Rethinking Bayesian Optimization for Co-Optimizing LLM Training Configurations

Zhiliang Chen[1*], **Alfred Wei Lun Leong**[1*], **Shao Yong Ong**[1*], **Apivich Hemachandra**[1], **Gregory Kang Ruey Lau**[1], **Chuan-Sheng Foo**[2], **Zhengyuan Liu**[2], **Nancy F. Chen**[2], **Bryan Kian Hsiang Low**[1]
[1]Department of Computer Science, National University of Singapore, Singapore
[2]Institute for Infocomm Research, A*STAR, Singapore
{chenzhiliang,alfred_leong,ongshaoyong,apivich,gregorylau}@u.nus.edu
lowkh@comp.nus.edu.sg

Abstract

Fine-tuning an LLM to maximize performance on a downstream task requires finding the optimal data and model configuration. This is a black-box optimization problem that Bayesian optimization (BO) addresses by sequentially evaluating training configurations and using observed performance feedback to adaptively guide the search towards better configurations. However, directly applying BO to the joint data-model space suffers from two fundamental challenges: the prohibitive cost of full LLM training at each iteration, and the large number of BO iterations required for effective exploration due to the high problem dimensionality. This paper introduces JoBS, a BO-based approach that co-optimizes data and model configurations without requiring a full training run at every iteration. JoBS allocates an initial portion of the optimization budget to learn a scaling-law-inspired performance predictor that estimates fully trained LLM performance from only a small number of training steps. The remaining budget is then used to run BO with this predictor, eliminating the need for full training runs and enabling JoBS to explore significantly more data and model configurations within the same budget. We analyze JoBS' average regret and derive the optimal budget allocation between predictor learning and BO iterations. Empirically, JoBS outperforms independent data and model optimization methods and existing multi-fidelity BO baselines across a diverse set of downstream tasks. Our code is available at https://github.com/a35453779/JoBS.

1 Introduction

Fine-tuning an LLM for a downstream task hinges on two intertwined decisions: which *training data* to use, and how to configure the *model architecture*. From the *data* perspective, performance can be improved by selecting more relevant or higher-quality training data [21, 44, 46] or by optimizing the mixture of data domains [2, 5, 26, 45, 47]. From the *model* perspective, a range of methods have been proposed to identify the most suitable architecture or fine-tuning configuration [14, 29, 40, 52].

Despite their individual successes, existing data and model optimization methods are developed in isolation, ignoring their *interdependence* that critically affects performance. The optimal data configuration (e.g., training data mixture) depends on the chosen model configuration (e.g., fine-tuning configuration), but determining the optimal model configuration also depends on the chosen

*Equal contribution

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

data configuration. This leads to a *chicken-and-egg dilemma*, as optimizing either component independently leads to sub-optimal LLM performance [4], as demonstrated empirically in Sec. 6.

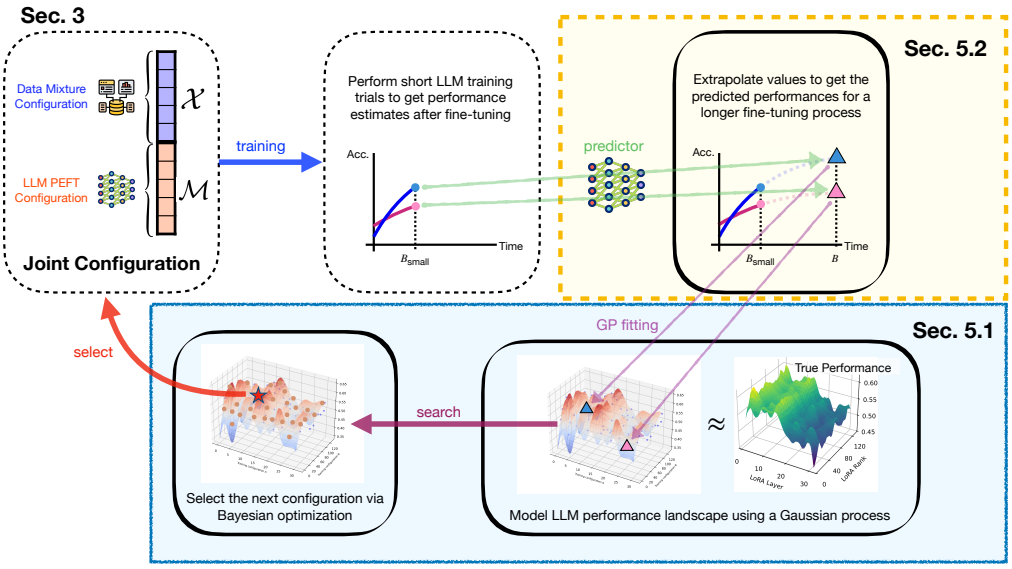


Figure 1: JoBS optimally balances budget between learning a performance predictor and running Bayesian optimization iterations.

Our work tackles this by *co-optimizing* the data and model configurations of LLMs to maximize a downstream task metric. In practice, these metrics, such as user ratings, engagement, or benchmark accuracy, are observed as feedback from the deployed LLM across successive training iterations. This formulation yields a classic *black-box optimization problem* as the relationship between the training configuration and metric feedback has no closed-form expression. Bayesian optimization (BO) is a principled approach for such problems, using observed performance feedback to adaptively guide the search toward better configurations. Crucially, this iterative structure allows BO to directly exploit downstream task feedback — an advantage that non-adaptive methods lack by design. For instance, gradient-based data selection methods [44] cannot utilize black-box feedback, limiting their effectiveness in this setting [5, 11, 31]. A naive approach would be to directly apply BO over the joint data-model space. However, this suffers from two fundamental challenges: the prohibitive cost of full LLM training at every iteration, and the high dimensionality of the joint search space, which demands many iterations to explore effectively. Together, these render naive BO computationally infeasible under a finite *optimization budget*.

Our method, **Joint Bayesian Optimization with a Scaling-law-inspired predictor** (JoBS), addresses both challenges by allocating an initial portion of the optimization budget to learn a scaling-law-inspired performance predictor that estimates the performance of a fully trained LLM for any training configuration from only a small number of training steps. The remaining budget is then used to run BO with this predictor, eliminating the need for full LLM training at each iteration and enabling significantly more BO iterations within the same optimization budget. Importantly, even if the predictor's estimates introduce errors, BO naturally absorbs this as observation noise, ensuring theoretical convergence guarantees (Sec. 5.4).

The main contributions of this paper are:

- We propose JoBS, an algorithm that co-optimizes LLM data and model configurations by running BO (Sec. 5.1) with a scaling-law-inspired performance predictor (Sec. 5.2) that predicts the performance of a fully trained LLM for any training configuration from only a small number of training steps. This reduces the cost of each BO iteration, enabling significantly more iterations within the same optimization budget.
- We empirically analyze the prediction error and identify an inherent tradeoff: more initial full-training runs improve predictor accuracy, but reduce the number of subsequent BO iterations available under a fixed optimization budget (Sec. 5.3).

- We theoretically analyze how predictor error propagates into JoBS' average regret, and derive the optimal budget allocation between predictor learning and BO iterations to minimize regret (Sec. 5.4).
- We empirically demonstrate that JoBS consistently outperforms independent data and model optimization (Sec. 6.2), vanilla BO, and multi-fidelity BO (Sec. 6.3) baselines across a diverse set of downstream tasks, with ablation studies validating our theoretical finding that an optimal budget allocation exists (Sec. 6.4).

2 Related work

Data and model optimization. Data optimization methods select or reweight training data to improve downstream performance: gradient-based selection (**LESS**) [44] identifies influential data points by aligning training gradients with task gradients; influence functions (**IF**) [21] rank data points by their estimated effect on model outputs; distributionally robust mixing (**DoReMi**) [45] optimizes domain weights to perform well across task distributions; and diversity-based selection (**Diversity**) [38] maximizes the log-determinant score of the selected subset. Model optimization methods identify suitable architectures or fine-tuning configurations: differentiable architecture search (**DARTS**) [25] relaxes discrete architecture choices into a continuous optimization; **AutoLoRA** [52] automatically optimizes LoRA rank; and **RoBoT** [14] selects configurations using training-free metrics. Crucially, these methods optimize data and model configurations in isolation, ignoring their critical interdependence. Furthermore, they cannot exploit iterative black-box metric feedback (e.g., user ratings or benchmark scores) to guide optimization, and may require access to task gradients, which are unavailable in black-box settings. These methods will serve as baselines in our experiments (Sec. 6.1).

Multi-fidelity BO. BO has been widely adopted for optimizing black-box functions where evaluations are costly [31]. Multi-fidelity BO methods [22, 32, 43, 49] incorporate fidelity as an additional input dimension and use cost-aware acquisition functions to determine which fidelity to query at each iteration. Examples of such methods include **MF-KG** [43], which uses knowledge gradients, and **MF-UCB** [31], which uses upper confidence bound (UCB). However, these methods suffer from three key limitations. First, they rely on a GP to jointly model low- and high-fidelity observations, which is often insufficiently expressive to capture the complex relationship between fidelities. Second, few works provide theoretical backing on the optimal number of queries per fidelity level. Third, freeze-thaw methods [22, 32] assume monotonically improving performance via an exponential decay kernel, but this assumption breaks down when suboptimal data mixtures cause LLM performance to *decrease* with more training steps (Fig. 3). Early-stopping BO [9] shares the same limitation of relying on a GP across different fidelities. A detailed comparison is provided in App. B.

LLM scaling laws. Existing scaling laws [3, 16, 20, 30, 42, 50] establish symbolic formulas that extrapolate LLM performance from a small number of training steps. However, as shown in Sec. 4.2, these formulas do not transfer to our setting as they are defined w.r.t. a fixed training configuration (e.g., a fixed training data pool), whereas we require performance predictions across a wide range of LLM training configurations. Notably, Yen et al. [49] explore the optimization of data mixtures using neural networks to predict LLM performance, but do not account for the cost of training the predictor within the optimization budget.

