVI(q=5)	6.33 min
PES_MO	16.6 s
MES_MO	9.14 s
MES_MO(q=5)	53 s*
JES_MO	12.8 s
DMO-Het	
RANDOM	0.00541 s
TS	2.03 s
KG	25.2 s
LogEI	3.44 s
LogNEI	6.59 s
LogNEI+KN	5.98 s
LogNEI+MLHGP(em=5)	7.69 s
UCB	2.96 s
UCB(beta=2.0)	3.38 s
PES	4.38 s
MES	3.11 s
GIBBON	2.92 s
JES	4.42 s

*Ran on H100. †Ran on H200. Otherwise ran on L40

contribute to making capable LLMs more accessible to smaller organizations and researchers with limited resources. On the negative side, accelerating LLM development pipelines could indirectly contribute to faster proliferation of capable language models, amplifying existing societal concerns around misuse.

Table 10: Mean wall-clock time per BO step — PO.

Method	***PO-128***	***PO-256***	***PO-512***	***PO-768***
RANDOM	0.00746 s	0.00966 s	0.0146 s	0.0112 s
TS	8.19 s	9.04 s	8.76 s*	4.98 s†
LogEI	10.3 s	12 s	6.82 s*	1.76 s†
LogNEI	8.55 s	10.6 s	7.15 s*	6.54 s†
LogNEI(q=5)	30.3 s	59.4 s*	1.01 min*	40.7 s†
UCB	7.65 s	9.34 s	7.22 s*	4.65 s†
GIBBON	9.57 s	9.94 s	7.34 s*	4.89 s†
TuRBO+LogNEI	9.8 s	10.3 s	8.85 s*	3.84 s †
TuRBO+LogNEI(q=5)	14.9 s	16.1 s	13.5 s*	7.4 s†
BAxUS+LogNEI	7.77 s	7.04 s*	7.13 s*	4.55 s†
BAxUS+LogNEI(q=5)	8.15 s	7.21 s*	9.79 s*	7.68 s†

*Ran on H100. †Ran on H200. Otherwise ran on L40

Table 11: Compute Usage Summary

Stage	**Model**	**Training**	**Eval**
HPO	8B	29 H200 GPU-days	11 L40 GPU-days
	4B	21 H200 GPU-days	7 L40 GPU-days
DMO	4B	66 H200 GPU-days	29 L40 GPU-days
PO	14B	–	21 H200 GPU-days