Bayesian Optimization with Early Trial Termination for Speeding Up Parallel Neural Network Training

Apivich Hemachandra **Yizhan Han** **See-Kiong Ng** **Bryan Kian Hsiang Low**
Department of Computer Science
National University of Singapore

Abstract

Training of large neural networks (NNs) is often done in parallel on multiple GPUs. While existing parallel training frameworks easily allow NN training using multi-dimensional parallelism, the challenge remains in finding the optimal hyperparameters, such as the best balance between the sizes of various parallelism dimensions, which would result in the highest training throughput. Due to the large number of possible parallelism configurations (PCs) for a given training scenario, an exhaustive search over them is prohibitively costly. Existing PC optimization methods either require running NN training trials on a large number of PCs, each of which is costly, or approximate the throughput using only (incomplete) domain knowledge, which may be inaccurate and hardware-specific. To overcome these issues, this paper presents OPPA that can boost the efficiency of Bayesian optimization for optimizing the PC by novelly exploiting (a) the domain knowledge of parallel NN training via parallelism-informed prior beliefs that are general in catering to a variety of parallel NN training scenarios, and (b) early termination of trials involving suboptimal PCs. Despite incorporating these nontrivial efficiency tricks, OPPA is still theoretically guaranteed to achieve sublinear regret. We empirically show that OPPA finds optimal PCs more efficiently than existing methods for parallel training of NNs with varying architectures, training frameworks, and multi-GPU hardware setups.

1 Introduction

Modern advances in deep learning have arisen from the ability to scale neural networks (NNs) to larger sizes. In natural language processing, for example, transformer models [5, 35], large language models (LLMs) [23, 34], and multimodal LLMs [20, 24], which comprise millions to billions of parameters, have shown tremendous success in tasks such as text classification, text generation, and language understanding. These large NNs often cannot be trained on standard machines with a single processor. To scale up, it is necessary to distribute the NN training workload across a cluster of machines and parallelize the training process. Different parallelism techniques for NN training have been proposed, such as that of data [26, 43], pipeline [10, 22], tensor [31], and their combinations which are also referred to as *multi-dimensional parallelism* [17, 28, 31].

To fully utilize the given hardware for reducing the time of NN training, one can maximize its *throughput*, i.e., average number of training steps being run per unit time. The training throughput depends on the selected *parallelism configuration* (PC), which, in large-scale parallel NN training frameworks [13, 17, 28, 31], is specified by several hyperparameters such as the sizes of various parallelism dimensions. In practice, it is **challenging to accurately determine how the choice of PC affects the training throughput** since such a complex relationship depends on the NN architecture, training data, compute and communications hardware, and the exact implementation of the parallel NN training framework. *Existing works* that approximate (and optimize) the training throughput of a

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

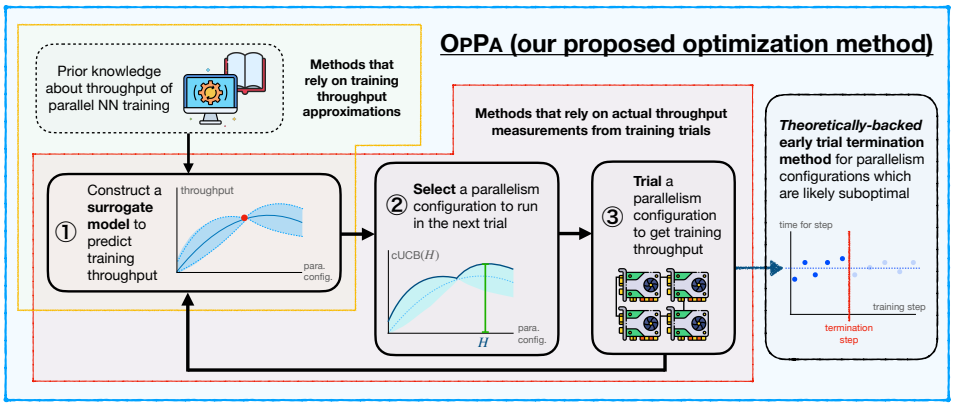


Figure 1: The key idea of OPPA is to exploit domain knowledge of parallel NN training & early trial termination to rapidly find the parallelism configuration (PC) with the highest training throughput.

PC [15, 41, 44] *require strong assumptions* on the hardware and parallel NN training implementation, and may not capture all the complex intricacies of parallel NN training. So, it may not be reliable to rely on any of these works alone to recover the optimal PC directly.

In practice, **the most reliable method to consider all possible factors and their intricate interactions during parallel training is to run real training trials** with each PC on the real hardware. Unfortunately, due to the large number of possible PCs, an exhaustive search over them is prohibitively costly. To circumvent this, existing parallel NN training frameworks have selected a subset of candidate PCs to trial using simplistic optimization algorithms, but *cannot utilize measured throughputs nor prior domain knowledge*, hence inefficiently needing excessive trials to find the optimal PC (Sec. 2).

To efficiently search for the PC with the highest throughput, it is desirable to be able to adaptively select promising PCs for trialing and filter out poor candidates using information from trialed PCs. Given these considerations, black-box optimization methods such as Bayesian optimization (BO) (Sec. 2) would be well-suited for the task. However, *a naive use of vanilla BO is still inefficient* as it does not exploit the inherent characteristics of PC trialing nor that of the parallel training process.

This paper presents the OPTIMIZER FOR PARALLELISM CONFIGURATIONS (OPPA) that can adaptively and rapidly optimize the PC for efficient parallel NN training. OPPA (Fig. 1) boosts the efficiency of BO by novelly exploiting *early termination of trials involving suboptimal PCs* and *parallelism-informed prior beliefs* that are general in catering to a variety of parallel NN training scenarios. Despite incorporating these nontrivial efficiency tricks, OPPA is still theoretically guaranteed to achieve sublinear regret. In Sec. 3, we first formulate the problem of finding the optimal PC as one of black-box optimization with black-box constraints. Then, we develop surrogate models representing *parallelism-informed prior beliefs* (based on domain knowledge of parallel NN training) that are general in catering to a variety of NN architectures, parallel NN training frameworks, and multi-GPU hardware setups (Sec. 4.1) and used by BO to select promising PCs for trialing (Sec. 4.2). We also describe the process of trialing a PC and *propose a novel technique to early terminate trials involving suboptimal PCs* with both theoretical and empirical justifications (Sec. 4.3). In Sec. 5, we empirically show that OPPA can be easily used on top of existing parallel NN training frameworks to find a good PC for training NNs more efficiently than existing methods and vanilla BO.

2 Background and Related Works

Parallel NN training on GPUs. A large NN can be effectively trained by distributing its workload across multiple GPUs. Different parallelism dimensions split the workload differently, which affects the amounts of computation per GPU, memory required in each GPU, and communication between GPUs. The most basic parallelism technique is data parallelism (DP) [16] where a batch of training data is split and distributed to each GPU to be separately processed by its local NN replica before gathering the gradients from all GPUs to update the NN weights. While DP is simple, it requires

replicating the NN onto each GPU, thus incurring additional memory. To resolve this, techniques, such as the Zero Redundancy Optimizer (ZeRO) [26] in the DeepSpeed package or Fully-Sharded Data Parallel (FSDP) [43], have been proposed to perform some sharding of NN weights or gradients to avoid full NN replication. Furthermore, tensor parallelism (TP) [2, 31] and pipeline parallelism (PP) [10, 22] have been proposed to shard the tensors in the NN and the NN execution pipelines onto multiple GPUs, respectively. The specific implementations of DP, TP, and PP, such as which tensors (e.g., NN weights, gradients) are sharded or how many shards they are partitioned into, are often manually specified by the user to control the overall throughput. App. A discusses DP, TP, and PP more. Besides, other types of parallelism (e.g., sequence [18], expert [27]) have also been developed.

Multi-dimensional parallelism. Several frameworks [17, 28, 31] have also been developed to combine different types of parallelism into the same training process. These frameworks provide simple interfaces for users to specify the desired *parallelism configuration* (PC) that comprises hyperparameters controlling the execution of the parallel training process, a subset of which specifies the sizes of various parallelism dimensions. These frameworks then automatically handle tensor sharding and execute the parallel training pipeline as per the specified PC. These frameworks may also manage training on multi-node or even heterogeneous hardware. While these frameworks allow practitioners to easily execute parallel NN training, *finding the PC with the highest throughput is challenging* since it depends non-trivially on the NN architecture, training data, GPU specifications, communication bandwidth between GPUs, and the exact implementation of the parallel NN training framework [17, 19, 36]. For example, DP is ineffective for large NNs since excessive NN replications can cause out-of-memory errors. Also, PP is less effective on smaller NNs as the communication costs between pipeline stages may dominate the computation of the fragmented pipeline.

Optimizing PC. The most accurate way to find the optimal PC would be to trial all possible PCs on the real training hardware and determine which one yields the highest training throughput. However, this is prohibitively costly since there can be a large number of possible PCs and trialing each PC can be computationally costly. To circumvent this, frameworks such as NeMo [13] and DeepSpeed [28] have implemented methods for automatic PC tuning[1] based on running NN training trials for a few training steps on a number of PCs. The PCs trialed are often either selected non-adaptively (e.g., based on random selection) or adaptively based on a simple surrogate function. However, these methods *cannot efficiently use the measured throughputs of trialed PCs* to perform informed optimization, and therefore still require a large number of NN training trials to obtain a good PC.

Since NN training trials are costly, we can consider constructing a surrogate model to approximate the throughputs of different PCs [15, 41, 44], which allows the use of domain knowledge to filter out suboptimal PCs and run fewer or even no trials at all. These methods, however, have *implicitly assumed that the surrogate model correctly predicts the actual training throughputs*. This is not possible in practice since the surrogate model cannot capture all the complex intricacies of parallel training. Furthermore, a fixed surrogate model cannot be easily extended to include new hyperparameters into the PC, which is important especially with the ever-growing parallel training literature.

Overview of BO. To efficiently search for the PC with highest throughput, we utilize BO [7, 8]. BO aims to maximize an expensive-to-query black-box function $\mathcal{R}$ (representing the throughput over the space of all possible PCs) whose derivative is unknown. The black-box function $\mathcal{R}$ is modeled as a realization of a Gaussian process (GP) whose prior belief is fully specified by its prior means and covariances [29]. Given the measured throughputs of trialed PCs, we can obtain a predictive distribution of the throughputs (of untrialed PCs) in the form of a GP posterior belief specified by its posterior means and covariances. Then, the BO algorithm selects a PC for trialing that maximizes a given acquisition function (e.g., expected improvement [11], upper confidence bound [33]) based on the GP posterior belief. Such an acquisition function trades off between exploring unique PCs that have not been queried to predict $\mathcal{R}$ better vs. exploiting PCs likely to have high values of $\mathcal{R}$ so as to efficiently find the optimal PC. App. B gives a more technical overview of GP modeling and BO.

BO is used in a wide range of black-box optimization problems, such as experimental design [14, 25] and material design [42]. More relevant to our work here, BO is also commonly used to optimize the NN architecture to achieve the best performance in a given task [32]. Finding the optimal PC, however, differs in two aspects: (a) The hyperparameters in a PC affecting the training throughput have better-defined mechanics (even if not completely known) that can be partially described based on

[1] https://docs.nvidia.com/nemo-framework/user-guide/latest/usingautoconfigurator.html, https://www.deepspeed.ai/tutorials/autotuning/

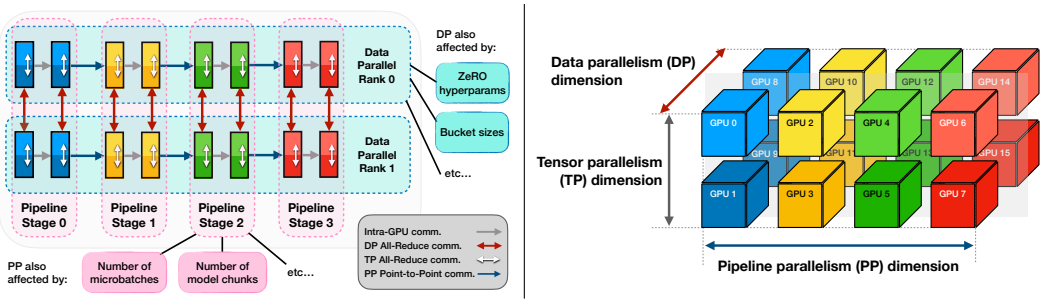


Figure 2: (Left) Visualization of a parallelism configuration (PC) with tunable hyperparameters. (Right) Visualization of GPU allocation for 3D parallelism according to the dimension sizes.

domain knowledge. Modeling via a GP allows us to incorporate this knowledge through an informed choice of the GP prior belief, which can reduce the number of trials required. (b) Trialing a PC allows for repeated measurements of the time taken per training step, which are used to calculate the throughput. Running many training steps is costly and may be unnecessary, so an early termination of trials can be adaptively applied to reduce the number of training steps needed to be run. We will elaborate more in the sections to follow.

3 Problem Setting

A parallelism configuration (PC) comprises several tunable hyperparameters found in typical parallel NN training frameworks and a subset of them determines the *sizes of various parallelism dimensions*, as visualized in Fig. 2. In our work here, we will mainly consider 3D parallelism where `dp`, `tp`, and `pp` denote the sizes of the data, tensor, and pipeline parallelism dimensions, respectively; note that their product $\texttt{dp} \cdot \texttt{tp} \cdot \texttt{pp}$ equals the number `n_gpus` of available GPUs. Also, as demonstrated in our experiments (Sec. 5), other types of parallelism can be included in the PC as well. The remaining hyperparameters determine the *specific implementations of the parallel NN training framework*, including (but not limited to) hyperparameters in the ZeRO optimizer, the number of model chunks for PP, whether communication overlaps are used, and whether gradient checkpointing is used. App. C.1 lists all of the hyperparameters considered in the PC for our problem, including additional parallelism dimensions that we consider in the experiments.

Let $\mathcal{H} \subset \mathbb{Z}^d$ be a set of all possible PCs such that each PC denotes a vector of hyperparameters in integers. The goal of our problem is to find the optimal PC $H^* \in \mathcal{H}$ with the *highest training throughput* (i.e., largest average number of training steps being run per unit time) and its *maximum memory usage not exceeding the available memory size* M_0 (i.e., fitting on the given GPUs). For any PC $H \in \mathcal{H}$, let $\mathcal{R}(H)$ and $\mathcal{M}(H)$ be the throughput and the maximum memory usage, respectively. Then, our problem of finding the optimal PC $H^* \in \mathcal{H}$ can be formulated as one of constrained maximization:

$$\max_{H \in \mathcal{H}} \mathcal{R}(H) \quad \text{subject to} \quad \mathcal{M}(H) \leq M_0 \,.$$

When querying a PC H, we run a short NN training trial (instead of training the NN till convergence) to obtain sufficiently accurate estimates of the throughput and maximum memory usage. To estimate the throughput of H, we measure the time taken $t_1, t_2, \ldots, t_{q_{\max}}$ for $q_{\max}$ consecutive training steps and use them to estimate the throughput: $\mathcal{R}(H) \approx \bar{r}_{q_{\max}} \triangleq q_{\max}^{-1} \sum_{j=1}^{q_{\max}} t_j^{-1}$. As shown in Fig. 3, the time taken for each training step fluctuates due to factors like additional overheads during process initialization, fluctuating communication speeds, and unpredictable system behaviors occurring during training. So, it is more reliable to estimate the throughput by averaging the time taken over multiple (rather than a few) training steps. While we can run the full trial with $q_{\max}$ training steps, we can also terminate the trial early after $q < q_{\max}$ steps at the cost of a higher variance of $\bar{r}_q$.

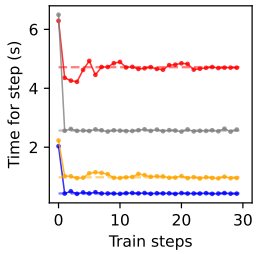


Figure 3: Time taken for each training step for a few training trials with various PCs. Each color represents a different PC used, and the dashed line represents the throughput estimate for that PC. Note the fluctuations in the time taken for each training step.

When designing our method, we note two other characteristics of the PC optimization problem:

A Since $\mathcal{R}$ and $\mathcal{M}$ depend on many factors and their intricate interactions that may be too complex to model or be exactly known in practice, **their exact forms are only partially known** given the incomplete domain knowledge. So, a suitable choice of the surrogate models for $\mathcal{R}$ and $\mathcal{M}$ has to **consider how the actual measurements may deviate from the domain knowledge**. Additionally, the surrogate models should *be able to quantify the confidence* of their predictions.

B Though a PC can be (and should be) trialed on real hardware, *running a full trial incurs a high cost*. This is especially true with suboptimal PCs since the same number of training steps on a suboptimal PC would incur more time. This motivates us to design an optimization algorithm such that *promising PCs are prioritized for trialing* while **PCs, which are likely to be suboptimal, are not trialed or are only trialed for fewer training steps and hence shorter time**.

4 Method

In this section, we describe OPPA, which exploits parallelism-informed GP prior beliefs and early trial termination to boost the performance of BO in optimizing the PC for parallel NN training. As summarized in Algorithm 1, OPPA alternates between 3 steps: ① modeling the training throughput and maximum memory usage based on measurements via *parallelism-informed GP prior beliefs*, ② selecting the next PC to trial via BO, and ③ running an NN training trial (with *early termination if suboptimal*) to obtain measurements for the (estimated) throughput and maximum memory usage of selected PC.

Algorithm 1 OPPA (shortened version of Algorithm 4)

Generate $\mathcal{H}$ of all valid PCs
for $i = 1, 2, \ldots$ **do**
  // Step ① – Modeling $\mathcal{R}$ and $\mathcal{M}$ (Sec. 4.1)
  Obtain GP predictive/posterior beliefs for $\mathcal{R}$ and $\mathcal{M}$ (17) from *parallelism-informed GP prior beliefs* using measurements of trialed PCs
  // Step ② – Selecting the next PC to trial (Sec. 4.2)
  Select $H_i \in \mathcal{H}$ with cUCB acquisition function (22)
  // Step ③ – Trialing the selected PC (Sec. 4.3)
  for training steps $q = 1, \ldots, q_{\max}$ **do**
    Measure time $t_{i,q}$ taken for training step q
    if $I_{i,q} = 0$ **then terminate trial** // from (2)
  if budget exhausted **then break**
return H_i with the best (estimated) measured throughput when its trial is not terminated early

4.1 Constructing Surrogate Models

In Step ①, surrogate models are constructed to predict the throughput and the maximum memory usage. As suggested in A, to explicitly capture the deviations from the domain knowledge, we decompose the actual throughput $\mathcal{R}$ and maximum memory usage $\mathcal{M}$ into

$$\mathcal{R}(H) = \widehat{\mathcal{R}}(H; \theta_{\mathcal{R}}) + f_{\mathcal{R}}(H) \qquad \text{and} \qquad \mathcal{M}(H) = \widehat{\mathcal{M}}(H; \theta_{\mathcal{M}}) + f_{\mathcal{M}}(H) \tag{1}$$

where the functions $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ (with parameters $\theta_{\mathcal{R}}$ and $\theta_{\mathcal{M}}$) capture the domain knowledge of parallel NN training, and $f_{\mathcal{R}}$ and $f_{\mathcal{M}}$ capture the unknown deviations from the domain knowledge.