3 Problem setup

We consider two types of training components: **data** $\mathcal{X}$ and **model** $\mathcal{M}$. Given these training components, we define a training process P_B that fine-tunes an LLM for B training steps to produce trained weights $\theta_{\mathcal{X},\mathcal{M},B} \triangleq P_B(\mathcal{X}, \mathcal{M})$, which are evaluated under a predefined downstream metric $\mathcal{L}$. Our formulation accommodates a broad class of downstream metrics, including black-box metrics such as benchmark accuracy or evaluation loss. Formally, this yields a black-box optimization problem over training configurations $\mathcal{X}, \mathcal{M}$:

$$\max_{\mathcal{X},\mathcal{M}} \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B}) \,. \tag{1}$$

Data $\mathcal{X}$. We assume access to n training datasets $\mathcal{D} \triangleq D_1 \cup D_2 \cup \cdots \cup D_n$ drawn from n distinct domains (e.g., General Knowledge, Math), and define the data component as a subset $\mathcal{X} \subseteq \mathcal{D}$. An optimal $\mathcal{X}$ selects data samples that are relevant to the target task [5] or of higher quality [37, 44, 51]. In this work, we consider $\mathcal{X}$ as the *training data mixture* [5], represented by a probability simplex ($\mathcal{X} \in \Delta^{n-1} \subset \mathbb{R}^n$). This data mixture represents the proportion of each data domain used to train the LLM.

Model $\mathcal{M}$. Our formulation supports optimization over arbitrary model configurations. In this work, we focus primarily on parameter-efficient fine-tuning (PEFT) using low-rank adaptation (LoRA) [17], but also show that JoBS extends to full-parameter fine-tuning (see App. M.2 for details). Concretely, the model configuration $\mathcal{M} \in \mathbb{R}^m$ is a vector of dimension m comprising: (1) the LLM *layers* to which LoRA is applied, (2) the LLM *modules* to which LoRA is applied (e.g., Q, K-projection [36]), and (3) the *LoRA hyperparameters*, including rank, α, and dropout [17].

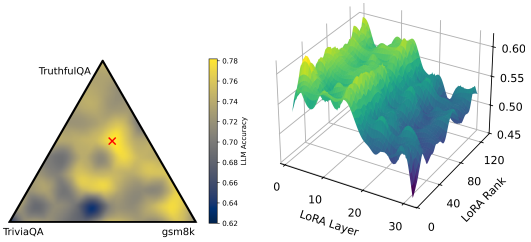
Optimization budget C. A key aspect of our work is the incorporation of an explicit total optimization budget $C \geq B$, which prior work [49] does not consider. Without such a budget constraint, brute-force search could be applied over all possible training configurations [1, 23]. For instance, given a budget of $C = 50000$ steps with each full training run costing $B = 1000$ steps, only $C/B = 50$ training configurations can be evaluated. Our algorithm accommodates other budget metrics as well, such as FLOPs or tokens, as detailed in App. G.

4 Preliminary findings

To motivate the design of JoBS, we first present empirical findings on the structure of the performance landscape and the feasibility of using neural networks as performance predictors.

4.1 Performance landscape

We examine the performance landscape of our optimization problem. In Fig. 2, we fine-tune a `Llama-3-8B-Instruct` [13] model across varying data mixtures and LoRA configurations and use GSM8K [8] accuracy as the downstream metric (averaged over 5 trials). The landscape reveals two key properties. First, jointly optimizing LoRA layer and rank yields over 10% performance improvement over an arbitrarily chosen configuration (Fig. 2b). Second, the landscape varies rather smoothly across the input dimensions (Fig. 2a), suggesting that it can be effectively modeled by a smooth surrogate function [11], making BO well-suited for this problem.



(a) LLM performance w.r.t. data mixtures.

(b) LLM performance w.r.t. LoRA configurations.

Figure 2: LLM performance varies with different configurations.

4.2 Extrapolating LLM performance

Next, we investigate whether LLM performance can be predicted from only a small number of training steps. Fig. 3 shows that performance trajectories vary substantially across different training configurations.

For instance, a good training configuration (green) yields improved performance as training steps increase, whereas a suboptimal training configuration (orange) causes performance to *decrease*. This is unsurprising as poorly chosen data mixtures can degrade LLM performance on downstream tasks. To capture this variability, JoBS uses a flexible neural network (Sec. 5.2) that predicts performance from only a small number of training steps, across diverse training configurations. By contrast, existing scaling law formulas [3, 50] are defined for *fixed* training configurations and cannot be applied here. While our predictor lacks the symbolic representation of classical scaling laws, its primary role is to provide useful signals to guide optimization.

5 Introducing JoBS

JoBS consists of two main steps: (i) **Predictor learning.** We allocate an initial portion of the budget to learn a performance predictor that estimates the fully trained LLM performance from a small number of training steps (Sec. 5.2). (ii) **BO with predictor.** We iteratively evaluate candidate training configurations following the standard BO feedback loop (Sec. 5.1). Unlike vanilla BO, which requires the LLM to be trained to completion at each iteration, JoBS queries the predictor to estimate the fully trained LLM performance after only a small number of training steps.

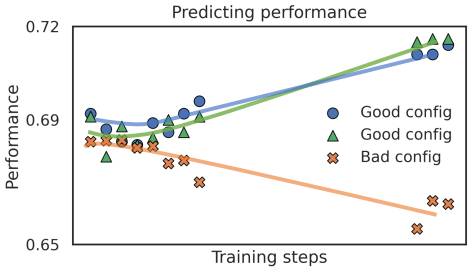


Figure 3: Training a neural network to predict fully trained LLM performance from a small number of training steps.

A key question is *how much of the optimization budget to allocate to learning the predictor*. This involves a clear tradeoff: allocating more budget to learn the predictor leaves fewer BO iterations to effectively search for the optimal training configuration, while reducing the learning budget results in a less accurate predictor that does not faithfully recover the true performance landscape. We analyze the performance tradeoff between the number of initial full training runs and remaining BO iterations both theoretically (Sec. 5.4) and empirically (Sec. 6.4), deriving the optimal budget allocation. Notably, JoBS does not require a perfect performance predictor because the BO framework naturally treats the prediction error as *observation noise*, and we show theoretically that JoBS converges to the optimal configuration despite this noise.

5.1 BO as the backbone of JoBS

We first provide an overview of the standard BO algorithm in a sequential feedback setting. We consider LLM performance as a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ over the space of inputs $x = [\mathcal{X}, \mathcal{M}] \in \mathbb{R}^d$ where $d = n + m$ (see Sec. 3). We treat our objective function in Eq. 1 as a *black-box function* whose maximizer $x^* \triangleq \operatorname{argmax}_x \mathcal{L}(x)$ we want to recover. In line with existing work, we model $\mathcal{L}$ as a surrogate *Gaussian process* (GP) [41]. At each iteration $t = 1, 2, \ldots, T$, we use an LLM training configuration x_t to obtain a *noisy* realization (after training with x_t) of the LLM performance $y_t \triangleq \mathcal{L}(x_t) + \epsilon_t$, which we assume is corrupted with sub-Gaussian noise ϵ_t (e.g., normally-distributed noise) to form the sample (x_t, y_t). Consistent with the work of Chowdhury and Gopalan [6], our GP is specified by its *prior* mean $\mu(x)$ and covariance $\kappa(x, x')$ for all $x, x' \in \mathbb{R}^d$, where κ is a *kernel* function that characterizes the correlation between two inputs x and x'.

Given the noisy observations $\boldsymbol{y}_t \triangleq [y_\tau]^\top_{\tau=1,\ldots,t}$ at inputs $x_1, \ldots, x_t$, the posterior belief of $\mathcal{L}$ at any new input x' is a Gaussian distribution with the *posterior* mean and variance given by

$$\begin{aligned} \mu_t(x') &\triangleq \kappa_t^\top(x')(K_t + \zeta I)^{-1}\boldsymbol{y}_t \\ \sigma_t^2(x') &\triangleq \kappa(x', x') - \kappa_t^\top(x')(K_t + \zeta I)^{-1}\kappa_t(x') \end{aligned} \tag{2}$$

where $\kappa_t(x') \triangleq [\kappa(x', x_\tau)]^\top_{\tau=1,\ldots,t}$ is a column vector, $K_t \triangleq [\kappa(x_\tau, x_{\tau'})]_{\tau,\tau'\in 1,\ldots,t}$ is a $t \times t$ covariance matrix, and $\zeta > 0$ is a free hyperparameter [6]. By learning the correlation between inputs and observations, modeling $\mathcal{L}$ with a GP allows us to capture the relationship between training configurations and LLM performance.

Using BO for data and model co-optimization. At each iteration, BO proposes a candidate data and model configuration by maximizing an acquisition function (e.g., *upper confidence bound* (UCB) [31]); the LLM is then trained under that configuration, and its downstream task performance is observed. We measure convergence via average regret after T iterations, given by $\mathcal{R}_T \triangleq T^{-1}\sum_{t=1}^{T}(\mathcal{L}(x^*) - \mathcal{L}(x_t))$, where $\mathcal{L}(x^*)$ is the optimum [35]. More details on BO are provided in App. D.

5.2 Amortization with performance predictor

Directly applying BO requires training an LLM to completion at every iteration, which is computationally prohibitive under a finite optimization budget. To amortize this cost, we learn a performance predictor that estimates fully trained LLM performance from a small number of training steps, allowing us to run significantly more BO iterations within the same budget. In Sec. 5.4, we further provide

a theoretical analysis of the computational cost tradeoff associated with learning this predictor, which is absent in prior literature [33, 49].

For the predictor to generalize, it must estimate LLM performance for different training configurations, which are not known in advance. To this end, JoBS learns a neural network that takes *any* training configuration $[\mathcal{X}, \mathcal{M}]$ and a sequence of performance observations $\{\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},b})\}_{b \le B_{\text{small}}}$ up to $B_{\text{small}} < B$ training steps, and predicts the fully trained LLM performance $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$. For instance, in our experiments we use multiple sequential observations up to $B_{\text{small}} = 100$ steps to predict final performance at $B = 1000$ steps. Examples of predictions are shown in Fig. 3. As expected, our ablation studies in Sec. 6.4 show that a larger B_{small} yields more accurate predictions at the expense of consuming more optimization budget.

Learning the predictor consists of two stages. (i) **Data collection:** To ensure diversity, we sample a random Sobol sequence [28] of N training configurations over $\mathcal{X}$ and $\mathcal{M}$. For each configuration, we train the LLM to completion, recording performance at several sequential steps up to B_{small} as well as the final performance $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ at B steps; these full training observations also warm-start the GP surrogate (Sec. 5.1). (ii) **Training:** JoBS fits a neural network $\mathcal{F} : (\mathcal{X}, \mathcal{M}, \{\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},b})\}_{b \le B_{\text{small}}}) \mapsto \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ on these observations, mapping performance after a small number of training steps to fully trained LLM performance. Once learned, $\mathcal{F}$ replaces full training runs: at every subsequent BO iteration, the LLM is trained for only B_{small} steps and $\mathcal{F}$ predicts the fully trained LLM performance.

