BOLT: A BENCHMARK TO DEMOCRATIZE BLACK-BOX OPTIMIZATION RESEARCH FOR EXPENSIVE LLM TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization of LLM training and inference configurations, such as hyperparameters, data mixtures, prompts, and parallelism strategies, is critical to performance. However, it is often done heuristically, which can lead to suboptimal outcomes. These tasks are *noisy*, *expensive*, and *derivative-free*, making them a natural fit for sample-efficient Bayesian optimization (BO) and other black-box optimization (BBO) methods. This direction remains underexplored, largely because LLM training and inference are prohibitively expensive for most BBO researchers, and new methods are often evaluated only on synthetic test functions and small-scale datasets that fail to capture the challenges of LLM optimization. This impedes the development of BBO methods and makes their effectiveness on modern LLM tasks difficult to assess. We introduce BOLT, the first LLM-centric benchmark that democratizes LLM research for the BBO community. BOLT covers a broad set of well-motivated LLM optimization problems, involving multi-fidelity, multi-objective, heteroscedastic noise, high-dimensional search spaces, and black-box constraints. Each problem is grounded in thousands of real LLM experiments, and is fully reproducible and accessible through lightweight emulators or tabular datasets. We benchmark a wide range of BO and BBO methods on BOLT. BO methods generally outperform other baselines, but no single method wins across tasks, and methods remain far from optimal on some problems. These gaps underscore the need for benchmarks grounded in modern LLM tasks. An anonymized version of BOLT is available at `https://github.com/anonom799/bolt`.

1 INTRODUCTION

Optimizing hyperparameters, data mixtures, prompts, and parallelism strategies for LLMs are critical decisions in any LLM pipeline and often have substantial computational cost. These decisions are often made by heuristics or grid search, which can be inefficient and suboptimal (Albalak et al., 2024). BO and other BBO methods offer a principled alternative. They handle noisy observations, require no gradients, and are designed to find good solutions in few evaluations (Frazier, 2018; Garnett, 2023). However, researchers in the BBO community often lack computational resources to validate new methods on LLM problems. As a result, many methods are only validated on numerical or classical optimization benchmarks (e.g., BBOB (Hansen et al., 2009), HPO-B (Arango et al., 2021)), making it difficult to reliably assess their effectiveness on LLM tasks. This impedes the development of BBO methods and leaves the potential of BBO for LLMs largely untapped.

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 introduces of four task families of LLM optimization tasks (Sec. 3.1): LoRA hyperparameter optimization (HPO), data mixture optimization (DMO), prompt optimization (PO), and parallelism configuration optimization (PCO), all of which are grounded in real LLM objectives and are active areas of research in LLMs (Tribes et al., 2024; Xie et al., 2023; Chen et al., 2024; Hemachandra et al., 2026). Each task family consists of multiple problem instances, framed as derivative-free optimization problems that span key challenges

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.

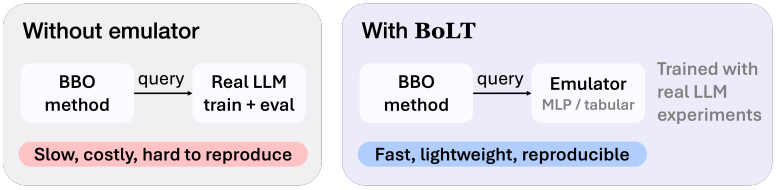


Figure 1: BOLT enables accessible and reproducible evaluation on emulated LLM tasks for BBO methods.

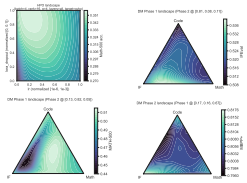


Figure 2: 2D slices of emulator landscapes

in black-box optimization research, including multi-fidelity, multi-objective, heteroscedastic noise, high-dimensional, and black-box constrained settings.

BOLT lowers the barrier for BBO community to adopt LLM tasks as standard, reproducible benchmarks via tabular datasets of precomputed evaluations and *emulators*. In particular, BOLT consist of a suite of emulators (Sec. 3.2), which are surrogate models trained on over 22k real LLM evaluations, with noise levels set to match those measured on the true objectives. Each emulator is validated to ensure that conclusions drawn are consistent with the true objectives. The emulators require *little compute and time to query*, providing access to validated LLM-grounded objectives and enabling reproducible comparisons without large-scale compute.

In Sec. 4, we perform *extensive empirical evaluation of BO and black-box baselines* on the LLM tasks presented in BOLT. BO methods generally outperform standard HPO and evolutionary baselines, while our LLM tasks expose structural and practical challenges for existing methods. These results suggest that BO is a promising but underexplored direction for LLM optimization, and underscore the importance of principled, accessible, and reproducible benchmarking to realize this potential.

The anonymized version of our benchmark is available at `https://github.com/anonom799/bolt`, and emulator models and tabular data are available on the Huggingface Hub at `https://huggingface.co/collections/anonom799/bolt-models`. Links to the code for data collection and experiments, along with usage examples for BOLT's API, are provided in Sec. A.

2 Preliminaries and Related Works

Black-box optimization (BBO). 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 $\varepsilon \sim \mathcal{N}(0, \sigma_{\text{N}}^2)$, and σ_{N} is the noise standard deviation. For instance, finding the hyperparameter that produces the best LLM performance is one such black-box optimization problem.

BBO for LLM. A growing body of work frames LLM optimization problems as BBO. Zhou et al. (2023b) and Yang et al. (2024) cast prompt optimization as a black-box problem with the LLM itself as optimizer. Subsequent work has proposed surrogate-based approaches, such as BO with Gaussian process or NTK kernels (Chen et al., 2024; Hu et al., 2024), neural surrogates over prompt embeddings (Lin et al., 2024a), and neural bandits (Wu et al., 2024), to improve sample efficiency.

HPO for LLM fine-tuning is 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 (Snoek et al., 2012; Shahriari et al., 2015), and recent work extends this to LLM settings. Jang et al. (2024) applied multi-objective BO to jointly optimize hyperparameters and fusion weights, while Seong-Eun et al. (2026) injected LoRA domain knowledge to achieve strong configurations in 30 iterations.

Beyond prompting and HPO, BO has been applied to data mixture optimization. Yen et al. (2025) proposed a multi-fidelity framework that models uncertainty across mixtures and scales, while Chen et al. (2026) used BO to re-weight post-training mixtures, achieving performance gains. Additionally, there are several under-explored use cases for BO in LLM training, such as to maximize the throughput of parallel (multi-GPU) LLM training, as explored by Hemachandra et al. (2026) who investigated how to combine BO with prior domain knowledge about parallel LLM training. These works indicate that BO is effective for a range of LLM optimization problems, with its sample efficiency

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 (cont.)
	HPO-MF-Disc	8	mixed variables, MF (disc.)
Data mixtures	DMO	6	simplex constraint
	DMO-MO	6	simplex constraint, MO
	DMO-Het	6	simplex constraint, het. noise
Prompts	PO-{128,256,512,768}	128/256/512/768	high-dimensional
Parallelism config.	PCO-{16,32,64}	8	black-box constraint

and principled uncertainty quantification. Our benchmark complements these efforts by enabling accessible and reproducible comparison of BBO methods across these emerging problems.

BBO benchmarks. BBO methods are commonly evaluated on synthetic functions, such as Ackley, Branin-Hoo, and Hartmann-6 (Bingham & Surjanovic, 2013), or COCO's (Hansen et al., 2021) 24 BBOB functions (Hansen et al., 2009) and their noisy, large-scale, and bi-objective variants. Synthetic functions are convenient as they are cheap to evaluate, analytically tractable, and have known global optima, but they may not represent real-world BBO challenges well (Palar et al., 2019).

Several real-world BBO benchmarks exist, each targeting specific problem settings. Liang et al. (2021) introduced five datasets in the material science domain and Kumar et al. (2020) 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 (Bliek et al., 2023) provides four expensive real-world problems, such as wind farm layout optimization, with mixed and conditional variables. Dreczkowski et al. (2023) 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 for HPO in machine learning. HPOBench (Eggensperger et al., 2021) and YAHPO Gym (Pfisterer et al., 2022) provide surrogate- and tabular-based HPO problems on classical ML models, with support for multi-fidelity and multi-objective settings. HPO-B (Arango et al., 2021) is another black-box HPO benchmark derived from OpenML. These benchmarks have driven meaningful progress in HPO research. However, they 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.

3 BOLT

BOLT provides a suite of LLM problems for reproducible and accessible benchmarking of BO and BBO methods. We introduce four task families, which include optimization of **LoRA hyperparameters**, **data mixtures**, **prompts**, and **parallelism configurations**. An overview is in Table 1. We provide detailed descriptions, problem formulation, and problem variants for each task family in Sec. 3.1, and more details about the emulators in Sec. 3.2.

BOLT is designed around three principles. It is **grounded in real LLM development workflows**, where each task family corresponds to a critical decision in the LLM pipeline. Objective landscapes are obtained from real LLM experiments, so comparisons on BOLT are meaningful for practical LLM research. BOLT also offers **structured coverage of BO challenges**, including mixed continuous-discrete-categorical search spaces, multi-fidelity observations, multi-objective trade-offs, heteroscedastic noise, constraints, and high dimensions. Finally, BOLT enables **reproducibility through emulators and tabular data**. Running BBO experiments with LLM training or inference in the loop is expensive and often irreproducible due to under-specified details. Thus, every black-box function is backed by a deterministic *emulator*, a surrogate model trained on real experimental runs, or by tabular data, so BBO methods can be evaluated at negligible cost without large-scale compute.

3.1 TASK FAMILIES

Hyperparameter optimization (HPO). Hyperparameters such as learning rate and batch size strongly affect LLM training outcomes (Li et al., 2025a). Our HPO task optimizes LLM performance over a moderate-dimensional search space of LoRA fine-tuning hyperparameters (rank, alpha, target modules), learning rate, and batch size.

We collect datapoints from real LLM training runs with different hyperparameters. Each run fine-tunes a model on a fixed 100k subset of OpenMathInstruct-2 (Toshniwal et al., 2025) using LoRA, and the fine-tuned model is evaluated on MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024) using the `lm-eval` harness framework Gao et al. (2024). In total, this yielded around 8k observations per emulator across all runs and checkpoints. We trained 2 emulators (HPO-4B and HPO-8B), one for each model size (Qwen3-4B and Qwen3-8B (Yang et al., 2025)). 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 8.

Problem variants: The base *HPO* problem is single-fidelity at the full training budget, with a 7-dimensional search space. The other two add a fidelity parameter indicating the compute budget (training tokens or model size) at which the LLM performance is observed. *HPO with continuous token fidelity* (HPO-MF-Cont) varies the number of training tokens seen, while *HPO with discrete model-size fidelity* (HPO-MF-Disc) switches between a 4B low-fidelity and 8B high-fidelity model. Both are 8-dimensional, due to additional fidelity parameter. Evaluating at a higher fidelity costs more optimization budget, but yields more accurate observations.

Data mixture optimization (DMO). Optimization over data mixtures can improve performance for both pretraining (Xie et al., 2023; Liu et al., 2025) and supervised fine-tuning (SFT) (Chen et al., 2026; Li et al., 2025b). Further gains are possible by adjusting mixtures across training stages (Olmo et al., 2025; Zhang et al., 2026), at the cost of a larger search space, or by optimizing over multiple, possibly competing, objectives. Our DMO problem settings span all of these research areas.

Data points for the emulator were collected from real LLM training runs across different training data mixtures. Each training data mixture contains 10k samples randomly drawn from three TULU-3 SFT data domains (instruction following, math, and code) (Lambert et al., 2025) at a specified mixing proportion. We perform SFT on a pretrained Qwen3-4B model and evaluate on three objectives: IFEval (Zhou et al., 2023a), MATH-500 (4-shot), and MBPP+ (Austin et al., 2021; Liu et al., 2023). The DMO-4B emulator is trained on 5931 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 space in Table 9). Evaluations comprise IFEval, MATH-500, and MBPP+, which are combined into different objectives for each problem variant.

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. The base *DMO* problem defines a single objective as the unweighted mean of the three scores. *Multi-objective DMO* (DMO-MO) instead returns the three scores directly via $f_{\text{DMO-MO}} : \mathcal{X}_{\text{DMO}} \to [0,1]^3$, targeting Pareto-front methods such as multi-objective BO (Daulton et al., 2021). *DMO with heteroscedastic noise* (DMO-Het) uses the same unweighted mean objective as DMO, but adds noise $\varepsilon \sim \mathcal{N}(0, \sigma^2_{\text{DMO}}(x))$ to the MATH-500 objective, with $\sigma_{\text{DMO}}(x)$ predicted from the noise emulator, making it a testbed for noise-aware methods (Kersting et al., 2007; Lázaro-Gredilla & Titsias, 2011).

Prompt optimization (PO). Well-chosen prompts can substantially improve performance at inference time (Zhou et al., 2023b), but the prompt space is hard to search. The text space is discrete and unstructured, while its embedding space is high dimensional. Prompt selection at test time is also time-sensitive, and repeated long-context inferences are computationally costly. Our PO dataset makes it assesible for researchers to evaluate PO methods without repeated, expensive inference.