Domain knowledge. The functions $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ aim to, respectively, *predict* $\mathcal{R}$ and $\mathcal{M}$ using only the domain knowledge of parallel NN training. Hence, *they are not expected to give completely accurate predictions but instead capture general trends in any given NN training scenario*; any factors not accounted for would be captured in the unknown deviation terms anyway, which will be described below. In practice, OPPA allows the predictions by $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ to be improved through incorporating additional domain knowledge from practitioners with more expertise in parallel NN training. Consequently, OPPA would be able to find the optimal PC more rapidly.

For OPPA, we provide default choices of $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ by considering parallel NN training under general, idealized conditions. Doing so allows $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ to be general enough to immediately cater to parallel NN training scenarios of varying model, data, hardware, and network configurations, while also having a simple analytical form that is cheap to compute. For $\widehat{\mathcal{R}}$, we consider the time taken per training step for the computation $\widehat{\mathcal{T}}_{\text{comp}}$ and for the communication $\widehat{\mathcal{T}}_{\text{comm}}$, which can be combined to approximate the throughput: $\widehat{\mathcal{R}}(H; \theta_{\mathcal{R}}) = \left[\widehat{\mathcal{T}}_{\text{comp}}(H; t_{\text{comp}}) + \widehat{\mathcal{T}}_{\text{comm}}(H; \mathbf{C})\right]^{-1}$ where $\theta_{\mathcal{R}} = \{t_{\text{comp}}, \mathbf{C}\}$ are *learnable parameters* (as opposed to fixed constants) that aim to estimate factors such as the approximate communication speed between GPUs or the communication load required by each type of parallelism. For $\widehat{\mathcal{T}}_{\text{comp}}$, we consider the additional computation time arising from the pipeline bubble in PP [22]. For $\widehat{\mathcal{T}}_{\text{comm}}$, inspired by [38], we consider an idealized training scenario

visualized in Fig. 2, which assumes the NN to contain roughly identical blocks, and considers the communication from the All-Reduce operations involved in DP and TP, and point-to-point communications involved in PP. We separately model the intra- and inter-node communications, collapsing the constants for specific communication types into $\mathbf{C}$. Further details for the construction of $\widehat{\mathcal{R}}$ are in App. D.1. For $\widehat{\mathcal{M}}$, we consider the memory required per GPU to store the NN weights and gradient values for backpropagation, and learnable parameters to estimate the overall memory required by them, as detailed in App. D.2. In our experiments (Sec. 5), we will show that our choices of $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ enable the GP prior beliefs to capture the general trends of the throughput and maximum memory usage sufficiently well, especially when combined with the learned unknown deviations, while not needing much additional computational costs during the PC optimization process.

Unknown deviations from domain knowledge. Due to incomplete domain knowledge to fully describe a parallel NN training process, we learn the deviations $f_{\mathcal{R}}$ and $f_{\mathcal{M}}$ of the actual measurements from the domain knowledge using real NN training trials by modeling them as realizations of GPs. The benefits of using a GP are twofold: (a) A GP is typically expressive enough to model unknown functions without an analytical form. (b) A GP can quantify the uncertainty of its predictions, allowing the surrogate model to determine how much it does not know about the throughput of any PC and subsequently be exploited by OPPA to potentially trial PCs with more uncertain throughput.

Specifically, we model $f_{\mathcal{R}}$ and $f_{\mathcal{M}}$ as realizations of GPs "centered" around zero prior means and with prior covariances defined using the Matérn kernel [29] where the covariance/correlation of any two PCs depends on the Euclidean distance between their corresponding vectors. The kernel is characterized by smoothness $\nu = 5/2$ and lengthscale and signal variance hyperparameters $\theta_k = \{\ell, \sigma_k\}$ that can be learned using the actual measurements, as described below. In App. D.3, we give the expression for the Matérn kernel and justify its use in our work here.

The additive decomposition of the domain knowledge and the unknown deviations in (1) implies that $\mathcal{R}$ and $\mathcal{M}$ are samples from **parallelism-informed GP prior beliefs** given by

$$\mathcal{R} \sim \mathcal{GP}\left(\widehat{\mathcal{R}}(\cdot\,;\,\theta_{\mathcal{R}}), k(\cdot,\cdot\,;\,\theta_k)\right) \qquad \text{and} \qquad \mathcal{M} \sim \mathcal{GP}\left(\widehat{\mathcal{M}}(\cdot\,;\,\theta_{\mathcal{M}}), k(\cdot,\cdot\,;\,\theta_k)\right).$$

Using the measurements of trialed PCs, the values of parameters $\{\theta_{\mathcal{R}}, \theta_{\mathcal{M}}\}$ in $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ and Matérn kernel hyperparameters θ_k can be estimated by maximizing the log marginal likelihood [29], as described in (16) in App. D.4, and the GP posterior beliefs for the throughput and maximum memory usage of any PC $H \in \mathcal{H}$ can be obtained from the prior beliefs, as stated in (17) to (21) in App. D.4.

4.2 Selecting the Next PC to Trial

In Step ②, the next PC $H_i \in \mathcal{H}$ to trial in round i is selected to maximize the constrained upper confidence bound (cUCB) acquisition function [33, 37] based on the surrogate models constructed in ①, as detailed further in App. E.1. cUCB trades off between *exploring* untrialed PCs to improve the accuracy of the surrogate models for $\mathcal{R}$ and $\mathcal{M}$ vs. *exploiting* promising PCs likely to have high throughputs so as to guide the PC optimization process to efficiently find the PC with the highest training throughput [8, 11], hence satisfying the requirement in B.

4.3 Trialing the Selected PC with Early Termination If Suboptimal

Finally, in Step ③, an NN training trial is run on the PC H_i selected in Step ② for q_{max} training steps, each of which takes time $t_{i,q}$ for $q = 1, \ldots, q_{\text{max}}$. These time measurements are then used to obtain an estimate $\bar{r}_{i,q_{\text{max}}}$ of the actual throughput $\mathcal{R}(H_i)$ where $\bar{r}_{i,q} = q^{-1}(t_{i,1}^{-1} + \ldots + t_{i,q}^{-1})$ and whose variance $\sigma^2_{\bar{r}_{i,q}}$ is defined in App. F.1. In practice, as shown earlier in Fig. 3, the actual time measurements for a sequence of training steps may contain outlier(s) that can skew a throughput estimate. As discussed in App. F.2, the estimated throughput is made more accurate by removing the outlier values of the time measurements. Meanwhile, the maximum memory usage m_i across all training steps is retrieved by calling `torch.cuda.max_memory_allocated()`.

Early trial termination. In practice, some PCs do not need to be trialed for the full q_{max} steps since fewer training steps might be sufficient to determine that the PC is suboptimal. This is because $\bar{r}_{i,q}$ is an estimator of the actual throughput $\mathcal{R}(H_i)$ whose variance $\sigma^2_{\bar{r}_{i,q}}$ decreases as q increases. Intuitively, this implies after a certain number of steps, $\bar{r}_{i,q}$ can estimate $\mathcal{R}(H_i)$ with a small enough variance for us to determine that H_i would not improve upon the best PC found so far.

To save time on such trials, we consider early trial termination where an NN training trial only continues if the estimated throughput meets some threshold. Formally, define an indicator variable

$$I_{i,q} = \mathbb{1}\left[(q \leq q_{\text{min}}) \vee \left(\bar{r}_{i,q} \geq \max_{l \in \{1,\ldots,i-1\}} \bar{r}_{l,\hat{q}_l} + \tau_q\right)\right]. \tag{2}$$

That is, $I_{i,q} = 1$ if (i) not more than q_{min} training steps have been run, or (ii) the estimated throughput of H_i is higher than any of that obtained in the previous steps by some margin $\tau_q > 0$. We continue to run the NN training trial at step q if $I_{i,q-1} = 1$, and terminate the trial at step $q = \hat{q}_i$ if $I_{i,\hat{q}_i} = 0$ (i.e., when H_i is unlikely to improve upon the best PC found so far). As explained in App. F.3.1, this intuitively corresponds to when more training steps are unlikely to provide further information about the optimal PC. This saves computation time in practice while still allowing OPPA to recover the optimal PC, as formalized in the result below:

Theorem 4.1 (Informally stated in terms of $\mathcal{R}$)**.** *Consider the practical setting of $\mathcal{M}(H) \leq M_0$ for all $H \in \mathcal{H}$. Let N be the number of PCs trialed so far. Under mild conditions on $\mathcal{R}$ and the time measurements $t_{i,q}$, there exists some choice of $\{\tau_q\}_{q=1}^{q_{\text{max}}}$ such that with high probability, the cumulative regret is $\sum_{i=1}^{N}\left(\mathcal{R}(H^*) - \mathcal{R}(H_i)\right) = \widetilde{\mathcal{O}}\left(\sqrt{N/q_{\text{min}}}\right)$, and for $i = 2,\ldots,N$, if $\mathcal{R}(H_i) < \max_{k \in \{1,\ldots,i-1\}} \mathcal{R}(H_k)$, then $\hat{q}_i < q_{\text{max}}$.*

In App. F.3.2, we prove a more general version of Thm. 4.1 such that $\mathcal{R}$ can instead be any function with a bounded RKHS norm, and provide an empirical verification for early trial termination. Thm. 4.1 considers the practical setting of $\mathcal{M}(H) \leq M_0$ for all $H \in \mathcal{H}$ because running the NN training trial on the PC H when $\mathcal{M}(H) > M_0$ would lead to an out-of-memory error and hence the throughput cannot be measured; so, our theoretical analysis can omit the case of $\mathcal{M}(H) > M_0$ for convenience. Despite incorporating early trial termination to gain efficiency in practice, Thm. 4.1 guarantees that OPPA can still achieve sublinear regret, that is, it can recover the optimal PC with the highest training throughput. Thm. 4.1 also reveals that PCs whose throughput is lower than that of PCs already trialed would likely have their training trials terminated early, hence allowing OPPA to save time and compute resources from trialing suboptimal PCs, as mentioned in B. App. G gives the complete pseudocode for OPPA that incorporates steps ①, ②, and ③.

5 Experiments and Discussion

This section evaluates the performance of OPPA in finding the optimal PC for parallel training of NNs with varying architectures, training frameworks, and multi-GPU hardware setups. App. H.1 discusses these various parallel NN training scenarios in detail. The performance of OPPA is compared against that of RANDOM (i.e., random PC selection), XGBOOST (i.e., *adaptive* PC selection based on the XGBOOST surrogate model [3] that is used by DEEPSPEED [28]), R&M-HAT (i.e., 1-shot selection of best PC based on surrogate models $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$), and VANILLA-BO (i.e., BO without modifications). App. H.2 describes these methods in detail, including how the surrogate models are trained.

To evaluate their performances, we plot the best (estimated) measured throughput (i.e., average number of training steps being run per second) achieved by each method (Sec. 4.3) vs. its incurred time (rather than vs. the number of PCs it has trialed). We do so since the time incurred by a method better reflects its cost of optimizing the PC for parallel NN training by capturing both the time needed to perform the optimization/selection and the time taken to run the NN training trials. The latter includes the time taken to initialize the NN training, run the training steps, and switch to the NN training trial of each subsequent selected PC, and is longer if the method trials more PCs or suboptimal PCs. Nonetheless, we give plots for the best (estimated) measured throughput vs. number of trialed PCs in App. I.1. The NN loss is independent of the selected PC and hence not reported.

Single-node setups. Fig. 4a shows results for finding the optimal PC for parallel training of the BERT model [5] on a single node of 8 GPUs using the COLOSSAL-AI framework [17]. It can be observed that OPPA, which exploits parallelism-informed GP prior beliefs and early trial termination, outperforms all other tested methods, including vanilla BO. We find that OPPA automatically prioritizes PCs with only DP, which matches our intuition of DP being adequate for smaller NNs. Fig. 4b shows similar results for the Qwen model [39] where OPPA again finds a better PC than all other methods. As the Qwen model is larger, OPPA now prefers a mix of DP and PP to reduce memory use while incurring minimal communications across GPUs. App. I.2 shows the PCs selected by OPPA.

To visualize the efficiency gains of OPPA, Fig. 5a shows that OPPA is able to trial many more PCs in a given time period compared to the other methods due to the early termination of suboptimal trials

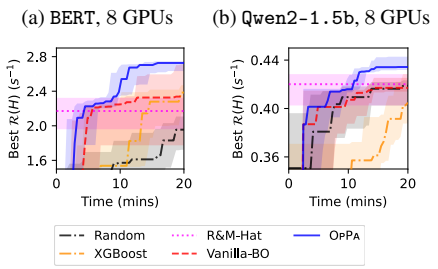


Figure 4: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across 10 repeated experiments with error bands showing lower and upper quartiles) vs. its incurred time for parallel NN training on a single-node setup using the COLOSSAL-AI framework [17].

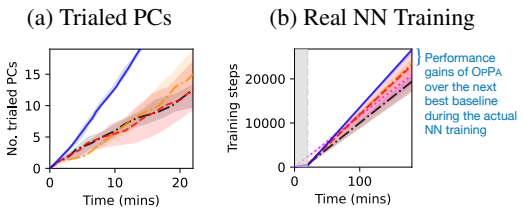


Figure 5: Comparing the efficiency of OPPA with that of other tested methods based on the experiments in Fig. 4a. Fig. 5a shows the no. of PCs each method has trialed. Fig. 5b shows the no. of training steps that can be run by each method during PC optimization (i.e., shaded gray region on the left) and in the subsequent parallel NN training using the optimized/selected PC (unshaded region on the right). See Fig. 4 for the legend.

to avoid wasting time: Though OPPA switches from one NN training trial to the next more often, this switching cost is less than the time saved by not running all training trials in full, allowing OPPA to trial more PCs overall. Along with the parallelism-informed GP prior beliefs to efficiently filter out suboptimal PCs, OPPA returns a PC with a higher throughput, which in turn allows more training steps to be run in the subsequent parallel NN training even after just 20 minutes of PC optimization, as shown in Fig. 5b. This demonstrates the effectiveness of OPPA in speeding up parallel NN training.

Generality in catering to various NN architectures and parallel training frameworks. In App. I.3, we report results on similar experiments as that in Fig. 4, except that the parallel NN training is performed using the NEMO framework [13] where the size of the sequence parallelism dimension [18] is included in the PC to be optimized. In App. I.4, we report results comparing the performances of OPPA and the other tested methods in optimizing the PC for parallel training of vision transformers and mixture of experts (MoE); the latter includes the size of expert parallelism dimension [27] in the PC. In these parallel NN training scenarios, OPPA outperforms the other methods, demonstrating that its surrogate models (Sec. 4.1) *do not overfit to a specific scenario*, but instead can cater to a variety of NN architectures, parallel training frameworks, and even unseen hyperparameters in a PC.

Accuracy of surrogate models. Fig. 6a compares the throughputs predicted by OPPA's surrogate model (in Step ① of Algorithm 1) with the actual measured throughputs of trialed and (randomly sampled) untrialed candidate PCs based on the experiments in Fig. 4b. It can be observed that the predictions correlate well with the measured throughputs, especially in the space of PCs with high throughputs that are trialed more often, which allows OPPA to better identify the optimal PC. In contrast, R&M-HAT's surrogate model $\widehat{R}$ (Fig. 6b) can only capture general trends of actual throughput $\mathcal{R}$ but not all the complex intricacies of parallel NN training, while VANILLA-BO's GP surrogate model (i.e., a GP posterior belief) is not informed by the prior domain knowledge of parallel NN training (Fig. 6c). As a result, their surrogate models cannot predict $\mathcal{R}$ well in the space of PCs with high throughput and hence cannot identify the optimal PC well. App. I.5 further discusses the accuracy of these surrogate models for $\mathcal{R}$ and $\mathcal{M}$ across various parallel NN training scenarios.

Effect of each efficiency trick in OPPA. Fig. 7 shows results from our ablation studies to isolate the effect of each proposed efficiency trick on OPPA's performance. It can be observed that without early trial termination, BO would incur more time in suboptimal trials and thus in finding the optimal PC. Without the parallelism-informed GP prior beliefs, BO would be less informed about promising PCs for trialing and hence incur more time to find the optimal PC. App. I.7 gives more results. App. I.8 shows the effect of $q_{\min}$ (2) on OPPA's performance, revealing minimal degradation for a small $q_{\min}$.

Multi-node setups. We also tested OPPA in optimizing the PC on multi-node setups. In these cases, the additional communication costs makes the throughput computation less straightforward, while larger number of feasible PCs complicates the optimization problem. In Fig. 8a, we show the results for optimizing the PC for training a smaller NN [9] with 1 billion parameters on a commodity cluster with 16 GPUs, which is a typical setup found by practitioners with existing hardware in practice. Meanwhile, Figs. 8b and 8c are the results for optimizing the PC to train larger NNs [34, 40] on

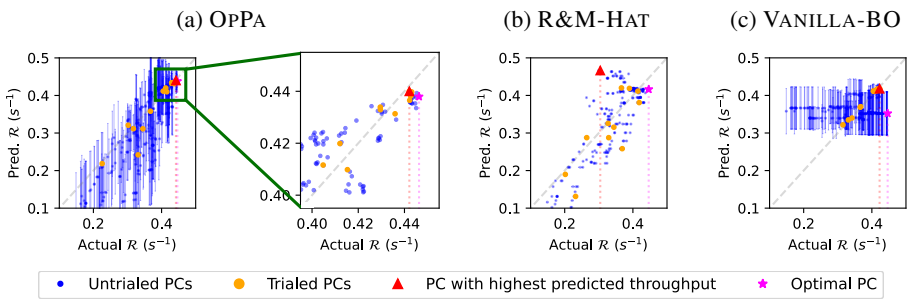


Figure 6: Graphs of throughputs predicted by different methods' surrogate models vs. actual measured throughputs after 20 NN training trials based on an experiment in Fig. 4b. The zoomed-in region (green box) shows PCs with high predicted and measured throughputs. Each untrialed PC in Figs. 6a and 6c is associated with error bars (i.e., omitted from zoomed-in region to ease interpretation) showing twice the GP posterior std. dev., which are not in Fig. 6b as $\widehat{R}$ gives deterministic predictions.

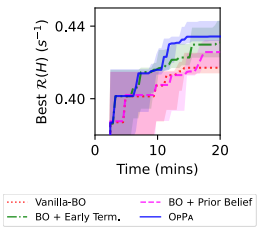


Figure 7: Effects of early trial termination and parallelism-informed GP prior beliefs on the ability of OPPA to optimize the throughput for `Qwen2-1.5b` training.