5.3 Prediction error and performance tradeoff

Prediction error. The accuracy of $\mathcal{F}$ directly influences the quality of BO iterations in JoBS, making it important to characterize how prediction error behaves in practice. Fig. 4 shows that as N increases, prediction accuracy improves, reflected in higher R^2 scores and tighter error concentration around zero. Empirically, the prediction error is well-approximated by a normal distribution, whose variance decreases with increasing N. Crucially, JoBS converges both theoretically (Theorem 5.1) and empirically (Fig. 5), despite this prediction noise.

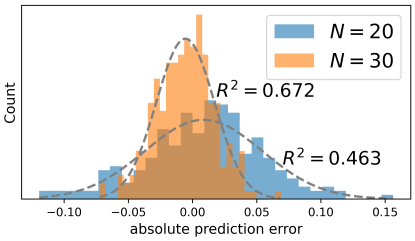


Figure 4: Predictor error w.r.t. N

Tradeoff. However, increasing N comes at a cost. Under a fixed optimization budget C, collecting N full training runs each requiring B steps consumes an initial budget of $N \times B$, leaving only $\lfloor (C - NB)/B_{\text{small}} \rfloor$ BO iterations remaining. Thus, a larger N produces a more accurate predictor but allows fewer BO iterations, and vice versa. In the next section, we formally analyze how the choice of N influences the convergence of JoBS.

5.4 Theoretical analysis of JoBS

We analyze how the regret bound of JoBS varies w.r.t. the prediction error and the choice of N. While the prediction error cannot be precisely quantified analytically, Fig. 4 suggests it is R-sub-Gaussian [6] with increasing variance as N decreases. The following assumption formalizes this characteristic based on the empirical observations in Fig. 4.

Assumption. Let ϵ be the prediction error of the performance predictor learned from N full training runs, whose empirical distribution is shown in Fig. 4. We assume ϵ is R-sub-Gaussian with $R = \frac{k}{\sqrt{N}}$ for some constant k, reflecting that prediction error decreases as N grows.

With this assumption, the following theorem characterizes JoBS' average regret given an optimization budget C and N initial full training runs.

Theorem 5.1. *Let $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ be the performance landscape of the LLM training configuration with bounded RKHS norm: $\|\mathcal{L}\|_\kappa = \sqrt{\langle \mathcal{L}, \mathcal{L} \rangle_\kappa} \le \mathcal{B}$ w.r.t. kernel κ. Assume our performance predictor is learned from N full training runs and makes predictions $\mathcal{F}(\mathcal{X}, \mathcal{M}, \{\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},b})\}_{b \le B_{small}}) = \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B}) + \epsilon$ from $B_{small} < B$ training steps where ϵ satisfies the assumption above. Then, running JoBS over LLM training configurations $\mathcal{X}, \mathcal{M}$ until an optimization budget of C is exhausted yields $T = \left\lfloor \frac{C - NB}{B_{small}} \right\rfloor$ BO iterations, and with the IGP-UCB acquisition function [6], JoBS achieves the*

following average regret with probability at least $1-\delta$*:*

$$\mathcal{R}_T = \mathcal{O}\left(\sqrt{\frac{B_{small}}{C - NB - B_{small}}}\left(\mathcal{B}\sqrt{\gamma_T} + \frac{k}{\sqrt{N}}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right)\right), \tag{3}$$

where γ_T *is the maximum information gain of* $\mathcal{L}$ *after* $T = \left\lfloor \frac{C-NB}{B_{small}} \right\rfloor$ *BO iterations.*

Theorem 5.1 implies JoBS is essentially no-regret, as the average regret reduces to zero with increasing budget C. The proof is provided in App. E. This bound is also tighter than the average regret bound of vanilla BO [6] under some reasonable assumptions (see App. E.1).

Optimal budget allocation. Theorem 5.1 presents a compute-performance tradeoff: a larger N yields more accurate predictions, *reducing* the $k/\sqrt{N}$ term in Eq. 3, but also reduces the number of BO iterations available, *increasing* the left term in Eq. 3. Using the experimental values $B_{\text{small}} = 100$, $C = 50000$, and $B = 1000$, **the optimal** N **that minimizes the average regret** $\mathcal{R}_T$ **lies approximately between** 25 **and** 35 (depending on other constants). We empirically illustrate this tradeoff and provide practical guidelines for choosing N in our ablation studies (Sec. 6.4).

6 Experiments

We empirically validate JoBS across a variety of downstream task metrics and settings. Our experiments are organized into three parts: **(i)** comparison against all combinations of independent data and model optimization methods, **(ii)** comparison against vanilla and multi-fidelity BO baselines, and **(iii)** ablation studies to isolate the contributions of key design choices, and computational cost analysis.

In our main experiments, we fine-tune `Llama-3-8B-Instruct` with LoRA [17]. Extensions to different model families (`Qwen`), larger models (14B, 32B), and full-parameter fine-tuning are provided in App. M.2, where JoBS remains consistently effective. Our objective is to maximize downstream task performance after $B = 1000$ training steps, and within an optimization budget of $C = 50000$ training steps. We evaluate benchmark accuracy using `lm-evaluation-harness` [12] across six language tasks (Fig. 5). We use $N = 30$ initial full training runs to train the performance predictor, which makes predictions after $B_{\text{small}} = 100$ steps — enabling $4\times$ more BO iterations within the same budget. We use the UCB acquisition function [31] with an increasing β_t schedule, and a mixed kernel that combines a Hamming-distance kernel with an SE kernel (App. G). Ablation studies in Sec. 6.4 justify the choices of N and B_{small}, and full experimental details are in App. I.

Data configuration. Data configuration $\mathcal{X}$ is a mixture over 9 training domains: **Wikitext** [27], **GSM8K** [8], **PubmedQA** [18], **SciQ** [39], **TriviaQA** [19], **TruthfulQA** [24], **MMLU** [15], **AI2 ARC** [7], and **CommonsenseQA** [34]. We construct a fine-tuning dataset consisting of 10000 data points by randomly sampling from these training datasets [5, 45, 48], with train and test splits kept separate to prevent data contamination.

Model configuration. Model configuration $\mathcal{M}$ comprises five LoRA hyperparameters: the LLM layers to which LoRA is applied, the LLM modules to which LoRA is applied (e.g., Q, K-projection), rank, α, and dropout. This gives a total of 10 model configuration dimensions. Further experimental details are provided in App. I.

6.1 Baselines

We compare JoBS against three categories of baselines (described in Sec. 2). **Data optimization:** LESS, DoReMi, Influence Function (IF), and Diversity. **Model optimization:** DARTS, AutoLoRA, and RoBoT. **BO methods:** vanilla BO, MF-KG, and MF-UCB, with discrete fidelities of $B_{\text{small}} = 100$ and $B = 1000$. Since task gradients are not available in our black-box setting, gradient-dependent baselines use training gradients as a proxy. Further implementation details are provided in App. J.

Since JoBS is, to our knowledge, the first method to co-optimize data and model configurations with iterative black-box feedback, no existing baseline offers a direct comparison in this setting. Our experiments therefore serve two purposes: **(i)** to assess the value of co-optimization by comparing against all combinations of data and model optimization baselines, and **(ii)** to evaluate the advantage of our predictor-based approach over kernel-based multi-fidelity BO methods.

6.2 Comparison with data and model optimization methods

Table 1: GSM8K accuracy (higher is better) for all combinations of data and model optimization methods vs. JoBS.

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	JoBS
Default	$71.9_{\pm 1.5}$	$71.1_{\pm 1.2}$	$72.3_{\pm 3.2}$	$68.7_{\pm 1.0}$	$74.4_{\pm 2.0}$	-
DARTS	$73.1_{\pm 0.9}$	$71.7_{\pm 0.7}$	$74.7_{\pm 1.4}$	$69.3_{\pm 0.5}$	$66.8_{\pm 0.8}$	-
AutoLoRA	$72.9_{\pm 1.2}$	$75.2_{\pm 0.4}$	$70.9_{\pm 0.8}$	$68.5_{\pm 0.5}$	$74.0_{\pm 0.6}$	-
RoBoT	$71.8_{\pm 0.7}$	$72.7_{\pm 1.6}$	$74.0_{\pm 1.9}$	$73.1_{\pm 1.6}$	$70.2_{\pm 1.8}$	-
JoBS	-	-	-	-	-	**86.4**$_{\pm 1.2}$

We evaluate all combinations of conventional data and model optimization methods applied independently to `Llama-3-8B-Instruct`. These are strong baselines under the standard paradigm of independent data and model optimization. However, they *cannot* exploit iterative black-box feedback or model the interdependence between data and model configurations, which are precisely the capabilities that JoBS introduces. Table 1 shows that JoBS consistently outperforms all such combinations, achieving the best accuracy of 86.4% on GSM8K and highlighting the clear value of joint co-optimization with iterative feedback. Additional results for other language tasks are provided in App. M.1.

6.3 Comparison with BO methods

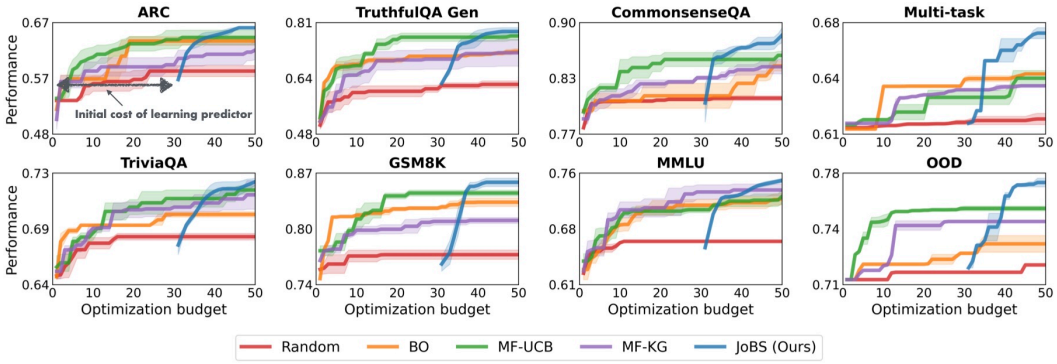


Figure 5: Best-found LLM performance against optimization budget (training steps, in thousands) for JoBS and BO baselines across six language tasks, as well as out-of-domain (OOD) and multi-task settings. JoBS consistently achieves the highest performance across all settings. JoBS' curve begins later as an initial portion of the budget is allocated to predictor learning.