Table 2: 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

We pre-generated a corpus of $5{,}014$ candidate prompts for mathematical reasoning. Each prompt is embedded using EmbeddingGemma (Schechter Vera et al., 2025), which supports Matryoshka Representation Learning (MRL) (Kusupati et al., 2022) for meaningful representations at varying dimensionalities. Candidates are evaluated on MATH-500 (0-shot) to isolate the prompt's influence from few-shot demonstrations. Example prompts and scores are in Table 2. We use tabular data rather than a learned emulator here as interpolated embeddings cannot be decoded back into natural language prompts, and selecting from a candidate pool is standard in prompt optimization (Zhou et al., 2023b; Hu et al., 2024; Lin et al., 2024a; Shi et al., 2024). Further details are in Sec. C.3.

Problem formulation and variants: 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 under prompt i from Qwen3-14B. Acquisition functions optimize over the discrete candidate set. *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.

Parallelism configuration optimization (PCO). The leap in LLM performance is partially due to the ability to scale up LLM training by parallelizing it across many GPUs (Kuchaiev et al., 2019; Huang et al., 2019; Rasley et al., 2020; Zhao et al., 2023). Hyperparameters in a parallelism configuration (PC) controls how the LLM training is parallelized, and affects the training throughput (i.e., number of training steps ran per second) and the maximum GPU memory usage, and can be difficult to optimize. Methods for PCO have been introduced, and generally involves adaptive querying of PCs (DeepSpeed, 2026), a surrogate model based on knowledge about parallel LLM training (Li et al., 2022; Zheng et al., 2022; Lin et al., 2024b), or both (Hemachandra et al., 2026).

We consider PCO for parallel training of LLMs on the NeMo framework (Kuchaiev et al., 2019), where each problem variant involves training on different hardware setups, LLMs, batch sizes and sequence lengths. Since the search space is tabular and discrete, we define a candidate set of PCs and performed an exhaustive run to measure the training throughput and the maximum GPU memory usage of all PCs in the candidate set on real multi-GPU hardware. Further details are in Sec. C.4.

Problem formulation and variants: The search space is $\mathcal{X}_{\text{PCO}} \subset \mathbb{Z}^d$, where each point $x \in \mathcal{X}_{\text{PCO}}$ is a d-dim vector of integers representing a PC, with $|\mathcal{X}_{\text{PCO}}|$ in the order of 10^2. The objective $f_{\text{PCO}} : \mathcal{X}_{\text{PCO}} \to (0, \infty)$ is the training throughput for a given PC. The black-box constraint $c_{\text{PCO}} : \mathcal{X}_{\text{PCO}} \to [-1, 0) \cup \{m_{\text{OOM}}\}$ gives the maximum GPU memory usage normalized to be between -1 and 0, or gives value $m_{\text{OOM}} > 0$ if the training would result in an OOM error. If x results in an OOM error, its throughput is never measured, so $f_{\text{PCO}}(x)$ is estimated as the throughput of its nearest feasible neighbor. Acquisition functions optimize over the discrete candidate set, and each query for x returns both values of $f_{\text{PCO}}(x)$ and $c_{\text{PCO}}(x)$. PCO is defined as a constrained optimization problem where the goal is to maximize $f_{\text{PCO}}(x)$ such that $c_{\text{PCO}}(x) < 0$. *PCO-16*, *PCO-32* and *PCO-64* correspond to PCO problem variants for training on 16, 32 and 64 GPUs respectively, with different training setups. In this task, $d = 8$.

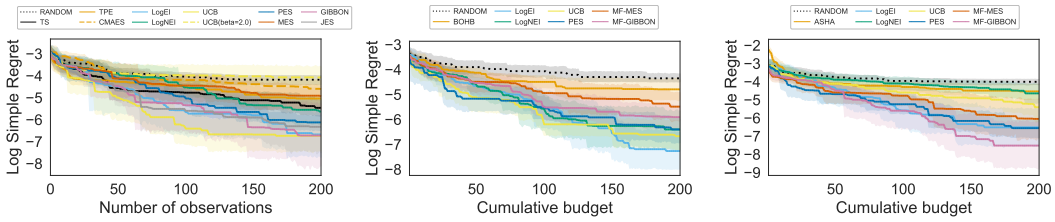
3.2 Emulators

Using emulators in place of expensive oracles is well established in optimization benchmarking (Häse et al., 2021). HPOBench and YAHPO Gym both offer surrogate models, and YAHPO Gym shows surrogate-based benchmarks match real functions more closely than tabular ones. Data points for the HPO and DMO objective emulators were selected via random sampling and adaptive strategies targeting high-variation regions and high-performing configurations.

Emulators are validated on a Sobol-sampled held-out test set. We report Spearman's rank correlation ρ and R^2 to assess candidate ranking and calibration in Table 3. HPO-4B/8B are 3-layer, 1024-width

Table 3: 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}	R^2_{train} / R^2_{test}	n_{train} / n_{test}
HPO-8B	HPO, HPO-MF-Cont, HPO-MF-Disc (HF)	MATH-500	0.972 / 0.940	0.946 / 0.926	8513 / 785
HPO-4B	HPO-MF-Disc (LF)		0.956 / 0.907	0.936 / 0.786	7970 / 819
DMO-4B	DMO, DMO-MO, DMO-Het	IFEval	0.776 / 0.762	0.713 / 0.643	5931 / 788
		MATH-500	0.819 / 0.835	0.642 / 0.664	
		MBPP+	0.793 / 0.761	0.670 / 0.638	
DMO-Noise	DMO-Het	MATH-500 σ	0.826 / –	0.611 / –	100 / –



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

Figure 3: Results on BoLT's HPO problems. MF variants are plotted against cumulative budget, where each observation's cost reflects compute cost, normalized so the highest fidelity's cost is 1.

MLPs and DMO-4B is a 3-layer 512-width MLP. The MLPs were chosen via sweep in Sec. B.2. Train and test metrics are close, indicating little overfitting. HPO emulators achieve high ρ and R^2, while DMO's emulators are moderate but still reasonable, reflecting its more complex landscape (Fig. 2). We further validate emulator fidelity on regions where BBO methods query in Sec. B.4.

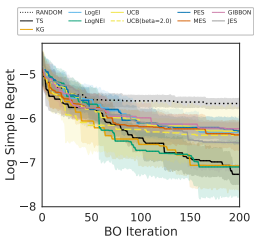
For DMO-Het, the noise emulator is a non-parametric kernel regression model fitted on 100 points, each estimated from 5 repeated evaluations. The noise model architecture was selected via 10-fold cross-validation. More details and analysis on emulators are in Sec. B.

4 Experiments

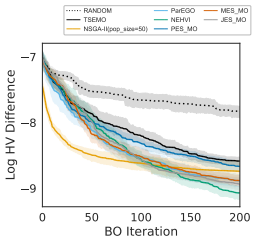
We benchmark a range of common BO and BBO methods on BoLT. Noise levels σ_N are obtained empirically for HPO and DMO problems (see Sec. D.1), except for the hetereoscedastic problem DMO-Het, where the math objective uses a noise model $\sigma_{\text{DMO}}(x)$. PO and PCO experiments set σ_N as 0.001. For single-objective problems, we plot the log simple regret, $\log(f^* - \max_{t' \le t} f(x_{t'}))$, where $x_{t'}$ is the observed point at iteration t'. For multi-objective problems, we plot the log hypervolume difference, $\log(\text{HV}(\text{PF}_t) - \text{HV}^*)$, where PF_t is the noiseless Pareto frontier at iteration t. Both are standard metrics in the optimization literature (Srinivas et al., 2010; Daulton et al., 2021). Each method has 10 initial observations. GP-based methods are implemented in BoTorch (Balandat et al., 2020) with an RBF kernel and dimensionality-scaled lengthscale prior (Hvarfner et al., 2024). HPO, DMO, and PCO experiments are averaged over 10 runs, and PO over 5. 95% confidence intervals are shaded. Further details are in Sec. D.

HPO experiments and analysis. We compare Thompson sampling (TS), expected improvement (EI, NEI), upper confidence bound (UCB), knowledge-gradients (KG), entropy-search methods (PES, MES, GIBBON), and common HPO baselines (TPE, CMA-ES, BOHB, ASHA). For multi-fidelity problems, we adapt acquisition functions to be cost-aware Frazier et al. (2009) and report results against cumulative budget. The cost of each multi-fidelity evaluation is $(1-\gamma) \cdot \text{fid}(x) + \gamma$, where $\text{fid}(x) \in [0, 1]$ is the fidelity parameter and γ is the estimated relative compute cost of the lowest fidelity. More details are in Sec. D.2. Results are in Fig. 3, and FLOPs-based results in Fig. 16.

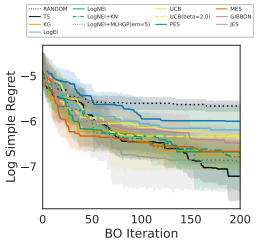
GP-based BO methods generally outperform non-GP-based HPO baselines. (MF-)GIBBON, UCB, and LogEI are generally the stronger performers across all three settings at 200 iterations. While



(a) DMO



(b) DMO-MO



(c) DMO-Het

Figure 4: Results on BOLT's DMO problems. NSGA2 has 50 observations per iteration, while all other methods use only 1 observation per iteration.

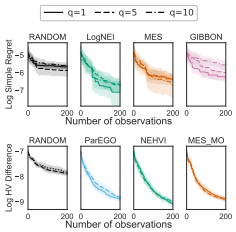


Figure 5: Different batch sizes on DMO (top) and DMO-MO (bottom).

non-GP baselines have substantially faster wall-clock times (Table 17), this overhead is negligible compared to the far costlier, 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. 3a), TPE outperforms most methods under 30 iterations but is subsequently overtaken while UCB ($\beta = 2$) outperforms until around iteration 10. We further analyze UCB's budget dependency through its β parameter that balances exploration and exploitation in Sec. D.3.

MF methods are sensitive to cost scale. 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, but this choice can significantly affect performance and fidelity allocation across methods, affecting the transferability to new problems. Analysis of different cost scales on both MF problems is in Sec. D.4, and fidelity allocation in Sec. D.5.

DMO experiments and analysis. On DMO, we evaluate single- and multi-objective BO methods, and an evolutionary baseline NSGA-II. Entropy-based methods (MES, PES, JES) approximate the optimum's distribution over a candidate set, so we replace the standard box-uniform grid with Dirichlet-sampled points to respect the simplex constraints. For DMO-Het, we add the noise-aware MLHGP which learns a noise model with EM (Kersting et al., 2007), and a GP with known ground-truth noise. More details are in Sec. D.6 and results are in Fig. 4.

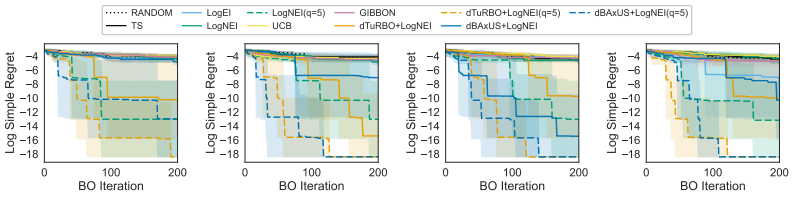
A mix of winners. TS leads single-objective problems at iteration 200. KG and LogNEI also performed well on DMO, and JES on DMO-Het. For multi-objective DMO-MO, NSGA-II(pop_size=50) leads until iteration 100, after which BO methods (NEHVI, JES_MO, ParEGO, MES_MO), overtake it, likely due to more informative sampling despite using $50\times$ fewer function evaluations per iteration.

Noise-aware methods yield modest improvements under heteroscedastic noise. When noise is known (LogNEI+KN) or modelled (LogNEI+MLHGP), performance is better than homoscedastic LogNEI in 200 iterations, indicating that good noise modelling is beneficial. MLHGP takes around 100 iterations to learn a good noise model, which we analyze in Sec. D.7.

Parallel batched candidates can reduce time cost with little performance loss. Since LLM finetuning and evaluation are often conducted in parallel, we compared single versus batched observations per iteration for selected acquisition functions in Fig. 5 on both DMO and DMO-MO, with the same observation budget of 200. Batched BO candidates are selected via sequential greedy conditioning (Balandat et al., 2020). We observed that using a batch size of 5 or 10 to run evaluations in parallel is often comparable to single sequential observations, other than for GIBBON.

PO experiments and analysis. PO is high-dimensional, so we include trust-region and subspace-reduction methods, TuRBO (Eriksson et al., 2019) and BAxUS (Papenmeier et al., 2022). We adapt them for discrete candidate sets as dTuRBO and dBAxUS, where the trust-region hyper-rectangle is replaced with an adaptive k-nearest-neighbors set. More details are in Sec. D.8. Batch size 5 for selected acquitision functions are also evaluated for parallel settings. Results are in Fig. 6.

Standard acquisition functions fail to scale. Their regret plateau between -3 and -5 within 50 iterations, with negligible improvement thereafter, and are indistinguishable from random. A notable



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

Figure 6: Results on BOLT's PO problems. ($q = 5$) indicates batch size 5. dTURBO and dBAxUS are our adapted methods for a discretized space.

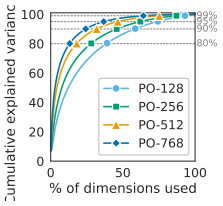
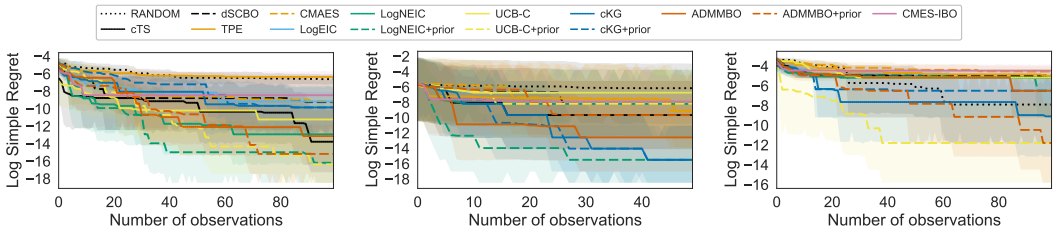


Figure 7: PCA explained variance for PO.



