BAYESIAN OPTIMIZATION WITH EARLY TRIAL TERMINATION FOR SPEEDING UP PARALLEL NEURAL NETWORK TRAINING

Anonymous authors
Paper under double-blind review

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 (Devlin et al., 2019; Vaswani et al., 2017), large language models (LLMs) (OpenAI et al., 2024; Touvron et al., 2023), and multimodal LLMs (Liu et al., 2023; Radford et al., 2021), 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 (Rajbhandari et al., 2020; Zhao et al., 2023), pipeline (Huang et al., 2019; Narayanan et al., 2019), tensor (Shoeybi et al., 2020), and their combinations, also referred to as *multi-dimensional parallelism* (Li et al., 2023a; Rasley et al., 2020; Shoeybi et al., 2020).

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 (Kuchaiev et al., 2019; Li et al., 2023a; Rasley et al., 2020; Shoeybi et al., 2020), 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 PC (Li et al., 2022; Zhang et al., 2024; Zheng et al., 2022) *require strong assumptions* on the hardware and parallel NN

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

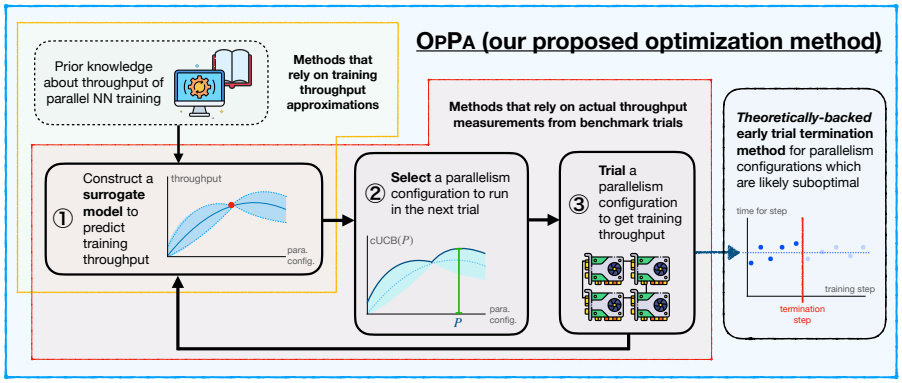


Figure 1: The key idea of OPPA is to exploit domain knowledge of parallel NN training & early trial termination to rapidly find the parallelism configuration (PC) with the highest training throughput.

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 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 methods, but *cannot utilize measured throughputs nor prior domain knowledge*, hence 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) (Li et al., 2020) 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) (Rajbhandari et al., 2020) or Fully-Sharded Data Parallel (FSDP) (Zhao et al., 2023), have been proposed to perform sharding of

NN weights or gradients to avoid full NN replication. Furthermore, tensor parallelism (TP) (Shoeybi et al., 2020; Bian et al., 2021) and pipeline parallelism (PP) (Huang et al., 2019; Narayanan et al., 2019) 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. Sec. A discusses DP, TP, and PP more. Besides, other types of parallelism (e.g., sequence Li et al. (2023b), expert Rajbhandari et al. (2022)) have also been developed.

Multi-dimensional parallelism. Several frameworks (Rasley et al., 2020; Shoeybi et al., 2020; Li et al., 2023a) 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. 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 (Li et al., 2023a; Lin et al., 2024; Wagenländer et al., 2024). 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 time between pipeline stages may dominate the computation of the 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 DEEPSPEED (Rasley et al., 2020) have implemented PC optimization methods (DeepSpeed, 2026) 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 focus on constructing a more accurate surrogate model to approximate the throughputs of different PCs (Li et al., 2022; Zheng et al., 2022; Zhang et al., 2024) and select the optimal PC with fewer or even no training trials at all. These methods, however, *implicitly assumes 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. Also, 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 (Gelbart et al., 2014; Frazier, 2018). 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 (Rasmussen & Williams, 2006). 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 (Jones et al., 1998), upper confidence bound (Srinivas et al., 2012)) 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. Sec. 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 (Lei et al., 2021; Rainforth et al., 2023) and material design (Zhang et al., 2020). 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 (Snoek et al., 2012). 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 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

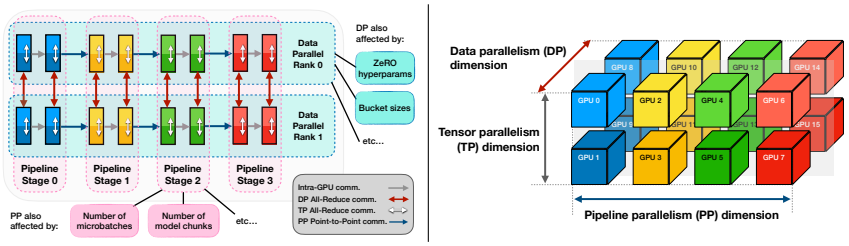


Figure 2: (Left) Visualization of a parallelism configuration (PC) with tunable hyperparameters. (Right) Visualization of GPU allocation for 3D parallelism according to the dimension sizes.

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. Sec. 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 $\mathcal{P} \subset \mathbb{Z}^d$ be a set of all possible PCs such that each PC denotes a vector of hyperparameters. The goal of our problem is to find the optimal PC $P^* \in \mathcal{P}$ with the *highest 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 $P \in \mathcal{P}$, let $\mathcal{R}(P)$ and $\mathcal{M}(P)$ be the throughput and the maximum memory usage, respectively. Then, our problem of finding the optimal PC $P^* \in \mathcal{P}$ can be formulated as of a constrained maximization:

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

When querying a PC P, 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 P, 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: $\mathcal{R}(P) \approx \bar{r}_{q_{\max}} \triangleq q_{\max}^{-1} \sum_{j=1}^{q_{\max}} t_j^{-1}$. 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 $\bar{r}_q$. When designing our method, we note two other characteristics of the PC optimization problem:

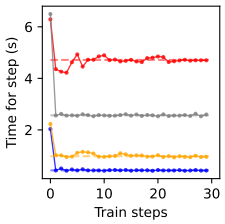


Figure 3: Time taken for each training step for a few training trials with various PCs. Each color represents a different PC used, and the dashed line represents the throughput estimate for that PC. Note the fluctuations in the time taken for each training step.