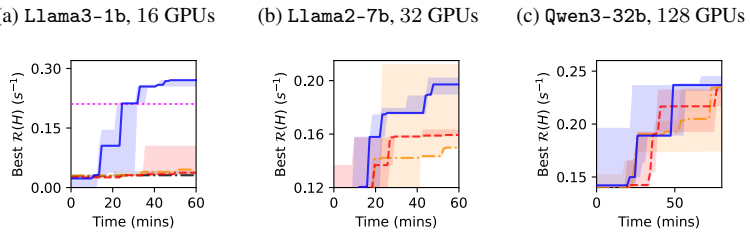


Figure 8: Graphs of best (estimated) measured throughput (higher is better) achieved by each method vs. its incurred time for parallel NN training on a multi-node setup using (a, b) COLOSSAL-AI [17] and (c) NEMO [13] with the sizes of sequence and context parallelism dimensions also being tunable hyperparameters in the PC. See Fig. 4 for the legend. Some methods were not run for results in Figs. 8b and 8c due to the available computational budget.

high-performance computing clusters with 32 and 128 GPUs, respectively. In all of these cases, OPPA outperforms other baselines, and also does so more consistently (i.e., smaller variance in the resulting throughput), demonstrating its robustness. This trend is observed across different scales of hardware setup, and even across parallel NN training frameworks or when including new parallelism dimensions in the PC (e.g., the experiment for Fig. 8c which includes six types of parallelism). We also see that vanilla BO outperforms other non-BO methods due to its ability to balance exploration and exploitation, however remaining worse than OPPA.

In Fig. 9, we also demonstrate that OPPA outperforms other methods that do not use training trials (and instead use a more complex approximation of training throughput compared to $\widehat{\mathcal{R}}$) such as AMP [15] and NNSCALER [19]. This shows the superiority of our adaptive approach which can effectively incorporate the measured throughput from real NN training trials into existing domain knowledge, and the non-trivial efficiency tricks exploited by OPPA that allow for this boost in performance.

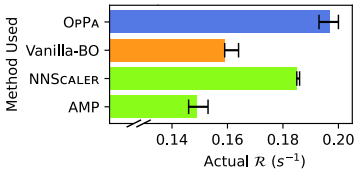


Figure 9: Obtained $\mathcal{R}$ from OPPA compared to AMP [15] and NNSCALER [19] for `Llama2-7b` training with 32 GPUs. The results reported for BO and OPPA are the same from Fig. 8b.

6 Conclusion

In this paper, we have described OPPA, which improves upon constrained BO by exploiting parallelism-informed GP prior beliefs and an early trial termination mechanism to efficiently optimize the parallelism configuration and achieve the highest training throughput across a vari-

ety of NN architectures, parallel NN training frameworks, and multi-GPU hardware setups. As demonstrated, OPPA can also be easily adapted to optimize other unseen hyperparameters in a PC as well. We think that the parallelism-informed GP prior beliefs can be embedded with more domain knowledge of specific parallel NN training frameworks or knowledge of specific NN architectures, which would boost the performance of OPPA further.

Acknowledgments

The computational work for this article was partially performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg). This project was supported by the the Swiss AI Initiative through a grant from the Swiss National Supercomputing Centre (CSCS) under project ID a139 on Alps.

References

[1] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Proc. NeurIPS*, 2020.

[2] Z. Bian, Q. Xu, B. Wang, and Y. You. Maximizing Parallelism in Distributed Training for Huge Neural Networks. 2021. arXiv:2105.14450.

[3] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proc. SIGKDD*, 2016.

[4] J. Couto. Kernel k-means for categorical data. In *Proc. IDA*, 2005.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*, 2019.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. ICLR*, 2021.

[7] P. I. Frazier. A Tutorial on Bayesian Optimization. 2018. arXiv:1807.02811.

[8] M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Proc. UAI*, 2014.

[9] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang,

S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma. The Llama 3 Herd of Models. 2024.

[10] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. GPipe: efficient training of giant neural networks using pipeline parallelism. In *Proc. NeurIPS*, 2019.

[11] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[12] J. Kirschner and A. Krause. Information Directed Sampling and Bandits with Heteroscedastic Noise. In *Proc. COLT*, 2018.

[13] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, P. Castonguay, M. Popova, J. Huang, and J. M. Cohen. NeMo: a toolkit for building AI applications using Neural Modules. 2019. arXiv:1909.09577.

[14] B. Lei, T. Q. Kirk, A. Bhattacharya, D. Pati, X. Qian, R. Arroyave, and B. K. Mallick. Bayesian optimization with adaptive surrogate models for automated experimental design. *npj Computational Materials*, 7(1):1–12, Dec. 2021.

[15] D. Li, H. Wang, E. Xing, and H. Zhang. AMP: automatically finding model parallel strategies with heterogeneity awareness. In *Proc. NeurIPS*, 2022.

[16] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. PyTorch distributed: experiences on accelerating data parallel training. In *Proc. VLDB*, 2020.

[17] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You. Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training. In *Proc. ICPP*, 2023.

[18] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You. Sequence parallelism: Long sequence training from system perspective. In *Proc. ACL*, 2023.

[19] Z. Lin, Y. Miao, Q. Zhang, F. Yang, Y. Zhu, C. Li, S. Maleki, X. Cao, N. Shang, Y. Yang, W. Xu, M. Yang, L. Zhang, and L. Zhou. nnScaler: constraint-guided parallelization plan generation for deep learning training. In *Proc. OSDI*, 2024.

[20] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. In *Proc. NeurIPS*, 2023.

[21] A. Makarova, I. Usmanova, I. Bogunovic, and A. Krause. Risk-averse Heteroscedastic Bayesian Optimization. In *Proc. NeurIPS*, 2021.

[22] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. PipeDream: generalized pipeline parallelism for DNN training. In *Proc. SOSP*, 2019.

[23] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, L. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, L. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. J. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong,

L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. GPT-4 Technical Report. 2024. arXiv:2303.08774.

[24] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proc. ICML*, 2021.

[25] T. Rainforth, A. Foster, D. R. Ivanova, and F. B. Smith. Modern Bayesian Experimental Design. 2023. arXiv:2302.14545.

[26] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. ZeRO: memory optimizations toward training trillion parameter models. In *Proc. SC*, 2020.

[27] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *Proc. ICML*, 2022.

[28] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proc. SIGKDD*, 2020.

[29] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 978-0-262-18253-9.

[30] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. ICLR*, 2017.

[31] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. 2020. arXiv:1909.08053.

[32] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. NeurIPS*, 2012.

[33] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *IEEE Transactions on Information Theory*, 58:3250–3265, 2012.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models. 2023.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All you Need. In *Proc. NeurIPS*, 2017.

[36] M. Wagenländer, G. Li, B. Zhao, L. Mai, and P. Pietzuch. Tenplex: Dynamic Parallelism for Deep Learning using Parallelizable Tensor Collections. In *Proc. SOSP*, 2024.

[37] J. T. Wilson, R. Moriconi, F. Hutter, and M. P. Deisenroth. The reparameterization trick for acquisition functions. 2017.

[38] D. Xiong, L. Chen, Y. Jiang, D. Li, S. Wang, and S. Wang. Revisiting the Time Cost Model of AllReduce. 2024. arXiv:2409.04202.

[39] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan. Qwen2 Technical Report. 2024.

[40] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu. Qwen3 Technical Report. 2025.

[41] S. Zhang, L. Diao, C. Wu, Z. Cao, S. Wang, and W. Lin. HAP: SPMD DNN Training on Heterogeneous GPU Clusters with Automated Program Synthesis. In *Proc. EuroSys*, 2024.

[42] Y. Zhang, D. W. Apley, and W. Chen. Bayesian Optimization for Materials Design with Mixed Quantitative and Qualitative Variables. *Scientific Reports*, (1):4924, 2020.

[43] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. In *Proc. VLDB*, 2023.

[44] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, J. E. Gonzalez, and I. Stoica. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *Proc. OSDI*, 2022.

A Additional Discussion on Types of Parallelism

In this section, we provide additional details on the types of parallelism that can be deployed for parallel training of NNs.

Data parallelism. The most basic type of parallelism technique is data parallelism (DP) [16] where a batch of training data is split and distributed to each GPU. Each batch is then fed into the local replicas of the NNs, before the gradients from each GPU are synchronized. While simple and often the fastest, the naive DP approach requires replicating the model on each GPU, which takes up additional memory on each GPU. Several techniques have since been proposed to perform DP with sharded models, including the Zero Redundancy Optimizer (ZERO) introduced by [26] in the DEEPSPEED package, and Fully-Sharded Data Parallel (FSDP) introduced by [43]. While these techniques reduces the maximum memory usage for DP, their effectiveness can still heavily depend on the choice of hyperparameters. For example, ZERO can be tuned so that one or more of the optimizer states, the gradients or the NN weights are sharded between GPUs. The choice of sharded items may reduce the memory usage on each GPU at the cost of increased communication between GPU which in turn reduces the throughput of the NN training.

Tensor parallelism. Another parallelism technique is tensor parallelism (TP) where individual tensors of the NN weights are sharded across multiple GPUs, so that the matrix multiplication operations are instead done in a distributed manner, allowing operations involving larger tensors to be possible. TP initially involved sharding a tensor along a single dimension (i.e., either row-wise or column-wise sharding) [31], however has since also incorporated sharding tensors across multiple dimensions as well [2].

Pipeline parallelism. In pipeline parallelism (PP) [10, 22] we instead partition the NN along its execution pipeline with each NN partition running synchronously with microbatches of training data. The gradients are accumulated for each microbatch and updated at the end of each training step. By sharding the NN and training data into smaller chunks, the GPU memory required at any one time becomes lower, allowing for the training of larger NNs at the cost of more sequential operation rounds and higher cost of communication between GPUs. The tradeoff between training speed and maximum memory usage can be further controlled based on the size of microbatches and the number of model chunks.

B Technical Primer on Gaussian Processes and Bayesian Optimization

In this section, we provide a technical overview of Gaussian process (GP) regression and on Bayesian optimization (BO). The contents are adapted from [7, 29].

A Gaussian process (GP) $\mathcal{GP}(\mu_{\text{prior}}, k)$ with prior mean μ_{prior} and prior covariance k is a random process where for any subset of input $\mathbf{X}$, its corresponding output is given by a normal distribution $f(\mathbf{X}) \sim \mathcal{N}\big(\mu_{\text{prior}}(\mathbf{X}), k(\mathbf{X}, \mathbf{X})\big)$. The prior mean $\mu_{\text{prior}}(x)$ describes the expected value of the random function $f(x)$ at a certain input, while the prior covariance $k(x, x')$ roughly captures the covariance between between $f(x)$ and $f(x')$.

Suppose that we have an unknown function f drawn from the GP. Assume we are given a set $\boldsymbol{D} = (\mathbf{X}, \mathbf{y}) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of inputs and corresponding measurements, where $y_i = f(x_i) + \epsilon_i$ are noisy measurements of the actual function with Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \lambda_i)$. Then, when performing Bayesian inference, we can express the GP posterior mean and covariance as

$$\mu(x) = \mu_{\text{prior}}(x) + k(x, \mathbf{X})\big(k(\mathbf{X}) + \text{diag}(\boldsymbol{\lambda})\big)^{-1}(\mathbf{y} - \mu_{\text{prior}}(x)) \,, \tag{3}$$

$$\sigma^2(x) = k(x, x) - k(x, \mathbf{X})\big(k(\mathbf{X}) + \text{diag}(\boldsymbol{\lambda})\big)^{-1}k(\mathbf{X}, x) \tag{4}$$

where $\text{diag}(\boldsymbol{\lambda})$ is a diagonal matrix with values $\lambda_1, \ldots, \lambda_n$ on the diagonals.

In practice, the prior mean and covariance may have hyperparameters θ, which specify the prior assumption of the functions drawn from the random process. For example, many kernels representing the prior covariance include lengthscale as a hyperparameter, which govern how correlated the function output is when a certain input dimension is further away. One method to find the optimal kernel hyperparameters is by finding the hyperparameters which maximize the marginal log-likelihood $\log p(\boldsymbol{D}|\theta)$ of the data $\boldsymbol{D}$.

In Bayesian optimization (BO), the goal is to find the maxima of the unknown function f, where the function is assumed to be black-box with no analytical form. To do so, we can learn more about f by querying it at different inputs, and perform Bayesian inference to update our belief on the unknown function.

Given the measurements $\boldsymbol{D}_t = (\mathbf{X}_t, \mathbf{y}_t) = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ in round t of data selection, GP regression can be performed to obtain a posterior mean μ_t and variance σ_t^2. The next input x_{t+1} to query can be selected as the input which maximizes some acquisition function. Examples of such acquisition function include the expected improvement [11]

$$\mathrm{EI}_t(x) = \mathbb{E}_{y' \sim \mathcal{N}(\mu_{t-1}(x), \sigma_{t-1}^2(x))}\big[\max(0, y' - \max_{y \in \mathbf{y}_{t-1}} y)\big]$$

or the upper confidence bound [33]

$$\mathrm{UCB}_t(x) = \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$$

where $\beta_t > 0$ is a constant that may vary with t. In all of these acquisition functions, a tradeoff is performed between selecting inputs that the GP is uncertain about (i.e., with high $\sigma_{t-1}^2(x)$) to learn more about those unknown region, and selecting inputs in regions where the function value is known to be higher (i.e., with high $\mu_{t-1}(x)$).

C Additional Discussion on Problem Setting

C.1 Hyperparameters Considered

In Table 1, we list several hyperparameters which we include in the parallelism configuration (PC) and the range of the values. Note that the hyperparameters are constrained to valid PC; for example, we ensure that $\texttt{dp} \cdot \texttt{tp} \cdot \texttt{pp} \cdot \texttt{cp} = \texttt{n\_gpus}$ to ensure that each dimension do not exceed number of GPUs. In our experiments, this is done by generating all the feasible PCs before starting the optimization. Some hyperparameters are also set to their default value when not in use; for example, if $\texttt{pp} = 1$ (i.e., no PP used) then we restrict $\texttt{mb} = \texttt{mc} = 1$ such that PCs are not duplicated. In the code, we generate all possible PCs beforehand so we can ensure that all PCs chosen will be valid according to the constraints.

Table 1: Tunable hyperparameters in a parallelism configuration (PC).

Hyperparameter	Description	Feasible Values	Notes
DP size (`dp`)	Data parallelism degree	[1, `n_gpus`]	
TP size (`tp`)	Tensor parallelism degree	[1, `n_gpus`]	Requires $\texttt{dp} \cdot \texttt{tp} \cdot \texttt{pp} \cdot \texttt{cp} = \texttt{n\_gpus}$
PP size (`pp`)	Pipeline parallelism degree	[1, `n_gpus`]	
CP size	Context parallelism degree	[1, `n_gpus`]	Only NeMo and only for some experiments
SP size	Sequence parallelism degree	[1, `n_gpus`]	Only NeMo and must be $\leq$ `tp`
EP size	Expert parallelism degree	[1, `n_gpus`]	Only NeMo and only for MoE
DP bucket size	Size for gradient reduction buckets (MB)	[1, 4096]	Only Colossal-AI
ZeRO stage	ZeRO stage used	[0, 3]	For Colossal-AI, must be ≤ 2 For NeMo, this controls FSDP sharding instead
ZeRO bucket size	Bucket size for ZeRO communication	[1, 4096]	
Overlap ZeRO communication	Whether to overlap ZeRO communication	True / False	
Overlap DP AllGather	Whether to overlap AllGather	True / False	
# microbatches (`mb`)	Number of microbatches per forward pass	$\leq$ batch size	Only considered for Colossal-AI
# model chunks (`mc`)	Number of model chunks for pipelining	$\leq$ # transformer blocks	
Overlap P2P for PP	Overlap PP communication or not	True / False	
Grad. checkpointing	Whether gradient checkpointing is enabled	True / False	

C.2 Throughput vs. Time Per Training Step

We explain why we choose to maximize throughput instead of minimizing time per training step.

As an example, suppose we consider three PCs H, H', H'' where the times per training step are given by $\mathcal{T}(H) = 0.3$, $\mathcal{T}(H') = 0.4$, and $\mathcal{T}(H'') = 0.5$. In this case, H would be the best PC out of the three. We see here the gap of the time per training step between H and H' is $\mathcal{T}(H') - \mathcal{T}(H) = 0.1$, and the same gap size for H' and H'' of $\mathcal{T}(H'') - \mathcal{T}(H') = 0.1$. Meanwhile, the gap between the throughput of the two PCs would be $\mathcal{R}(H) - \mathcal{R}(H') = \mathcal{T}(H)^{-1} - \mathcal{T}(H')^{-1} = 3.\bar{3} - 2.5 = 0.8\bar{3}$ and $\mathcal{R}(H') - \mathcal{R}(H'') = \mathcal{T}(H)^{-1} - \mathcal{T}(H')^{-1} = 2.5 - 2 = 0.5$. We can see that the gap between the best PC becomes enhanced when we consider the throughput, when we compare it with the relative gap size of the training step time.

More concretely, if we have a PC which requires time t per training step and another PC which requires Δt less time per training step, then the throughput would be different by roughly $(\Delta t)/t^2$. When t becomes smaller, the change in throughput will also increase but at an increasing rate. This therefore means that by modeling the throughput, the scores of the good PCs will be more clearly separated.

Additionally, we also consider a maximizing the throughput since throughput would be bounded by $[0, r_{\text{max}}]$, rather than the time per training step which would be unbounded on one end, i.e., be in the interval $[t_{\text{min}}, \infty)$ which makes suboptimal PCs easier to handle. Also, we frame our problem as a maximization in order to be consistent with Bayesian optimization works which typically considers a maximization problem.

D Additional Discussion of Surrogate Models used by OPPA

D.1 Construction of $\widehat{R}$

In this section, we elaborate on the construction of $\widehat{R}$, which eventually is used as the parallelism-informed prior mean for the GP prior belief of the throughput. In summary, we design $\widehat{R}$ to incorporate the following characteristics.

- *Parallelism coverage.* We model DP/TP traffic via All-Reduce–style collectives and PP via point-to-point transfers, using a placement-aware split of intra- and inter-node links. For computation costs, we also explicitly consider the pipeline bubbles.
- *Topology and protocol awareness.* The communication term uses canonical ring/tree scaling with hierarchical aggregation (node-local first, then cross-node), while collapsing ZERO stages into a single All-Reduce operation at the bandwidth level (bytes over the wire are of the same order) and letting stage-specific latency/overlap differences be absorbed by $\mathbf{C}$ (e.g., event counts, bucket sizes, communication overhead for each parallelism dimension, etc.).
- *Hardware agnosticism via learning.* Rather than hard-coding device/network constants, we use a set of learnable parameters whose values are learned from the estimates of the throughput. This keeps $\widehat{R}$ generalized across NNs, data types, and hardware setup while preserving the correct asymptotic trends for how each hyperparameters affect the overall throughput.