(a) PCO-16 (b) PCO-32 (c) PCO-64

Figure 8: Results on BOLT's PCO problems.

exception is LogEI at PO-768, which descends to around -7, likely benefiting from richer expressivity of the full embedding dimensionality, though it remains far behind trust-region methods.

Batching helps even without trust regions. Log simple regret of LogNEI($q = 5$) reaches approximately -12.5 for all problems without any local search constraints and far exceeds its sequential counterpart, showing that parallel acquisition provides meaningful gains independent of trust regions.

Local search methods lead to substantial performance gains. Across all dimensions, dTuRBO+LogNEI and dBAxUS+LogNEI reach log simple regret of -10 to -15, showing the k-NN adaptation preserves the effectiveness of local search in discretized search spaces. Combined with batch evaluation, dTuRBO+LogNEI($q = 5$) and dBAxUS+LogNEI($q = 5$) achieve the lowest regret, approaching the optimal solution within 200 iterations across almost all PO problems.

Linear subspace reduction may be unnecessary for LLM embedding spaces. At PO-128, dBAxUS underperforms dTuRBO, as PO-128 embeddings need 74.2% of principal components to explain 95% of variance (Fig. 7), violating BAxUS's low-dimensional linear active-subspace assumption (Papenmeier et al., 2022). At higher dimensions, variance concentrates in fewer components and dBAxUS recovers. This suggests subspace reduction is most effective when the embedding space has dimensional redundancy, but LLM embeddings may not always have this property.

PCO experiments and analysis. We evaluated constrained BO methods: LogEIC (Gardner et al., 2014), LogNEIC (Letham et al., 2019), UCB-C (Nguyen et al., 2024), dSCBO (Eriksson & Poloczek, 2021) (discretized adaptation), cTS, CMES-IBO (Takeno et al., 2022), cKG (Ungredda & Branke, 2024), ADMMBO (Ariafar et al., 2019), and BBO baselines (TPE, CMA-ES). For LogNEIC, UCB-C, cKG, and ADMMBO, we include a variant with GP prior means designed from PC knowledge (Hemachandra et al., 2026). More details in Sec. D.9. Regret plots are in Fig. 8, and Fig. 22 shows how close each method's final recommendation is to the optimum across seeds.

The parallelism prior generally helps. On PCO-16, it raises how often UCB-C, LogNEIC, and ADMMBO find the optimum (3→8, 5→8, 5→7 of 10 seeds), giving the three lowest-regret methods. On PCO-64, it raises UCB-C and ADMMBO from 0 and 1 to 5/10 seeds. On PCO-32, LogNEIC+prior leads for the first $\sim$40 iterations and finds the optimum in 8/10 seeds, up from 3.

Performance depends on the landscape. PCO-32 has a sharp optimum, while PCO-16 and PCO-64 have near-optimal plateaus (Sec. D.10), where many methods stall within 2.5% of the optimum without reaching it. On PCO-64, most are even outperformed by random search. No method is best

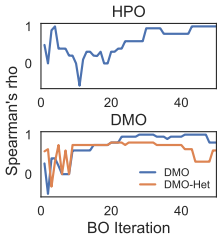


Figure 9: ρ on mean ranks between real and emulated runs.

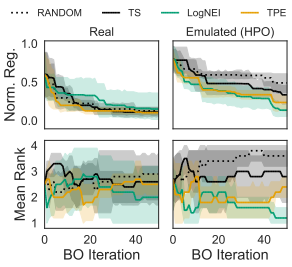


Figure 10: Real versus emulated for HPO.

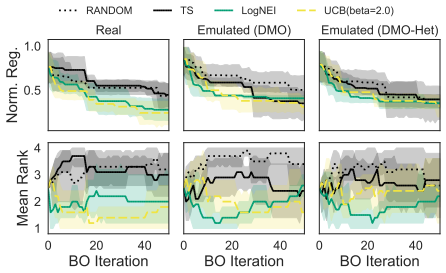


Figure 11: Real versus the emulated DMO and DMO-Het for data mixtures.

on all three problems, but a prior variant is best or tied for best on each: LogNEIC+prior (with cKG) on PCO-32, and UCB-C+prior on PCO-16 and PCO-64.

Transferability to real LLM training. We run selected methods on HPO, DMO, and DMO-Het for 50 iterations over 5 seeds with both real LLM evaluation in the loop and with emulator outputs, to assess how well emulator rankings transfer to real LLM training. Methods are ranked by normalized regret, $(y_{\text{best}} - y_{\text{min}})/(y_{\text{max}} - y_{\text{min}})$, as in HPO libraries like YAHPO Gym, because real runs only yield observed values, not the noiseless objective required for simple regret. We do not use normalized regret in previous sections as it is less stable across seeds than simple regret (Sec. D.11).

As seen in Fig. 9, rankings stabilize and largely agree after around 30 iterations for HPO and 20 for DMO. For HPO in Fig. 10, random performs worst and LogNEI best in both settings, while TPE and TS rankings are tighter in the real runs. LogNEI's worse mean normalized regret on the real experiments stems from a single bad seed, and it still achieves the best mean ranking at 50 iterations. DMO and DMO-Het results are in Fig. 11. DMO's rankings match the real runs more closely than DMO-Het's, with both agreeing on LogNEI $>$ UCB ($\beta = 2$) $>$ TS $>$ Random after iteration 20. DMO-Het's main discrepancy is underestimating UCB, likely because its exploration is more sensitive to the noise model, with noise errors amplified in its confidence bounds.

5 Conclusion

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 shows that mature BO methods transfer to LLM tasks, but no single method wins across tasks, and strong performance often requires adapting to LLM-specific structure, such as simplex-constrained mixtures, high-dimensional discrete prompt spaces, and near-optimal plateaus. Domain knowledge also helps, as parallelism priors improve several constrained BO methods. These challenges reflect the structure of real LLM pipeline optimization problems and highlight gaps for the BBO community to tackle.

Limitations and future extensions. To collect sufficient data to fit reliable emulators for BOLT, we introduced measures such as smaller training token budgets, reduced maximum token limits, and the exclusion of chain-of-thought reasoning in HPO tasks. 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, etc.) 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.

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

AI USE STATEMENT

In this work, we used generative AI tools for assisting the implementation of data collection code and methods benchmarked, refining data collection methodology, formatting of datasets, and some data analysis. We have not used generative AI tools for creating synthetic datasets, and mathematical claims, proofs, hypotheses, and translation are not applicable to this work. Additionally, we used generative AI tools for identifying relevant literature and editing for readability. We have reviewed all AI-assisted work. For example, we verified the methods' implementation against released codes, if available, and looked through the codes manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our dataset and emulators are trained on our own generated LLM results, and does not have privacy, consent, copyright issues. It also does not have potential harmful consequences, beyond those from standard BBO and LLM research, to the best of our knowledge.

REPRODUCIBILITY STATEMENT

We provide extensive implementation details for each set of experiments, including training and inference configurations, kernel choices, constants for noise, cost scale, and more, especially in our appendix. Our codes, emulators, and datasets are also publicly available.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proc. SIGKDD*, pp. 2623–2631, 2019.

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *arXiv:2402.16827*, 2024.

Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Unexpected improvements to expected improvement for Bayesian optimization. In *Proc. NeurIPS*, volume 36, pp. 20577–20612, 2023.

Sebastian Pineda Arango, Hadi Samer Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A Large-Scale Reproducible Benchmark for Black-Box HPO based on OpenML. In *Proc. NeurIPS Track on Datasets and Benchmarks*, 2021.

Setareh Ariafar, Jaume Coll-Font, Dana Brooks, and Jennifer Dy. ADMMBO: Bayesian optimization with unknown constraints using ADMM. *Journal of Machine Learning Research*, 20(123):1–26, 2019.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv:2108.07732*, 2021.

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Proc. NeurIPS*, volume 33, pp. 21524–21538, 2020.

Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective Bayesian optimization. In *Proc. NeurIPS*, volume 32, 2019.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proc. NeurIPS*, volume 24, 2011.

Derek Bingham and Sonja Surjanovic. Virtual library of simulation experiments: Test functions and datasets, 2013. URL https://www.sfu.ca/~ssurjano/optimization.html.

Laurens Bliek, Arthur Guijt, Rickard Karlsson, Sicco Verwer, and Mathijs De Weerdt. Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *Applied Soft Computing*, 147:110744, 2023.

Eric Bradford, Artur M Schweidtmann, and Alexei Lapkin. Efficient multiobjective optimization employing Gaussian processes, spectral sampling and a genetic algorithm. *Journal of global optimization*, 71(2):407–438, 2018.

Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *Proc. NeurIPS*, volume 24, 2011.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. InstructZero: Efficient Instruction Optimization for Black-Box Large Language Models. In *Proc. ICML*, volume 235, pp. 6503–6518. PMLR, 2024.

Zhiliang Chen, Gregory Kang Ruey Lau, Chuan Sheng Foo, and Bryan Kian Hsiang Low. DUET: Optimizing LLM Training Data Mixtures via Noisy Feedback from Unseen, Downstream Evaluation Tasks. In *Proc. ICLR*, volume 2026, pp. 68858–68885, 2026.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *Proc. NeurIPS*, volume 34, pp. 2187–2200, 2021.

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

DeepSpeed. Autotuning – DeepSpeed, 2026. URL https://www.deepspeed.ai/tutorials/autotuning/.

Kamil Dreczkowski, Antoine Grosnit, and Haitham Bou Ammar. Framework and benchmarks for combinatorial and mixed-variable Bayesian optimization. In *Proc. NeurIPS Track on Datasets and Benchmarks*, volume 36, pp. 69464–69489, 2023.

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, Rene Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO. In *Proc. NeurIPS Track on Datasets and Benchmarks*, 2021.

David Eriksson and Martin Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proc. UAI*, pp. 493–503. PMLR, 2021.

David Eriksson and Matthias Poloczek. Scalable constrained Bayesian optimization. In *International conference on Artificial Intelligence and Statistics*, pp. 730–738. PMLR, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Proc. NeurIPS*, volume 32, 2019.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proc. ICML*, pp. 1437–1446. PMLR, 2018.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proc. ICML*, pp. 13481–13544. PMLR, 2024.

Peter Frazier, Warren Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

Peter I Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811*, 2018.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jacob Gardner, Matt Kusner, Zhixiang Xu, Kilian Weinberger, and John Cunningham. Bayesian optimization with inequality constraints. In *Proc. ICML*, pp. 937–945. PMLR, 2014.

Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

Nikolaus Hansen. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pp. 75–102, 2006.

Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. *Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions*. PhD thesis, INRIA, 2009.

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

Florian Häse, Matteo Aldeghi, Riley J Hickman, Loïc M Roch, Melodie Christensen, Elena Liles, Jason E Hein, and Alán Aspuru-Guzik. Olympus: a benchmarking framework for noisy optimization and experiment planning. *Machine Learning: Science and Technology*, 2(3):035021, 2021.

Apivich Hemachandra, Yizhan Han, See-Kiong Ng, and Bryan Kian Hsiang Low. Bayesian Optimization with Early Trial Termination for Speeding Up Parallel Neural Network Training. In *ICML 2026 DEMO Workshop*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proc. NeurIPS Track on Datasets and Benchmarks*, 2021.

Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective Bayesian optimization. In *Proc. ICML*, pp. 1492–1501. PMLR, 2016.

José M Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Proc. NeurIPS*, volume 27, 2014.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. Localized zeroth-order prompt optimization. In *Proc. NeurIPS*, volume 37, pp. 86309–86345, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Proc. NeurIPS*, 2019.

Carl Hvarfner, Frank Hutter, and Luigi Nardi. Joint entropy search for maximally-informed Bayesian optimization. In *Proc. NeurIPS*, volume 35, pp. 11494–11506, 2022.

Carl Hvarfner, Erik Orm Hellsten, and Luigi Nardi. Vanilla Bayesian optimization performs great in high dimensions. In *Proc. ICML*, pp. 20793–20817. PMLR, 2024.

Chaeyun Jang, Hyungi Lee, Jungtaek Kim, and Juho Lee. Model fusion through Bayesian optimization in language model fine-tuning. In *Proc. NeurIPS*, volume 37, pp. 29878–29912, 2024.

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

Kristian Kersting, Christian Plagemann, Patrick Pfaff, and Wolfram Burgard. Most likely heteroscedastic Gaussian process regression. In *Proc. ICML*, pp. 393–400, 2007.

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.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Proc. NeurIPS*, volume 35, pp. 22199–22213, 2022.

Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, Patrice Castonguay, Mariya Popova, Jocelyn Huang, and Jonathan M. Cohen. NeMo: A toolkit for building AI applications using neural modules. arXiv:1909.09577, 2019.

Abhishek Kumar, Guohua Wu, Mostafa Z Ali, Rammohan Mallipeddi, Ponnuthurai Nagaratnam Suganthan, and Swagatam 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.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *Proc. NeurIPS*, volume 35, pp. 30233–30249, 2022.