In-domain setting. We compare JoBS against the three BO baseline methods (**vanilla, MF-KG, MF-UCB**) described in Sec. 6.1 across six language tasks, where the downstream evaluation task (GSM8K) is part of the training domain. All methods are allocated the same optimization budget of $C = 50000$ steps, with two discrete fidelities at $B_{\text{small}} = 100$ and $B = 1000$ steps for multi-fidelity methods. Note that JoBS' curve begins later as an initial portion of the budget is allocated to predictor learning. As shown in Fig. 5, JoBS consistently outperforms all baselines across all six tasks. Outperforming vanilla BO confirms that reducing the per-iteration cost via the performance predictor enables more effective exploration within the same budget, while outperforming multi-fidelity BO baselines shows that the performance predictor is more effective than using a GP kernel for modeling the relationship between low- and high-fidelity observations [43]. Further details on the comparison with multi-fidelity BO are provided in App. B.

Out-of-domain setting. We evaluate JoBS in an out-of-domain (OOD) setting where the target benchmark (GSM8K) is withheld from the pool of training data domains, forcing the optimizer to rely on knowledge transfer from related domains. As shown in Fig. 5 (bottom-right), JoBS outperforms all BO baselines even in this harder setting, consistently finding configurations that generalize to the held-out domain. This indicates that the performance predictor generalizes beyond the training

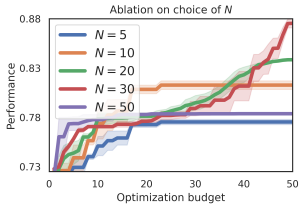
domains, and JoBS remains effective provided the training domains have sufficient coverage of the target task distribution, which is a reasonable assumption in practical deployment scenarios.

Multi-task setting. We evaluate JoBS in a multi-task setting where the downstream metric is an aggregation of multiple benchmarks to demonstrate its flexibility to arbitrary black-box objectives. Specifically, we evaluate on five benchmarks—GSM8K, TruthfulQA, TriviaQA, CommonsenseQA, and MMLU—using the test splits that are excluded from training, and aggregate their accuracies into a single composite objective (average normalized score). As shown in Fig. 5 (top-right), JoBS remains effective in this setting, highlighting that JoBS is agnostic to the form of the downstream metric and naturally extends to arbitrary black-box objectives, including composite ones. JoBS is also compatible with the multi-objective BO setting [10], though a thorough investigation is left for future work.

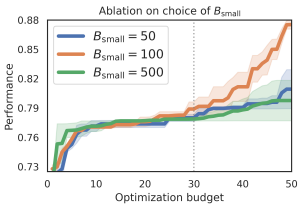
Extended settings. We further evaluate JoBS on different model families (Qwen), larger models (14B, 32B), full-parameter fine-tuning, and evaluation loss as the metric. As shown in App. M.2, JoBS remains consistently effective across different model families, model sizes, training settings, and downstream metrics, demonstrating its flexibility to various settings.

6.4 Ablation studies and computational cost

We conduct several ablation studies (using Llama-3-8B-Instruct evaluated on CommonsenseQA) to investigate key design choices in JoBS, and additionally analyze its computational cost.



(a) Varying N



(b) Varying B_{small}

Method	Time (hours)
LESS	16.3
DoReMi	18.5
IF	52
Diversity	122
JoBS	13.3

(c) Wall-clock runtime

Figure 6: (a) Ablation on choice of N. (b) Ablation on choice of B_{small}. (c) Wall-clock runtime of JoBS vs. data optimization methods.

Effects of varying N. Fig. 6a illustrates the tradeoff between the number of initial full training runs N used to train the performance predictor $\mathcal{F}$ and the number of available BO iterations under a fixed budget. A larger N improves predictor accuracy, but consumes more budget, while a smaller N allows more BO iterations but yields a less accurate predictor. As such, a value of N that is too small or too large degrades JoBS' performance. Empirically, $N = 30$ strikes the best balance under our experimental setting, which is consistent with our theoretical findings in Sec. 5.4.

Effects of varying B_{small}. Fig. 6b shows a similar tradeoff for B_{small}. A larger B_{small} gives more accurate predictions but reduces the number of available BO iterations, while a smaller B_{small} allows more BO iterations but yields a less accurate predictor. From our experiments, we find $B_{\text{small}} = 100$ steps (10% of $B = 1000$) provides the best balance between prediction accuracy and BO iterations.

Computational costs. Lastly, Fig. 6c shows that JoBS has a lower wall-clock runtime than all data optimization baselines under the same budget $C = 50000$ steps. A detailed computational cost analysis is provided in App. K, and qualitative case studies of the optimal training configurations found by JoBS are presented in App. L.

7 Conclusion and future work

We introduced JoBS, a BO-based approach that co-optimizes data and model configurations for LLMs. JoBS learns a scaling-law-inspired performance predictor to estimate fully trained LLM performance from only a small number of training steps, eliminating the need for full LLM training at every iteration. We showed that JoBS enjoys theoretical convergence guarantees and derived the optimal budget allocation between predictor learning and BO iterations. One limitation is that JoBS

requires iterative deployment to optimize the training configuration, though as we discuss in App. H, this reflects real-world LLM development cycles.

Several promising directions remain. Can existing scaling law formulas be directly incorporated into GP priors to further improve optimization efficiency? Can the performance predictor be reused across different tasks to amortize its training cost? Addressing these questions would further reduce the computational demands of `JoBS`, bringing principled black-box co-optimization closer to the practical requirements of LLM development.

References

[1] Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in llm pre-training. *arXiv:2505.13738*, 2025.

[2] Mayee F. Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Ré. Aioli: A unified optimization framework for language model data mixing. *arXiv:2411.05735*, 2025.

[3] Yangyi Chen, Binxuan Huang, Yifan Gao, Zhengyang Wang, Jingfeng Yang, and Heng Ji. Scaling laws for predicting downstream performance in llms. *arXiv:2410.08527*, 2025.

[4] Zhiliang Chen, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Towards AutoAI: Optimizing a machine learning system with black-box and differentiable components. In *Proc. ICML*, 2024.

[5] Zhiliang Chen, Gregory Kang Ruey Lau, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Duet: Optimizing training data mixtures via feedback from unseen evaluation tasks. *arXiv:2502.00270*, 2025.

[6] Sayak Ray Chowdhury and Aditya Gopalan. On kernelized multi-armed bandits. In *Proc. ICML*, 2017.

[7] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457*, 2018.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021.

[9] Zhongxiang Dai, Haibin Yu, Bryan Kian Hsiang Low, and Patrick Jaillet. Bayesian optimization meets Bayesian optimal stopping. In *Proc. ICML*, 2019.

[10] Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Multi-objective bayesian optimization over high-dimensional search spaces. *arXiv:2109.10964*, 2022.

[11] Peter I. Frazier. A tutorial on bayesian optimization. *arXiv:1807.02811*, 2018.

[12] 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.

[13] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[14] Zhenfeng He, Yao Shu, Zhongxiang Dai, and Bryan Kian Hsiang Low. Robustifying and boosting training-free neural architecture search. *arXiv:2403.07591*, 2024.

[15] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv:2009.03300*, 2021.

[16] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv2203.15556*, 2022.

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

[18] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. *arXiv:1909.06146*, 2019.

[19] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv:1705.03551*, 2017.

[20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

[21] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv:1703.04730*, 2020.

[22] Dong Bok Lee, Aoxuan Silvia Zhang, Byungjoo Kim, Junhyeon Park, Steven Adriaensen, Juho Lee, Sung Ju Hwang, and Hae Beom Lee. Cost-sensitive freeze-thaw bayesian optimization for efficient hyperparameter tuning. *arXiv:2510.21379*, 2025.

[23] Houyi Li, Wenzhen Zheng, Qiufeng Wang, Zhenyu Ding, Haoying Wang, Zili Wang, Shijie Xuyang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable scale: Part ii, farseer: A refined scaling law in large language models. *arXiv:2506.10972*, 2025.

[24] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv:2109.07958*, 2022.

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2019.

[26] 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. *arXiv:2407.01492*, 2025.

[27] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv:1609.07843*, 2016.

[28] Vu Nguyen, Sunil Gupta, Santu Rana, Cheng Li, and Svetha Venkatesh. Practical batch bayesian optimization for less expensive functions. *arXiv:1811.01466*, 2018.

[29] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv:1811.12808*, 2020.

[30] Mustafa Shukor, Louis Bethune, Dan Busbridge, David Grangier, Enrico Fini, Alaaeldin El-Nouby, and Pierre Ablin. Scaling laws for optimal data mixtures. *arXiv:2507.09404*, 2025.

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

[32] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *1406.3896*, 2014.

[33] Kevin Swersky, Yulia Rubanova, David Dohan, and Kevin Murphy. Amortized bayesian optimization over discrete spaces. In *Proc. UAI*, 2020.

[34] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv:1811.00937*, 2019.

[35] Sebastian Shenghong Tay, Chuan-Sheng Foo, Daisuke Urano, Richalynn Leong, and Bryan Kian Hsiang Low. Bayesian optimization with cost-varying variable subsets. In *Proc. NeurIPS*, 2023.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Neurips*, 2017.

[37] Jingtan Wang, Xiaoqiang Lin, Rui Qiao, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Helpful or harmful data? fine-tuning-free shapley attribution for explaining language model predictions. *arXiv:2406.04606*, 2024.

[38] Peiqi Wang, Yikang Shen, Zhen Guo, Matthew Stallone, Yoon Kim, Polina Golland, and Rameswar Panda. Diversity measurement and subset selection for instruction tuning datasets. *arXiv:2402.02318*, 2024.

[39] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Workshop on Noisy User-generated Text*, 2017.

[40] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv:1910.11858*, 2020.

[41] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[42] Chuhan Wu and Ruiming Tang. Performance law of large language models. *arXiv:2408.09895*, 2024.

[43] Jian Wu, Saul Toscano-Palmerin, Peter I. Frazier, and Andrew Gordon Wilson. Practical multi-fidelity bayesian optimization for hyperparameter tuning. *arXiv:1903.04703*, 2019.