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 should be trialed on real hardware, *running a full trial incurs a high cost*. This is especially true with suboptimal PCs since each training step on a suboptimal PC would incur more time. This motivates us to design an optimization method 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 𝒫 of all valid PCs
for i = 1, 2, ... do
    // Step ① – Modeling ℛ and ℳ (Sec. 4.1)
    Obtain GP predictive/posterior beliefs for ℛ and 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 P_i ∈ 𝒫 with cUCB acquisition function (22)
    // Step ③ – Trialing the selected PC (Sec. 4.3)
    for training steps q = 1, ..., q_max do
        Measure time t_{i,q} taken for training step q
        if I_{i,q} = 0 then terminate trial    // from (2)
    end for
    if time budget exhausted then break
end for
return P_i with the best (estimated) measured throughput
```

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

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

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. This 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: $\widehat{\mathcal{R}}(P;\theta_{\mathcal{R}}) = \left[\widehat{\mathcal{T}}_{\text{comp}}(P;\theta_{\mathcal{R}}) + \widehat{\mathcal{T}}_{\text{comm}}(P;\theta_{\mathcal{R}})\right]^{-1}$ where $\theta_{\mathcal{R}}$ are *learnable parameters* (as opposed to fixed constants) that estimate factors such as the communication speed between GPUs or the size of message being communicated. For $\widehat{\mathcal{T}}_{\text{comp}}$, we consider the additional time arising from the PP pipeline bubble (Narayanan et al., 2019). For $\widehat{\mathcal{T}}_{\text{comm}}$, inspired by Xiong et al. (2024), we consider an idealized training scenario (as 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 $\theta_{\mathcal{R}}$. Further details about $\widehat{\mathcal{R}}$ are in Sec. D.1. For $\widehat{\mathcal{M}}$, we consider the memory required per GPU to store the

NN weights and gradients for backpropagation, using learnable parameters to estimate the overall memory required, as detailed in Sec. 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 (Rasmussen & Williams, 2006) 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 $\theta_k = \{\boldsymbol{\ell}, \sigma_k\}$ that can be learned using the actual measurements, as described below. In Sec. 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

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

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 (Rasmussen & Williams, 2006) (see (16) in Sec. D.4), and the GP posterior beliefs for the throughput and maximum memory usage of any PC $P \in \mathcal{P}$ can be obtained from the prior beliefs (see (17) to (21) in Sec. D.4).

4.2 Selecting the Next PC to Trial

In Step ②, the next PC $P_i \in \mathcal{P}$ to trial in round i is selected to maximize the constrained upper confidence bound (cUCB) acquisition function (Srinivas et al., 2012; Wilson et al., 2017) based on the surrogate models constructed in ①, as detailed further in Sec. E.1. cUCB trades off between *exploring* uncertain 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 optimal PC (Jones et al., 1998; Gelbart et al., 2014), 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 P_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 $\mathcal{R}(P_i)$ 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 Sec. 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 Sec. 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 $\mathcal{R}(P_i)$ 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 $\mathcal{R}(P_i)$ with a small enough variance for us to determine that P_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 (a) no more than q_{min} training steps have been run, or (b) the estimated throughput of P_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 $I_{i,\hat{q}_i} = 0$. As explained in Sec. 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 below:

Theorem 4.1 (Informally stated in terms of $\mathcal{R}$)**.** *Consider the practical setting of $\mathcal{M}(P) \leq M_0$ for all $P \in \mathcal{P}$. 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 $\sum_{i=1}^{N} (\mathcal{R}(P^*) - \mathcal{R}(P_i)) = \widetilde{\mathcal{O}}(\sqrt{N/q_{\text{min}}})$, and for $i = 2, \ldots, N$, if $\mathcal{R}(P_i) < \max_{k \in \{1,\ldots,i-1\}} \mathcal{R}(P_k)$, then $\hat{q}_i < q_{\text{max}}$.*

In Sec. 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. Note that Thm. 4.1 assumes the setting where $\mathcal{M}(P) \leq M_0$ for all $P \in \mathcal{P}$ because running the NN training trial on the PC P when $\mathcal{M}(P) > M_0$ would lead to an out-of-memory error and hence the throughput cannot be measured; so, our theoretical analysis can omit the case of $\mathcal{M}(P) > M_0$ 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.

Sec. G gives the complete pseudocode for OPPA with steps ①, ②, and ③. In Sec. H, we elaborate how OPPA is different from existing BO approaches and why it is more suited for PC optimization.

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. Sec. I.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 (Chen & Guestrin, 2016) that is used by DEEPSPEED (Rasley et al., 2020; DeepSpeed, 2026)), 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). Sec. I.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 Sec. J.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 (Devlin et al., 2019) on a single node of 8 GPUs using the COLOSSAL-AI framework (Li et al., 2023a). We observe that OPPA, which exploits parallelism-informed GP prior beliefs and early trial termination, outperforms all other methods, including vanilla BO. We find that OPPA 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 (Yang et al., 2024) where OPPA again finds a better PC than other methods. As the Qwen model is larger, OPPA now prefers a mix of DP and PP to reduce memory use while minimizing communications across GPUs. Sec. J.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 to avoid wasting time: Though OPPA switches from one NN training trial to the next more often, this

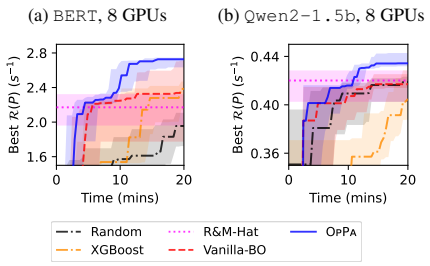


Figure 4: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (plotted as medians across 10 repeated experiments with error bands showing lower and upper quartiles) vs. incurred time for the optimization on a single-node setup using COLOSSAL-AI framework (Li et al., 2023a).

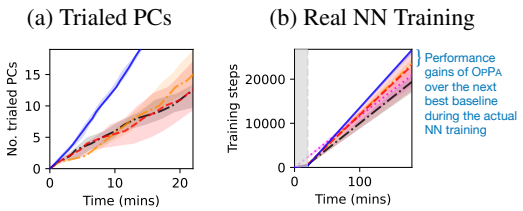


Figure 5: Comparing the efficiency of OPPA with that of other tested methods based on the experiments in Fig. 4a. Fig. 5a shows the no. of PCs each method has trialed. Fig. 5b shows the no. of training steps that can be run by each method during PC optimization (i.e., shaded gray region on the left)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