Nathan Lambert, Jacob Daniel Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hanna Hajishirzi. TÜLU 3: Pushing Frontiers in Open Language Model Post-Training. In *Proc. COLM*, 2025.

Miguel Lázaro-Gredilla and Michalis K Titsias. Variational heteroscedastic Gaussian process regression. In *Proc. ICML*, pp. 841–848, 2011.

Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14, 06 2019. doi: 10.1214/18-BA1110.

Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. AMP: Automatically finding model parallel strategies with heterogeneity awareness. In *Proc. NeurIPS*, 2022.

Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable Scale: Part I, Step Law – Optimal Hyperparameter Scaling Law in Large Language Model Pretraining. *arXiv:2503.04715*, 2025a.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *machine learning and systems*, volume 2, pp. 230–246, 2020.

Yuan Li, Zhengzhong Liu, and Eric Xing. Data Mixing Optimization for Supervised Fine-Tuning of Large Language Models. In *Proc. ICML*, pp. 35419–35437. PMLR, 2025b.

Qiaohao Liang, Aldair E Gongora, Zekun Ren, Armi Tiihonen, Zhe Liu, Shijing Sun, James R Deneault, Daniil Bash, Flore Mekki-Berrada, Saif A Khan, Kedar Hippalgaonkar, Benji Maruyama, Keith A Brown, John Fisher III, and Tonio Buonassisi. Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains. *npj Computational Materials*, 7(1):188, 2021.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *Proc. ICLR*, 2024.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Use Your INSTINCT: INSTruction optimization for LLMs usIng Neural bandits Coupled with Transformers. In *Proc. ICML*, pp. 30317–30345. PMLR, 2024a.

Zhiqi Lin, Youshan Miao, Quanlu Zhang, Fan Yang, Yi Zhu, Cheng Li, Saeed Maleki, Xu Cao, Ning Shang, Yilei Yang, Weijiang Xu, Mao Yang, Lintao Zhang, and Lidong Zhou. nnScaler: Constraint-guided parallelization plan generation for deep learning training. In *Proc. OSDI*, 2024b.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Proc. NeurIPS*, volume 36, pp. 21558–21572, 2023.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. RegMix: Data Mixture as Regression for Language Model Pre-training. In *Proc. ICLR*, 2025.

Henry B Moss, David S Leslie, Javier Gonzalez, and Paul Rayson. GIBBON: General-purpose Information-Based Bayesian OptimisatioN. *Journal of Machine Learning Research*, 22(235):1–49, 2021.

Quoc Phong Nguyen, Wan Theng Ruth Chew, Le Song, Bryan Kian Hsiang Low, and Patrick Jaillet. Optimistic Bayesian optimization with unknown constraints. In *Proc. ICLR*, volume 2024, pp. 2797–2808, 2024.

Masahiro Nomura, Shuhei Watanabe, Youhei Akimoto, Yoshihiko Ozaki, and Masaki Onishi. Warm starting CMA-ES for hyperparameter optimization. In *AAAI conference on Artificial Intelligence*, volume 35, pp. 9188–9196, 2021.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3. *arXiv:2512.13961*, 2025.

Pramudita Satria Palar, Rhea Patricia Liem, Lavi Rizki Zuhal, and Koji Shimoyama. On the use of surrogate models in engineering design optimization and exploration: The key issues. In *Proc. GECCO companion*, pp. 1592–1602, 2019.

Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. Increasing the scope as you learn: Adaptive Bayesian optimization in nested subspaces. In *Proc. NeurIPS*, volume 35, pp. 11586–11601, 2022.

Leonard Papenmeier, Matthias Poloczek, and Luigi Nardi. Understanding high-dimensional Bayesian optimization. In *Proc. ICML*, pp. 47902–47923. PMLR, 2025.

Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization. In *Proc. International Conference on Automated Machine Learning*, volume 188, pp. 3/1–39. PMLR, 2022.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proc. SIGKDD*, 2020.

Henrique* Schechter Vera, Sahil* Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, Alice Lisak, Min Choi, Lucas Gonzalez, Omar Sanseviero, Glenn Cameron, Ian Ballantyne, Kat Black, Kaifeng Chen, Weiyi Wang, Zhe Li, Gus Martins, Jinhyuk Lee, Mark Sherwood, Juyeong Ji, Renjie Wu, Jingxiao Zheng, Jyotinder Singh, Abheesht Sharma, Divya Sreepat, Aashi Jain, Adham Elarabawy, AJ Co,

Andreas Doumanoglou, Babak Samari, Ben Hora, Brian Potetz, Dahun Kim, Enrique Alfonseca, Fedor Moiseev, Feng Han, Frank Palma Gomez, Gustavo Hernández Ábrego, Hesen Zhang, Hui Hui, Jay Han, Karan Gill, Ke Chen, Koert Chen, Madhuri Shanbhogue, Michael Boratko, Paul Suganthan, Sai Meher Karthik Duddu, Sandeep Mariserla, Setareh Ariafar, Shanfeng Zhang, Shijie Zhang, Simon Baumgartner, Sonam Goenka, Steve Qiu, Tanmaya Dabral, Trevor Walker, Vikram Rao, Waleed Khawaja, Wenlei Zhou, Xiaoqi Ren, Ye Xia, Yichang Chen, Yi-Ting Chen, Zhe Dong, Zhongli Ding, Francesco Visin, Gaël Liu, Jiageng Zhang, Kathleen Kenealy, Michelle Casbon, Ravin Kumar, Thomas Mesnard, Zach Gleicher, Cormac Brick, Olivier Lacombe, Adam Roberts, Yunhsuan Sung, Raphael Hoffmann, Tris Warkentin, Armand Joulin, Tom Duerig, and Mojtaba Seyedhosseini. EmbeddingGemma: Powerful and Lightweight Text Representations. *arXiv:2509.20354*, 2025.

Baek Seong-Eun, Lee Jung-Mok, Kim Sung-Bin, and Tae-Hyun Oh. Efficient hyper-parameter search for LoRA via language-aided Bayesian optimization. *arXiv:2602.11171*, 2026.

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

Chengshuai Shi, Kun Yang, Zihan Chen, Jundong Li, Jing Yang, and Cong Shen. Efficient prompt optimization through the lens of best arm identification. In *Proc. NeurIPS*, volume 37, pp. 99646–99685, 2024.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. NeurIPS*, volume 25, 2012.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. ICML*, 2010.

Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In *Proc. ICML*, pp. 9334–9345. PMLR, 2020.

Shion Takeno, Tomoyuki Tamura, Kazuki Shitara, and Masayuki Karasuyama. Sequential and parallel constrained max-value entropy search via information lower bound. In *Proc. ICML*, pp. 20960–20986. PMLR, 2022.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. OpenMathInstruct-2: Accelerating AI for Math with Massive Open-Source Instruction Data. In *Proc. ICLR*, 2025.

Christophe Tribes, Sacha Benarroch-Lelong, Peng Lu, and Ivan Kobyzev. Hyperparameter optimization for large language model instruction-tuning. In *AAAI Conference on Artificial Intelligence*, 2024.

Ben Tu, Axel Gandy, Nikolas Kantas, and Behrang Shafei. Joint entropy search for multi-objective Bayesian optimization. In *Proc. NeurIPS*, volume 35, pp. 9922–9938, 2022.

Juan Ungredda and Juergen Branke. Bayesian optimisation for constrained problems. *ACM Transactions on Modeling and Computer Simulation*, 34(2):1–26, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proc. ACL (volume 1: long papers)*, pp. 2609–2634, 2023.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *Proc. ICML*, pp. 3627–3635. PMLR, 2017.

Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with ease? efficient ordering-aware automated selection of exemplars. *Proc. NeurIPS*, 37:122706–122740, 2024.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. *Proc. NeurIPS*, 36:69798–69818, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv:2505.09388*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *Proc. ICLR*, 2024.

Thomson Yen, Andrew Wei Tung Siah, Haozhe Chen, C Daniel Guetta, Tianyi Peng, and Hongseok Namkoong. Data mixture optimization: A multi-fidelity multi-scale Bayesian framework. In *Proc. NeurIPS*, volume 38, pp. 98818–98842, 2025.

Yang Zhang, Amr Mohamed, Hadi Abdine, Guokan Shang, and Michalis Vazirgiannis. Beyond random sampling: Efficient language model pretraining via curriculum learning. In *Proc. EACL (Volume 1: Long Papers)*, pp. 5776–5794, 2026.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. In *Proc. VLDB*, 2023.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *Proc. OSDI*, 2022.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv:2311.07911*, 2023a.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *Proc. ICLR*, 2023b.

A More details on BoLT's codes and usage

A.1 Codes for BoLT's data collection and experiments

Data collection codes are in `https://github.com/anonom799/bolt-data` and experiment codes and results are in `https://github.com/anonom799/bolt-experiments`.