To model the throughput, we will estimate the time per training step then compute its reciprocal. In our approximation, we decompose the time per training step into the time required for the computation time (which includes the amount of tensor operations required and possibly any sequential dependency in those operations) and the communication time (which is from any intra- and inter-node communications).

To predict the computation time, we assume an idealized training scenario on infinitely many processors such that DP and TP can be perfectly parallelized. Meanwhile, PP uses an interleaved schedule which incurs additional computation time from the microbatches being run sequentially, creating a pipeline bubble when the first microbatch is being fed through the pipeline [22]. This additional computation time from PP, visualized in Fig. 10, is roughly equal to

$$\widehat{\mathcal{T}}_{\text{comp}}(H; t_{\text{comp}}) = \frac{t_{\text{comp}}}{\texttt{n\_gpus}} \cdot \left(\texttt{mb} + \frac{\texttt{pp} - 1}{\texttt{mc}} \right) \tag{5}$$

where `mb` is the number of microbatches used in PP (set to 1 when PP is not used), `mc` is the number of model chunks for PP (also set to 1 when PP is not used), and $t_{\text{comp}} = t_f + t_b$ is the total time to perform the forward and backward passes.

To predict the communication time, inspired by the model visualized in Fig. 2, we assume that DP and TP involve All-Reduce communications and PP P2P communications, where all these costs are modeled separately. We use an extended (α, β, γ) model discussed [38], where the intra- and inter-node communications are characterized by the latency α_{intra} and α_{inter}, the per-byte bandwidth cost β_{intra} and β_{inter}, the incast overhead γ_{intra} and γ_{inter}, and the memory access overhead δ_{intra} and δ_{inter}. We assume that the inter-node communication costs will be larger than their intra-node counterparts.

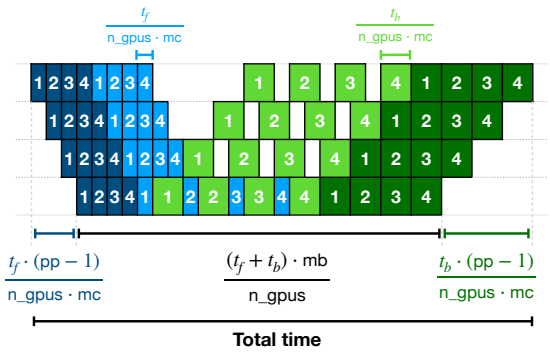


Figure 10: Predicted computation time for PP where t_f and t_b are the time taken for the respective forward and backward stages when $\texttt{pp} = \texttt{n\_gpus} = 4$, $\texttt{mb} = 4$, and $\texttt{mc} = 2$.

For DP and TP, we assume that the Ring All-Reduce implementation is used where the cost to gather and scatter data of size N across D GPUs is given by

$$C_{\text{AR}}(D, N, \alpha, \beta, \gamma, \delta) = 2(D-1)\alpha + \frac{D-1}{D} N(2\beta + \gamma + 3\delta) \, .$$

For DP, we consider the gradient synchronization from the All-Reduce procedure. The data size per GPU for DP All-Reduce, $N_{\texttt{dp}}$, is

$$N_{\texttt{dp}} = \lambda_Z \frac{M_{\text{model}}}{\texttt{tp} \cdot \texttt{pp}}$$

where M_{model} is a learnable parameter representing the total size of the NN weights, and λ_Z is a tiny fudge for ZERO flavor while staying in AR-land, and it accounts for param-All-Gather + grad-Reduce-Scatter volume equivalence with small overhead for ZERO-3.

Suppose G_{node} is the number of GPUs per node. The overall communication cost from DP $\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}$ would then depend on the configuration of the nodes.

- In a **hierarchical** (multi-node scenario with $\texttt{dp} > G_{\text{node}}$) system, the cost is given by

$$\begin{aligned}\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(H; \mathbf{C}) = {} & C_{\text{AR}}(G_{\text{node}}, N_{\texttt{dp}}, \alpha_{\text{intra}}, \beta_{\text{intra}}, \gamma_{\text{intra}}, \delta_{\text{intra}}) \\ & + C_{\text{AR}}(\lceil \texttt{dp}/G_{\text{node}} \rceil, N_{\texttt{dp}}/G_{\text{node}}, \alpha_{\text{inter}}, \beta_{\text{inter}}, \gamma_{\text{inter}}, \delta_{\text{inter}}). \end{aligned} \tag{6}$$

- In a **flat inter-node** (multi-node scenario with $\texttt{dp} \leq G_{\text{node}}$), the cost is given by

$$\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(H; \mathbf{C}) = C_{\text{AR}}(\texttt{dp}, N_{\texttt{dp}}, \alpha_{\text{inter}}, \beta_{\text{inter}}, \gamma_{\text{inter}}, \delta_{\text{inter}}). \tag{7}$$

- In a **flat intra-node** (single-node scenario), the cost is given by

$$\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(H; \mathbf{C}) = C_{\text{AR}}(\texttt{dp}, N_{\texttt{dp}}, \alpha_{\text{intra}}, \beta_{\text{intra}}, \gamma_{\text{intra}}, \delta_{\text{intra}}). \tag{8}$$

Depending on the hardware used, the appropriate cost for the scenario can be selected.

For TP, we consider the cost from frequent activation communication (e.g., All-Reduce per layer). Let $M_{\text{act},\texttt{tp}}$ be a learnable parameter representing the characteristic data size for one such TP All-Reduce operation. Let $O_{\texttt{tp/mb}}$ be the learnable parameter representing the number of these operations per microbatch. The total number of TP communication operations is $N_{\texttt{tp,ops}} = O_{\texttt{tp/mb}} \cdot \texttt{mb}$, and the cost of a single TP All-Reduce operation is $C_{\text{AR}}(\texttt{tp}, M_{\text{act},\texttt{tp}}, \alpha_{\text{eff}}, \beta_{\text{eff}}, \gamma_{\text{eff}}, \delta_{\text{eff}})$ where effective parameters $\alpha_{\text{eff}}, \beta_{\text{eff}}, \gamma_{\text{eff}}, \delta_{\text{eff}}$ are chosen as intra-node or inter-node based on whether the $\texttt{tp}$ group spans multiple nodes (i.e., if $N_H > 1$ and $\texttt{tp} > G_{\text{node}}$) or not. Then, the total TP communication cost is the number of communication operations multiplied by the cost per communication operation, or

$$\widehat{\mathcal{T}}_{\text{comm},\texttt{tp}}(H; \mathbf{C}) = N_{\texttt{tp,ops}} \cdot C_{\text{AR}}(\texttt{tp}, M_{\text{act},\texttt{tp}}, \alpha_{\text{eff}}, \beta_{\text{eff}}, \gamma_{\text{eff}}, \delta_{\text{eff}}). \tag{9}$$

For PP, we consider the point-to-point (P2P) transfers of activations and gradients between $\texttt{pp}$ pipeline stages where the cost to transfer data of size N is given by

$$C_{\text{P2P}}(N, \alpha, \beta) = \alpha + N \cdot \beta. \tag{10}$$

Let $M_{\text{act,pp}}$ be a learnable parameter representing the characteristic data size for one P2P transfer (e.g., the size of an activation tensor). The number of communication boundaries is $\texttt{pp}-1$. Communication occurs for each of $\texttt{mb}$ microbatch, in both forward (activations) and backward (gradients) directions. The total number of P2P communications is given by $N_{\text{pp,transfers}} = 2 \cdot \max(\texttt{pp}-1, 0) \cdot \texttt{mb}$ where a single P2P transfer uses effective latency α_{eff} and effective per-unit-data cost $\beta_{\text{pp,eff}}$ (derived from base β parameters), chosen as intra-node or inter-node based on whether communicating stages are on different nodes (approximated if $N_H > 1$). Given this, the overall cost of all communications related to PP would be given by

$$\widehat{\mathcal{T}}_{\text{comm,pp}}(H; \mathbf{C}) = N_{\text{pp,transfers}} \cdot C_{\text{P2P}}(M_{\text{act,pp}}, \alpha_{\text{eff}}, \beta_{\text{pp,eff}}). \tag{11}$$

Here, if $\texttt{pp} = 1$, then $\widehat{\mathcal{T}}_{\text{comm,pp}} = 0$.

When combining the communication costs for all three types of parallelism, considering in practice large chunks of communication overlap with compute, we attach non-overlap factors κ and obtain

$$\widehat{\mathcal{T}}_{\text{comm}}(H; \mathbf{C}) = \kappa_{\texttt{dp}} \cdot \widehat{\mathcal{T}}_{\text{comm,dp}}(H; \mathbf{C}) + \kappa_{\texttt{tp}} \cdot \widehat{\mathcal{T}}_{\text{comm,tp}}(H; \mathbf{C}) + \kappa_{\texttt{pp}} \cdot \widehat{\mathcal{T}}_{\text{comm,pp}}(H; \mathbf{C}) \tag{12}$$

where $\mathbf{C} = \{M_{\text{model}}, M_{\text{act,tp}}, M_{\text{act,pp}}, O_{\texttt{tp/mb}}, \alpha_{\text{intra}}, \beta_{\text{intra}}, \gamma_{\text{intra}}, \delta_{\text{intra}}, \alpha_{\text{inter}}, \beta_{\text{inter}}, \gamma_{\text{inter}}, \delta_{\text{inter}}\}$ are learnable parameters related to the various costs as mentioned above which are to be inferred based on the measured throughput. Given (5) and (12), we can construct the prior mean for throughput to be

$$\widehat{\mathcal{R}}(H; \{t_{\text{comp}}, \mathbf{C}\}) = \left[\widehat{\mathcal{T}}_{\text{comp}}(H; t_{\text{comp}}) + \widehat{\mathcal{T}}_{\text{comm}}(H; \mathbf{C})\right]^{-1}. \tag{13}$$

D.2 Construction of $\widehat{M}$

We now briefly elaborate on the construction of $\widehat{M}$, and is eventually used as the parallelism-informed prior mean for the GP prior belief of the maximum memory usage.

As discussed in the main paper, we only consider the GPU memory that is required to store the NN weights, and those to store the gradients. For NN weights, its sharding can be done in the pipeline itself or within the layers (i.e., by sharding the tensors which make up the NN weights), allowing us to approximate the memory required for storing the NN weights per GPU to be inversely proportional to $\texttt{pp} \cdot \texttt{tp}$. Note that assuming the simplest DP implementation, the NN weights are duplicated and stored on each DP dimension, and so the maximum memory usage is not affected by the DP. In practice, we can also make the memory required in this case also inversely proportional to the DP dimension size only in the case where ZeRO-3 is used.

Meanwhile, assuming we have a fixed number of GPUs, the maximum memory required for storing the gradients will roughly be proportional to how many model parameters a certain GPU has to perform the forward and backward passes for (which is inversely proportional to $\texttt{tp} \cdot \texttt{pp}$ as before), times how many training samples the GPU has to process at any one time (which is inversely proportional to the number of microbatches).

Combining these two factors, can write the maximum memory usage as

$$\widehat{\mathcal{M}}(H; \theta_{\mathcal{M}}) = \min\left\{m_1 \cdot (\texttt{pp} \cdot \texttt{tp})^{-1} + m_2 \cdot (\texttt{pp} \cdot tpar \cdot \texttt{mb})^{-1} + m_3,\ M_0\right\} \tag{14}$$

where, m_1 captures the memory used for storing NN weights, and m_2 captures the memory used for storing the gradients, m_3 are any other additional memory overheads unaccounted for by our model. Combined, $\theta_{\mathcal{M}} = \{m_1, m_2, m_3\}$ are learnable parameters that are to be inferred based on the measured maximum memory usage. Note that since we cannot measure maximum memory usage above values of M_0, we apply the $\min$ function to clip the GP prior beliefs.

D.3 Kernel for Modeling the Unknown Deviations

Given a PC $H \in \mathbb{Z}^d$, we use the Matérn kernel [29] which is given by

$$k(H, H') = \sigma_k^2 k_{\text{Matérn},\nu}(H, H'; \ell) = \sigma_k^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\, d_\ell(H, H')\right)^\nu K_\nu\left(\sqrt{2\nu}\, d_\ell(H, H')\right)$$

where Γ is the Gamma function, K_ν is the modified Bessel function, σ_k is the kernel scaling constant,

$$d_\ell(H, H') = \left(H - H'\right)^\top \mathbf{L}^{-2} \left(H - H'\right) \tag{15}$$

is the distance between two PC (in the vector representation) and $\mathbf{L} = \text{diag}(\boldsymbol{\ell}) = \text{diag}([\,\ell_1 \cdots \ell_p\,])$ is the lengthscale for each of the input dimension.

Justification for kernel based on Euclidean distance. Note that in our setup, we use a kernel based on the Euclidean distance despite a discrete search space. This is because we expect each value within the PCs are typically correlated according to its order (e.g., throughput when $\texttt{dp} = 1$ will correlate with when $\texttt{dp} = 2$, which then correlates with when $\texttt{dp} = 4$, etc.). In this regard, there exists some correlation between PCs that have similar values under our embedding (even if the values are all discrete), justifying our use of GPs.

We also observe this later in Table 6 which shows that the lengthscales learned by the GP are larger in some values even without explicit treatment of the discrete values, suggesting existence of learned interpolations by OPPA. Additionally, later in Fig. 20, we demonstrate that OPPA which uses a Matérn kernel outperforms OPPA which instead uses a categorical based on the Hamming distance as in [4], which empirically further justifies our kernel choice.

D.4 GP Posterior Beliefs for Throughput and Maximum Memory Usage

Suppose we are in the ith round. We let $\mathbf{H}_i = [H_1 \cdots H_i]$ be the list of PCs, $\bar{\mathbf{r}}_i = [\bar{r}_{1,\hat{q}_1} \cdots \bar{r}_{i,\hat{q}_i}]$ and $\sigma^2_{\bar{\mathbf{r}}_i} = [\sigma^2_{\bar{r}_{1,\hat{q}_1}} \cdots \sigma^2_{\bar{r}_{i,\hat{q}_i}}]$ be the measured throughput and the corresponding variance, and $\mathbf{m}_i = [m_1 \cdots m_i]$ be the measured maximum memory usage values.

Given the data, we first find the optimal hyperaparameters $\theta = \{\theta_{\mathcal{R}}, \theta_{\mathcal{M}}, \theta_k\}$. This is done by maximizing the log marginal likelihood [29], or

$$\theta = \arg\max_{\theta'} \; \log p(\bar{\mathbf{r}}_i, \bar{\mathbf{m}}_i | \bar{\mathbf{H}}_i, \theta). \tag{16}$$

The GP posterior beliefs for the throughput and maximum memory usage of any PC H' are then given by normal distributions

$$\mathcal{N}\big(\mu_{\mathcal{R},i}(H'), \sigma^2_{\mathcal{R},i}(H')\big) \quad \text{and} \quad \mathcal{N}\big(\mu_{\mathcal{M},i}(H'), \sigma^2_{\mathcal{M},i}(H')\big) \tag{17}$$

where the mean and variance for the throughput of any PC H' can then be defined as

$$\mu_{\mathcal{R},i}(H') = \widehat{\mathcal{R}}(\mathbf{H}_i; \theta_{\mathcal{R}}) + k(H', \mathbf{H}_i; \theta_k)\big(k(\mathbf{H}_i, \mathbf{H}_i; \theta_k) + \text{diag}(\sigma^2_{\bar{\mathbf{r}}_i})\big)^{-1}\big(\bar{\mathbf{r}}_i - \widehat{\mathcal{R}}(\mathbf{H}_i; \theta_{\mathcal{R}})\big), \tag{18}$$

$$\sigma^2_{\mathcal{R},i}(H') = k(H', H'; \theta_k) - k(H', \mathbf{H}_i; \theta_k)\big(k(\mathbf{H}_i, \mathbf{H}_i; \theta_k) + \text{diag}(\sigma^2_{\bar{\mathbf{r}}_i})\big)^{-1} k(\mathbf{H}_i, H'; \theta_k), \tag{19}$$

and the GP posterior mean and variance for the maximum memory usage of any PC H' can then be defined as

$$\mu_{\mathcal{M},i}(H') = \widehat{\mathcal{M}}(\mathbf{H}_i; \theta_{\mathcal{M}}) + k(H', \mathbf{H}_i; \theta_k)\big(k(\mathbf{H}_i, \mathbf{H}_i; \theta_k) + \lambda I\big)^{-1}\big(\mathbf{m}_i - \widehat{\mathcal{M}}(\mathbf{H}_i; \theta_{\mathcal{M}})\big), \tag{20}$$

$$\sigma^2_{\mathcal{M},i}(H') = k(H', H'; \theta_k) - k(H', \mathbf{H}_i; \theta_k)\big(k(\mathbf{H}_i, \mathbf{H}_i; \theta_k) + \lambda I\big)^{-1} k(\mathbf{H}_i, H'; \theta_k) \tag{21}$$

where λ is to make the matrix invertible.

E Additional Discussion of PC Selection Method in OPPA

E.1 Constrained UCB Acquisition Function

The next PC $H_i \in \mathcal{H}$ to trial in round i is selected to maximize the constrained upper confidence bound (cUCB) acquisition function [33, 37], which given by

$$\text{cUCB}_i(H) \triangleq \mathbb{E}_{\hat{r}_{H,i-1}, \hat{m}_{H,i-1}}\big[X_{H,i-1} + \beta_i \big| X_{H,i-1} - \mathbb{E}[X_{H,i-1}]\big|\big] \tag{22}$$

where $\hat{r}_{H,i-1} \sim \mathcal{N}\big(\mu_{\mathcal{R},i-1}(H), \sigma^2_{\mathcal{R},i-1}(H)\big)$ and $\hat{m}_{H,i-1} \sim \mathcal{N}\big(\mu_{\mathcal{M},i-1}(H), \sigma^2_{\mathcal{M},i-1}(H)\big)$ are sampled from their respective GPs as written in (17), and $X_{H,i-1} = \hat{r}_{H,i-1} \cdot \big(1 - \texttt{sigmoid}(\hat{m}_{H,i-1})\big)$. Note that in the case that memory constraint is not violated (i.e., when $\texttt{sigmoid}(\hat{m}_{H,i-1}) \approx 0$), the objective in (22) can be reduced to the analytical UCB objective (i.e., $\text{cUCB}_i(H) \approx \mu_{\mathcal{R},i-1}(H) + \beta_i \sigma_{\mathcal{R},i-1}(H)$).

E.2 Random Sampling For Additional Exploration

In OPPA, we sometimes select PCs at random for additional exploration. There are two scenarios which triggers a random selection of PC in OPPA.