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 Sec. J.3, we report results on similar experiments as that in Fig. 4, except that the parallel NN training is performed using the NEMO framework (Kuchaiev et al., 2019) where the PC now also considers the sequence parallelism (Li et al., 2023b). In Sec. J.4, we report results comparing the performances of OPPA and the other tested methods in optimizing the PC for parallel training of vision transformers (ViT) and mixture of experts (MoE); the latter includes the size of expert parallelism dimension (Rajbhandari et al., 2022) in the PC. In these scenarios, OPPA outperforms the other methods, showing 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. Sec. J.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. Sec. J.7 gives more results. Sec. J.8 shows the effect of $q_{\min}$ (2) on OPPA's performance, revealing minimal degradation for a small $q_{\min}$.

Multi-node setups. The additional inter-node communication makes calculating the throughputs less straightforward, and the larger number of candidate PCs complicates the optimization. Fig. 8a shows results for PC optimization for parallel training of a smaller NN (Grattafiori et al., 2024) with 1 billion parameters on a commodity cluster with 16 GPUs, which is a typical setup the practitioners use on existing hardware. Figs. 8b and 8c show results for PC optimization for parallel training of larger NNs (Touvron et al., 2023; Yang et al., 2025) on high-performance computing clusters with 32 and 128 GPUs, respectively. In all these parallel NN training scenarios (including *different scales* of

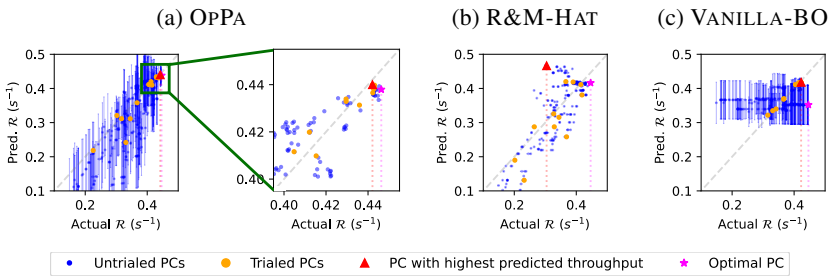


Figure 6: Graphs of throughputs predicted by different methods' surrogate models vs. actual measured throughputs after 20 NN training trials based on an experiment in Fig. 4b. The zoomed-in region (green box) shows PCs with high predicted and measured throughputs. Each untrialed PC in Figs. 6a and 6c 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 $\widehat{\mathcal{R}}$ gives deterministic predictions.

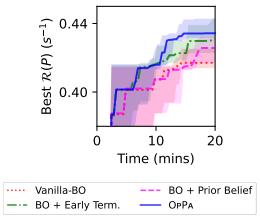


Figure 7: Effects of early trial termination & parallelism-informed GP prior beliefs on best (estimated) measured throughput achieved by OPPA vs. its incurred time based on the experiments in Fig. 4b.

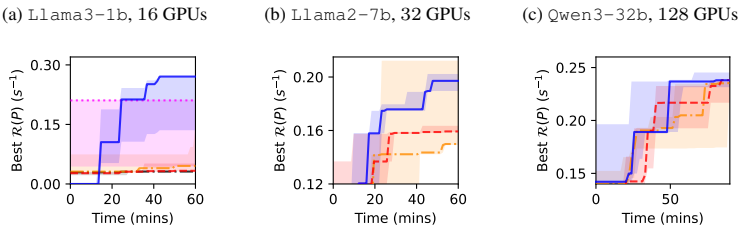


Figure 8: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (plotted as medians across 5 repeated experiments with error bands showing lower and upper quartiles) vs. incurred time for optimization on a multi-node setup using the (a, b) COLOSSAL-AI (Li et al., 2023a) and (c) NEMO Kuchaiev et al. (2019) frameworks (with the latter including sizes of sequence and context parallelism dimensions as hyperparameters in the PC). See Fig. 4 for the legend. Figs. 8b and 8c do not show results all methods due to limited compute resources.

multi-GPU hardware setups, training frameworks, and sizes of *new parallelism dimensions* as tunable hyperparameters in the PC (Fig. 8c)), OPPA outperforms all other tested methods and also performs more reliably (with smaller error bands). Also, though vanilla BO performs worse than OPPA, it outperforms other non-BO methods due to its ability to trade off between exploration vs. exploitation.

Fig. 9 shows that OPPA outperforms methods that do not utilize NN training trials but instead use more complex approximations of training throughputs of different PCs compared to $\widehat{\mathcal{R}}$ (e.g., AMP (Li et al., 2022), NNSCALER (Lin et al., 2024)). This is due to OPPA being able to adaptively select promising PCs using the measured throughputs from real NN training trials and exploit the nontrivial tricks to boost the efficiency of BO.

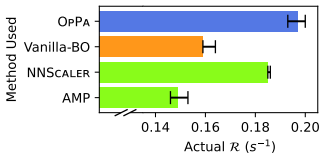


Figure 9: Best (estimated) measured throughput achieved by each method (i.e., plotted as medians across 5 repeated experiments with error bars showing the quartiles) based on results in Fig. 8b.

6 CONCLUSION

This paper describes OPPA that boosts the efficiency of constrained BO by novelly exploiting parallelism-informed prior beliefs (Sec. 4.1) and an early trial termination mechanism (Sec. 4.3) to rapidly find the PC with the highest training throughput for parallel training of NNs with varying architectures, training frameworks, and multi-GPU hardware setups. Despite so, Thm. 4.1 theoretically guarantees that OPPA achieves sublinear regret. As shown in Sec. 5, OPPA can be easily adapted to cater to unseen hyperparameters in a PC. In practice, the parallelism-informed prior beliefs can incorporate additional domain knowledge of specific NN architectures or parallel training frameworks, which would trade off generality for a greater efficiency of OPPA.

AI USE STATEMENT

We have not used generative AI tools for the theoretical claims, data generation, or interpreting the results. We used generative AI tools for code generation (particularly for the parallel NN training and its integration into the optimization loop) and for minor adjustments of grammar and phrasing in the paper. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work does not involve human subjects, potentially harmful insights, methodologies and applications, potential conflicts of interest and sponsorship, discrimination/bias/fairness concerns, privacy and security issues, legal compliance, or research integrity issues.

REPRODUCIBILITY STATEMENT

For the theoretical results, the full result along with assumptions are elaborated in Sec. F.3. For the empirical results, the code to replicate the results are provided in the supplementary material, and the hardware and training setup ran are elaborated in Sec. I.

REFERENCES

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Proc. NeurIPS*, 2020.

Zhengda Bian, Qifan Xu, Boxiang Wang, and Yang You. Maximizing parallelism in distributed training for huge neural networks. arXiv:2105.14450, 2021.

Mickaël Binois, Jiangeng Huang, Robert B. Gramacy, and Mike Ludkovski. Replication or Exploration? Sequential Design for Stochastic Simulation Experiments. *Technometrics*, 2019.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. SIGKDD*, 2016.

Julia Couto. Kernel k-means for categorical data. In *Proc. IDA*, 2005.

Zhongxiang Dai, Quoc Phong Nguyen, Sebastian Shenghong Tay, Daisuke Urano, Richalynn Leong, Bryan Kian Hsiang Low, and Patrick Jaillet. Batch Bayesian Optimization for Replicable Experimental Design. In *Proc. NeurIPS*, 2023.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16×16 words: Transformers for image recognition at scale. In *Proc. ICLR*, 2021.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proc. ICML*, 2018.

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

Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. Bayesian optimization with unknown constraints. In *Proc. UAI*, 2014.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai

Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The Llama 3 herd of models. arXiv:2407.21783, 2024.

Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 2021.

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

Hideaki Ishibashi, Masayuki Karasuyama, Ichiro Takeuchi, and Hideitsu Hino. A stopping criterion for Bayesian optimization by the gap of expected minimum simple regrets. In *Proc. AISTATS*, volume 206, 2023.

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