A.2 BoLT usage examples

```python
from bolt import HPO
import torch

# Instantiate a problem with default noise_std
problem = HPO()

# 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

B.1 Emulator Data Points

For the HPO and DMO objective emulators, data points are selected via random sampling and two adaptive sampling strategies: one maximizes emulator gradient magnitudes to target high-variation regions, and one maximizes predicted objectives to anchor high-performing configurations. Random sampling provides broad coverage while the adaptive strategies concentrate effort where it matters most.

B.2 Emulator MLP Sweep and Training

Emulator sweep. The MLP configuration for each HPO and DMO objective was selected via a sweep of MLP configurations on depth and width, shown in Table 4 and Table 5. The final selected MLP configurations are 3-layer, 1024-width MLPs for HPO-4B and HPO-8B emulators, and 3-layer, 512-width MLP for DMO-4B.

Emulator training. The training recipes are optimized for each emulator. HPO-4B is trained for 4k steps, with cosine scheduler, HPO-8B for 20k steps with no scheduler, both at learning rate 10^{-4}. For DMO-4B, it was trained for 50k steps, with a learning rate of 3×10^{-4} and a cosine scheduler.

B.3 Emulator analysis

Actual and predicted rank comparison of data points in the test set is shown Fig. 12, 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.

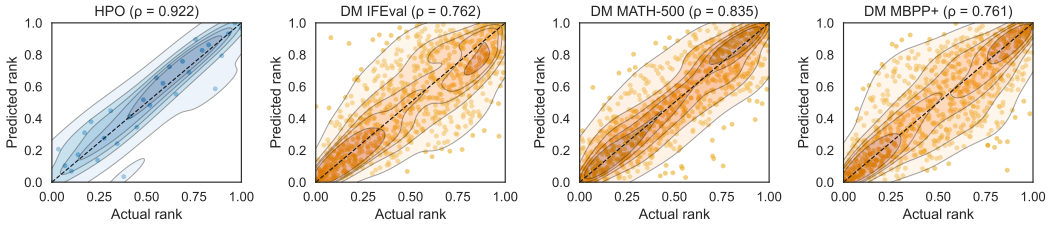


Figure 12: Real and predicted ranks on test set.

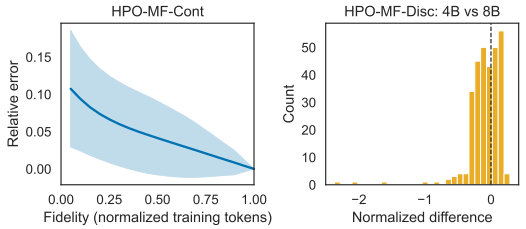


Figure 13: Differences in emulated objective at different fidelities

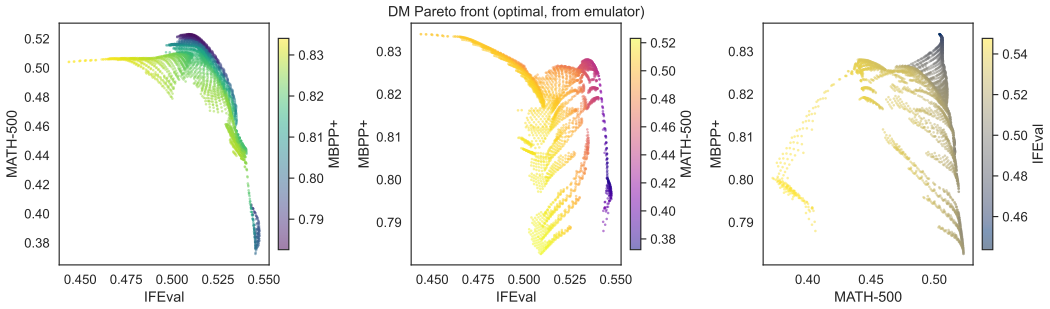


Figure 14: Estimated optimal Pareto front, as 2D slices of a 3D space

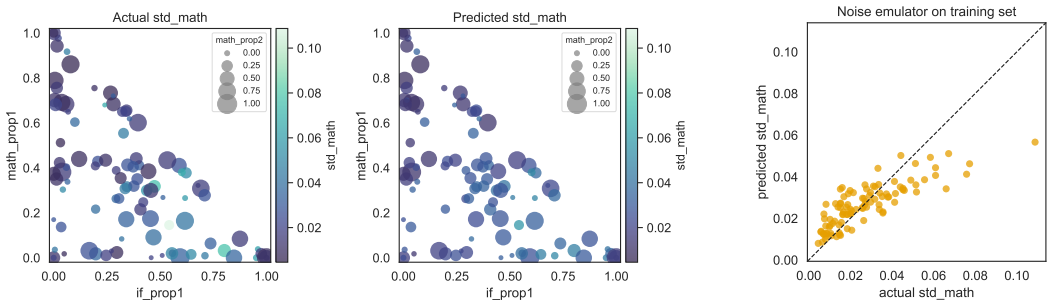


Figure 15: Noise emulator predictions for MATH-500 standard deviation. if_prop1 indicates mixture proportion of information following data at stage 1 of the curriculum, while math_prop1 and math_prop2 represent math data proportions at stage 1 and 2 respectively.

Table 4: MLP sweep for HPO emulators. 5 seeds per configuration. The R^2 and Spearman's ρ are calculated on a held-out test set (819 for HPO-4B, 785 for HPO-8B). ρ_{25}: Spearman over the top quartile of the true objective. ρ_{mt}: Spearman at the full token budget. Best value per column, per panel is bolded and second-best underlined. Final selected MLP configuration is also bolded.

Depth	Width	R^2	ρ	ρ_{25}	ρ_{mt}
HPO-4B					
2	256	.785±.021	.898±.009	.538±.035	.934±.016
2	512	.803±.012	.904±.002	.634±.016	.933±.016
2	1024	.802±.009	.908±.004	**.660**±.006	.938±.012
3	256	.812±.015	.902±.004	.555±.008	.926±.027
3	512	.821±.016	.905±.003	.618±.017	.928±.007
3	**1024**	.835±.007	.908±.004	.634±.006	**.940**±.009
4	256	.814±.011	.904±.007	.544±.043	.928±.035
4	512	.820±.011	.906±.006	.611±.028	.936±.009
4	1024	**.841**±.008	.908±.004	.631±.011	.935±.005
5	256	.799±.009	.899±.012	.533±.032	.932±.031
5	512	.817±.010	.907±.003	.593±.034	.939±.007
5	1024	.836±.005	**.909**±.002	.604±.026	.935±.004
HPO-8B					
2	256	.914±.015	.933±.006	.772±.024	.881±.018
2	512	.916±.012	.941±.007	**.810**±.024	.885±.005
2	1024	.912±.014	.944±.003	.808±.012	.887±.004
3	256	.911±.024	.934±.009	.762±.037	.873±.018
3	512	.920±.010	.940±.007	.781±.014	.881±.009
3	**1024**	**.927**±.006	**.946**±.005	.804±.030	**.893**±.009
4	256	.911±.017	.937±.008	.732±.044	.883±.029
4	512	.920±.012	.939±.007	.774±.038	**.893**±.013
4	1024	.926±.007	.942±.007	.789±.016	.884±.008
5	256	.905±.005	.935±.007	.709±.048	.876±.011
5	512	.920±.010	.938±.007	.777±.033	.876±.010
5	1024	.924±.009	.941±.008	.767±.031	.888±.008

Table 5: MLP sweep for the DMO-4B emulator. 5 seeds per configuation, on 788 held-out Sobol points. R^2 / ρ / ρ_{25} are unweighted means over ifeval / math / code. Best value per column bolded and second-best underlined. Selected MLP configuration is also in bold.

Depth	Width	R^2	ρ	ρ_{25}
2	256	.644±.001	.785±.001	.446±.002
2	512	.646±.001	**.786**±.000	.447±.003
2	1024	.646±.000	**.786**±.000	.446±.001
3	256	.645±.001	.784±.001	.446±.003
3	**512**	**.647**±.001	**.786**±.000	**.448**±.003
3	1024	**.647**±.001	.785±.000	.444±.001
4	256	.642±.001	.782±.001	.441±.003
4	512	**.647**±.001	.785±.000	.445±.000
4	1024	**.647**±.001	.785±.000	.444±.003
5	256	.639±.000	.781±.001	.439±.002
5	512	.645±.000	.784±.000	.442±.004

Multi-fidelity settings. We analyze the HPO emulators under multi-fidelity settings in Fig. 13. 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. 14, 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+

Method	CV R^2	CV ρ	Train R^2	Train ρ
Ridge (Linear)	0.269	0.518	0.339	0.579
Ridge (Quadratic)	**0.338**	0.549	0.471	0.680
Kernel ridge (Laplacian)	0.298	**0.598**	**0.611**	**0.826**
Kernel ridge (RBF)	0.283	0.575	0.449	0.694
Kernel ridge (Quadratic)	0.233	0.540	0.344	0.639
GP (Matérn-5/2)	0.242	0.564	0.549	0.774
GP (RBF)	0.233	0.551	0.494	0.743

Table 6: Noise model ablation to model Math-500 objective's standard deviation on the 6 mixture-proportion features (100 configs, 5 seeds each). Cross-validation (CV) columns report the mean of R^2/Spearman ρ over random 10-fold (90/10) splits. Train columns are fit on all 100 configs.

Table 7: Emulator fit on the points visited by the real BBO runs (DMO: TS, LogNEI, UCB($\beta = 2.0$); HPO: TS, LogNEI, TPE). Brackets are 95% bootstrap CIs.

Task	**Objective**	n	R^2	Spearman ρ	RMSE
DMO	IFEval	798	0.68 [0.62, 0.72]	0.70 [0.65, 0.74]	0.013
	MATH-500	798	0.64 [0.59, 0.69]	0.70 [0.66, 0.74]	0.022
	MBPP+	798	0.61 [0.56, 0.65]	0.76 [0.72, 0.79]	0.009
	Mean	798	0.60 [0.55, 0.63]	0.73 [0.70, 0.77]	0.009
HPO	MATH-500	800	0.50 [0.40, 0.58]	0.66 [0.61, 0.70]	0.044

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 $k = 10$ and the Spearman'ρ metric in Table 6. The final model was trained on all 100 data samples using the selected hyperparameters: a Laplacian kernel with $\gamma = 0.5$. 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. 15 on 3 dimensions. 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 100-point training set.

B.4 Emulator validation on real BBO runs

We also validated the emulators in regions where BBO algorithms query, based on the real BBO runs from experiments in Figs. 9 to 11. We report the R^2, Spearman's ρ, and RSME between the real and emulated objectives, of the configurations visited by BBO methods on real LLM runs, in Table 7. BO methods tend to concentrate their evaluations near the optimum, and scores are often similar, so these points can be hard to differentiate. Even so, the emulators explain 50% to 68% of the variance in real scores (R^2), rank them with Spearman ρ of 0.66 to 0.76, and show near-zero bias, with RMSE of at most 0.022 on DMO and 0.044 on HPO.

Table 8: 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 Details on data generation for emulators

C.1 Hyperparameter optimization

OpenMathInstruct-2 (Toshniwal et al., 2025) 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, using 2 H200 for each training run. Checkpoints are saved every 200 steps for evaluation. We evaluate these checkpoints to collect multiple observations for each training run, and used early stopping to increase coverage of the hyperparameter space within a fixed compute budget.

HPO's mixed-variable search space is specified in Table 8.

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` (Lambert et al., 2025). 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 (Gao et al., 2024). 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+.

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 (Schechter Vera et al., 2025). 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 10 were also added to the candidate set. Meta-prompts used to generate the rest of the prompts are shown in Table 11.

Table 9: 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.			

Table 10: Prompts from prior prompt optimization methods.

Method	Prompt
Zero-Shot CoT (Kojima et al., 2022)	"Let's think step by step."
APE (Zhou et al., 2023b)	"Let's work this out in a step by step way to be sure we have the right answer."
OPRO (Yang et al., 2024)	"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 (Fernando et al., 2024)	"SOLUTION:"
ZOPO (Hu et al., 2024)	"Let's find the solution by using the given information."
INSTINCT (Lin et al., 2024a)	"Let's think about it."
InstructZero (Chen et al., 2024)	"Let's use the instruction to solve the problem."
Plan-and-Solve+ (Wang et al., 2023)	"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 (Hendrycks et al., 2021) using the `lm-eval` library (Gao et al., 2024), 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 $32,768$ (Yang et al., 2025). 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.

Table 11: 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

C.4 Parallelism configuration optimization