[44] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv:2402.04333*, 2024.

[45] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *arXiv:2305.10429*, 2023.

[46] Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. Data selection for language models via importance resampling. *arXiv:2302.03169*, 2023.

[47] Wanyun Xie, Francesco Tonin, and Volkan Cevher. Chameleon: A flexible data-mixing framework for language model pretraining and finetuning, 2025.

[48] Jiasheng Ye, Peiju Liu, Tianxiang Sun, Yunhua Zhou, Jun Zhan, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. *arXiv:2403.16952*, 2024.

[49] Thomson Yen, Andrew Wei Tung Siah, Haozhe Chen, Tianyi Peng, Daniel Guetta, and Hongseok Namkoong. Data mixture optimization: A multi-fidelity multi-scale bayesian framework, 2025.

[50] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv:2402.17193*, 2024.

[51] Chi Zhang, Huaping Zhong, Kuan Zhang, Chengliang Chai, Rui Wang, Xinlin Zhuang, Tianyi Bai, Qiu Jiantao, Lei Cao, Ju Fan, Ye Yuan, Guoren Wang, and Conghui He. Harnessing diversity for important data selection in pretraining large language models. In *Proc. ICLR*, 2025.

[52] Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv:2403.09113*, 2024.

A Impact statement

JoBS reduces the computational cost of fine-tuning LLMs by eliminating the need for full training runs at every optimization iteration. The primary societal benefit is democratization, as by making principled LLM optimization more computationally accessible, our method lowers the barrier for smaller research groups and organizations to develop high-quality, task-specific models without requiring large-scale compute infrastructure. We also show empirically that JoBS improves the overall performance for LLMs in the sequential feedback setting.

B Additional related works on multi-fidelity BO

Multi-fidelity BO methods typically offer the option to query a black-box function at either high or low fidelity. High-fidelity queries faithfully evaluate the true objective function but are more expensive, while low-fidelity queries provide cheaper but approximate estimates. Multi-fidelity BO incorporates fidelity as an *additional input dimension* and uses a cost-aware acquisition function [43] to determine which fidelity to query at every iteration.

In our setting, observing LLM performance after a fraction of the training steps can be considered a low-fidelity evaluation, while observing the LLM performance after full training can be viewed as a high-fidelity evaluation. The key distinction between JoBS and standard multi-fidelity BO lies in how fidelities are modeled. Rather than jointly modeling low and high-fidelity observations with a GP (by treating fidelity as an additional input dimension), JoBS explicitly models the relationship between low and high-fidelity observations using an expressive neural network — the performance predictor — which is *decoupled from the GP surrogate*. This decoupling is important, as a GP is often not expressive enough to jointly model the relationship between fidelities, which explains why multi-fidelity BO methods underperform in our experiments. The same limitation applies to early-stopping BO [9], which relies on a GP to model across fidelities. Furthermore, most multi-fidelity BO works do not analyze the optimal number of queries per fidelity. JoBS addresses this by providing theoretical guidelines for choosing an optimal number of high-fidelity observations to query (Sec. 5.4).

A related line of work, freeze-thaw BO [32, 22], decides at each iteration whether to continue training an existing configuration or to explore a new one. We find this approach ill-suited to our setting for two reasons. First, freeze-thaw methods require storing multiple copies of fine-tuned LLMs simultaneously, which is memory-intensive at the current scale of LLMs. Second, they typically model performance improvement with an exponential decay kernel, which assumes monotonically improving performance with more training. However, Fig. 3 shows that in our setting, the change in LLM performance can *decrease* with more training steps when the chosen data mixture poorly matches the downstream task. As such, the exponential decay kernel introduced in these works is unable to capture this behavior and is ill-suited to our setting.

C Downstream task metrics and sequential feedback setting

JoBS uses BO as its algorithmic backbone, leveraging iterative downstream task metric feedback to improve the training configuration. Crucially, JoBS is agnostic to the form of this feedback. In the most practical deployment scenario, the metric could reflect real-world user behavior, such as user ratings or time spent on a chatbot, aggregated over a set of users.

In our experiments, we consider three forms of downstream task metrics to demonstrate JoBS' versatility: downstream evaluation loss, performance on a specific QA benchmark, and aggregated performance across multiple QA benchmarks (i.e., a multi-task setting). Importantly, the specific benchmark used for evaluation is never revealed to our optimization algorithm (only its numerical output is observed), mirroring real-world conditions where the evaluation data distribution is unknown [5].

D Additional background on Bayesian optimization

At every BO iteration, the BO algorithm proposes the next configuration x_{t+1} as the configuration which maximizes some acquisition function, such as the *upper confidence bound* (UCB) [31], given by $x_{t+1} = \text{argmax}_x \, \mu_t(x) + \beta_{t+1}\sigma_t(x)$, where β_{t+1} is an exploration parameter.

A common kernel choice is the *squared exponential* (SE) kernel $\kappa(x, x') \triangleq \exp(-\|x - x'\|_2^2/(2m^2))$ with a *length-scale* hyperparameter m that can be learned via maximum likelihood estimation from observations.

E Proof of Theorem 5.1

Theorem 5.1. *Let $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ be the performance landscape of the LLM training configuration with bounded RKHS norm: $\|\mathcal{L}\|_\kappa = \sqrt{\langle \mathcal{L}, \mathcal{L} \rangle_\kappa} \leq \mathcal{B}$ w.r.t. kernel κ. Assume our performance predictor is learned from N full training runs and makes predictions $\mathcal{F}(\mathcal{X}, \mathcal{M}, \{\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},b})\}_{b \leq B_{small}}) = \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B}) + \epsilon$ from $B_{small} < B$ training steps where ϵ satisfies the assumption above. Then, running JoBS over LLM training configurations $\mathcal{X}, \mathcal{M}$ until an optimization budget of C is exhausted yields $T = \left\lfloor \frac{C - NB}{B_{small}} \right\rfloor$ BO iterations, and with the IGP-UCB acquisition function [6], JoBS achieves the following average regret with probability at least $1 - \delta$:*

$$\mathcal{R}_T = \mathcal{O}\left(\sqrt{\frac{B_{small}}{C - NB - B_{small}}} \left(\mathcal{B}\sqrt{\gamma_T} + \frac{k}{\sqrt{N}} \sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)} \right) \right), \tag{3}$$

where γ_T is the maximum information gain of $\mathcal{L}$ after $T = \left\lfloor \frac{C - NB}{B_{small}} \right\rfloor$ BO iterations.

Proof. To begin, recall that we are trying to maximize our LLM performance, a black-box function $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ (Sec. 3). Using our scaling law predictor (Sec. 5.2), we instead train our LLM for B_{small} training steps (or time) and observe $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B_{\text{small}}})$. We then use the performance predictor to produce $\mathcal{F}(\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B_{\text{small}}}))$, estimating the LLM performance if we trained it fully for B training steps. Since we are predicting the LLM performance, our model prediction is noisy with $\mathcal{F}(\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B_{\text{small}}})) = \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B}) + \epsilon$. Hence, we only have access to a noisy estimate of our black-box function: $\mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B}) + \epsilon$. The prediction error is also R-sub-Gaussian as suggested from our empirical findings (Fig. 4). As such, this formulation is in line with the BO algorithmic framework introduced in Sec. 5.1, where we obtain noisy observations of the true underlying function.

Next, we aim to draw the relationship between the choice of N and our predictor's error ϵ and consequently, JoBS' regret. First, we present the following lemma from [6]:

Lemma E.1. *Let $||f||_\kappa = \sqrt{\langle f, f \rangle_\kappa} \leq \mathcal{B}$. Also, assume that the observation noise associated with each BO iteration is R-sub-Gaussian with $R > 0$. Then with probability at least $1 - \delta$, the following holds for BO iteration $t \leq T$:*

$$|\mu_t(x) - f(x)| \leq \left(\mathcal{B} + R\sqrt{2(\gamma_t + 1 + \ln(1/\delta))} \right) \sigma_t(x) \tag{4}$$

where γ_t is the maximum information gain after t observations and $\mu_t(x), \sigma_t^2(x)$ are mean and variance of posterior distribution of GP defined in Eq. 2 with $\zeta = 1 + 2/T$.

In our setting, set $f = \mathcal{L}$ (our LLM performance after fine-tuning) and $x = \mathcal{X}, \mathcal{M}$ (our LLM training configuration). This lemma indicates that our estimated mean $\mu_t(x)$ of our performance landscape from our fitted GP over historical observations of LLM performance deviates from the true LLM performance $f(x) = \mathcal{L}(\theta_{\mathcal{X},\mathcal{M},B})$ by at most the right-hand side term in Lemma E.1.

We are now ready to prove Theorem 5.1. First, we observe that the next LLM training configuration x_t at each BO iteration t is chosen via the IGP-UCB acquisition function (i.e., $x_t = \text{argmax}_x \, \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$ and $\beta_t = \mathcal{B} + R\sqrt{2(\gamma_{t-1} + 1 + \ln(1/\delta))}$ where the observation noise associated with each BO iteration is R-sub-Gaussian). Thus, we can see that at each iteration $t \geq 1$, we have $\mu_{t-1}(x_t) + \beta_t \sigma_{t-1}(x_t) \geq \mu_{t-1}(x^*) + \beta_t \sigma_{t-1}(x^*)$. It then follows that for all $t \geq 1$ and with probability at least $1 - \delta$,

$$\begin{aligned} f(x^*) - f(x_t) &\overset{(1)}{\leq} \beta_t \sigma_{t-1}(x_t) + \mu_{t-1}(x_t) - f(x_t) \\ &\overset{(2)}{\leq} \beta_t \sigma_{t-1}(x_t) + \mu_{t-1}(x_t) + (\beta_t \sigma_{t-1}(x_t) - \mu_{t-1}(x_t)) \\ &\leq 2\beta_t \sigma_{t-1}(x_t) \end{aligned} \tag{5}$$

where $\overset{(1)}{\leq}$ uses the fact that via Lemma E.1 and our acquisition function, $f(x^*) \leq \beta_t \sigma_{t-1}(x^*) + \mu_{t-1}(x^*) \leq \beta_t \sigma_{t-1}(x_t) + \mu_{t-1}(x_t)$ and $\overset{(2)}{\leq}$ once again uses Lemma E.1.