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proc. NeurIPS*, 2018.

Johannes Kirschner and Andreas Krause. Information directed sampling and bandits with heteroscedastic noise. In *Proc. COLT*, 2018.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proc. AISTATS*, 2017.

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

Bowen Lei, Tanner Quinn Kirk, Anirban Bhattacharya, Debdeep Pati, Xiaoning Qian, Raymundo Arroyave, and Bani K. Mallick. Bayesian optimization with adaptive surrogate models for automated experimental design. *npj Computational Materials*, 7(194), 2021.

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

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. PyTorch distributed: Experiences on accelerating data parallel training. In *Proc. VLDB*, 2020.

Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-AI: A unified deep learning system for large-scale parallel training. In *Proc. ICPP*, 2023a.

Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective. In *Proc. ACL*, 2023b.

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

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Proc. NeurIPS*, 2023.

Anastasia Makarova, Ilnura Usmanova, Ilija Bogunovic, and Andreas Krause. Risk-averse heteroscedastic Bayesian optimization. In *Proc. NeurIPS*, 2021.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proc. SOSP*, 2019.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng,

Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C. J. Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 technical report. arXiv:2303.08774, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proc. ICML*, 2021.

Tom Rainforth, Adam Foster, Desi R. Ivanova, and Freddie Bickford Smith. Modern Bayesian experimental design. *Statist. Sci.*, 39(1):100–114, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *Proc. SC*, 2020.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proc. ICML*, 2022.

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

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. ICLR*, 2017.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv:1909.08053, 2020.

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

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, 2012.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh

Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. NeurIPS*, 2017.

Marcel Wagenländer, Guo Li, Bo Zhao, Luo Mai, and Peter Pietzuch. Tenplex: Dynamic parallelism for deep learning using parallelizable tensor collections. In *Proc. SOSP*, 2024.

James T. Wilson. Stopping Bayesian optimization with probabilistic regret bounds. In *Proc. NeurIPS*, 2024.

James T. Wilson, Riccardo Moriconi, Frank Hutter, and Marc Peter Deisenroth. The reparameterization trick for acquisition functions. arXiv:1712.00424, 2017.

Jian Wu, Saul Toscano-Palmerin, Peter I. Frazier, and Andrew Gordon Wilson. Practical Multi-fidelity Bayesian Optimization for Hyperparameter Tuning. *arXiv:1903.04703*, 2019.

Dian Xiong, Li Chen, Youhe Jiang, Dan Li, Shuai Wang, and Songtao Wang. Revisiting the time cost model of AllReduce. arXiv:2409.04202, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. arXiv:2407.2178, 2024.

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

Shiwei Zhang, Lansong Diao, Chuan Wu, Zongyan Cao, Siyu Wang, and Wei Lin. HAP: SPMD DNN training on heterogeneous GPU clusters with automated program synthesis. In *Proc. EuroSys*, 2024.

Yichi Zhang, Daniel W. Apley, and Wei Chen. Bayesian optimization for material design with mixed quantitative and qualitative variables. *Scientific Reports*, 10(4924), 2020.

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

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

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) (Li et al., 2020) where a batch of training data is split and distributed to each GPU and then fed into its local replica of the NN, after which the NN gradients from all the GPUs are synchronized. While simple and often the fastest, the naive DP technique requires replicating the NN 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) (Rajbhandari et al., 2020) in the DeepSpeed package and the fully-sharded data parallel (FSDP) technique (Zhao et al., 2023). Though these techniques reduce the maximum memory usage for DP, their effectiveness can still depend heavily on the choice of their hyperparameter values. For example, the hyperparameters of ZeRO can be tuned so that one or more of the optimizer states, NN gradients, or NN weights are sharded between GPUs. The choice of sharded items may reduce the memory usage on each GPU at the cost of increased communications between GPUs, which in turn reduces the throughput of the parallel 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 matrix multiplication operations can be performed in a distributed manner, allowing the possibility of operations involving larger tensors. Initially, TP involves sharding a tensor along a single dimension (i.e., either row-wise or column-wise sharding) (Shoeybi et al., 2020). However, it has since also incorporated sharding tensors across multiple dimensions as well (Bian et al., 2021).

Pipeline parallelism. In pipeline parallelism (PP) (Huang et al., 2019; Narayanan et al., 2019), 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 memory required at any one time becomes smaller, allowing for the training of larger NNs at the cost of more sequential operation rounds and communications between GPUs. The trade-off between training speed and maximum memory usage can be further controlled based on the size of microbatches and the number of NN chunks.

B Technical Primer on Gaussian Process and Bayesian Optimization

In this section, we provide a technical overview of Gaussian process (GP) regression and Bayesian optimization (BO), which is adapted from (Rasmussen & Williams, 2006; Frazier, 2018).

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 $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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

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

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

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 learnable parameters θ, which specify the prior assumption of the functions drawn from the random process. For example, many kernels representing the prior covariance include lengthscale, which govern how correlated the function output is when

a certain input dimension is further away. One method to find the optimal θ is by finding θ which maximizes the marginal log-likelihood $\log p(\mathcal{D}|\theta)$ of the observed data $\mathcal{D}$.

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 $\mathcal{D}_t = (\mathbf{X}_t, \mathbf{y}_t) = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ 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 (Jones et al., 1998)

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

or the upper confidence bound (Srinivas et al., 2012)

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