In Table 12, we list the training configuration of each problem variant, including the hardware setup, LLM, batch size and sequence length.

In Table 13, we list the hyperparameters considered in the parallelism configuration (PC) for the PCO problem. Note that we apply logarithm to several of the hyperparameters due to their value generally being considered on an exponential scaling. In practice, we require several constraints to be met for the PC, namely (a) $\texttt{ldp} + \texttt{ltp} + \texttt{lpp} + \texttt{lcp} = \lg \texttt{ngpus}$ so that the GPUs are fully utilized; (b) if $\texttt{ldp} = 0$ then $\texttt{fsdp} = 0$ as a default value; and (c) if $\texttt{lpp} = 0$ then $\texttt{lvp} = 0$ as a default value. Additionally, we apply additional constraints to further prune the candidate space, namely (a) setting $\texttt{ldp} > 0$ and $\texttt{ltp} \leq 2$ to encourage some DP and discourage too much TP; (b) setting

Table 12: Training setup for each PCO problem variant.

Problem variant	PCO-16	PCO-32	PCO-64
GPU type	NVIDIA H100	NVIDIA GH200	NVIDIA GH200
# GPUs (`ngpus`)	16	32	64
# Nodes	2	8	16
LLM	Llama-2-7B	Qwen3-14B	Qwen3-32B
Batch size	512	512	512
Sequence length	2048	8192	8192
Size of $\mathcal{X}_{\text{PCO}}$	919	379	780

Table 13: Hyperparameters in the parallelism configuration for the PCO problem. Here, all logarithms are of base 2, and we use $\lg x$ to denote log base 2 of x. Note that `fsdp` hyperparameter encodes discrete setting for FSDP (which are equivalent to the ZeRO stages); setting `fsdp` to 0, 1, 2 or 3 is equivalent to using the flags `no_shard`, `optim`, `optim_grads` and `optim_grads_params` for the sharding strategy respectively (however $\texttt{fsdp} = 1$ is omitted). Some of the hyperparameters (e.g., `lacc`, `lbsz`) may have larger search ranges, however is restricted in our setting to values that likely give rise to an optimal PC.

Parameter	**Description**	**Range**
`ldp`	Log of data parallelism (DP) size	$[0, \lg \texttt{ngpus}]$
`ltp`	Log of tensor parallelism (TP) size	$[0, \lg \texttt{ngpus}]$
`lpp`	Log of pipeline parallelism (PP) size	$[0, \lg \texttt{ngpus}]$
`lcp`	Log of context parallelism (CP) size	$[0, \lg \texttt{ngpus}]$
`lsp`	Log of sequence parallelism (SP) size	$[0, \texttt{ltp}]$
`lbsz`	Log of # DP bucket size (in MB)	$\{6, 7, 8, 9\}$
`fsdp`	FSDP stage	$\{0, 2, 3\}$
`lvp`	Log number of virtual pipeline chunks for PP	$\{0, 1, 2\}$
`lacc`	Log of # of gradient accumulation steps	$\{0, 1, 2\}$

$\texttt{lcp} = 1$ for PCO-16 due to shorter sequence length; and (c) setting $\texttt{lacc} = 1$ for PCO-32 and PCO-64. Since the candidate set is pre-generated, these constraints are automatically incorporated into the optimization problem and do not need to explicitly considered when using the benchmark.

OOM configurations. Configurations that run out of memory have no measured throughput. We set $c_{\text{PCO}}(x) = m_{\text{OOM}} = 0.2$, and $f_{\text{PCO}}(x)$ to the throughput of their nearest feasible configuration, or the mean throughput if there are ties.

D More experiment details

Optimal values $f*$ and HV^* are found via exhaustive evaluation for tabular data, or gradient ascent of 200 best points found via grid evaluation for MLP emulators. 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. Initial observations are not included in the regret plots.

D.1 Empiricial noise levels

Noise levels σ_N for the HPO and DMO experiments were empirically found, via 5 repeated evaluations at matched configurations. Details in Table 14. We use empirically measured noise levels to better emulate real LLM problems.

Table 14: Empirically measured noise standard deviations σ_N for BOLT problems, estimated from 5 repeated evaluations at matched configurations.

Family	Problem	Measured σ_N	# Configs ($\times 5$ seeds)
Hyperparameters	HPO	0.0290	30
	HPO-MF-Cont	0.0220	90
	HPO-MF-Disc	0.0290	30
Data mixtures	DMO	0.0114	100
	DMO-MO	(0.0131, 0.0274, 0.0101)	100
	DMO-Het	$\sqrt{\sigma_{\text{if}}^2 + \sigma_{\text{math}}(x)^2 + \sigma_{\text{code}}^2}/3$ $\sigma_{\text{if}} = 0.0131$, $\sigma_{\text{code}} = 0.0101$ (measured) $\sigma_{\text{math}}(x)$ from fitted noise model	100

D.2 HPO EXPERIMENTS

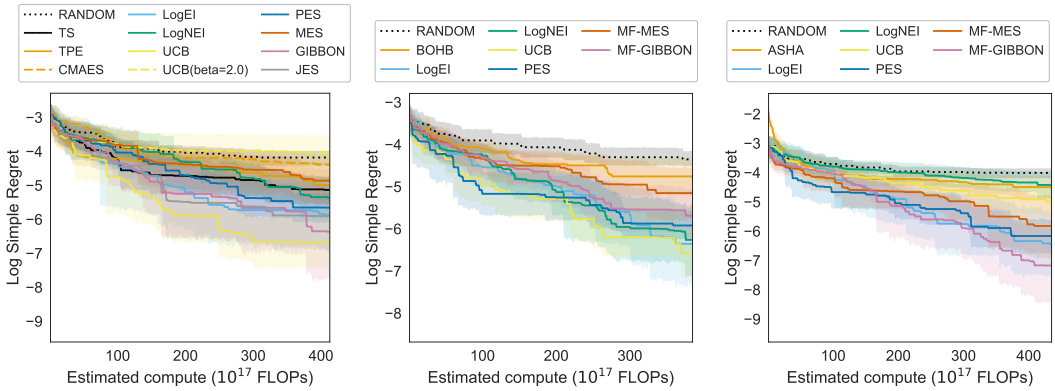
For the single-fidelity HPO problem, we compare random sampling, Thompson sampling (TS) (Chapelle & Li, 2011), log expected improvement (LogEI) (Jones et al., 1998; Ament et al., 2023), log noisy expected improvement (LogNEI) (Ament et al., 2023), upper confidence bounds (UCB) (Srinivas et al., 2010), predictive entropy search (PES) (Hernández-Lobato et al., 2014), max-value entropy search (MES) (Wang & Jegelka, 2017), GIBBON (Moss et al., 2021), and joint entropy search (JES) (Hvarfner et al., 2022). Knowledge gradient (KG) (Frazier et al., 2009) was excluded as it was prohibitively slow for mixed-variable search spaces. These methods were implemented using the BoTorch library (Balandat et al., 2020).

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 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) (Balandat et al., 2020).

We also include dedicated HPO baselines from popular HPO libraries: tree-structured Parzen estimator (TPE) (Bergstra et al., 2011) and covariance matrix adaptation evolution strategy (CMA-ES) (Nomura et al., 2021; Hansen, 2006) from Optuna (Akiba et al., 2019) for the single-fidelity setting problem, BOHB (Falkner et al., 2018) from SMAC3 (Lindauer et al., 2022) for the continuous-fidelity setting, and asynchronous successive halving algorithm (ASHA) (Li et al., 2020) 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.

For multi-fidelity problems, each evaluation consumes budget $\lambda(x) = (1-\gamma)\cdot\text{fid}(x)+\gamma$, normalized so that the highest fidelity costs 1. We set γ from estimated training compute. For HPO-MF-Cont, all fidelities fine-tune the same model (Qwen3-8B), so training FLOPs grow linearly with the number of training tokens, at roughly 30 GFLOP/token for LoRA fine-tuning (about twice the forward-pass cost). The intercept $\gamma = 0.05$ accounts for per-run fixed overhead (model loading and the MATH-500 evaluation pass), which we estimate at about 5% of a full-fidelity run. For HPO-MF-Disc, both fidelities train on the same token budget and differ only in model size, thus the cost ratio matches the ratio of per-token training FLOPs. Counting non-embedding parameters and the LM head, Qwen3-4B and Qwen3-8B have 4.02B and 7.57B parameters, which leads to about 16.1 and 30.3 GFLOP/token respectively (4 FLOPs per parameter per token), and $\gamma = 16.1/30.3 \approx 0.53$.

We compare MF-MES (Takeno et al., 2020), 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 $c(x) = \alpha \cdot \text{fid}(x) + 1$, where $\text{fid}(x)$ 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. For HPO-MF-Cont, we set $\alpha = 0.005$ for NEI, PES, and MF-GIBBON, $\alpha = 0.01$ for EI, and $\alpha = 0.05$ for UCB and MF-MES. For HPO-MF-Disc, we set $\alpha = 0.005$ for PES and MF-GIBBON, $\alpha = 0.01$ for NEI, UCB, and MF-MES, and $\alpha = 0.05$ for EI. A sensitivity analysis over α is in Sec. D.4.



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

Figure 16: Results on BOLT's HPO problems, against FLOPs

FLOPs-based results are additionally shown in Fig. 16, where the x-axis is measured against the estimated compute budget rather than BO iterations or MF budget. FLOPs as the budget are common for HPO practitioners. For the FLOPs plots, each query's compute cost is estimated by an analytical estimate of its LoRA fine-tuning FLOPs, which accounts for model size, training tokens and LoRA depth. Unlike the budget $\lambda(x)$, it counts training compute only and excludes fixed overhead such as model loading and evaluation. The differences are minor and the analysis on the simple regret plots measured against BO iterations hold.

D.3 Discussion on UCB's exploration parameter

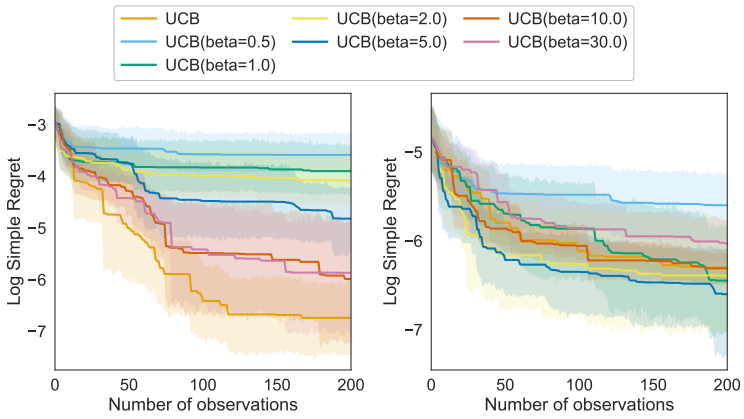
Fig. 17 compares UCB with a fixed β against the scheduled β_t (Srinivas et al., 2010). Our implementation differs slightly, where we use the input dimension d in place of the number of candidate points for β_t, which makes it slightly smaller.

A fixed β allows direct control over the exploration-exploitation tradeoff and thus the convergence behavior. Smaller β reduces exploration, which can speed up early progress but tends to stall at worse solutions in both problems, 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 if a fixed β is preferred, β should be chosen with the compute budget in mind, to balance exploration and exploitation effectively.

On HPO, the schedule outperforms every fixed β, while on DMO, differences are small and a moderate fixed β (2–5) is competitive. We include UCB ($\beta = 2$) in the main regret plots for reference. β of 2 is smaller than scheduled β_t so it tends to be better in early iterations.

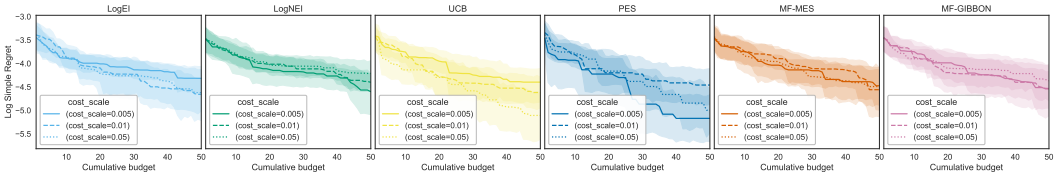