1. In the first few chosen PCs. This is because in the beginning there are no PCs which can be used to infer the hyperparameters of the prior belief, therefore a few PCs are chosen at random to kick-off the modeling process and provide a reasonably diverse set of samples for inferring the hyperparameters well.
2. When too many out-of-memory errors have been encountered in a row. This is because any out-of-memory trials will not result in a usable training data for the throughput modeling and possibly minimal data for the maximum memory GP which does not aid the GP model. When too many such cases are encountered, we attempt to do random exploration so that the model can receive some information that can be used to model better with and find new feasible PCs.

For the random selection process, we select a PC using a weighted random strategy, where we first select a parallelism dimension configuration uniformly at random, then select the remaining hyperparameters uniformly at random such that they are feasible within the selected parallelism dimension configuration.

F Additional Discussion of PC Trialing Method

F.1 Throughput Estimation

Given the time $t_{i,1}, \ldots, t_{i,q}$ required for q training steps, we can estimate the throughput and its predicted variance as

$$\bar{r}_{i,q} = \frac{1}{q}\sum_{j=1}^{q}\frac{1}{t_{i,j}}, \quad \text{and} \quad \sigma^2_{\bar{r}_{i,q}} = \frac{1}{q(q-1)}\sum_{j=1}^{q}\left(\frac{1}{t_{i,j}} - \bar{r}_{i,q}\right)^2. \tag{23}$$

Note that $\bar{r}_{i,q}$ can be viewed as an unbiased estimator of the reciprocal of the time per training step (i.e., $\mathbb{E}[\bar{r}_{i,q}] = 1/\mathcal{T}(H_i)$ where $\mathcal{T}(H_i)$ is the time per training step). Meanwhile, $\sigma^2_{\bar{r}_{i,q}}$ is the variance of the estimator, which roughly scales inversely with the number of samples q.

F.2 Outlier Removal

As demonstrated in Fig. 3, not all training time measurements will be representative of the actual throughput. We therefore perform two actions. First, we remove the first training step $t_{i,1}$ since it typically corresponds to a warm-up for the training and therefore will usually be an anomaly measurement. Second, we compute the median and the inter-quartile range (IQR), and remove all measurements which are away from the median by at least $2 \times IQR$. This is a standard method for outlier removal in practice. However, we choose a looser threshold for outliers to account for fluctuating measurements. This generally has little effect on the overall optimization since most training steps tend to not fluctuate by too much anyway. This is demonstrated by Fig. 11 which shows that the predicted throughput would have little difference across the threshold of outlier removal, showing robustness of our method to the outlier removal process. The remaining training points are then used to compute (23).

F.3 Additional Discussion on Early Trial Termination Mechanism

In this section, we attempt to prove Thm. 4.1. To do so, we will consider a more general case for an arbitrary unknown function f. We do so since the problem setting of sequential repeated measurements is useful in many problem settings beyond optimizing the PC, such as in repeated scientific experiments where we can decide on-the-fly whether the same experimental setup should be repeated or not. We will first provide an intuitive justification for why early trial termination can be applied, followed by a theoretical proof for Thm. 4.1 and ablation studies on synthetic functions.

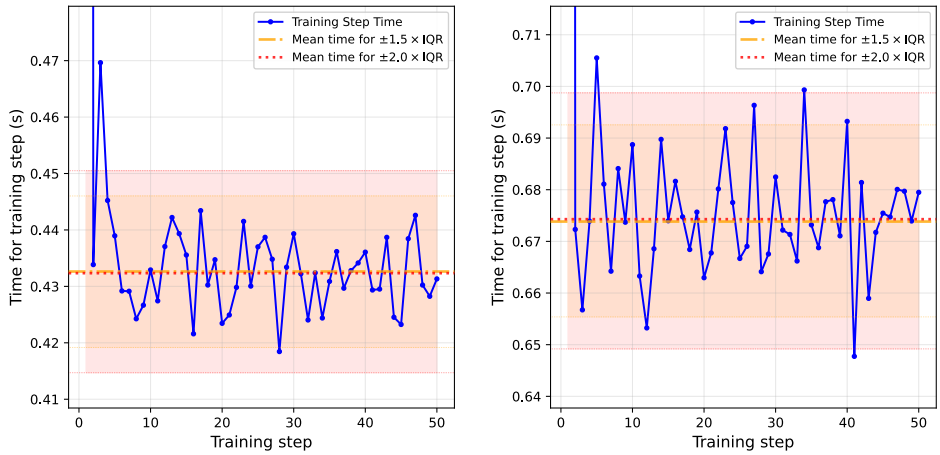


Figure 11: Example of the outlier removal process based on the IQR on different random PC trials. Here, we show the effect of the threshold selected vs. the mean of the measurements, demonstrating the robustness with respect to the different thresholds.

F.3.1 Intuition Underlying Early Trial Termination Mechanism

Before stating the formal proof, we provide some intuition behind why the early trial termination mechanism saves on resources while also not affecting the overall performance. In Fig. 12, we present a toy example of a BO iteration where an input is repeatedly queried.

Here, we notice several things. We first see that for a certain input, more repetitions of the query will result in a lower standard deviation for the measurement. This is simply from the fact that when we take the mean of several i.i.d. measurements, the mean will tend towards the actual value, and the variance of the mean predictor will decay at a rate inversely proportional to the number of repeated measurements. This is reflected in the plots where the measurement at $x = 2$ has smaller and smaller standard deviation as we have more repeats, and as we will show in Corollary F.2. In these cases, we see that some repeated queries are therefore necessary to reduce the noise of the measurements.

However, as we perform more repeats, there is a diminishing effect on how much more information is gained from each additional repeats. In the diagram, we see that after 20 repeated measurements (as in Fig. 12c), the posterior GP would barely change. Similarly, we see that at 20 repeated measurements, we are able to somewhat conclude that the global optimum is less likely to be in the neighborhood of $x = 2$, as reflected by the corresponding probability density function, and that this belief remains when we continue up to 100 repeated measurements (as in Fig. 12d). This shows the additional repeated measurements are unlikely to improve the predictions by the GP and the confidence of the optimum. We therefore can see that by stopping after 20 repeated measurements, we would have been able to eventually make the same conclusion about the optimum in the end while saving on resources that would have been incurred from additional repeated measurements.

Finally, we note that additional repeats are more beneficial to recovering the optimum value in the case that it improves upon the best measurement so far. In Fig. 13, we see that additional repeated measurements at $x = 2$ eventually results in diminishing returns, as seen in the probability distribution of the maximum value of $f(x)$ being similar between using 20 repeats and 100 repeats. Meanwhile, in Fig. 14 where we instead repeatedly query the input $x = 4.2$, we continue to gain information about the maximum value even when we use 100 repeats, as seen by the sharper probability distribution for the belief of the maximum $f(x)$. We therefore can see that additional repeats are less informative if they are for suboptimal inputs. We can therefore use this idea to form a metric to determine if a trial should be terminated early.

F.3.2 Proof of Thm. 4.1

With this intuition, we will now formally prove the performance of the early trial termination mechanism. For completeness, we first state the assumptions for the function and the measurements

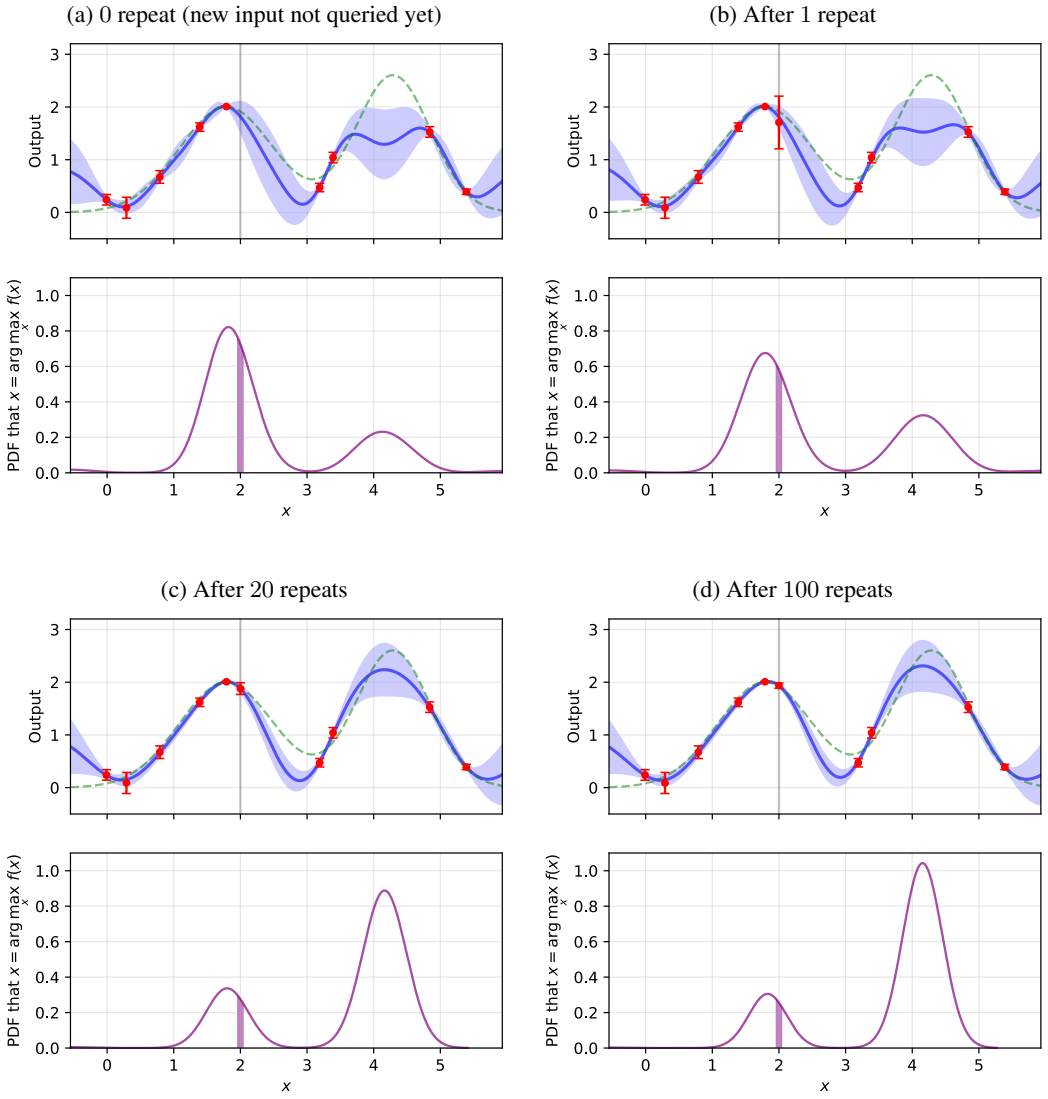


Figure 12: Demonstration of why early termination of certain trials does not affect final result. Figs. 12a to 12d represent the modeled GP and the predicted optimum when a certain input, in this case at $x = \arg\max_{x'} \mu_t(x') + \beta_t \sigma_t(x') = 2$, is repeatedly queried for 0, 1, 20, and 100 times. For each subfigure, the top diagram represents the modeled GP where the dashed green curve represents the actual function, the solid blue line and error bands represent the GP mean and standard deviation respectively, and the red points represent the predicted mean and standard deviation (the latter which shrinks with more repeats). In the bottom diagram, we show the probability of each input being the optimal (i.e., probability that $x = \arg\max_{x'} f(x')$), while highlighting the interval around the input $x = 2$.

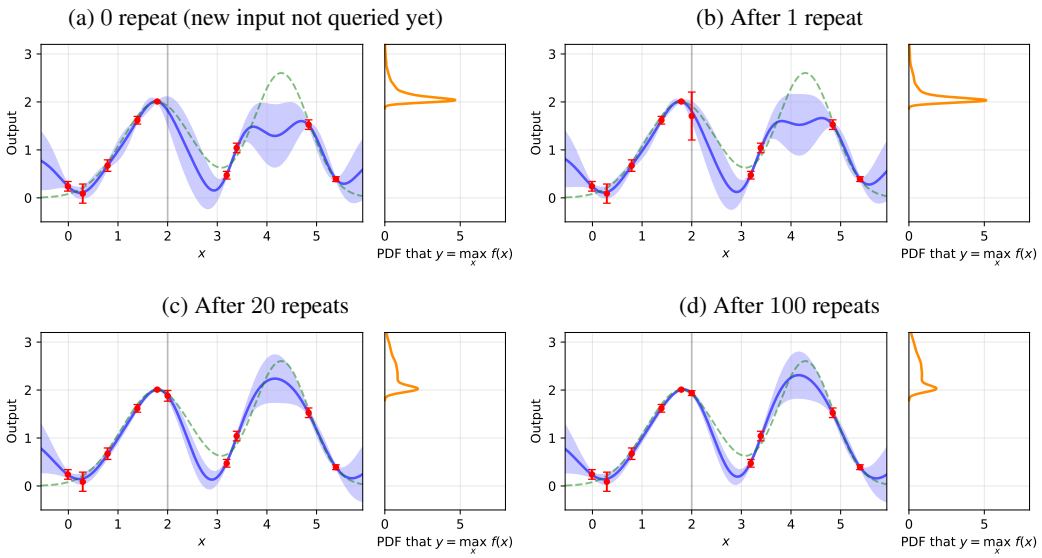


Figure 13: Additional demonstration of why early termination of certain trials does not affect the obtained optimal value. The setup is as done previously in Fig. 12 with queries at $x = 2$. For each subfigure, the left plot represents the modeled GP as shown previously in Fig. 12. In the right diagram, we show the probability density function for the belief of the maximum value $\max_x f(x)$ according to the GP.

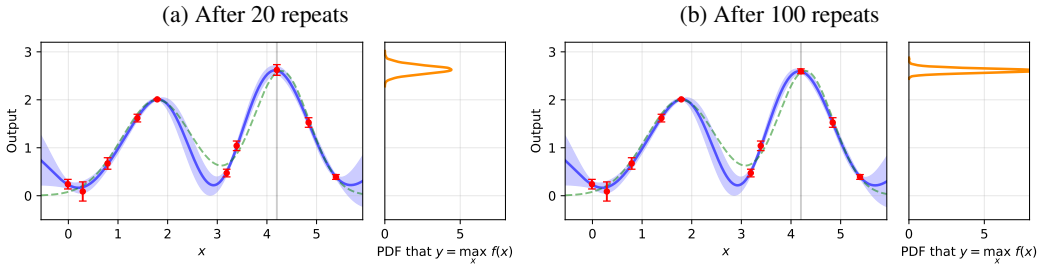


Figure 14: Additional demonstration of why early termination of certain trials does not affect the obtained optimal value. The setup is similar to as done previously in Fig. 12, however instead with queries at $x = 4.2$, and with only 20 repeats and 100 repeats shown (comparable to Figs. 13c and 13d). The information in each subfigure is the same as in Fig. 13.

which follow from other BO works [12, 21, 33] however with additional assumptions on repeated measurements from the same input.

Assumption F.1. Let $f \sim \mathcal{GP}(0, k)$ be an unknown function whose RKHS norm $\|f\|_{\mathcal{H}} \leq B$ is bounded. The BO algorithm is as noted in Algorithm 3, where in each BO iteration i, an input $x_i \in \mathcal{X}$ is selected, and $\hat{q}_i \leq q_{\max}$ noisy outputs $y_{i,j} = f(x_i) + \varepsilon_{i,j}$ are returned, where $\varepsilon_{i,j} \sim \mathcal{N}(0, s^2)$ are i.i.d. noise.

Note that in this following proof, we consider any function f which has a bounded RKHS norm. While this property also applies for the unknown throughput $\mathcal{R}$, it will also be satisfied by other black-box functions in general as well. Our theoretical results in this section will consider Algorithm 3. Formally, we will analyze Algorithm 3 which does early trial termination on some of the rounds. This is in contrast with Algorithm 2 which repeats each measurement $q_{\max}$ times regardless of input.

We first state a result regarding the mean estimator of $f(x_i)$.

Algorithm 2 GP-UCB with Repeated Measurements

$\mathcal{D}_0 \leftarrow \emptyset$
for $i = 1, \ldots, N$ **do**
  Fit GP on $\mathcal{D}_{i-1}$ to obtain mean μ_{i-1} and variance σ^2_{i-1}
  $x_i \leftarrow \arg\max_{x \in \mathcal{X}} \mu_{i-1}(x) + \beta_i \sigma_{i-1}(x)$
  for $j = 1, \ldots, q_{\text{max}}$ **do**
    Sample measurement $y_{i,j}$
  $\bar{y}_{i,q_{\text{max}}} \leftarrow q_{\text{max}}^{-1} \sum_{j=1}^{q_{\text{max}}} y_{i,j}$
  $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(x_i, \bar{y}_{i,q_{\text{max}}})\}$

Algorithm 3 Modified GP-UCB with Repeated Measurements and Early Trial Termination

$\mathcal{D}_0 \leftarrow \emptyset$
for $i = 1, \ldots, N$ **do**
  Fit GP on $\mathcal{D}_{i-1}$ to obtain mean μ_{i-1} and variance σ^2_{i-1}
  $x_i \leftarrow \arg\max_{x \in \mathcal{X}} \mu_{i-1}(x) + \beta_i \sigma_{i-1}(x)$
  for $j = 1, \ldots, q_{\text{max}}$ **do**
    Sample measurement $y_{i,j}$
    $\bar{y}_{i,j} \leftarrow j^{-1} \sum_{j'-1}^{j} y_{i,j'}$
    $\mathcal{D}'_{i,j'} \leftarrow \mathcal{D}_i \cup \{(x_i, \bar{y}_{i,j})\}$
    if $j > q_{\text{min}}$ and $\bar{y}_{i,q} < \max_{j<i} \bar{y}_{j,\hat{q}_j} + \tau_q$ **then**
      break // Early trial termination mechanism
  $\mathcal{D}_i \leftarrow \mathcal{D}'_{i,j'}$

Corollary F.2. *Suppose we define*

$$\bar{y}_{i,q} = \frac{1}{q} \sum_{j=1}^{q} y_{i,j}. \tag{24}$$

Then, the expected value of the mean estimator $\bar{y}_{i,q}$ is $\mathbb{E}[\bar{y}_{i,q}] = f(x_i)$, while the variance of the mean estimator is given by $\mathbb{V}[\bar{y}_{i,q}] = s^2/q$.

Proof. The results are direct consequences of the summation of expected values and variances of independent random variables, and the fact that $\mathbb{V}[y_{i,j}] = s^2$ by assumption. □

We will now prove the first half of Thm. 4.1. Given the results in Corollary F.2, we next show that we are able to obtain a GP with good predictive bounds even if some trials are terminated early, i.e., not run for up to q_{max} repeats.