where $\beta_t > 0$ is a constant that may vary with t. In all of these acquisition functions, a trade-off 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 in the Parallelism Configuration

In Tab. 1, we list the hyperparameters in the parallelism configuration (PC) and the range of their values. Note that there exists constraints on the feasible hyperparameter values; for example, the fact that we must have $\texttt{dp} \cdot \texttt{tp} \cdot \texttt{pp} \cdot \texttt{cp} = \texttt{n_gpus}$, or that ZeRO-3 may only be compatible with some parallelism dimensions depending on the training framework. In our experiments, we generate all of the feasible candidate PCs before starting the optimization, such that they can be checked against these constraints. Some hyperparameters are also set to a specific default value when it will not affect the parallel NN training instance in that case; for example, if $\texttt{pp} = 1$ (i.e., PP not performed) then we require $\texttt{mb} = \texttt{mc} = 1$. In the actual implementation, we generate all of the feasible PCs beforehand, and perform optimization over the (finite) PC candidates.

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 P, P', P'' where the times per training step are given by $\mathcal{T}(P) = 0.3$, $\mathcal{T}(P') = 0.4$, and $\mathcal{T}(P'') = 0.5$. In this case, P would be the best PC out of the three. We see here the gap of the time per training step between P and P' is $\mathcal{T}(P') - \mathcal{T}(P) = 0.1$, and the same gap size for P' and P'' of $\mathcal{T}(P'') - \mathcal{T}(P') = 0.1$. Meanwhile, the gap between the throughput of the two PCs would be $\mathcal{R}(P) - \mathcal{R}(P') = \mathcal{T}(P)^{-1} - \mathcal{T}(P')^{-1} = 3.\bar{3} - 2.5 = 0.8\bar{3}$

and $\mathcal{R}(P') - \mathcal{R}(P'') = \mathcal{T}(P)^{-1} - \mathcal{T}(P')^{-1} = 2.5 - 2 = 0.5$. 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 differ 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. Framing our problem as a maximization also makes our problem consistent with Bayesian optimization works which typically considers a maximization problem.

D Additional Discussion of Surrogate Models used by OPPA

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

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

- *Individual modeling the effect on the throughput from each parallelism dimension.* We model communication from DP and TP via All-Reduce–style collectives, and communication from PP via point-to-point transfers. For computation costs, we also explicitly consider pipeline bubbles that occur during the computation.
- *General assumptions on the topology of the nodes.* 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 and letting stage-specific latency/overlap differences be absorbed by learnable parameters $\theta_{\mathcal{R}}$. We also the intra- and inter-node communications are separately modeled.
- *Generalized across different training scenarios.* Rather than hard-coding device and network constants, we use a set of learnable parameters whose values are learned from the estimates of the throughput. This keeps $\widehat{\mathcal{R}}$ generalized across NNs, data types, and hardware configurations while preserving the correct asymptotic trends for how each hyperparameter in the PC affects the throughput.

To model the throughput, we will instead consider 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 (which includes the tensor operations required and possibly any sequential dependency in those operations) and for the communication (which is from any intra- and inter-node communications).

To model 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 (Narayanan et al., 2019). This additional computation time from PP, visualized in Fig. 10, is roughly equal to

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

where $\texttt{mb}$ is the number of microbatches used in PP (set to 1 when PP is not used), $\texttt{mc}$ 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 model the communication time, as 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 a model inspired by Xiong et al. (2024), where the intra- and inter-node communications are

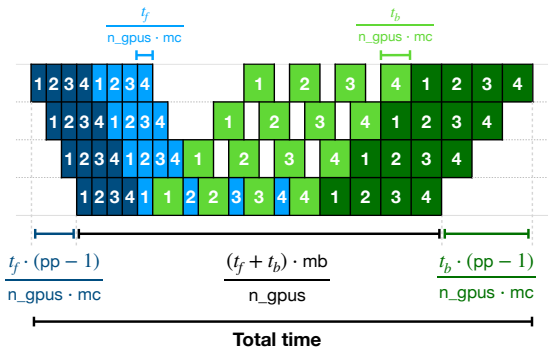


Figure 10: Predicted computation time for PP where t_f and t_b are the time taken for the respective forward and backward stages when $\texttt{pp} = \texttt{n_gpus} = 4$, $\texttt{mb} = 4$, and $\texttt{mc} = 2$.

characterized by the overhead α_{intra} and α_{inter}, and the communication time per unit of message size is given by β_{intra} and β_{inter}. We assume that the inter-node communication costs will be larger than their intra-node counterparts, i.e., $\alpha_{\text{intra}} \leq \alpha_{\text{inter}}$ and $\beta_{\text{intra}} \leq \beta_{\text{inter}}$.

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

$$C_{\text{AR}}(n, z; \alpha, \beta) = 2(n-1)\alpha + \frac{n-1}{n} z\beta.$$

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

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

where M_{model} is a learnable parameter representing the total size of the NN weights.

Suppose G_{node} is the number of GPUs per node. The overall communication time 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

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

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

$$\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(P; \theta_{\text{comm}}) = C_{\text{AR}}(\texttt{dp}, Z_{\texttt{dp}}; \alpha_{\text{inter}}, \beta_{\text{inter}}). \tag{7}$$

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

$$\widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(P; \theta_{\text{comm}}) = C_{\text{AR}}(\texttt{dp}, Z_{\texttt{dp}}; \alpha_{\text{intra}}, \beta_{\text{intra}}). \tag{8}$$

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

For TP, we consider the time from frequent activation communication (e.g., All-Reduce per layer). Let $M_{\text{act},\texttt{tp}}$ be a learnable parameter representing the data size for one 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 $C_{\text{AR}}(\texttt{tp}, M_{\text{act},\texttt{tp}}; \alpha_{\text{eff}}, \beta_{\text{eff}})$ where $(\alpha_{\text{eff}}, \beta_{\text{eff}})$ are set to $(\alpha_{\text{inter}}, \beta_{\text{inter}})$ if the $\texttt{tp}$ group spans multiple nodes (i.e., if $N_H > 1$ and $\texttt{tp} > G_{\text{node}}$), and $(\alpha_{\text{intra}}, \beta_{\text{intra}})$ otherwise. Then, the total communication time from TP is the number of communication operations multiplied by the time per communication operation, or

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

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

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

Let $M_{\text{act},\texttt{pp}}$ 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 $N_{\texttt{pp},\text{transfers}} = 2 \cdot \max(\texttt{pp} - 1, 0) \cdot \texttt{mb}$ where a single P2P transfer uses effective latency α_{eff} and effective per-unit-data cost β_{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 time for all communications related to PP would be given by

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