Next, we adjust the total number of BO iterations in our problem setting, depending on the chosen N. As mentioned in the analysis of Sec. 5.3, under an overall optimization budget of C, collecting N initial full-training runs leaves enough budget for $\lfloor \frac{C-NB}{B_{\text{small}}} \rfloor$ BO iterations. Hence, we analyze how JoBS' cumulative regret grows for up to $T = \lfloor \frac{C-NB}{B_{\text{small}}} \rfloor$ iterations. Using Eq. 5, we can bound the cumulative regret by

$$\sum_{t=1}^{T} r_t = \sum_{t=1}^{T} (f(x^*) - f(x_t)) \leq 2 \sum_{t=1}^{T} \beta_t \sigma_{t-1}(x_t) \tag{6}$$

where r_t is the per-iteration regret.

Since we know that $\sum_{t=1}^{T} \sigma_{t-1}(x_t) = \mathcal{O}(\sqrt{T\gamma_T})$ and used $\beta_t = \mathcal{B} + R\sqrt{2(\gamma_{t-1} + 1 + \ln(1/\delta))}$, the cumulative regret in Theorem 5.1 can be written as

$$\begin{aligned}
\mathcal{R}_T &= \sum_{t=1}^{T} r_t && (7)\\
&\leq 2 \sum_{t=1}^{T} \beta_t \sigma_{t-1}(x_t) && (8)\\
&\leq 2\mathcal{O}(\sqrt{T\gamma_T})(\mathcal{B} + R\sqrt{2(\gamma_T + 1 + \ln(1/\delta))}) && (9)\\
&= \mathcal{O}\left(\mathcal{B}\sqrt{T\gamma_T} + R\sqrt{T}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right) && (10)\\
&\overset{(1)}{=} \mathcal{O}\left(\mathcal{B}\sqrt{\left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor \gamma_T} + R\sqrt{\left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right) && (11)\\
&\overset{(2)}{=} \mathcal{O}\left(\mathcal{B}\sqrt{\left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor \gamma_T} + \frac{k}{\sqrt{N}}\sqrt{\left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right), && (12)
\end{aligned}$$

where $\overset{(1)}{=}$ uses the fact that if we collected N full-training runs initially, the number of BO iterations available is $T = \left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor$ and $\overset{(2)}{=}$ uses **Assumption 1** that our prediction error is R-sub-Gaussian with $R = \frac{k}{\sqrt{N}}$.

Lastly, because different numbers of initial full-training runs N will influence the number of remaining BO iterations, we derive the average regret w.r.t. the number of BO iterations by dividing the cumulative regret bounds obtained in Eq. (12) by $\left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor$ throughout:

$$\begin{aligned}
\frac{\mathcal{R}_T}{T} &\overset{(1)}{=} \mathcal{O}\left(\mathcal{B}\sqrt{\frac{\gamma_T B_{\text{small}}}{C - NB - B_{\text{small}}}} + \frac{k}{\sqrt{N}}\sqrt{\frac{B_{\text{small}}}{C - NB - B_{\text{small}}}}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right) && (13)\\
&= \mathcal{O}\left(\left(\sqrt{\frac{B_{\text{small}}}{C - NB - B_{\text{small}}}}\right)\left(\mathcal{B}\sqrt{\gamma_T} + \frac{k}{\sqrt{N}}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right)\right) && (14)
\end{aligned}$$

where $\overset{(1)}{=}$ divides the cumulative regret in Eq. (12) throughout by $T = \left\lfloor \frac{C-NB}{B_{\text{small}}} \right\rfloor$ and uses the fact that $\frac{1}{\lfloor a/b \rfloor} \leq \frac{b}{a-b}$ when $a > b > 0$ (which holds here since $C - NB > B_{\text{small}}$ in our setting).

One interesting point to note is that our algorithm incurs some regret when collecting N random training configurations initially. However, since our black-box function is bounded (under the SE-kernel) and N is not assumed to be dependent on the total optimization budget T, then the average regret incurred when collecting training configurations initially can be removed in the big-O notation. □

E.1 Comparison of regret bound in vanilla BO

We show the conditions in which the average regret upper-bound of JoBS is smaller than that of the vanilla BO case. To simplify notation, let T be the number of BO iterations available in the vanilla BO case. Recall that in normal BO (without any performance predictor), the cumulative regret is [4, 6]

$$\mathcal{O}\left(\mathcal{B}\sqrt{T\gamma_T} + R\sqrt{T}\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right). \tag{15}$$

Dividing by T, we obtain an average regret of:

$$\mathcal{O}\left(\frac{\mathcal{B}\sqrt{\gamma_T}}{\sqrt{T}} + \frac{R\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}}{\sqrt{T}}\right). \tag{16}$$

Consider that in our case, JoBS allocates a constant number of iterations (e.g., $N \ll T$) to learn the performance predictor, which yields noisy observations with error distribution that is R_1-sub-Gaussian, where $R_1 > R$. This means the predictor does not recover the original performance landscape perfectly. By doing so, each BO iteration consumes 10% of the original resources (since we only need to fine-tune an LLM to 10% of the original number of training steps) and hence, we can perform $10(T - N)$ more BO iterations. Assuming that the average regret of the first N iterations where we make random queries is constant (we can ignore it in the asymptotic bound), this implies the average regret bound of JoBS is:

$$\mathcal{O}\left(\frac{\mathcal{B}\sqrt{\gamma_T}}{\sqrt{10(T-N)}} + \frac{R_1\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}}{\sqrt{10(T-N)}}\right). \tag{17}$$

Note that this is essentially the same bound as Theorem 5.1 but without the complication of deriving the expression for the number of BO acquisition steps available under a limited optimization budget. Comparing Eq.(16) and Eq.(17), we see that the average regret bound of JoBS is smaller than vanilla BO with probability at least $1 - \delta$ if and only if:

$$\left(\mathcal{B}\sqrt{\gamma_T} + R_1\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right) < \left(\mathcal{B}\sqrt{\gamma_T} + R\sqrt{\gamma_T^2 + \gamma_T \ln(1/\delta)}\right)\sqrt{\frac{10(T-N)}{T}}. \tag{18}$$

The differences between the left and right side are highlighted in red and this inequality admits a neat form that is easy to interpret: R_1 is larger than R (i.e., using a predictor gives noisier observations), but as long as $\sqrt{\frac{10(T-N)}{T}}$ is much larger (i.e., we can run more BO iterations; it is almost 10 times larger in this case), the average regret of JoBS will be smaller than that found in the vanilla BO case.

F Computational complexity

The computational complexity of the BO algorithm is well studied [31]; a detailed summary is provided in Appendix F of Chen et al. [5]. Briefly, the dominant computational cost scales cubically with the number of iterations T, i.e., $\mathcal{O}(T^3)$, due to kernel matrix factorization (e.g., via Cholesky decomposition). The dependence on problem dimension n arises primarily from acquisition function optimization, which incurs $\mathcal{O}(c\,n T^2)$ per iteration, where c is the number of optimization restarts. In typical regimes where $n \ll T$, the cubic dependence on T dominates, yielding an overall complexity of $\mathcal{O}(T^3)$ with only linear dependence on n.

Beyond the BO overhead, each iteration requires training an LLM, which is the dominant practical cost. Naively, T BO iterations require T full LLM training runs. However, this cost need not be front-loaded. In realistic deployment scenarios, the LLM can be rolled out to collect metric feedback over an extended period, as is standard practice in real-world LLM development (e.g., ChatGPT and Gemini are both rolled out and improved over months or years).

G Practical design choices

We discuss several practical design choices involved in JoBS. First, we use a mixed kernel for the GP that combines a Hamming-distance kernel with an SE kernel, enabling more principled handling

Kernel	MMLU	TriviaQA	GSM8K
Hamming + RBF (SE2)	74.8 ± 0.6	72.6 ± 0.8	86.6 ± 0.8
RBF (SE2)	74.5 ± 0.3	72.0 ± 1.1	86.4 ± 1.2
Matérn	73.7 ± 0.8	71.0 ± 0.6	85.6 ± 0.7

Table 2: Performance across different kernels on MMLU, TriviaQA, and GSM8K.

of the mixed discrete-continuous input space (data mixtures are continuous variables while LoRA rank and layer are discrete variables). Second, data from prior training runs (earlier experiments or published works) can be used to learn the predictor at no additional budget cost, further improving prediction accuracy (Fig. 4). Lastly, we choose the number of training steps as the budget constraint in our problem formulation to keep the number of data samples used in the BO loop constant. However, JoBS' budget is a generalizable constraint, and other constraint choices such as total number of training tokens or floating-point operations (FLOPs) can be considered with little change to our formulation, since the number of data points and model size is approximately constant in our setting (the number of fine-tuning parameters is small relative to the model size).

H Further discussion of limitations

One limitation of JoBS is that it requires several rounds of feedback to iteratively optimize an LLM's training configurations. However, this characteristic is found in most black-box optimization settings (e.g., BO [31]) and is not unique to our work. In addition, we view the process of obtaining feedback as a natural part of the LLM deployment life-cycle. For instance, many commercial LLMs, such as ChatGPT, Gemini or DeepSeek, are iteratively improved over many deployment iterations by gathering feedback from users (in fact, it would not be surprising if such iterative optimization approaches are already employed in practice under-the-hood).

In addition, in our setting, obtaining feedback is the only way we can optimize the LLM training configurations. Other methods, such as data selection, cannot exploit feedback and hence are suboptimal in our setting.

I Additional experimental details

Search space. The data configuration $\mathcal{X}$ consists of 9 dimensions representing the mixing ratio (a probability simplex) over 9 training data domains. The model configuration $\mathcal{M}$ consists of 10 dimensions:

1. Number of LLM layers to apply LoRA to $\in [1, 31]$ (varies by LLM architecture),
2. Whether to apply LoRA to front or rear layers (binary),
3. Whether to apply LoRA to Q-projection (binary),
4. Whether to apply LoRA to K-projection (binary),
5. Whether to apply LoRA to V-projection (binary),
6. Whether to apply LoRA to MLP-up-projection (binary),
7. Whether to apply LoRA to MLP-down-projection (binary),
8. LoRA rank $\in [1, 256]$,
9. LoRA dropout $\in [0, 1]$, and
10. LoRA $\alpha \in [1, 500]$.