Lemma F.3. *Suppose we let*

$$\mu_i(x) = k(x, \mathbf{X}_i)\big(k(\mathbf{X}_i, \mathbf{X}_i) + s^2 \mathbf{Q}_i^{-1}\big)^{-1} \mathbf{y}_i \,, \tag{25}$$

$$\sigma_i^2(x) = k(x, x) - k(x, \mathbf{X}_i)\big(k(\mathbf{X}_i, \mathbf{X}_i) + s^2 \mathbf{Q}_i^{-1}\big)^{-1} k(\mathbf{X}_i, x) \tag{26}$$

where $\mathbf{X}_i = [x_1 \cdots x_i]$, $\mathbf{y}_i = [\bar{y}_{1,\hat{q}_1} \cdots \bar{y}_{i,\hat{q}_i}]$, and $\mathbf{Q}_i = \text{diag}([\hat{q}_1 \cdots \hat{q}_i])$. If

$$\beta_i = B + \sqrt{2 \log \frac{\det\left(k(\mathbf{X}_i, \mathbf{X}_i) + s^2 \mathbf{Q}_i^{-1}\right)^{1/2}}{\delta \det\left(s^2 \mathbf{Q}_i^{-1}\right)^{1/2}}} \tag{27}$$

then, with probability greater than $1 - \delta$, for all $x \in \mathcal{X}$ and all $i = 1, \ldots, N$, we have

$$\big|f(x) - \mu_{i-1}(x)\big| \leq \beta_i \sigma_{i-1}(x). \tag{28}$$

Proof. Given the variance of the mean predictor from Corollary F.2, our scenario can be thought of as having i function measurements with heteroscedastic noise with variances of $s^2/\hat{q}_1, \ldots, s^2/\hat{q}_i$. The variance bounds then follow directly from Lemma 7 in [12] where we substitute $\Sigma_i \to s^2 \mathbf{Q}_i^{-1}$ and $\lambda \to 1$. □

Corollary F.4. *Given Lemma F.3, for all $x' \in \mathcal{X}$, we have $f(x') - \mu_{i-1}(x_i) \leq \beta_i \sigma_{i-1}(x_i)$.*

Proof. We see that

$$\begin{aligned} f(x') - \mu_{i-1}(x_i) &\leq \mu_{i-1}(x') + \beta_i \sigma_{i-1}(x') - \mu_{i-1}(x_i) && \text{by Lemma F.3,} && (29)\\ &\leq \mu_{i-1}(x_i) + \beta_i \sigma_{i-1}(x_i) - \mu_{i-1}(x_i) && \text{by how } x_i \text{ is chosen,} && (30)\\ &= \beta_i \sigma_{i-1}(x_i). && && (31) \end{aligned}$$

□

We now show the cumulative regret of the problem. Let

$$\mathbb{I}[\mathbf{y}_X; \mathbf{f}_X] = \frac{1}{2} \sum_{x'_i \in X} \log\left(1 + \frac{\sigma^2_{i-1}(x'_i)}{s^2/\hat{q}_i}\right), \tag{32}$$

and

$$\gamma_i = \max_{X=[x'_1,\ldots,x'_i] \subset \mathcal{X}} \mathbb{I}[\mathbf{y}_X; \mathbf{f}_X] \tag{33}$$

be the maximum possible information gain across i rounds. We then prove the following result.

Theorem F.5. *Let $x^* = \arg\max_{x \in \mathcal{X}} f(x)$. With probability at least $1 - \delta$,*

$$\sum_{i=1}^{N} f(x^*) - f(x_i) \leq s \beta_N \sqrt{\frac{8N\gamma_N}{q_{\min}}}. \tag{34}$$

Proof. This result is similar to previous results for UCB-based methods, e.g., Theorem 3 in [33] or Corollary 9 in [12].

With probability at least $1 - \delta$, Lemma F.3 holds. We see that

$$\begin{aligned} \sum_{i=1}^{N} \big(f(x^*) - f(x_i)\big) &= \sum_{i=1}^{N} \underbrace{\big(f(x^*) - \mu_{i-1}(x_i)\big)}_{Corollary\ F.4} + \underbrace{\big(\mu_{i-1}(x_i) - f(x_i)\big)}_{Lemma\ F.3} && (35)\\ &\leq \sum_{i=1}^{N} 2\beta_i \sigma_{i-1}(x_i). && (36) \end{aligned}$$

Since

$$\begin{aligned} \sum_{i=1}^{N} \sigma^2_{i-1}(x_i) &\leq \sum_{i=1}^{N} \frac{s^2}{\hat{q}_i} \frac{\sigma^2_{i-1}(x_i)}{s^2/\hat{q}_i} && (37)\\ &\leq \frac{s^2}{q_{\min}} \sum_{i=1}^{N} \log\left(1 + \frac{\sigma^2_{i-1}(x_i)}{s^2/\hat{q}_i}\right) && (38)\\ &\leq \frac{2s^2}{q_{\min}} \gamma_N\,, && (39) \end{aligned}$$

we can rewrite (36) as

$$\begin{aligned} \sum_{i=1}^{N} \big(f(x^*) - f(x_i)\big) &\leq \sqrt{\sum_{i=1}^{N} 4\beta_i^2} \sqrt{\sum_{i=1}^{N} \sigma^2_{i-1}(x_i)} && \text{by Cauchy-Schwartz,} && (40)\\ &\leq \beta_N \sqrt{4N} \sqrt{\sum_{i=1}^{N} \sigma^2_{i-1}(x_i)} && \text{since } \beta_i \leq \beta_N, && (41)\\ &\leq s\beta_N \sqrt{\frac{8N\gamma_N}{q_{\min}}} && \text{by (39).} && (42) \end{aligned}$$

□

Remark F.6 (The regret bound in Thm. F.5 is lower when $q_{\min}$ increases). We investigate the upper bound in (42) further to see its dependence on $q_{\min}$.

For simplicity, we will let $\mathbf{K} = k(\mathbf{X}_N, \mathbf{X}_N)$. First, we see that

$$\log\det\left(\mathbf{K} + s^2\mathbf{Q}_N^{-1}\right) = \log\det s^2\mathbf{Q}_N^{-1} + \log\det\left(I + s^{-2}\mathbf{Q}_N\mathbf{K}\right) \tag{43}$$

$$\leq \log\det s^2\mathbf{Q}_N^{-1} + \log\det\left(I + s^{-2}q_{\max}\mathbf{K}\right), \tag{44}$$

which means that

$$\beta_N = B + \sqrt{2\log\frac{\det\left(\mathbf{K} + s^2\mathbf{Q}_N^{-1}\right)^{1/2}}{\delta\det\left(s^2\mathbf{Q}_N^{-1}\right)^{1/2}}} \tag{45}$$

$$= B + \sqrt{2\log\frac{1}{\delta} + \log\det\left(\mathbf{K} + s^2\mathbf{Q}_N^{-1}\right) - \log\det s^2\mathbf{Q}_N^{-1}} \tag{46}$$

$$\leq B + \sqrt{2\log\frac{1}{\delta} + \log\det\left(I + s^{-2}q_{\max}\mathbf{K}\right)}. \tag{47}$$

Furthermore, we see that

$$\gamma_N = \max_{X=[x_1',\ldots,x_N']\subset\mathcal{X}} \frac{1}{2}\sum_{x_i'\in X}\log\left(1 + \frac{\sigma_{i-1}^2(x_i')}{s^2/\hat{q}_i}\right) \tag{48}$$

$$\leq \max_{X=[x_1',\ldots,x_N']\subset\mathcal{X}} \frac{1}{2}\sum_{x_i'\in X}\log\left(1 + \frac{\sigma_{i-1}^2(x_i')}{s^2/q_{\max}}\right). \tag{49}$$

Therefore we see that both β_N and γ_N are upper bounded by terms which are independent of $q_{\min}$, and so the constants in the upper bound provided in Thm. F.5 do not hide any additional dependencies with respect to $q_{\min}$. This shows that the upper bound of cumulative regret from (42) decays at a rate of $1/\sqrt{q_{\min}}$.

We will now prove the second half of Thm. 4.1. We first prove the following result.

Lemma F.7. *Define* $c_1 \triangleq \sqrt{2\log(2Nq_{\max}/\delta)}$. *With probability at least* $1-\delta$, *for all* $i = 1,\ldots,N$ *and all* $q = 1,\ldots,q_{\max}$, *we have*

$$\left|\bar{y}_{i,q} - f(x_i)\right| \leq \frac{c_1 s}{\sqrt{q}}. \tag{50}$$

Proof. From Corollary F.2, we know that the $\bar{y}_{i,q}$ is normally distributed with mean $f(x_i)$ and standard deviation $s/\sqrt{q}$. By Chernoff bounds, for each $i = 1,\ldots,N$ and each $q = 1,\ldots,q_{\max}$, we would have $\bar{y}_{i,q} - f(x_i) > c_1 s/\sqrt{q}$, and $f(x_i) - y_{i,q} > c_1 s/\sqrt{q}$ where either event happens with probability no greater than $\delta N/2q_{\max}$. This means that with probability no greater than $\delta/Nq_{\max}$, we have $|y_{i,q} - f(x_i)| > c_1 s/\sqrt{q}$. Therefore, by union bound, we have for all $i = 1,\ldots,N$ and each $q = 1,\ldots,q_{\max}$, we would have $|\bar{y}_{i,q} - f(x_i)| \leq c_1 s/\sqrt{q}$ with probability greater than $1 - (\delta/Nq_{\max})(Nq_{\max}) = 1-\delta$. □

Theorem F.8. *Define*

$$\tau_q = c_1 s\left(\frac{1}{\sqrt{q_{\min}}} + \frac{1}{\sqrt{q}}\right). \tag{51}$$

Then, with probability at least $1-\delta$, *for all* $i \leq N$, *if we have* $f(x_i) < \max_{j<i} f(x_j)$, *then* $\hat{q}_i < q_{\max}$.

Proof. With probability at least $1-\delta$, Lemma F.7 applies.

To prove the statement above, we show its contrapositive. Suppose we have $\hat{q}_i = q_{\max}$, or that the trial for x_i does not terminate early. This implies that $\bar{y}_{i,q} \geq \max_{j<i}\bar{y}_{j,\hat{q}_j} + \tau_q$ for all $q = q_{\min},\ldots,q_{\max}$.

For any q in this range, we would then have

$$f(x_i) \geq \bar{y}_{i,q} - \frac{c_1 s}{\sqrt{q}} \tag{52}$$

$$\geq \max_{j<i} \bar{y}_{j,\hat{q}_j} + \tau_q - \frac{c_1 s}{\sqrt{q}} \tag{53}$$

$$\geq \max_{j<i} f(x_j) - \frac{c_1 s}{\sqrt{q_{\min}}} + \tau_q - \frac{c_1 s}{\sqrt{q}} \tag{54}$$

$$= \max_{j<i} f(x_j). \tag{55}$$

This proves the contrapositive which in turn proves the original statement. □

Finally, Thms. F.5 and F.8 can be combined with appropriate union bounds to achieve a more formal version of Thm. 4.1.

F.3.3 Ablation Studies on Toy Examples

In Fig. 15, we provide a brief empirical demonstration of efficiency gains due to early trial termination. We see that when the noise variance is too high, querying the function once per input would give measurements which are too noisy to give good information. Meanwhile, by repeating each query a maximum number of times, we can obtain a good estimate of the actual function and allow the BO algorithm to arrive at the optimal using few queries. However, it can be observed that when early trial termination is allowed, we can still arrive at the optimal input as before, while not requiring all queries to be repeated the maximum number of times. This shows that early trial termination allows for efficiency gains while minimally sacrificing on the actual optimization process.

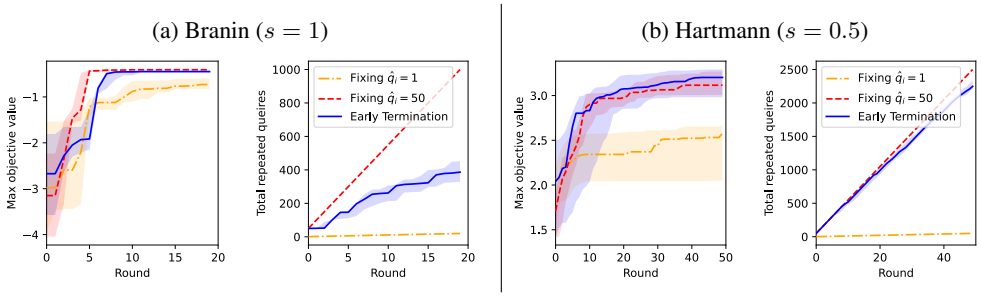


Figure 15: Results of BO with early trial termination (with $q_{\min} = 1$) on different synthetic functions where each query has different noise levels. For each case, we show the best queried objective value (left plot), and the cumulative number of total repeated queries made with each BO iteration (right plot).

In practice, since it is difficult to determine γ_N, β_N, s, and c_1 exactly, we instead fix β_i and τ_q to some constant. In OPPA, we choose $\beta_i = 1$ and $\tau_q = 10^{-3}$. We find that these values work well for our methods. Furthermore, in our proofs we do not consider the constrained BO setting. Despite this, the early trial termination can still be used in practice to achieve good results which we show in the experiments.

G Pseudocode and Hyperparameter Choices of OPPA

We present the pseudocode for OPPA in Algorithm 4.

In practice, we find that fixed hyperparameter values (i.e., setting $\beta_t = 1$, $\tau_q = 10^{-3}$, and $q_{\min} = 5$ in all iterations as done in our main experiments) give consistent results in most cases since the parameters are generally robust to different choices of hyperparameter values anyway. In our experiments, we only use a higher $q_{\min}$ value for MoE training (where we use $q_{\min} = 10$) to counteract the higher fluctuation in training load between each training step.

In our experiments, the value of $q_{\max}$ are set in accordance to Table 2 depending on the benchmark; however in practice can be set as large as the practitioner wants depending on how accurately they

would like to measure the actual throughput of the PC. As plotted in Fig. 16, we find that after a certain point, a larger number of repeats will typically not have large effects on the measured throughput of a PC, showing that OPPA is quite robust to different choices of $q_{\max}$. For OPPA, larger $q_{\max}$ is more tolerable since the early trial termination mechanism will not exhaust such budget anyway.

We also consider removing outliers which are beyond two times the IQR from the mean; this is be more lenient towards fluctuating training effects, however can also be set to 1.5 times the IQR as typically done as well with little consequences, as described in App. F.2.

Algorithm 4 OPTIMIZER FOR PARALLELISM CONFIGURATIONS (OPPA)

Generate all valid PCs $\mathcal{H}$
$i \leftarrow 1$
while time budget not exhausted **do**
  if $i < N_{\text{random}}$ **then**
    Select H_i randomly
  else
    // Step ① – Modeling throughput and memory usage
    Construct $\mu_{\mathcal{R},i-1}$ and $\sigma^2_{\mathcal{R},i-1}$ according to (18) and (19), respectively
    Construct $\mu_{\mathcal{M},i-1}$ and $\sigma^2_{\mathcal{M},i-1}$ according to (20) and (21), respectively
    // Step ② – Selecting the next PC to trial
    $H_i \leftarrow \underset{H \in \mathcal{H} \setminus \{H_1,\ldots,H_{i-1}\}}{\arg\max}$ $\text{cUCB}_i(H)$ where cUCB_{i-1} is defined in (22)
  // Step ③ – Trialing selected PC
  for $q = 1, \ldots, q_{\max}$ **do**
    Measure training step time as $t_{i,j}$
    Compute $\bar{r}_{i,q}$ and $\sigma^2_{\bar{r}_{i,q}}$ according to (23)
    if $I_{i,q} = 0$ **then**
      break // Early trial termination condition from (2)
  Measure maximum memory usage as m_i
  $\hat{q}_i \leftarrow q$ to track the number of training steps run in round i
  $i \leftarrow i + 1$
return H_{i^*} where $i^* = \arg\max_{i\,:\,(m_i < M_0) \wedge (\hat{q}_i = q_{\max})} \bar{r}_{i,\hat{q}_i}$

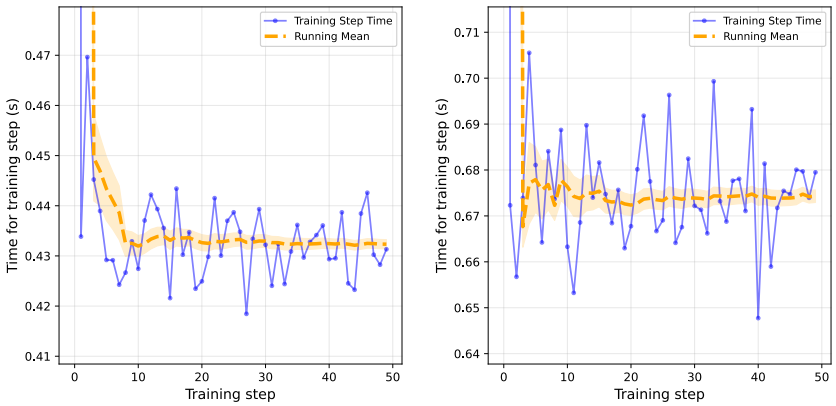


Figure 16: Effect of number of training steps repeatedly measured on the predicted throughput. The solid blue line represents the measured time per training step for an NN training trial. The solid orange line represents the average training time (i.e., reciprocal of throughput) predicted up to that certain training step.

H Additional Details on Experimental Setup

H.1 Training and Hardware Configurations

We list the models used in our experiments in Table 2, along with the allotted time budget for the optimization, and how many repeated measurements are allowed for each trial ($q_{\max}$). All models used are based on the transformer architecture. All of the LLMs have architectures based on the specified model, which are loaded from Huggingface. For the `ViT` case, we use a custom Vision Transformer with 24 hidden layers, 16 attention heads, hidden size of 1024, and intermediate layer hidden size of 4096. For the `MoE` case, we use a Mixture-of-Experts model with 16 hidden layers, 32 attention heads, 32 experts, hidden size of 1024, and intermediate layer hidden size of 1024.

Table 2: Details of models used in our experiments and the corresponding training scenario.

Case	Model	Batch size	Max. seq. length	Time budget	$q_{\max}$	# Rep. exps.
`BERT`	BERT Base Uncased (110M params)	256	256	20 mins	50	10
`Qwen2-1.5b`	Qwen-2 (1.5B params)	64	1024	20 mins	30	10
`Llama3-1b`	Llama 3 (1B params)	64	1024	60 mins	20	5
`Llama2-7b`	Llama 2 (7B params)	256	1024	60 mins	30	5
`Qwen3-32b`	Qwen-3 (32B params)	256	8192	80 mins	30	5
`ViT`	Custom ViT	256	N/A	30 mins	30	5
`MoE`	Custom MoE	64	1024	60 mins	30	5

In Table 3, we list the hardware setups used in our experiments. Here, we see that the experiments are conducted on multiple setups with varying number of nodes, GPU model, and communication bandwidth. The hardware setups used are based on the resources available to the authors. All of the NN training is done on either COLOSSAL-AI or on NEMO which we specify in each of the results.