where $(\alpha_{\text{eff}}, \beta_{\text{eff}})$ are set to $(\alpha_{\text{inter}}, \beta_{\text{inter}})$ or $(\alpha_{\text{intra}}, \beta_{\text{intra}})$ as previously. Here, if $\texttt{pp} = 1$, then $\widehat{\mathcal{T}}_{\text{comm},\texttt{pp}} = 0$.

When combining the communication time for all three types of parallelism, we obtain the total time spent on communication given by

$$\widehat{\mathcal{T}}_{\text{comm}}(P; \theta_{\text{comm}}) = \widehat{\mathcal{T}}_{\text{comm},\texttt{dp}}(P; \theta_{\text{comm}}) + \widehat{\mathcal{T}}_{\text{comm},\texttt{tp}}(P; \theta_{\text{comm}}) + \widehat{\mathcal{T}}_{\text{comm},\texttt{pp}}(P; \theta_{\text{comm}}) \tag{12}$$

where $\theta_{\text{comm}} = \{M_{\text{model}}, M_{\text{act},\texttt{tp}}, M_{\text{act},\texttt{pp}}, O_{\texttt{tp/mb}}, \alpha_{\text{intra}}, \beta_{\text{intra}}, \alpha_{\text{inter}}, \beta_{\text{inter}}\}$ are learnable parameters which are to be inferred based on the measured throughput. Given (5) and (12), we can construct the prior mean for throughput to be

$$\widehat{\mathcal{R}}(P; \theta_{\mathcal{R}}) = \left[\widehat{\mathcal{T}}_{\text{comp}}(P; t_{\text{comp}}) + \widehat{\mathcal{T}}_{\text{comm}}(P; \theta_{\text{comm}})\right]^{-1} \tag{13}$$

where $\theta_{\mathcal{R}} = \{t_{\text{comp}}, \theta_{\text{comm}}\}$ are learnable parameters.

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

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

As discussed previously, we only consider the memory required to store the NN weights and the NN 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 and a fixed training batch size, the maximum memory required for storing the NN gradients will roughly be proportional to how many NN weights each 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

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

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 $P \in \mathbb{Z}^d$, we use the Matérn kernel (Rasmussen & Williams, 2006) which is given by

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

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

$$d_\ell(P, P') = \big(P - P'\big)^\top \mathbf{L}^{-2} \big(P - P'\big) \tag{15}$$

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 Tab. 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 (Couto, 2005), 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 $\mathbf{P}_i = [P_1 \cdots P_i]$ 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 (Rasmussen & Williams, 2006), or

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

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

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

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

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

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

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

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

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

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 $P_i \in \mathcal{P}$ to trial in round i is selected to maximize the constrained upper confidence bound (cUCB) acquisition function (Srinivas et al., 2012; Wilson et al., 2017), which is given by

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

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

E.2 Random Sampling for Additional Exploration

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

1. In the first few chosen PCs. In the beginning, there are no PCs which can be used to infer the learnable parameters 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 learnable parameters.
2. When too many out-of-memory errors have been encountered in a row. Any out-of-memory trials will not yield a throughput estimate and provide minimal data for the maximum memory GP, making the surrogate modeling ability stagnate. When too many such cases are encountered, we perform some random exploration to make the process "unstuck" and potentially pick up more information on which PCs may actually be feasible.

For the random selection process, we select a PC using a weighted random strategy such that we are equally likely to select any of the configuration of size of parallelism dimension $(\texttt{dp}, \texttt{tp}, \texttt{pp})$.

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 predictive 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. 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 to obtain a better estimate of the throughput. First, we remove the measurement from the first training step $t_{i,1}$ since it typically corresponds to a warm-up for the training and therefore will usually be an anomalous 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 measurements 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

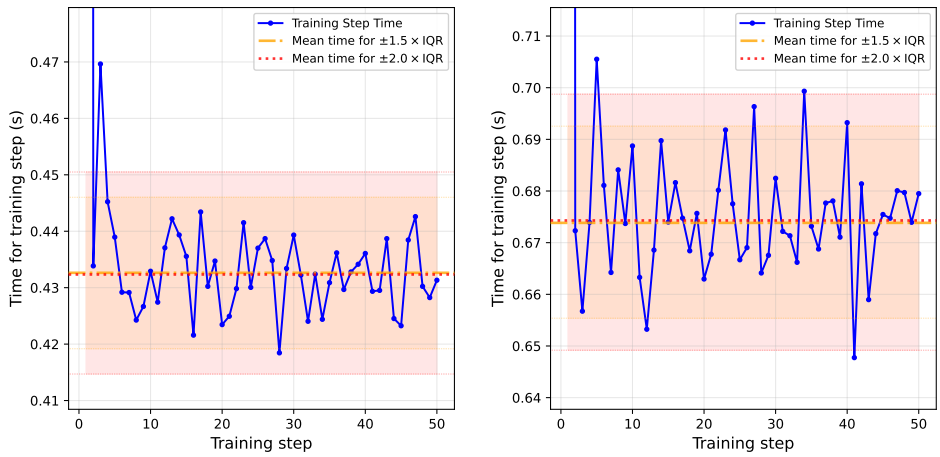


Figure 11: Example of the outlier removal process based on the IQR on different random PC trials. Here, we show the effect of the threshold selected vs. the mean of the measurements, demonstrating the robustness with respect to the different thresholds.

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.

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 repeated measurements, and as we will show in Corr. F.1. In these cases, we see that some repeated measurements are therefore necessary to reduce the noise of the estimate.

However, as we perform more repeats, there is a diminishing effect on how much more information is gained from each additional repeated measurement. 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

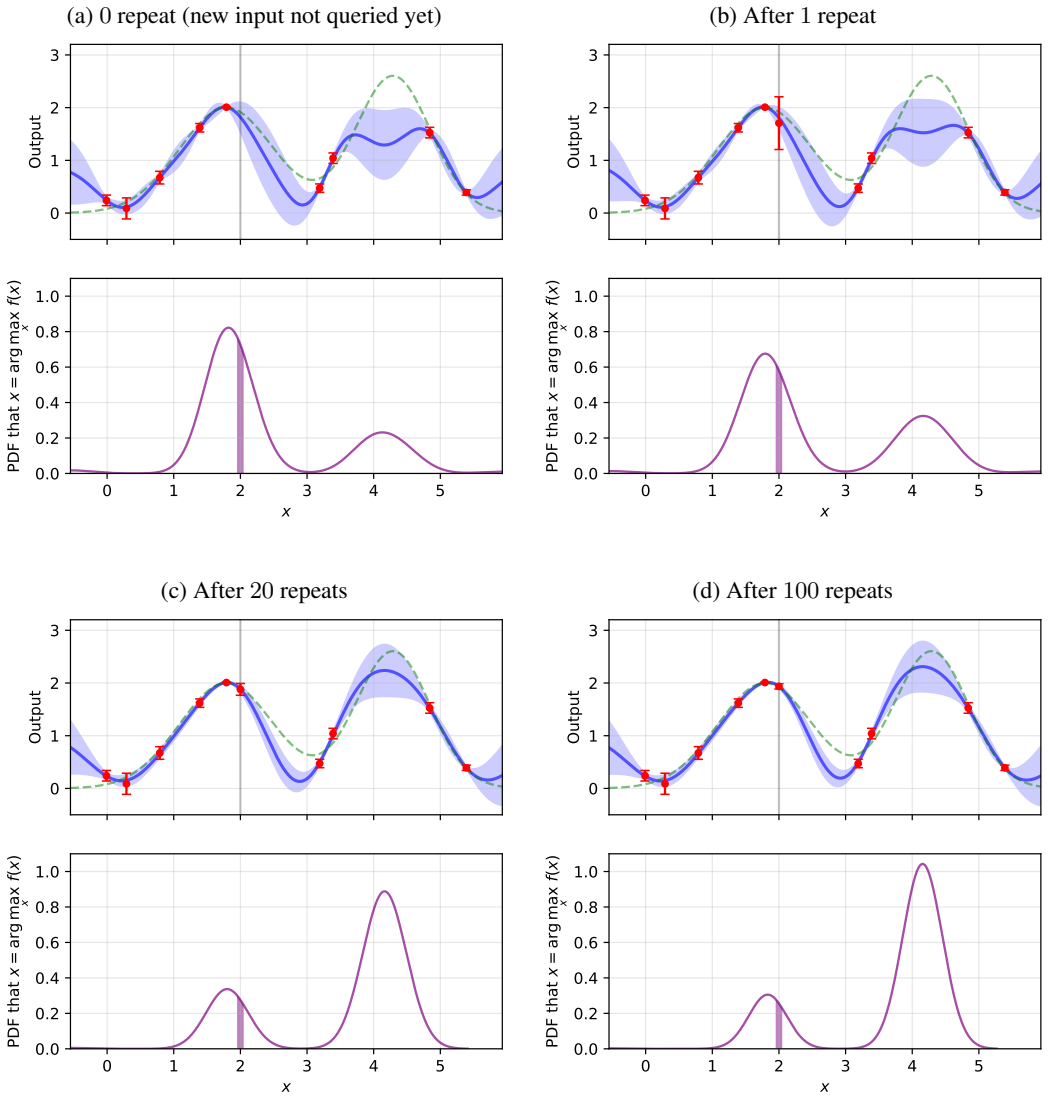


Figure 12: Demonstration of why early termination of certain trials does not affect final result. Figs. 12a to 12d represent the modeled GP and the predicted optimum when a certain input, in this case at $x = \arg\max_{x'} \mu_t(x') + \beta_t \sigma_t(x') = 2$, is repeatedly queried for 0, 1, 20, and 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 $x = 2$.

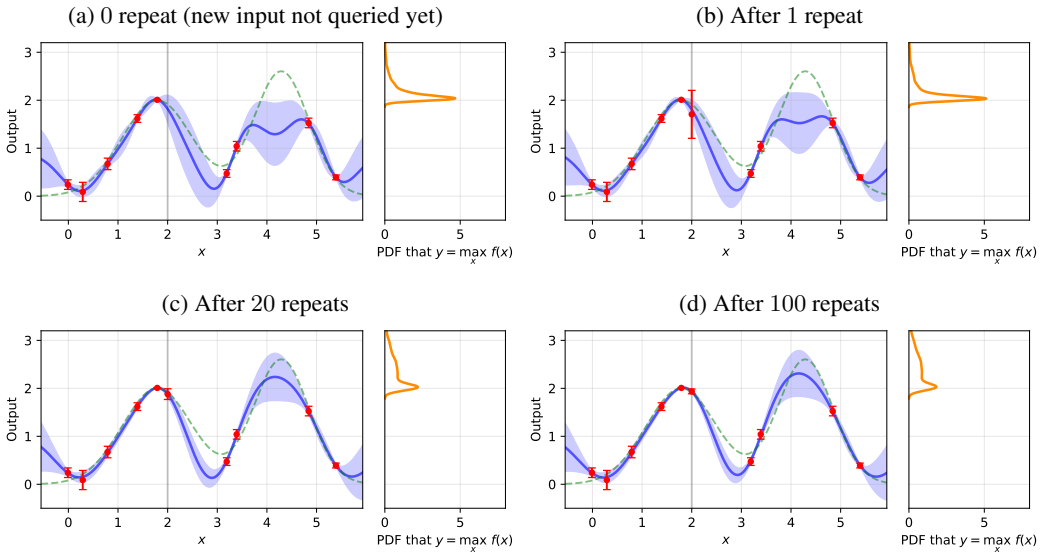


Figure 13: Additional demonstration of why early termination of certain trials does not affect the obtained optimal value. The setup is as done previously in Fig. 12 with queries at $x = 2$. For each subfigure, the left plot represents the modeled GP as shown previously in Fig. 12. In the right diagram,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

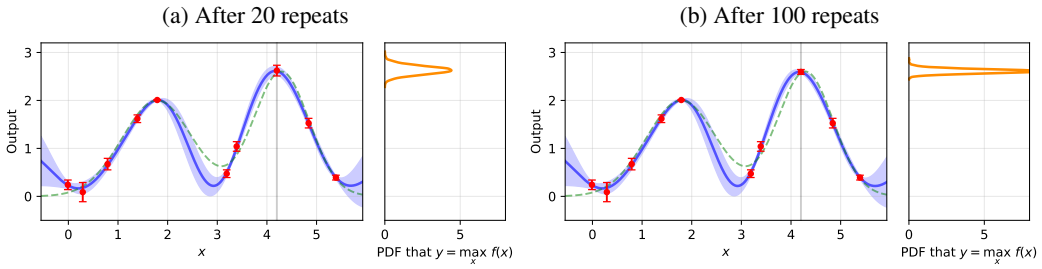


Figure 14: Additional demonstration of why early termination of certain trials does not affect the obtained optimal value. The setup is similar to as done previously in Fig. 12, however instead with queries at $x = 4.2$, and with only 20 repeats and 100 repeats shown (comparable to Figs. 13c and 13d).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

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 which follow from other BO works (Srinivas et al., 2012; Makarova et al., 2021; Kirschner & Krause, 2018) 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 $\|f\|_{\mathcal{P}} \leq B$ 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) measurements $y_{i,j} = f(x_i) + \varepsilon_{i,j}$ 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-