Default configurations. For baselines that do not optimize a particular component, we adopt the following default fine-tuning configuration: (a) LoRA applied to all LLM layers, (b) LoRA applied to all Q-, V-, K-, MLP-up-, MLP-down-projection, (c) rank $= 16$, (d) dropout $= 0.1$, (e) $\alpha = 16$, and (f) uniform data mixture. These defaults are consistent with standard configurations used in existing works and off-the-shelf fine-tuning frameworks. In our experiments, we use 8-shot prompting

with chain-of-thought reasoning, with $B_{\text{small}} = 100$ training steps, $B = 1000$ training steps and optimization budget $C = 50000$ training steps.

Performance predictor. To learn the performance predictor (Sec. 5.2), we sample a random Sobol sequence [28] of 30 LLM training configurations and fine-tune the LLM for $B = 1000$ training steps (approximately 1.5 training epochs). During training, we collect the LLM evaluation loss or performance at intervals of 25 training steps, up to $B_{\text{small}} = 100$ training steps. We construct the neural network input from observations at steps $25, 50, 75, 100$, together with the data and model configurations. The performance at $B = 1000$ steps serves as the neural network's prediction target. We fit a densely-connected neural network $\mathcal{F}$ with 3 layers of width 64 on this dataset.

BO setup. The $N = 30$ full training observations are also used to warm-start the GP surrogate, leaving a remaining budget of $\frac{50000-30\times1000}{100} = 200$ BO iterations. We use a mixed kernel combining a Hamming-distance kernel and an SE kernel for the GP, implemented via the `BoTorch` library. Kernel hyperparameters are estimated via maximum likelihood at the end of every iteration. All training configuration parameters are normalized to $[0, 1]$ before fitting the GP. Experiments were run on 4 `H200` GPUs and a training batch size of 64.

J Implementation details of baselines

LESS [44]. We pass each training data point from each data domain into the LLM to construct a gradient store [44]. Since we do not know the exact evaluation task (just like the setting considered in Chen et al. [5]), we form a validation mixture consisting of uniformly sampled data from each data domain, before retrieving a subset of 10000 data points from the gradient store with the highest feature similarity with the evaluation dataset. We follow the implementation given in `https://github.com/princeton-nlp/LESS`.

DoReMi [45]. We use DoReMi to optimize the data mixing ratio across the training data domains, according to the implementation given in `https://github.com/sangmichaelxie/doremi`. Again, since we do not have direct access to the evaluation task (as DoReMi requires an explicit validation dataset), we form the validation dataset using a uniform validation mixture similar to LESS (see above).

IF [21]. We compute the influence scores of each data point (following the implementation in `https://github.com/alstonlo/torch-influence`) w.r.t. a uniform validation mixture constructed from the training data domains (see above). Then, we retrieve the top 10000 data points with the highest influence scores.

Diversity [38]. We first project every data point into a continuous semantic space using an off-the-shelf embedding model. Then, we retrieve the data subset with the highest log-determinant value, which is derived by arranging data points' embeddings in a matrix and computing the determinant. There is a greedy implementation [38] that improves the computational efficiency.

DARTS [25]. We optimize the same model configurations detailed in App. I by introducing a differentiable parameter for each model component (e.g., LoRA rank). For example, LoRA rank is controlled by a single differentiable parameter, while LoRA layer selection is controlled by a softmax layer with weights, allowing us to drop certain LoRA layers (if the weight is decreased to 0 after update). We alternate between freezing the differentiable parameters that control the model configuration and model parameters that control the LLM behavior every 5 training steps. We optimize these objectives in an alternating fashion, switching between updating configuration parameters and model parameters every 5 steps.

AutoLoRA [52]. We follow the implementation in `https://github.com/ruz048/AutoLoRA`, which tunes the LoRA rank for each layer.

RoBoT [14]. We utilize the training-free proxies given in [14] and perform BO over the weights assigned to each training-free proxy. With the optimized weights, we then iterate through the model configurations to find the one with the best (weighted) training-free measure.

MF-KG [43]. The implementation is reproduced from `https://botorch.org/docs/tutorials/multi_fidelity_bo/`.

MF-UCB. We use a cost-aware UCB acquisition function that follows the framework in Yen et al. [49].

K Computational cost of JoBS vs. baselines

Quantitative comparison of wall-clock hours. JoBS is run for 50000 training steps in total. This equates to approximately 50000 seconds of wall-clock time (13.3 hours) using a H200 GPU. All model selection methods [14, 25] used in our paper are iterative in nature and require repeated fine-tuning of LLMs. Equal time budget was allocated to both JoBS and all baseline methods for a fair comparison. We found that given equal computation time, JoBS achieves better performance than all model selection methods. For data optimization methods (LESS, DoReMi, IF, Diversity), we record their wall-clock runtime in Table 3. In general, we found data optimization methods to be computationally more expensive than JoBS.

Table 3: Wall-clock runtime comparison of data selection methods vs. JoBS.

Method	Time (hours)
LESS	16.3
DoReMi	18.5
IF	52
Diversity	122
JoBS	**13.3**

L Optimal LLM training configurations found by JoBS compared to baselines

We show some of the optimal LLM training configurations found by JoBS vs. other baselines. We divided the configurations into two tables detailing the best data (Table 4) and model (Table 5) configurations found for the **GSM8K** evaluation task.

Table 4: Optimal data mixing ratio found by JoBS vs. other baselines. The columns denote the ratio allocated to each training domain.

	CQA	GSM8K	PubmedQA	SciQ	TrivQA	TruthQA	Wiki	MMLU	ARC
JoBS	0	0.34	0	0.10	0.19	0	0.21	0.16	0
DoReMi	0.08	0.11	0.18	0.05	0.08	0.14	0.04	0.16	0.13

Of particular interest is that JoBS optimizes the data mixture by placing greater weights on some data domains based on their evaluation performance on the downstream task (in this case, GSM8K). Specifically, JoBS successfully inferred (without knowing that the evaluation task is GSM8K) that domains such as GSM8K, SciQ, TriviaQA, Wikipedia, and MMLU contain some math information and therefore includes them in the optimized data mixture.

On the other hand, DoReMi is a distributionally robust data mixing method and results in a more uniform data mixing ratio. As a result, the output data mixtures are not specialized for any single evaluation task, leading to lower performance when evaluated on specific individual tasks, such as GSM8K.

Table 5: Optimal model configuration found by JoBS vs. other baselines.

	Rank	NumLayers	Order	Q	K	V	Up	Down	dropout	α
JoBS	36	25	1	1	0	1	1	0	0.112	64
DARTS	12	13	0	1	1	1	1	0	0.058	45

Next, we examine the optimal model configurations found by JoBS. We noticed that JoBS prefers a higher LoRA rank and number of layers (i.e., how many layers to apply LoRA), but chooses to apply

LoRA to only certain transformer layers. In particular, JoBS found that for the GSM8K evaluation task, fine-tuning Q-, V-, and MLP-up-projection layers is sufficient to achieve good fine-tuning performance, and we should fine-tune the rear layers instead of the front layers (Order $= 1$).

M Additional experimental results and discussion

M.1 Additional experiments against data and model optimization methods

Tables 6, 7, 8, 9, and 10 extend the evaluation from Table 1 to five other language tasks: TruthfulQA, CommonsenseQA, MMLU, ARC, and TriviaQA. JoBS consistently outperforms all combinations of data and model optimization methods confirming that co-optimizing data and model configurations yields meaningful gains over treating them independently.

Table 6: TruthfulQA [24].

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	BO	JoBS
Default	$59.8_{\pm 1.9}$	$60.7_{\pm 1.0}$	$62.9_{\pm 2.6}$	$62.3_{\pm 1.3}$	$64.5_{\pm 1.2}$	$74.8_{\pm 1.0}$	-
DARTS	$61.9_{\pm 1.3}$	$62.4_{\pm 0.6}$	$66.7_{\pm 1.3}$	$63.8_{\pm 0.5}$	$64.8_{\pm 1.2}$	$77.0_{\pm 0.9}$	-
AutoLoRA	$61.7_{\pm 1.0}$	$66.9_{\pm 1.2}$	$64.5_{\pm 1.1}$	$65.3_{\pm 1.2}$	$65.9_{\pm 0.6}$	$72.5_{\pm 0.5}$	-
RoBoT	$64.2_{\pm 0.6}$	$65.8_{\pm 0.7}$	$58.9_{\pm 1.3}$	$56.9_{\pm 1.0}$	$65.8_{\pm 0.6}$	$73.9_{\pm 1.3}$	-
BO	$66.8_{\pm 1.2}$	$67.9_{\pm 0.5}$	$69.8_{\pm 0.9}$	$70.9_{\pm 1.0}$	$64.7_{\pm 1.4}$	$76.1_{\pm 2.0}$	-
JoBS	-	-	-	-	-	-	$\mathbf{80.6}_{\pm 1.1}$

Table 7: CommonsenseQA [34].

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	BO	JoBS
Default	$76.3_{\pm 1.0}$	$73.0_{\pm 0.8}$	$74.2_{\pm 1.7}$	$79.3_{\pm 0.7}$	$77.4_{\pm 1.7}$	$80.6_{\pm 0.8}$	-
DARTS	$79.6_{\pm 1.3}$	$76.3_{\pm 1.7}$	$76.1_{\pm 1.1}$	$73.7_{\pm 1.2}$	$80.1_{\pm 1.1}$	$79.6_{\pm 0.6}$	-
AutoLoRA	$78.9_{\pm 0.9}$	$79.8_{\pm 0.4}$	$76.1_{\pm 0.5}$	$77.9_{\pm 1.2}$	$78.0_{\pm 1.0}$	$81.5_{\pm 1.0}$	-
RoBoT	$74.9_{\pm 0.8}$	$75.5_{\pm 0.9}$	$77.1_{\pm 0.9}$	$79.4_{\pm 1.5}$	$76.3_{\pm 0.9}$	$80.2_{\pm 0.2}$	-
BO	$79.7_{\pm 1.3}$	$79.4_{\pm 0.3}$	$77.0_{\pm 0.4}$	$81.1_{\pm 0.9}$	$79.4_{\pm 1.1}$	$80.7_{\pm 1.2}$	-
JoBS	-	-	-	-	-	-	$\mathbf{84.3}\pm 2.4$

Table 8: MMLU [15].