Table 3: Different hardware setups used in the experiments.

Config. Name	GPU Model (Memory)	GPUs per Node	# Node	Multi-Node Characteristic
8 GPUs	NVIDIA RTX A5000 (24GB)	8	1	-
16 GPUs	NVIDIA RTX 3080 (10GB)	8	2	Docker Overlay Network
32 GPUs	NVIDIA A100 (40GB)	4	8	High-Performance Compute
128 GPUs	NVIDIA GH200 (96GB)	4	32	High-Performance Compute

In all of our plots, we plot the median of whichever reported value with a solid line, and also the lower and upper quartiles with an error band around the solid line across some number of repeated experiments. We report the quantiles instead of the standard deviation since we find that the values are often asymmetrically skewed, and therefore opted to show the quartile values to more accurately represent the distribution of these values. The number of repeated experiments conducted for each scenario is based on how expensive the full optimization would take (with the more expensive scenarios being repeated for a fewer number of times).

In every run of the experiment, the best PC is the one that has the best measured throughput so far and whose trial has been run for $q_{\max}$ steps. In other words, any PC whose trial terminated early will not be considered when we report the best measured throughput so far.

H.2 Other Methods Run for Comparison

We list the other methods that we have run along with their implementation details here:

- RANDOM. This involves randomly selecting a PC from $\mathcal{H}$ to trial in each round until the time budget is exhausted.
- XGBOOST [3]. This method is used by DEEPSPEED [28] to configure the PC and adapted to work with the hyperparameters in our PC. The method involves training an XGBOOST surrogate model based on the measured throughput so far, then selecting the PC with the highest prediction from the surrogate according to the XGBOOST surrogate model to trial next. Like in the implementation from [28], the XGBOOST surrogate model does not model the throughput directly, but the relative rankings of the throughput value. This is repeated until time budget is exhausted.

- R&M-HAT. This involves performing curve-fitting directly on $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ as written in (13) and (14) respectively. The data used to fit are based on randomly selected PCs, which are trialed to obtain the throughput and maximum memory usage. We continue to select and trial PCs until the time budget is exhausted. We then find the optimal hyperparameters $\theta_{\mathcal{R}}$ and $\theta_{\mathcal{M}}$ for $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ respectively by performing curve-fitting with MSE loss, then use the fitted functions to perform a one-shot selection of the best PC.
- VANILLA-BO. This performs BO whose GP has a constant mean and a Matérn kernel with $\nu = 5/2$. The cUCB acquisition function is used for PC selection. The BO loop is implemented using BOTORCH [1].
- OPPA. This is the method proposed in Sec. 4 which involves modifying BO to include parallelism-informed GP prior beliefs and early trial termination being used when trialing the PCs. The hyperparameters for this method are set as stated in App. G.

We note that the methods we compare with are those which rely on measurements from short training trials, as opposed to the class of methods which only rely on constructing an approximation of the training throughput without directly observing any training trials. This is because the methods which only rely on constructing an approximation of the training throughput generally require different input data compared to OPPA, and tend to also operate on a different representation of PC, making them difficult to compare with OPPA. Nonetheless, we provide some comparisons with selected methods of such nature, namely, AMP [15] and NNSCALER [19] in Fig. 9.

I Additional Results

I.1 Plots of Best Measured Throughput vs. Other Quantities

In Fig. 17, we plot the best measured throughput vs. the number of training trials that have been run for experiments corresponding to Figs. 4 and 8. We see that in this view, OPPA still outperforms other methods. While the margin may be smaller in some instances, we see that OPPA is able to achieve the good results more consistently as seen by the error bands when compared to some of the other methods. This also demonstrates that the prior beliefs used in OPPA alone would have helped in achieving a better performance regardless of the early trial termination mechanism in OPPA.

In Fig. 18, we plot the best measured throughput vs. the number of training steps run in the training trials, where the difference in efficiency of OPPA becomes more pronounced. From Figs. 17 and 18, observe that OPPA is both more time- and query-efficient, which is useful when the overhead to run one trial is large, such as when the framework is adapted to run on a cluster with a job scheduler.

I.2 Optimal PCs recovered by OPPA

In Tables 4 and 5, we show the optimal PCs that were found by OPPA for different parallel NN training scenarios on the COLOSSAL-AI and NEMO frameworks, respectively. Note that the optimal PCs chosen match quite well with intuitive choices of PC optimization, such as prioritizing DP for smaller NNs and prioritizing TP or PP for larger NNs. Additionally, we also find that some choices made by OPPA may not match exactly with the intuition of PC optimization, such as when communication overlap is necessary or whether the DP bucket size affects overall throughput. We see that because NN training is not always done on the same hardware, intuitive choices of PC may not always give the maximum throughput, and so should instead be optimized based on actual measurements as well to obtain the most accurate information.

I.3 Generality of OPPA in Catering to Various Parallel Neural Network Training Frameworks

Another advantage of adaptively selecting real NN training trials to run can be seen when we change the parallel training framework used to train the NN. In Fig. 19, we run similar experiments as in Fig. 4, however instead performing training on NEMO [13]. In these experiments, we consider additional hyperparameters within our PC, such as the size of the sequence parallelism dimension. We see that compared to Fig. 4, the obtained maximum throughput are different because the training processes are implemented differently across the two frameworks, and the fact that the tunable

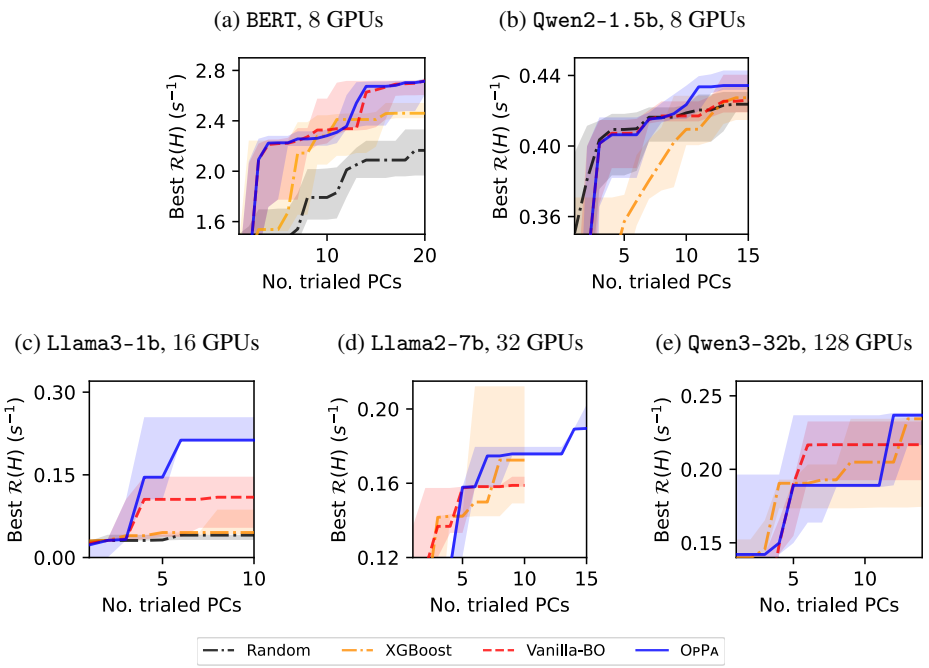


Figure 17: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across (a,b) 10 and (c,d,e) 5 repeated experiments with error bands showing lower and upper quartiles) vs. the number of PCs it has trialed for parallel NN training on (a,b) single- and (c,d,e) multi-node setups using COLOSSAL-AI framework [17]. These results are from the same experiments as in Figs. 4 and 8. Note that for the experiments on 32 GPUs, we are unable to run the optimization beyond the allotted time due to resource constraints and therefore are only able to plot some methods for a fewer number of trialed PCs.

Table 4: Example of optimal PCs selected by OPPA in different parallel NN training scenarios when training on COLOSSAL-AI framework. The listed PCs are based on the found optimal PCs across multiple repeated experiments. A range of value shows that a certain hyperparameter shows a spread across multiple trials (i.e., no strong preference towards one value), while a dash shows that the parallelism dimension associated with that hyperparameter is not used. Note that these may not be the best PCs for the parallel NN training scenarios, but are instead only examples of what are recovered by OPPA.

Hyperparameter	BERT, 8 GPUs (Fig. 4a)	Qwen2-1.5b, 8 GPUs (Fig. 4b)	ViT, 8 GPUs (Fig. 21)	Llama3-1b, 16 GPUs (Fig. 8a)	Llama2-7b, 32 GPUs (Fig. 8b)
DP size (dp)	8	4	8	1	2
TP size (tp)	1	1	1	1	1
PP size (pp)	1	2	1	16	16
DP bucket size	1 – 64	1 – 4096	1 – 64	-	1 – 4096
ZeRO stage	0	1	0	-	1
ZeRO bucket size	-	1 – 64	-	-	64 – 4096
Overlap ZeRO communication	-	True/False	-	-	False
Overlap DP AllGather	-	True/False	-	-	False
# microbatches (mb)	-	8	-	64	32
# model chunks (mc)	-	1 – 2	-	1	1
Overlap P2P for PP	-	True/False	-	False	True/False
Grad. checkpointing	False	False	True	False	False

hyperparameters in the PC are also different. Regardless, we see that even though most methods can eventually recover the optimal PC, OPPA is still able to do so more rapidly and, more importantly, much more consistently. The latter fact can be seen in the error bands of the other methods, which are much larger than that of OPPA. This shows that OPPA does not overfit to any one specific parallel NN training framework, but instead can simultaneously cater to various parallel NN training frameworks.

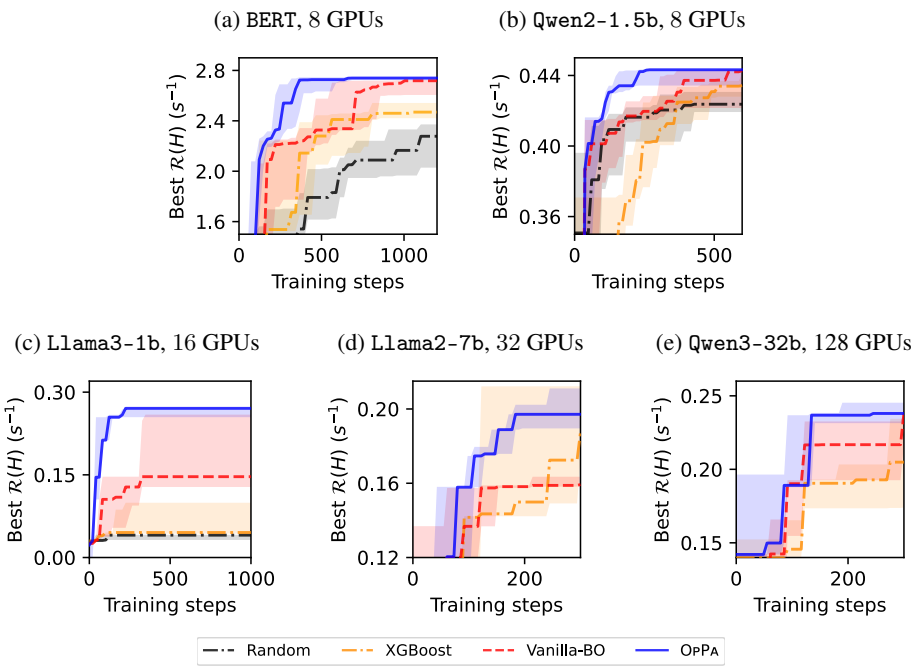


Figure 18: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across (a,b) 10 and (c,d,e) 5 repeated experiments with error bands showing lower and upper quartiles) vs. total number of training steps being run by the method for parallel NN training on (a,b) single- and (c,d,e) multi-node setups using COLOSSAL-AI framework [17]. These results are from the same experiments as in Figs. 4 and 8.

Table 5: Example of optimal PCs selected by OPPA in different parallel NN training scenarios when training on NEMO framework. The interpretation is as in Table 4.

Hyperparameter	BERT, 8 GPUs (Fig. 19a)	Qwen2-1.5b, 8 GPUs (Fig. 19b)	MoE, 8 GPUs (Fig. 22)	Qwen3-32b, 128 GPUs (Fig. 8c)
DP size (dp)	8	4	8	16–32
TP size (tp)	1	1	1	4
PP size (pp)	1	2	1	1 – 2
SP size	1	1	1	1 – 4
EP size	-	-	2	-
CP size	-	-	-	1
DP bucket size	1 – 64	1 – 4096	1 – 4096	1 – 4096
ZeRO stage	0	0	0	0
Overlap ZeRO communication	-	-	-	-
Overlap DP AllGather	False	False	True	True
# model chunks (mc)	-	1	-	-/4
Overlap P2P for PP	-	-/True/False	-	True/False
Grad. checkpointing	False	False	False	False

I.4 PC Optimization for Other NN Architectures

In Fig. 21, we presented the results for PC optimization for ViT model [6]. We see that the results here show that OPPA is able to select more optimal PCs compared to the other methods, consistent with other results in the paper.

Additionally, in Fig. 22, we present the results for PC optimization for transformer-based Mixture-of-Experts (MoE) model [30], which allows for expert parallelism (EP) to be utilized in addition to the existing DP, TP and PP. In our experiments, we consider parallel training of a model with 16 transformer units, 32 attention heads, and 32 experts. Each trial can be run for up to $q_{\max} = 50$ training steps, and in OPPA, we set $q_{\min} = 10$ to account for higher variation in training step times. In the results, we again see that OPPA consistently outperforms the other competing methods. Specifically, even if OPPA may eventually lead to similar results to vanilla BO at the end of the

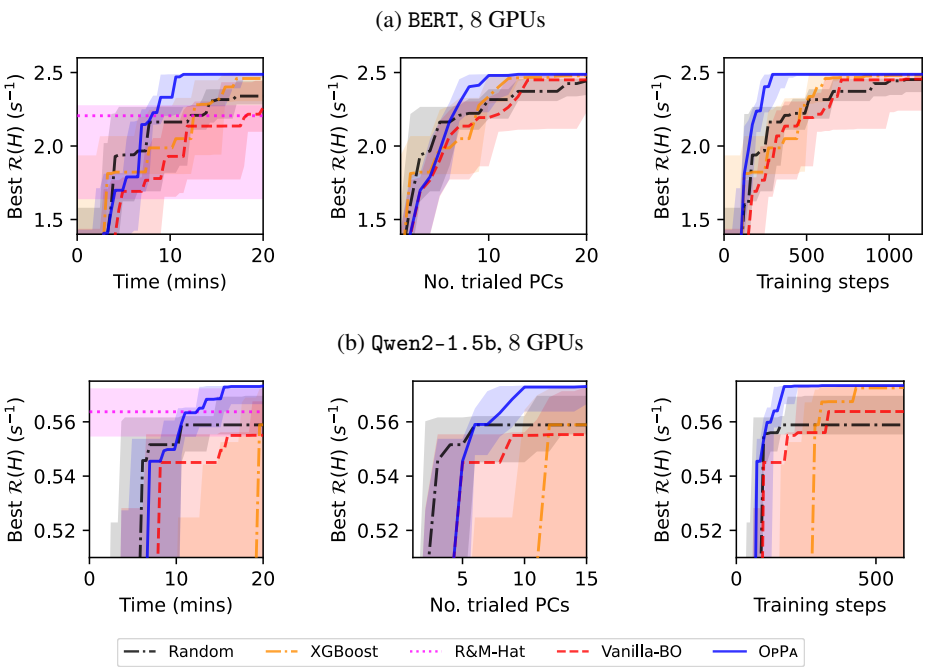


Figure 19: Graphs of best (estimated) measured throughput (higher is better) achieved by each method vs. its incurred time for parallel NN training on a single-node setup using the NEMO framework [13]. Both cases train the NN with identical architectures as the experiments in the respective Figs. 4a and 4b, and are run on a single node.

optimization process, OPPA is able to do so more consistently (lower variance across the repeated experiments), and is able to arrive at the optimal PC much more rapidly and efficiently. These results further demonstrate that OPPA can also generalize to other NN architectures as well.

I.5 Predicted Throughput and Memory by Parallelism-Informed GP Prior Beliefs

In Figs. 23 to 25, we compare the predicted throughput from different surrogate models with the measured throughput across different parallel NN training scenarios. We see that in this case, the parallelism-informed GP prior beliefs allow the throughput to be predicted adequately well, but more importantly, allow for the PC which achieves the highest throughput to also be identified correctly. We find that for BO, surrogate models only need to model the good PCs well in order to select a good PC in the end. Meanwhile, when not using the parallelism-informed prior beliefs, the resulting posterior beliefs learn the patterns much less efficiently or do not learn them at all. This correlates well with the results in the main text where vanilla BO selects a worse PC compared to OPPA. This is similar to the XGBOOST surrogate model which are unable to correctly learn the ranking of the PCs. Meanwhile, $\widehat{R}$ alone is only able to learn the general trend of the throughput, but is unable to capture all of the effects of hyperparameters within the PC, especially those which cannot be explicitly modeled by $\widehat{R}$.

In Fig. 26, we compare the modeled maximum memory with the actual measured value. We present both the actual predicted value and the binary classification results in the form of a confusion matrix. Here, we see a similar trend as before, where using a parallelism-informed prior belief is able to distinguish between PCs which will cause an out-of-memory error and those that will not. Meanwhile, using the GP alone without the parallelism-informed prior belief does not provide enough information for the maximum memory usage to be efficiently learned, while using $\widehat{M}$ alone is unable to capture all of the effects of hyperparameters within the PC.