D.4 Cost scale sensitivity

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. 18. The degree of sensitivity varies across acquisition functions and problems. The effect of cost scale α is method-dependent, and no single value is best across methods. On the continuous-fidelity problem (HPO-MF-Cont), PES and UCB are the most sensitive, and at their best α they reach the lowest final regret of all methods. Their preferred values differ, however, with $\alpha = 0.005$ for PES and $\alpha = 0.05$ for UCB. On the discrete-fidelity problem (HPO-MF-Disc), all methods show moderate sensitivity, with differences across α largely within the confidence intervals. Since the best cost scale varies by method and problem, α should be treated as a hyperparameter to tune rather than fixed a priori.

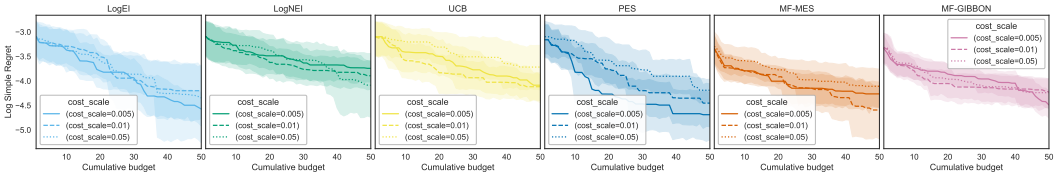


(a) HPO (b) DMO

Figure 17: Comparison of β for UCB on HPO and DMO base problems. "UCB" refers to using scheduled β_t from Srinivas et al. (2010)

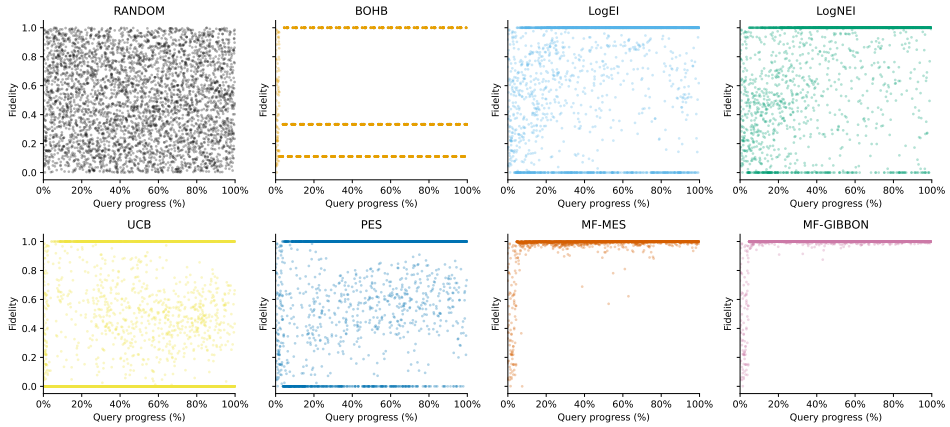


(a) HPO-MF-Cont

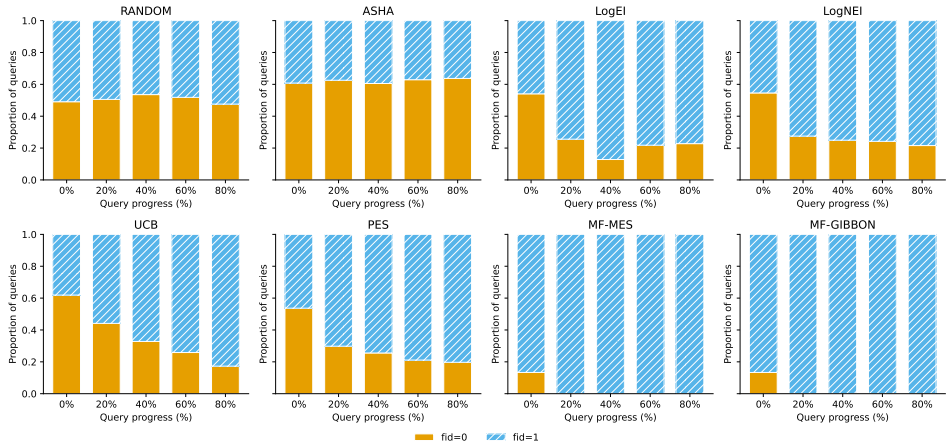


(b) HPO-MF-Disc

Figure 18: Performance on MF HPO problems on different cost scale.



(a) HPO-MF-Cont



(b) HPO-MF-Disc

Figure 19: Fidelity proportions of observations on MF HPO problems.

D.5 Fidelity proportions over iterations

Figure 19 shows how each method allocates queries across fidelity levels over the course of optimization. Recall the acquisition functions are all cost scaled. For HPO-MF-Cont, $\alpha = 0.005$ for NEI, PES, and MF-GIBBON, $\alpha = 0.01$ for EI, and $\alpha = 0.05$ for UCB and MF-MES. For HPO-MF-Disc, $\alpha = 0.005$ for PES and MF-GIBBON, $\alpha = 0.01$ for NEI, UCB, and MF-MES, and $\alpha = 0.05$ for EI.

For the continuous-fidelity setting (HPO-MF-Cont), BOHB concentrates queries at a few discrete fidelity levels set by its bracket schedule. MF-MES and MF-GIBBON query almost exclusively at fidelity 1 after initialization. In contrast, cost-scaled LogEI, LogNEI, UCB, and PES maintain a persistent cluster at the lowest fidelity alongside scattered intermediate-fidelity queries, most pronounced for UCB. These mixed-fidelity methods achieve the lowest regret on this task (Fig. 3b).

For the discrete-fidelity setting (HPO-MF-Disc), cost-scaled acquisition functions shift toward high-fidelity queries as optimization progresses, while MF-MES and MF-GIBBON query only the high fidelity after initialization. PES achieves the lowest regret, followed by MF-GIBBON (Fig. 3c).

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.

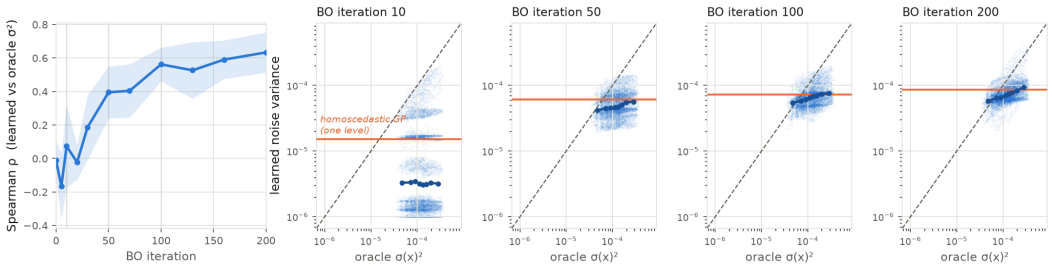


Figure 20: MLHGP noise model over BO iterations

D.6 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 (Bradford et al., 2018), NSGA2 (Deb et al., 2002), ParEGO (Knowles, 2006), NEHVI (Daulton et al., 2021), PESMO (Hernández-Lobato et al., 2016) MESMO (Belakaria et al., 2019), and JESMO (Tu et al., 2022).

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. 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.7 MLHGP NOISE ANALYSIS

We evaluate the fit of MLHGP's learned noise model over BO iterations in Fig. 20, using a held-out set of 1000 sampled test points. The noise model's Spearman ρ against the oracle noise variance improves steadily and becomes reasonably accurate by around iteration 100. This is consistent with LogNEI+MLHGP overtaking LogNEI in regret at roughly the same point. Unlike a homoscedastic noise estimate, which fits a single noise level everywhere, MLHGP correctly captures the varying noise levels across the input space. It does, however, tend to underestimate the noise magnitude, which likely pushes LogNEI toward more exploitative behavior.

D.8 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. We omit SAASBO (Eriksson & Jankowiak, 2021) due to its prohibitive cost, and MSR (Papenmeier et al., 2025) as it does not apply to discretized spaces.

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. Sobol candidate generation and sparse perturbation are omitted in the discretized adaptions.

We show how selected enhancements (batch size, dTuRBO, dBAxUS) affect the simple regret at iteration 200 in Table 15.

Table 15: Delta in log simple regret at iteration 200 vs LogNEI baseline (mean ± 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
`TuRBO`	-5.87 ± 7.55	-10.61 ± 7.10	-5.12 ± 7.78	-5.67 ± 7.67
`TuRBO+q=5`	$\mathbf{-14.05 \pm 0.63}^{\dagger}$	$\mathbf{-13.61 \pm 0.46}^{\dagger}$	$\mathbf{-13.78 \pm 0.41}^{\dagger}$	$\mathbf{-14.09 \pm 0.87}^{\dagger}$
`BAxUS`	-0.43 ± 1.09	-2.27 ± 6.57	-10.82 ± 6.72	-5.99 ± 7.38
`BAxUS+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.9 PCO EXPERIMENTS

We evaluated constrained BO methods: (Log)EIC (Gardner et al., 2014), (Log)NEIC (Letham et al., 2019), UCB-C (Nguyen et al., 2024), dSCBO (Eriksson & Poloczek, 2021), cTS, CMES-IBO (Takeno et al., 2022), cKG (Ungredda & Branke, 2024), ADMMBO (Ariafar et al., 2019), and BBO baselines (TPE, CMA-ES). Log variants for EIC methods are used for numerical stability (Ament et al., 2023). For LogNEIC, UCB-C, cKG, and ADMMBO, we include a variant with GP prior means formulated from parallelism configuration knowledge (Hemachandra et al., 2026). UCB-C, CMES-IBO, cKG, and ADMMBO were implemented on BoTorch's acquisition interface (Balandat et al., 2020), and cTS via BoTorch's `ConstrainedMaxPosteriorSampling`.

All acquisition functions are maximized over the discretized candidate set, and previously evaluated configurations remain in the pool. For CMES-IBO, constrained maxima are sampled directly from joint posterior samples over the candidate set rather than via random Fourier features. For cKG, inner maximizations are computed exactly over the candidate set. For TPE and CMA-ES, which search the integer bounding box, each proposal is mapped to the nearest valid configuration.

dSCBO is our discretized adaptation of SCBO, which extends cTS with a trust region. As with dTuRBO in PO experiments (Sec. D.8), the trust region is represented by the k nearest neighbors of the incumbent, with $k_{\text{init}} = \lfloor N/2 \rfloor$, $\tau_{\text{succ}} = 3$, and $\tau_{\text{fail}} = 10$. Distances are weighted by the objective GP's ARD lengthscales, following SCBO's trust-region shaping.

For LogNEIC, UCB-C, cKG, and ADMMBO, we include a variant that replaces the GPs' constant prior means with analytic cost models of parallel training (Hemachandra et al., 2026). The throughput mean models communication time for DP, TP, and PP (with separate intra- and inter-host latency and bandwidth) plus computation time, and the memory mean models parameter, activation, and overhead memory. Their parameters are learned jointly with the kernel hyperparameters. Following Hemachandra et al. (2026), the throughput mean is blended with a constant whose weight increases to one by iteration 20, so the prior guides early fits and gives way as data accumulates.

Each method gets 10 random initial points followed by 50 (PCO-32) or 100 (PCO-16, PCO-64) iterations, over 10 seeds. The different number of iterations are due to the different number of data points for each problem. PCO-32 has fewer points (379) than PCO-16 (919) and PCO-64 (780).

D.10 PCO LANDSCAPES

The problem landscapes are shown in Fig. 21. PCO-32 has a single sharp optimum, and its second-best configuration is about 4.6% below the optimum. PCO-16 and PCO-64 instead have flat near-optimal regions, with 16 of 632 and 6 of 560 feasible configurations within 2.5% of the optimum, respectively.

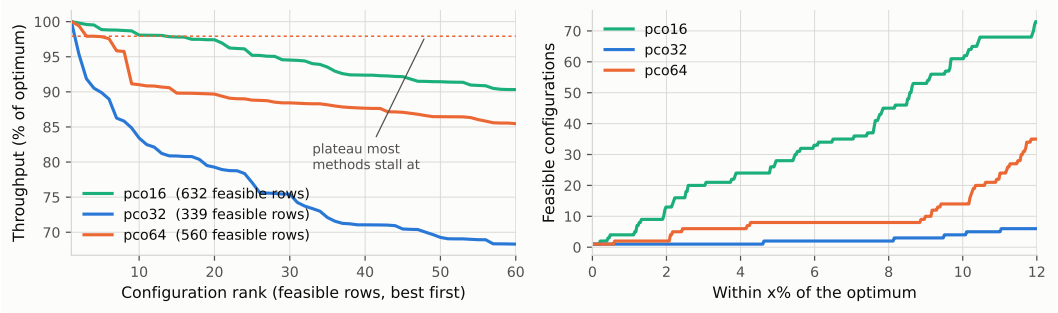
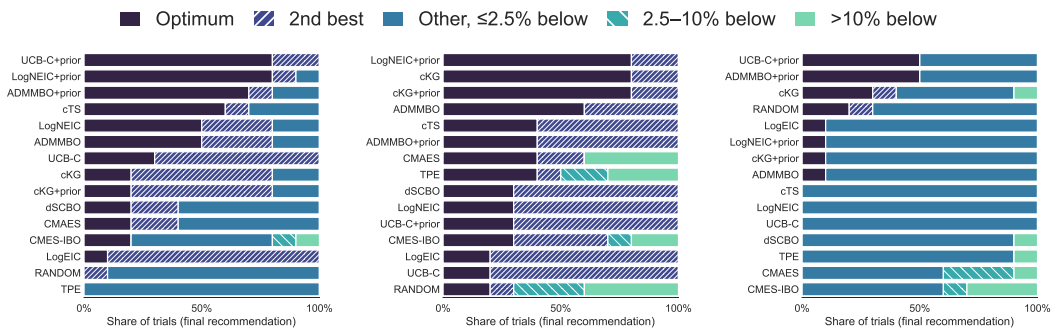


Figure 21: PCO's landscape which shows a plateau near the optimum for PCO-16 and PCO-64.



(a) PCO-16 at iteration 100 (b) PCO-32 at iteration 50 (c) PCO-64 at iteration 100

Figure 22: Quality of final recommendations across 10 seeds, grouped by their gap to the optimum.

On PCO-64, the second-best configuration is only about 0.6% below the optimum. Once a BO method reaches such a plateau, the small throughput differences within it can lead to over-exploitation.

Figure 22 supports this. On PCO-32, all BO methods except CMES-IBO recommend the optimum or second-best configuration in every seed, with LogNEIC+prior, cKG, and cKG+prior each finding the optimum in 8 of 10 seeds. In contrast, random search, TPE, CMA-ES, and CMES-IBO end more than 10% below the optimum in 2–4 of 10 seeds. On PCO-16 and PCO-64, nearly all final recommendations lie within 2.5% of the optimum, and many methods reach the plateau but not the peak. On PCO-64, UCB-C+prior and ADMMBO+prior find the optimum in 5 of 10 seeds, cKG in 3, and random search in 2, while cTS, LogNEIC, UCB-C, dSCBO, TPE, CMA-ES, and CMES-IBO never do. On PCO-16, UCB-C+prior and LogNEIC+prior find it in 8 of 10 seeds, while random search and TPE never do.