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	BO	JoBS
Default	$61.2_{\pm 1.3}$	$63.5_{\pm 0.9}$	$59.7_{\pm 1.8}$	$57.9_{\pm 0.6}$	$62.1_{\pm 1.4}$	$64.2_{\pm 1.2}$	-
DARTS	$58.3_{\pm 0.7}$	$61.0_{\pm 2.1}$	$62.9_{\pm 1.0}$	$55.7_{\pm 1.6}$	$60.1_{\pm 0.5}$	$63.4_{\pm 2.0}$	-
AutoLoRA	$62.5_{\pm 1.4}$	$64.3_{\pm 0.6}$	$60.8_{\pm 2.2}$	$58.2_{\pm 1.9}$	$63.7_{\pm 0.8}$	$61.5_{\pm 1.1}$	-
RoBoT	$59.9_{\pm 0.9}$	$60.7_{\pm 1.2}$	$63.4_{\pm 1.7}$	$61.5_{\pm 1.5}$	$58.3_{\pm 2.3}$	$62.1_{\pm 0.7}$	-
BO	$55.8_{\pm 1.8}$	$57.2_{\pm 0.4}$	$61.3_{\pm 1.2}$	$60.5_{\pm 1.6}$	$63.9_{\pm 1.0}$	$59.6_{\pm 1.5}$	-
JoBS	-	-	-	-	-	-	$\mathbf{69.5}_{\pm 0.8}$

Table 9: ARC [7].

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	BO	JoBS
Default	$54.7_{\pm 1.3}$	$59.2_{\pm 0.7}$	$61.4_{\pm 2.0}$	$52.8_{\pm 1.5}$	$60.6_{\pm 0.9}$	$62.3_{\pm 1.2}$	-
DARTS	$58.1_{\pm 0.8}$	$61.0_{\pm 1.6}$	$62.8_{\pm 0.5}$	$54.3_{\pm 2.1}$	$57.9_{\pm 1.7}$	$60.5_{\pm 0.6}$	-
AutoLoRA	$60.4_{\pm 1.1}$	$63.2_{\pm 0.9}$	$58.6_{\pm 1.9}$	$55.1_{\pm 0.8}$	$62.1_{\pm 2.0}$	$59.8_{\pm 1.0}$	-
RoBoT	$56.8_{\pm 1.5}$	$58.7_{\pm 1.4}$	$61.1_{\pm 1.2}$	$60.3_{\pm 0.7}$	$55.7_{\pm 2.2}$	$61.4_{\pm 1.3}$	-
BO	$52.6_{\pm 2.0}$	$55.9_{\pm 0.6}$	$59.7_{\pm 1.3}$	$58.5_{\pm 1.4}$	$63.4_{\pm 1.0}$	$57.4_{\pm 0.9}$	-
JoBS	-	-	-	-	-	-	$\mathbf{70.4}_{\pm 1.3}$

Table 10: TriviaQA Gen [19].

↓ **Model | Data** →	Default	LESS	DoReMi	IF	Diversity	BO	JoBS
Default	$55.5_{\pm 1.4}$	$57.2_{\pm 0.8}$	$53.1_{\pm 0.9}$	$55.8_{\pm 0.7}$	$58.9_{\pm 0.8}$	$65.0_{\pm 0.6}$	-
DARTS	$58.2_{\pm 0.8}$	$61.3_{\pm 1.2}$	$61.0_{\pm 0.7}$	$63.3_{\pm 1.0}$	$59.2_{\pm 0.6}$	$66.7_{\pm 1.8}$	-
AutoLoRA	$67.8_{\pm 1.4}$	$64.7_{\pm 0.9}$	$70.6_{\pm 2.2}$	$68.6_{\pm 1.7}$	$66.2_{\pm 1.5}$	$69.7_{\pm 2.4}$	-
RoBoT	$58.4_{\pm 1.5}$	$62.3_{\pm 1.7}$	$64.2_{\pm 1.4}$	$57.2_{\pm 1.2}$	$63.4_{\pm 1.5}$	$68.2_{\pm 1.3}$	-
BO	$70.7_{\pm 1.4}$	$66.7_{\pm 0.8}$	$72.5_{\pm 0.8}$	$71.7_{\pm 0.9}$	$74.7_{\pm 1.0}$	$72.7_{\pm 2.3}$	-
JoBS	-	-	-	-	-	-	**76.2**± 1.9

M.2 Extension to different model sizes, full-parameter fine-tuning and evaluation loss

We extend our experiments to larger models (`Qwen3-14B`) (Fig. 7), full-parameter fine-tuning (Table 11), and evaluation loss as the downstream metric (Fig. 8). In the full-parameter fine-tuning setting, LoRA is not used; instead, JoBS optimizes which LLM layers to fine-tune directly. Our results show that JoBS remains consistently effective across different model families, model sizes, training settings, and downstream metrics.

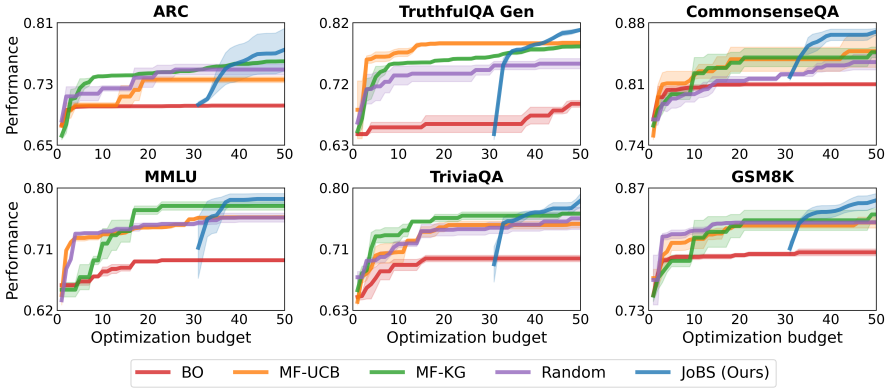


Figure 7: JoBS performance for LLM evaluation performance with `Qwen-14B`.

Table 11: JoBS applied to full-parameter fine-tuning of larger models.

↓ Model | Method →	LESS + AutoLoRA	DoReMi + DARTS	JoBS
Llama-3-8B-Instruct	0.73	0.76	**0.81**
Qwen3-14B	0.76	0.81	**0.83**
Qwen3-32B	0.86	0.82	**0.88**

M.3 Additional BO baselines

In Table 12, we jointly optimized LLM training configurations using several other naive approaches in our experiments. We experimented with 3 naive approaches: (a) **Random** randomly picks 500 different LLM training configurations, fine-tunes the LLM for 100 training steps each, use our performance scaling law predictor to predict and select the best-performing LLM training configuration. (b) **Random Data** performs JoBS on model configurations for only 10 iterations and repeats the experiment with 10 randomly chosen data configurations (this ensures the same amount of compute as performing JoBS on all LLM training configurations for 100 iterations). (c) **Random Model** repeats approach (b) on LLM training configurations instead. While these approaches serve as good sanity checks, they do not yield good LLM performances, largely because randomly selecting LLM training configurations does not exploit the learnt performance landscape from historically observed LLM performances.

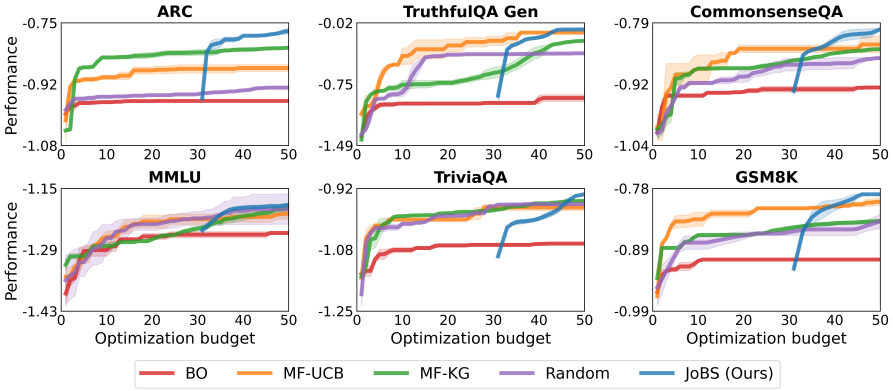


Figure 8: JoBS performance for LLM evaluation loss (instead of performance).

Table 12: Comparison of some naive baselines with JoBS (Higher is better), averaged over 5 trials. **Random Data** means that we randomly selected data mixtures and applied JoBS only on the model configurations (and vice versa for **Random Model**).

Model	Task	**Random Data**	**Random Model**	JoBS
Llama-3-8B-Instruct	GSM8K	$67.3_{\pm 1.6}$	$71.5_{\pm 0.9}$	$86.4_{\pm 1.2}$
	TruthfulQA	$59.8_{\pm 1.5}$	$64.2_{\pm 1.4}$	$80.6_{\pm 1.1}$
	CommonsenseQA	$76.4_{\pm 1.2}$	$76.3_{\pm 1.2}$	$84.3_{\pm 2.4}$
	MMLU	$66.4_{\pm 0.7}$	$63.1_{\pm 1.1}$	$69.5_{\pm 0.8}$
	ARC	$65.2_{\pm 1.7}$	$64.6_{\pm 0.6}$	$70.4_{\pm 1.3}$
	TriviaQA	$61.7_{\pm 2.4}$	$63.2_{\pm 1.5}$	$76.2_{\pm 1.2}$

N Asset licenses

Asset	**License**	**URL**
Wikitext	CC-BY-SA 3.0	https://huggingface.co/datasets/mindchain/wikitext2
GSM8K	MIT	https://huggingface.co/datasets/openai/gsm8k
PubmedQA	MIT	https://huggingface.co/datasets/qiaojin/PubMedQA
SciQ	CC-BY-NC 3.0	https://huggingface.co/datasets/allenai/sciq
TriviaQA	Apache 2.0	https://huggingface.co/datasets/mandarjoshi/trivia_qa
TruthfulQA	Apache 2.0	https://huggingface.co/datasets/domenicrosati/TruthfulQA
MMLU	MIT	https://huggingface.co/datasets/cais/mmlu
AI2 ARC	CC-BY-SA 4.0	https://huggingface.co/datasets/allenai/ai2_arc
CommonsenseQA	MIT	https://huggingface.co/datasets/tau/commonsense_qa
Llama-3-8B-Instruct	Llama 3 Community License	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
Qwen3	Apache 2.0	https://huggingface.co/collections/Qwen/qwen3
lm-evaluation-harness	MIT	https://github.com/EleutherAI/lm-evaluation-harness
BoTorch	MIT	https://github.com/meta-pytorch/botorch