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 $j = 1, \ldots, q_{\max}$ **do**
    Sample measurement $y_{i,j}$
  end for
  $\bar{y}_{i,q_{\max}} \leftarrow q_{\max}^{-1} \sum_{j=1}^{q_{\max}} y_{i,j}$
  $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(x_i, \bar{y}_{i,q_{\max}})\}$
end for

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 $j = 1, \ldots, q_{\max}$ **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 $j > q_{\min}$ and $\bar{y}_{i,q} < \max_{j<i} \bar{y}_{j,\hat{q}_j} + \tau_q$ **then**
      break // Early trial termination mechanism
    end if
  end for
  $\mathcal{D}_i \leftarrow \mathcal{D}'_{i,j'}$
end for

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 $f(x_i)$.

Corollary F.1. *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 Corr. F.1, 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.1. *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 $\mathbf{Q}_i = \mathrm{diag}([\hat{q}_1 \cdots \hat{q}_i])$. 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 at least $1-\delta$*, for all* $x \in \mathcal{X}$ *and all* $i = 1, \ldots, N$*, we have*

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

Proof. Given the variance of the mean predictor from Corr. F.1, our scenario can be viewed 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 (Kirschner & Krause, 2018) where we substitute $\Sigma_i \to s^2\mathbf{Q}_i^{-1}$ and $\lambda \to 1$. □

Corollary F.2. *Given Lemma F.1, 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

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

□

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.1. *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 (Srinivas et al., 2012) or Corollary 9 in (Kirschner & Krause, 2018).

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

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

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

$$\sum_{i=1}^{N}\big(f(x^*)-f(x_i)\big) \le \sqrt{\sum_{i=1}^{N} 4\beta_i^2}\sqrt{\sum_{i=1}^{N}\sigma_{i-1}^2(x_i)} \qquad \text{by Cauchy-Schwartz,} \tag{40}$$

$$\le \beta_N\sqrt{4N}\sqrt{\sum_{i=1}^{N}\sigma_{i-1}^2(x_i)} \qquad \text{since } \beta_i \le \beta_N, \tag{41}$$

$$\le s\beta_N\sqrt{\frac{8N\gamma_N}{q_{\text{min}}}} \qquad \text{by (39).} \tag{42}$$

□

Remark F.1 (The regret bound in Thm. F.1 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 $\boldsymbol{K} = k(\mathbf{X}_N, \mathbf{X}_N)$*. First, we see that*

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

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

which means that

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

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

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

Furthermore, we see that

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

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

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.1 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* $1/\sqrt{q_{\text{min}}}$.

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

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

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

Proof. From Corr. F.1, we know that the $\bar{y}_{i,q}$ is normally distributed with mean $f(x_i)$ and standard deviation $s/\sqrt{q}$. By Gaussian tail bounds, for each $i=1,\ldots,N$ and each $q=1,\ldots,q_{\text{max}}$, we would have the events $\bar{y}_{i,q}-f(x_i) > c_1 s/\sqrt{q}$ and $f(x_i)-y_{i,q} > c_1 s/\sqrt{q}$ each happening with probability no greater than $\delta/2Nq_{\text{max}}$. This means that with probability no greater than δ/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 $q=1,\ldots,q_{\text{max}}$, we would have $|\bar{y}_{i,q}-f(x_i)| \le c_1 s/\sqrt{q}$ with probability at least $1-(\delta/Nq_{\text{max}})(Nq_{\text{max}}) = 1-\delta$. □

Theorem F.2. *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.2 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

$$\begin{aligned} f(x_i) &\geq \bar{y}_{i,q} - \frac{c_1 s}{\sqrt{q}} && (52)\\ &\geq \max_{j<i} \bar{y}_{j,\hat{q}_j} + \tau_q - \frac{c_1 s}{\sqrt{q}} && (53)\\ &\geq \max_{j<i} f(x_j) - \frac{c_1 s}{\sqrt{q_{\min}}} + \tau_q - \frac{c_1 s}{\sqrt{q}} && (54)\\ &= \max_{j<i} f(x_j). && (55) \end{aligned}$$

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

Finally, Thms. F.1 and F.2 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.

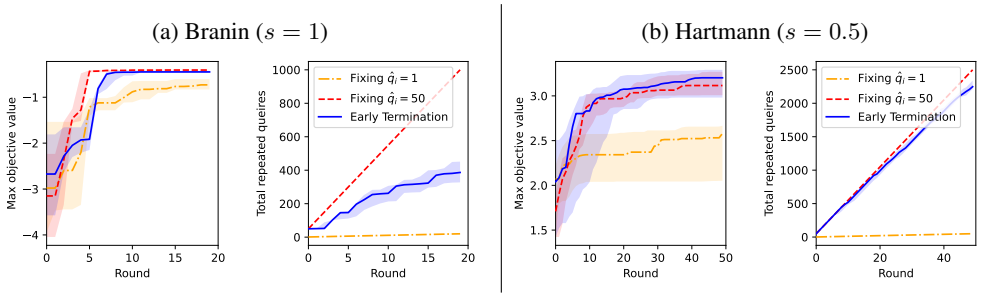


Figure 15: Results of BO with early trial termination (with $q_{\min} = 1$) on different synthetic functions where each query has different noise levels. For each case, we show the best queried objective value (left plot), and the cumulative number of total repeated queries made with each BO iteration (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 $q_{\text{min}} = 5$ in all iterations and training scenarios) give good results in all cases, demonstrating that the performance is likely robust to different hyperparameter values. In our experiments, we only use a higher q_{min} value for MoE training (where we use $q_{\text{min}} = 10$) 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 Tab. 2 depending on the benchmark; however in practice can be set as large as the practitioner wants depending on how accurately they would like the estimated throughput of the PC to be. 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 Sec. F.2.

Algorithm 4 Optimizer for Parallelism configurations (OpPa)

Generate all valid PCs $\mathcal{P}$
$i \leftarrow 1$
while time budget not exhausted **do**
  if $i < N_{\text{random}}$ **then**
    Select P_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
    $P_i \leftarrow \underset{P \in \mathcal{P} \setminus \{P_1,\ldots,P_{i-1}\}}{\arg\max}$ $\text{cUCB}_i(P)$ where cUCB_{i-1} is defined in (22)
  end if
  // Step ③ – Trialing selected PC
  for $q = 1, \ldots, q_{\text{max}}$ **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)
    end if
  end for
  Measure maximum memory usage as m_i
  $\hat{q}_i \leftarrow q$ to track the number of training steps run in round i
  $i \leftarrow i + 1$
end while
return P_{i^*} where $i^* = \arg\max_{i \,:\, (m_i < M_0) \wedge (\hat{q}_i = q_{\text{max}})} \bar{r}_{i,\hat{q}_i}$

H Comparison of OpPa with Other Existing Approaches

Given that we have presented the full implementation of OpPa, we review related approaches (particularly those from the wider BO literature) and discuss why they do not capture the PC optimization problem as directly as OpPa has.

We first note that PC optimization is closely related to problems tackled by multi-fidelity BO (MFBO) (Klein et al., 2017; Wu et al., 2019), where the method choose which measurements are observed and at which fidelity. However, MFBO relies on a different assumption than in PC optimization where for

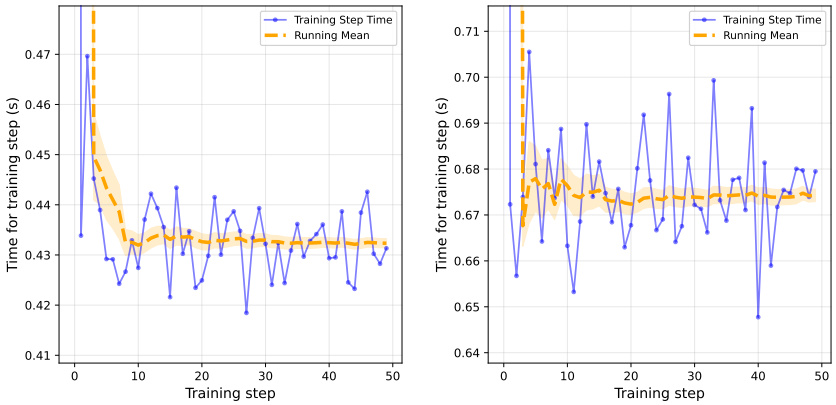