The prior helps most on the plateau problems, where it improves several methods. We hypothesize that its analytic throughput mean (Sec. D.9) separates near-optimal configurations whose throughputs differ too little for a constant-mean GP to distinguish from few observations. In contrast, cKG, which values queries by their effect on the final recommendation, performs best on the sharp-optimum PCO-32 but gains little from the prior on the plateaus. Overall, no method is robust across all three landscapes, and the plateaus in PCO-16 and PCO-64 expose a failure mode of current methods: they reach near-optimal regions but under-explore within them.

CMES-IBO and dSCBO are less suited to PCO's discretized candidate set and integer search space as they were designed for the continuous settings. CMES-IBO's acquisition function, a lower bound on the information gain, remains positive at observed points. This is harmless in continuous spaces, where exact repeats are unlikely, but over a discretized candidate set it leads to frequent re-querying (86%, 81%, and 87% of the budget on PCO-16, PCO-32, and PCO-64). dSCBO performs worse than cTS, which it extends with a trust region, most clearly on PCO-16. With our k-NN adaptation (Sec. D.9), the region is always non-empty, but two properties of PCO limit its benefit. First, the candidate set is small enough for cTS to search globally, so localizing search adds little

	split-half rank stability (Spearman ρ)		
Criterion	mean	median	frac. < 0
normalized regret	0.505	0.518	0.01
simple regret	**0.709**	0.718	0.00

Table 16: Ranking quality of simple vs. normalized regret on the HPO emulator.

benefit. Second, on a coarse integer lattice, the nearest configurations can still differ substantially in throughput, weakening the locality that trust regions rely on.

D.11 Simple regret vs normalized regret

We compare the stability of simple regret and normalized regret across seeds on the HPO emulator. Normalized regret was used to compare real and emulated BBO results in Figs. 10 and 11 as it is based only on observed values. Formally, normalized regret is defined as $(y_{\text{best}} - y_{\text{min}})/(y_{\text{max}} - y_{\text{min}})$, where y_{min} and y_{max} are the min and max of the best observed values across all iterations and methods. Methods are ranked by $\max_t y_t$, the best *observed* value, where $y_t = f(x_t) + \varepsilon_t$. A maximum over noisy draws selects for large positive ε, so the score carries an optimism bias even after averaging over BO seeds. On our HPO emulator, this overshoot can exceed the true performance gap between methods, and cause noise to influence the ranking.

Simple regret is the more stable ranking. Splitting the seeds into disjoint halves (5/5 seeds) and ranking the methods within each half, simple regret agrees across halves far more often than normalized regret does, as seen in Table 16. We therefore report simple regret wherever the noiseless objective is available, and normalized regret only on the real runs.

D.12 More evaluation metrics

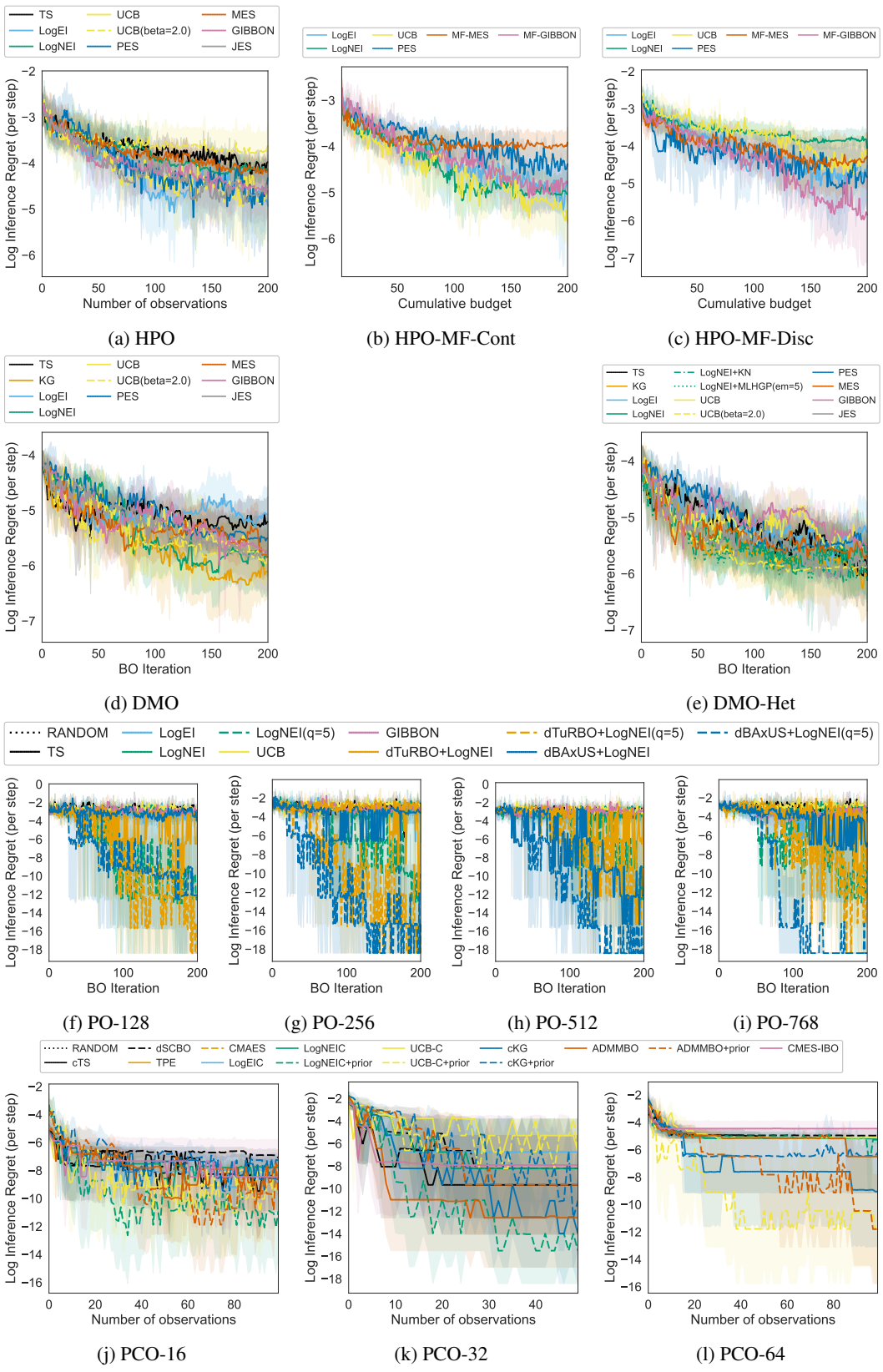
In Sec. 4, for single-objective problems, we evaluate performance using simple regret, which is defined as $f^* - \max_{t' \le t} f(x_{t'})$. While simple regret provides a clean picture of optimization progress over the budget t, it requires access to the noiseless f in their $\max$ term, which is computable 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 (Wang & Jegelka, 2017), and it utilizes the surrogate model in estimating the optimal point. We also report the *per-step* variant of simple regret $f^* - f(\tilde{x}_t)$, where $\tilde{x}_t \triangleq \arg\max_{t' \le t} y_{t'}$ is the best (noisy) observed point. Both metrics are not always monotonically decreasing, but matches practical evaluation conditions more closely.

Inference regret plots are Fig. 23 and per-step simple regret plots are in Fig. 24. Per-step simple regret closely follows simple regret across all problems, as observation noise is small relative to differences in objective values. Under inference regret, methods are less distinguishable on some problems like HPO, suggesting that their posterior means are similar despite different observations. Nevertheless, the main insights from Sec. 4 largely hold under both metrics: MF-GIBBON leads on HPO-MF-Disc, noise-aware LogNEI variants and UCB($\beta = 2$) are among the strongest on DMO-Het. Batched, dTuRBO, and dBAxUS methods remain essential for high-dimensional PO, and cKG remains the strongest on PCO-64. One main difference is on PCO-32, where cKG+prior leads under simple regret but not under inference regret, where ADMMBO(+prior) and LogNEIC+prior perform best.

D.13 Wall-clock time for experiments

Wall-clock runtimes are reported in Tables 17 to 19. These runtimes include emulator query time, which is negligible. Each entry is the total run time divided by the number of iterations (50 for `PCO-32`, 100 for `PCO-16`, `PCO-64`, 200 for the rest). Per-step cost grows super-linearly with the number of observations, as the surrogate is refitted on more data, so later steps take longer than this average suggests and earlier steps shorter. PCO experiments are ran on CPUs. Other experiments ran on L40 GPUs, with a small proportion using H100 or H200 GPUs to accommodate the higher VRAM



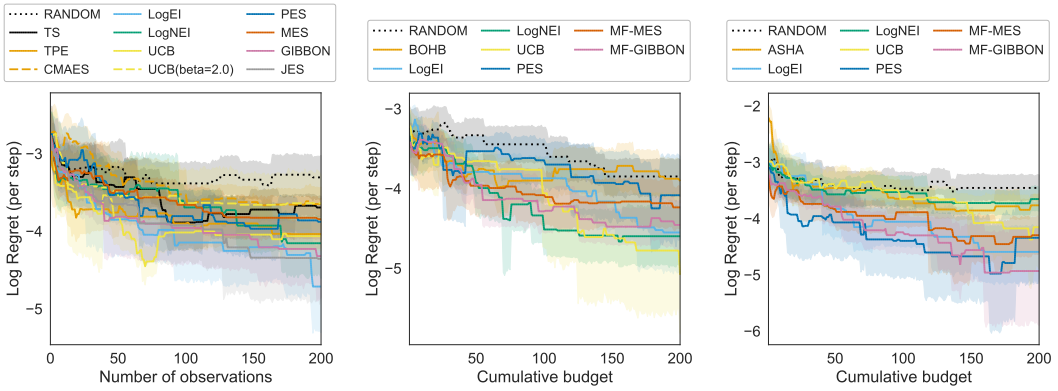
(a) HPO (b) HPO-MF-Cont (c) HPO-MF-Disc

(d) DMO (e) DMO-Het

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

(j) PCO-16 (k) PCO-32 (l) PCO-64

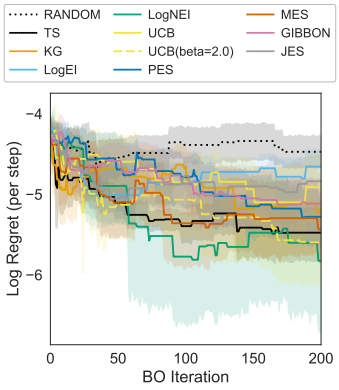
Figure 23: Results on BoLT's problems using inference regret. Random is not included as random sampling does not maintain a posterior mean.



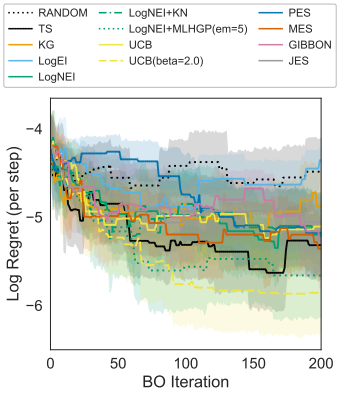
(a) HPO

(b) HPO-MF-Cont

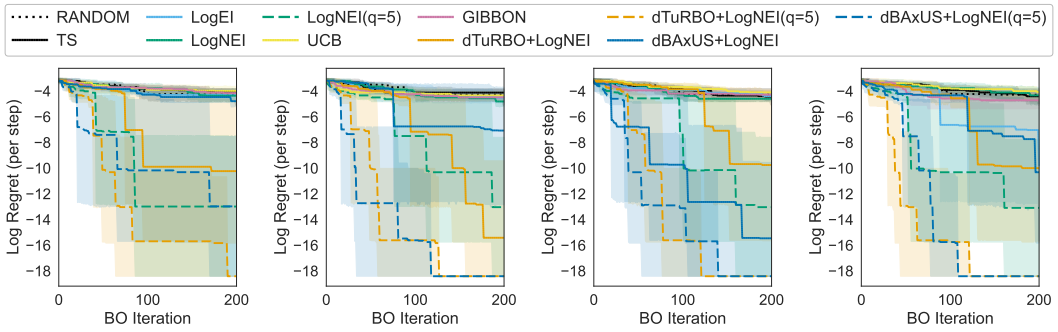
(c) HPO-MF-Disc



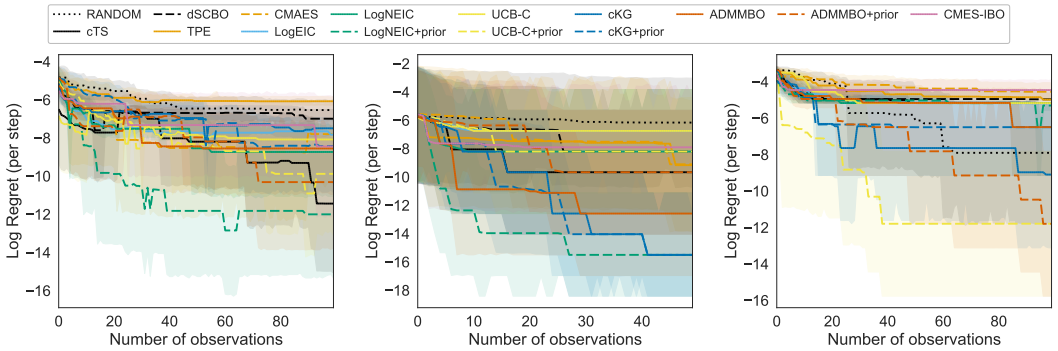
(d) DMO



(e) DMO-Het



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



(j) PCO-16 (k) PCO-32 (l) PCO-64

Figure 24: Results on BOLT's problems using per-step simple regret.

Table 17: Mean wall-clock time per BO step — HPO and DMO. All ran on L40s.

(a) **HPO**

Method	Time / step
HPO	
RANDOM	0.00184 s
TS	7.47 s
TPE	0.00987 s
CMAES	0.00299 s
LogEI	5.73 s
LogNEI	6.07 s
UCB	7.28 s
UCB(beta=2.0)	4.33 s
PES	13 s
MES	7.1 s
GIBBON	7.29 s
JES	8.48 s
HPO-MF-Cont	
RANDOM	0.00608 s
BOHB	0.151 s
LogEI	9.94 s
LogNEI	11.7 s
UCB	27.6 s
PES	18.9 s
MF-MES	9.72 s
MF-GIBBON	8.99 s
HPO-MF-Disc	
RANDOM	0.0039 s
ASHA	0.0111 s
LogEI	6.36 s
LogNEI	6.48 s
UCB	9.18 s
PES	16 s
MF-MES	7.69 s
MF-GIBBON	7.27 s

(b) **DMO**

Method	Time / step
DMO	
RANDOM	0.00561 s
RANDOM(q=5)	0.00596 s
RANDOM(q=10)	0.00542 s
TS	1.9 s
KG	21.6 s
LogEI	3.17 s
LogNEI	4.56 s
LogNEI(q=5)	19.2 s
LogNEI(q=10)	1.79 min
UCB	3.02 s
UCB(beta=2.0)	3.01 s
PES	4.66 s
MES	4.11 s
MES(q=5)	9.44 s
MES(q=10)	16.1 s
GIBBON	3.49 s
GIBBON(q=5)	18.2 s
GIBBON(q=10)	49.8 s
JES	3.75 s
DMO-MO	
RANDOM	0.0202 s
RANDOM(q=5)	0.0422 s
TSEMO	2.02 s
NSGA-II(pop_size=50)	1.2 s
ParEGO	3.15 s
ParEGO(q=5)	14.7 s
NEHVI	11.2 s
NEHVI(q=5)	4.82 min
PES_MO	31 s
MES_MO	4.68 s
JES_MO	6.24 s
MES_MO(q=5)	12.8 s
DMO-Het	
RANDOM	0.00596 s
TS	1.79 s
KG	21.9 s
LogEI	3.26 s
LogNEI	4.25 s
LogNEI+KN	4.28 s
LogNEI+MLHGP(em=5)	5.46 s
UCB	2.96 s
UCB(beta=2.0)	3 s
PES	4.6 s
MES	3.72 s
GIBBON	3.52 s
JES	3.73 s

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).

Table 18: 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†
dTuRBO+LogNEI	9.8 s	10.3 s	8.85 s*	3.84 s †
dTuRBO+LogNEI(q=5)	14.9 s	16.1 s	13.5 s*	7.4 s†
dBAxUS+LogNEI	7.77 s	7.04 s*	7.13 s*	4.55 s†
dBAxUS+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 19: Mean wall-clock time per BO step — PCO. All ran on CPU.

Method	***PCO-16***	***PCO-32***	***PCO-64***
RANDOM	0.00135 s	0.00117 s	0.00119 s
cTS	3.17 s	1.49 s	3.23 s
dSCBO	1.6 s	0.801 s	0.88 s
TPE	0.016 s	0.0108 s	0.0112 s
CMAES	0.00296 s	0.00252 s	0.00256 s
LogEIC	2.7 s	1.77 s	3.07 s
LogNEIC	2.73 s	2.14 s	3.25 s
LogNEIC+prior	7.79 s	7.71 s	7.66 s
UCB-C	3.6 s	2.57 s	2.68 s
UCB-C+prior	7.68 s	6.84 s	5.73 s
cKG	4.28 s	2.52 s	3.42 s
cKG+prior	8.19 s	6.92 s	8.46 s
ADMMBO	3.13 s	1.95 s	2.81 s
ADMMBO+prior	7.94 s	7.09 s	7.55 s
CMES-IBO	2.94 s	1.19 s	1.89 s

E COMPUTE USAGE

We detail the compute used for generation of BOLT's data, and BO experiments in Table 20. Additionally, in Table 21, we report the compute cost for the BBO evaluations with real LLMs, used in Figs. 10 and 11, to highlight how expensive evaluations are without our emulators and datasets.

Table 20: Compute Usage Summary

Stage	Model	Training	Eval
HPO	8B	31 H200 GPU-days	17 L40 GPU-days
	4B	62 H200 GPU-days	7 L40 GPU-days
DMO	4B	123 H200 GPU-days	107 L40 GPU-days
PO	14B	–	21 H200 GPU-days
PCO-16		–	57 H100 GPU-days
PCO-32		–	154 GH200 GPU-days
PCO-64		–	277 GH200 GPU-days

Table 21: Compute for real BBO runs in Figs. 10 and 11.

Problem	Training	Eval
HPO	53 H200 GPU-days	15 L40 GPU-hours
DMO	20 H200 GPU-days	32 L40 GPU-days