To additionally demonstrate the interpretability of using a GP as the surrogate model, in Table 6, we show the results for the lengthscales of the Matérn kernel (as given in (15)) which maximizes the log-likelihood. We see that PC hyperparameters that have larger effects on the resulting throughput or

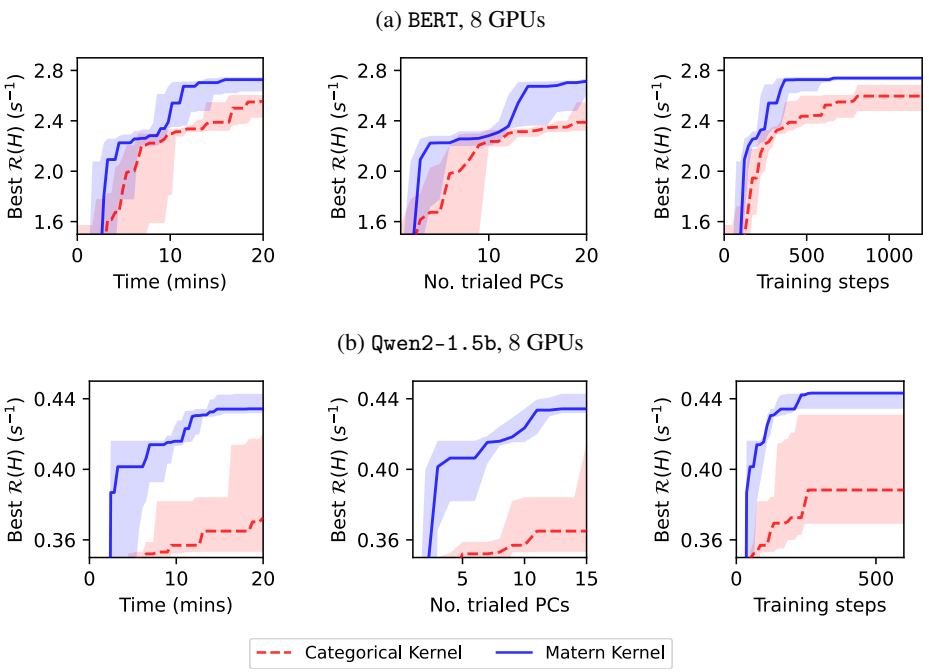


Figure 20: Graphs of best (estimated) measured throughput (higher is better) achieved by OPPA vs. its incurred time when changing from the Matérn kernel to the categorical kernel [4]. Both cases train the NN with identical architectures as the experiments in the respective Figs. 4a and 4b (with the results for the Matérn kernel being exactly the same as in those figures), and are run on a single node.

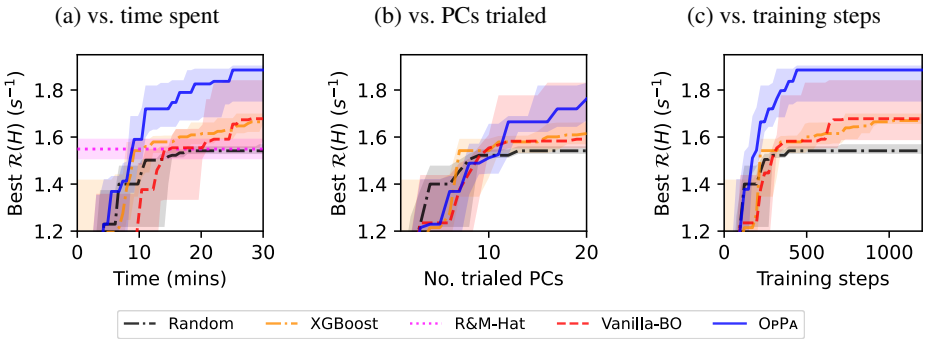


Figure 21: Results for optimizing the PC for training ViT model [6] on the COLOSSAL-AI training framework.

are less well-modeled by our parallelism-informed prior belief will typically correspond to the shorter lengthscales. For example, for the training of BERT, we see that the lengthscales corresponding to the TP dimension size and the number of microbatches (for PP) have shorter lengthscales. This matches our intuition where the throughput would be more sensitive to the change in TP or PP dimension sizes.

I.6 Effects of Prior Misspecification

To investigate the robustness of OPPA against a misspecified prior belief, we perform experiments to see how OPPA behaves as our parallelism-informed prior belief becomes increasingly inaccurate. To do so, we add perturbation terms (whose magnitude is about 25% and 50% that of the maximum throughput obtained) into $\widehat{R}$ and $\widehat{M}$. We present the best measured throughput across 5 repeated experiments in Table 7. We see that even when $\widehat{R}$ and $\widehat{M}$ are adversarially perturbed, we are still

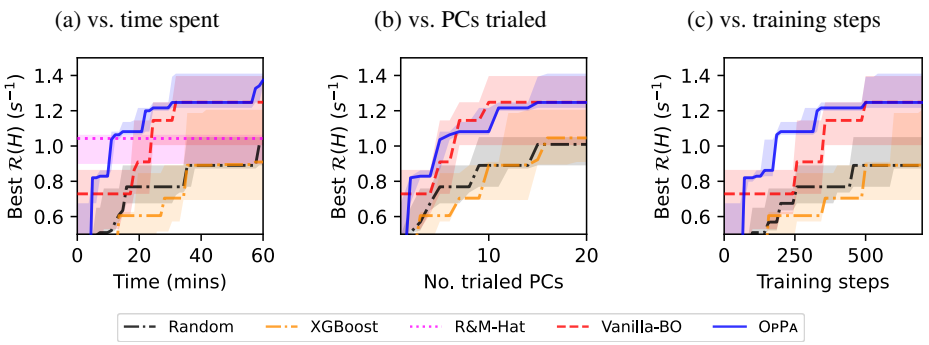


Figure 22: Results for optimizing the PC for training Mixture-of-Experts [30] on the NEMO training framework.

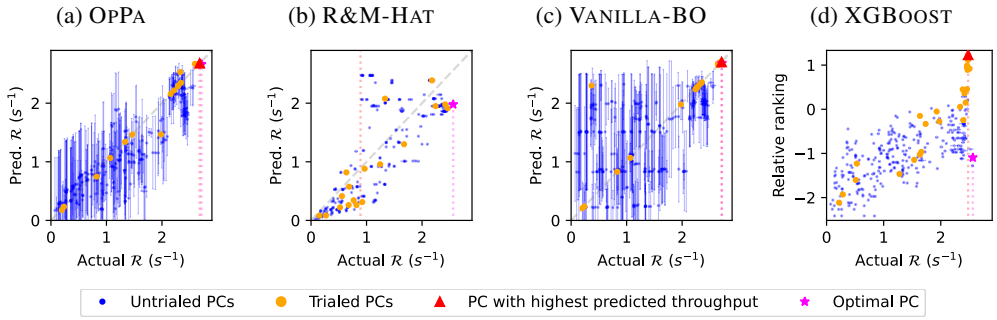


Figure 23: Comparison of modeled throughput vs. the actual throughput for training of `BERT` model on 8 GPUs and using COLOSSAL-AI framework after 20 NN training trials. Each untrialed PC for OPPA and VANILLA-BO is associated with error bars showing twice the GP posterior standard deviation.

able to obtain good performances even if the convergence is slightly slower. This suggests that even in this extreme case, the GP is able to correct for the inaccuracies in the parallelism-informed prior belief anyway.

In practice, prior misspecification is typically due to the parallelism-informed prior belief not being sufficiently complex to match the actual parallel NN training dynamics arising from incomplete or inaccurate domain knowledge about the actual system, rather than due to an adversarial construction of the cost function. This is the case for our choice of $\widehat{R}$ and $\widehat{M}$ where there is a discrepancy between the predicted throughput and maximum memory usage and the actual measurements, as demonstrated in Fig. 6b. Under practical scenarios, we therefore would not expect the results to be as extreme as what we have seen here, and that a GP should be able to effectively model the throughputs anyway.

I.7 Additional Results for Ablation Studies of Efficiency Tricks in OPPA

In Fig. 27, we demonstrate how early trial termination and parallelism-informed GP prior beliefs affect the best measured throughput. First, we see that the parallelism-informed prior beliefs does not harm the performance, and in fact shows clear benefit as seen in the `Qwen2-1.5b` scenario, which is due to the increased complexity in the training setup which benefit from additional domain knowledge being exploited by OPPA. Meanwhile, with early trial termination, it can be observed that the performance is consistently better in terms of time and number of training steps needed, while not sacrificing the performances when considering the number of PCs that are trialed to achieve a certain performance.

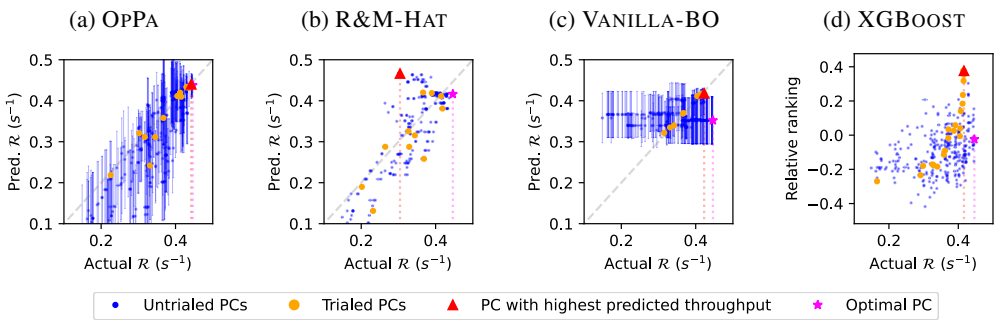


Figure 24: Comparison of modeled throughput vs. the actual throughput for training of `Qwen2-1.5b` model on 8 GPUs and using COLOSSAL-AI framework after 20 NN training trials. Each untrialed PC for OPPA and VANILLA-BO is associated with error bars showing twice the GP posterior standard deviation.

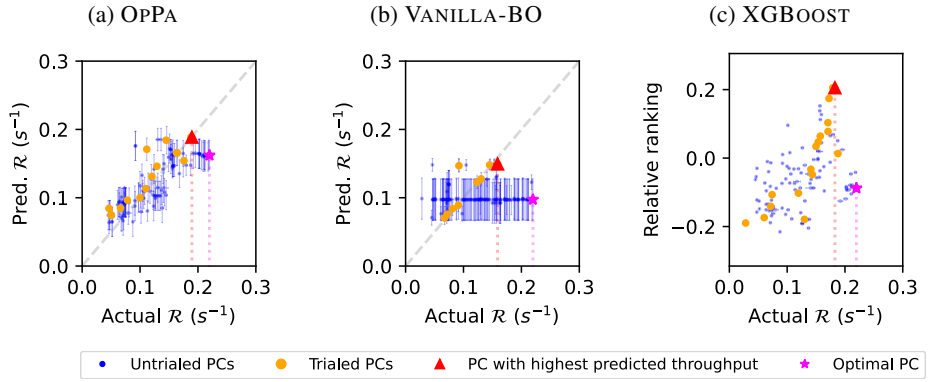


Figure 25: Comparison of modeled throughput vs. the actual throughput for training of `Llama-2` model on 32 GPUs and using COLOSSAL-AI framework for, in order, OPPA after 20 NN training trials, VANILLA-BO after 10 NN training trials and XGBOOST after 10 NN training trials. Each untrialed PC for OPPA and VANILLA-BO is associated with error bars showing twice the GP posterior standard deviation. Note that among the three methods, only OPPA ran for up to 20 trials given the time constraint.

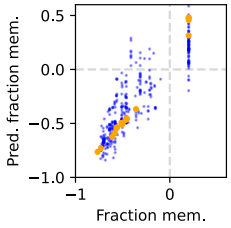
I.8 Effect of $q_{\min}$ on OPPA's Performance

In Fig. 28, we see how the choice of $q_{\min}$ affects the best measured throughput. For the `BERT` case, it can be observed that early trial termination clearly improves the time required for optimization, as seen where when $q_{\min} = q_{\max}$, the obtained PC is the worst. For a smaller $q_{\min}$, it can be observed that there is less difference in the best measured throughput, which is due to there still being sufficient trails to measure the throughput accurately. At smaller values of $q_{\min}$, it is also likely that the required time to trial a PC is dominated by the setup phase (i.e., starting the program script and loading the models) rather than the time for the training itself. However, when the number of training steps for the optimization is plotted, it can be observed that a smaller $q_{\min}$ will require fewer training steps to arrive at the same optimal PC. However, the performance worsens when $q_{\min}$ is excessively small, likely since the value obtained is too noisy to give good information. For the `Qwen2-1.5b` case, similar trends can be seen where reducing $q_{\min}$ is able to reduce the time and the number of training steps required to find the optimal PC up to a certain point.

J Limitations and Broader Impacts

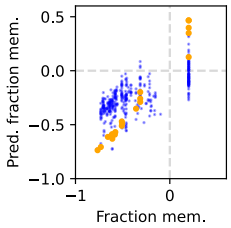
While there are many other factors of parallel NN training that we can optimize, we have mainly considered hyperparameters commonly found in existing parallel NN training frameworks that are usually manually tuned by practitioners. We believe that OPPA can also be extended to optimize

(a) OPPA



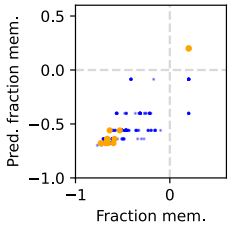
	Predicted Feasible	Predicted OOM
Actual Feasible	296	16
Actual OOM	4	84

(b) VANILLA-BO



	Predicted Feasible	Predicted OOM
Actual Feasible	309	3
Actual OOM	54	34

(c) R&M-HAT



	Predicted Feasible	Predicted OOM
Actual Feasible	312	0
Actual OOM	23	65

Figure 26: Comparison of modeled maximum memory usage vs. the actual maximum memory usage for training of `Qwen2-1.5b` model on 8 GPUs and on COLOSSAL-AI framework after 20 PCs trialed. In each row, the left hand graph shows the predicted vs. the actual value (with the predictive standard deviation omitted for clarity), while the right hand table is the confusion matrix when using the surrogate to classify between PCs which result in out-of-memory errors and those which do not.

in other more sophisticated PC search spaces that may arise in practice as well. In a wider picture, our work allows the training of NNs to be done more efficiently since it reduces the amount of time required to perform some number of training steps when training NNs on multiple GPUs. While this may allow faster development of NNs for both good and bad use cases, we still believe our work provides a net positive impact since the higher efficiency in NN training would reduce the required GPU compute time, directly making the training process more resource efficient.

Table 6: Example of the log lengthscales learned by the kernel of the GP for the throughput. The bolded value are to highlight hyperparameters with particularly shorter lengthscales.

Hyperparameter	BERT, 8 GPUs	Qwen2-1.5b, 8 GPUs
DP size (dp)	1.877	5.233
TP size (tp)	**−1.071**	6.024
PP size (pp)	2.293	**−1.849**
DP bucket size	3.484	5.994
ZeRO stage	0.434	5.279
ZeRO bucket size	3.402	6.516
Overlap ZeRO communication	3.949	7.581
Overlap DP AllGather	3.822	7.589
# microbatches (mb)	**−1.322**	**−1.741**
# model chunks (mc)	−0.293	0.124
Overlap P2P for PP	3.854	6.974
Grad. checkpointing	−0.334	6.586

Table 7: Effect of perturbing the parallelism-informed prior belief on the resulting optimization process. The values reported are the median throughput obtained for the PC found after certain number of minutes of running OPPA with the perturbed prior belief.

Percent perturbation magnitude	10 mins. of search	20 mins. of search
0 (original prior)	0.425	0.446
About 25	0.424	0.447
About 50	0.415	0.447

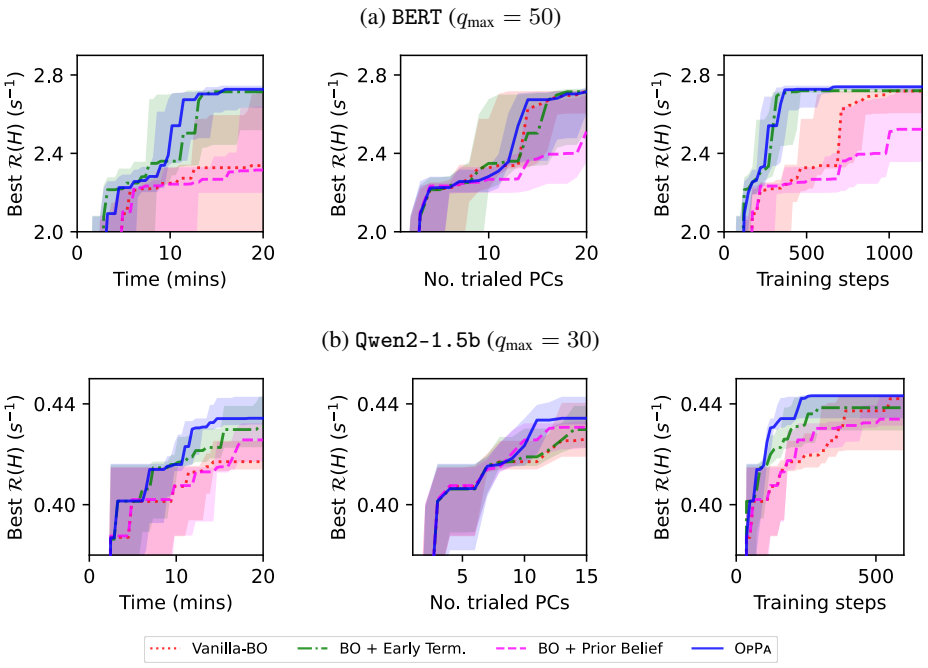


Figure 27: Effects of different efficiency tricks in OPPA on the optimal PC found. We present the measured throughput vs., from left to right, the time the method has been run for, the number of unique PCs trialed, and the number of training steps that have been run across all of the trials.

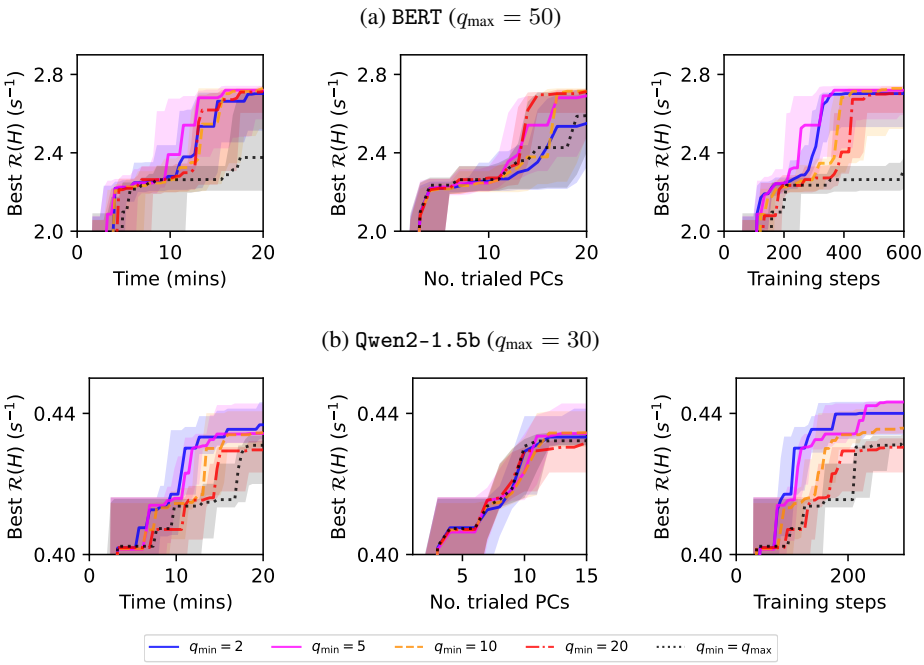


Figure 28: Effect of $q_{\min}$ on the optimal PC found. In each row, we present the results in a certain training setup, where we present the measured throughput vs., from left to right, the time the method has been run for, the number of unique PCs trialed, and the number of training steps that has been run across all of the trials.