Figure 16: Effect of number of training steps repeatedly measured on the predicted throughput. The solid blue line represents the measured time per training step for an NN training trial. The solid orange line represents the average training time (i.e., reciprocal of throughput) predicted up to that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

an input, the measurements that are made are not repeated on the same quantity but are instead made across fidelities. Here, MFBO would be at a disadvantage due to its looser assumptions. Additionally, methods such as Falkner et al. (2018) which parallelizes training trials would not yield benefits since we would likely only have one instance of the true hardware in practice, making parallelization infeasible.

Other works Binois et al. (2019); Dai et al. (2023) have also proposed methods which use repeated measurements of the same input. Here, in each query, the algorithms specify upfront which measurement to observe and how many repetitions. This is unsuitable for PC optimization, since the time required for initializing NN training is high (and often larger than the time per training step), and so it is more efficient to decide on the number of training steps required on-the-fly during the training loop to avoid repeated trials on the same PC during optimization. The early termination mechanism fits better for PC optimization, as the decision on how many repeated measurements is made during the training loop and incurs little overhead for the training process.

Our early stopping mechanism can also be compared with stopping criteria for BO (Ishibashi et al., 2023; Wilson, 2024), which focuses on when the BO loop itself should be terminated. While these methods and our early termination mechanism both reduce the amount of trialing that needs to be performed, they do so at two different stages of the optimization, and thus are not directly comparable. Additionally, the early termination mechanism is designed to exploit the sequential nature of NN training, and is simple enough to be used in the training loop itself. This is opposed to stopping criteria for BO which are too expensive to be repeatedly run during the NN training.

Additionally, PC optimization is related to the problems in AutoML (Kandasamy et al., 2018; He et al., 2021), which aim to optimize hyperparameters to achieve the best model performance. While AutoML is well-studied, most of the works focus on optimizing the actual model performance (e.g., predictive accuracy), with few methods (DeepSpeed, 2026) focusing on PC optimization for maximizing training throughput. The methods in existing AutoML works are also unable to directly exploit the characteristics of the PC optimization problem, namely (1) information from both the existing domain knowledge of parallel NN training and throughput measurements from training trials (whereas AutoML works would only focus on using the latter), and (2) the sequential and repeated training steps ran during training which improves the throughput estimate (which is not the case for validation loss, as typically considered by AutoML works).

I Additional Details on Experimental Setup

I.1 Training and Hardware Configurations

Tab. 2 lists the NNs used in our experiments along with the allotted time budget for the optimization process and how many repeated measurements are allowed for each NN training trial ($q_{\max}$). All NNs used are based on the transformer architecture. All of the LLMs have architectures based on the specified NNs loaded from Huggingface. For the `ViT` model, we use a custom vision transformer with 24 hidden layers, 16 attention heads, embedding dimension of 1024, and feed-forward hidden dimension of 4096. For the `MoE` model, we use a custom mixture-of-experts model with 16 hidden layers, 32 attention heads, 32 experts, embedding dimension of 1024, and feed-forward hidden dimension of 1024.

Table 2: Details of NNs used in our experiments and their corresponding training scenarios.

Case	NN	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 Tab. 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.

I.2 Tested Methods

We provide the implementation details of the tested methods below:

- Random. This involves randomly selecting a PC from $\mathcal{P}$ to trial in each round until the time budget (Tab. 2) is exhausted.
- XGBoost (Chen & Guestrin, 2016). This method is used by DeepSpeed (Rasley et al., 2020) 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 throughputs thus far and then selects the PC with the highest prediction from the surrogate according to the XGBoost surrogate model to trial next. Like in the implementation from (Rasley et al., 2020), the XGBoost surrogate model does not model the throughputs of different

PCs directly, but their relative rankings. This is repeated until the time budget (Tab. 2) 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 (Tab. 2) is exhausted. We then find the optimal hyperparameters $\theta_{\mathcal{R}}$ and $\theta_{\mathcal{M}}$, respectively, for $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{M}}$ 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 using a regular GP with a constant prior 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 (Balandat et al., 2020).
- 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 Sec. 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 (Li et al., 2022) and NNSCALER (Lin et al., 2024) in Fig. 9.

J ADDITIONAL RESULTS

J.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.

J.2 OPTIMAL PCS RECOVERED BY OPPA

In Tabs. 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. This fits our motivation that intuitive choices of PC may not always give the maximum throughput, which makes PC optimization based on actual measurements to be important in order for us to make the most informed choice on which PC is optimal.

J.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 with the parallel NN training being performed using NEMO (Kuchaiev et al.,

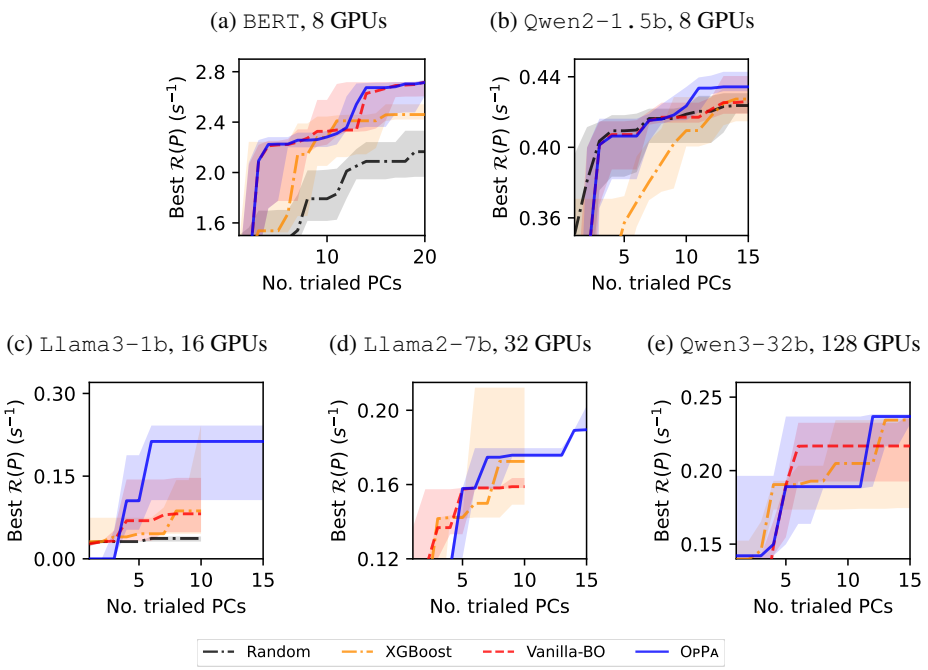


Figure 17: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across (a,b) 10 and (c,d,e) 5 repeated experiments with error bands showing lower and upper quartiles) vs. the number of PCs it has trialed for parallel NN training on (a,b) single- and (c,d,e) multi-node setups using COLOSSAL-AI framework (Li et al., 2023a). These results are from the same experiments in Figs. 4 and 8. For the experiments with 32 GPUs in Fig. 17d, some methods can only trial fewer PCs due to the allotted time budget (Tab. 2).

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

2019) instead. In these experiments, we have additional hyperparameters in the PC, namely the size of the sequence parallelism dimension. We see that compared to Fig. 4, the obtained maximum throughput values are different because the training processes are implemented differently across the two frameworks, and the fact that the hyperparameters in the PC that can be optimized are also different across training frameworks. Regardless, we see that even though other methods are able to eventually recover the optimal PC, OPPA is still able to do so more rapidly and, more importantly, more consistently. The latter fact can be seen in the error bands of the other methods, which are much

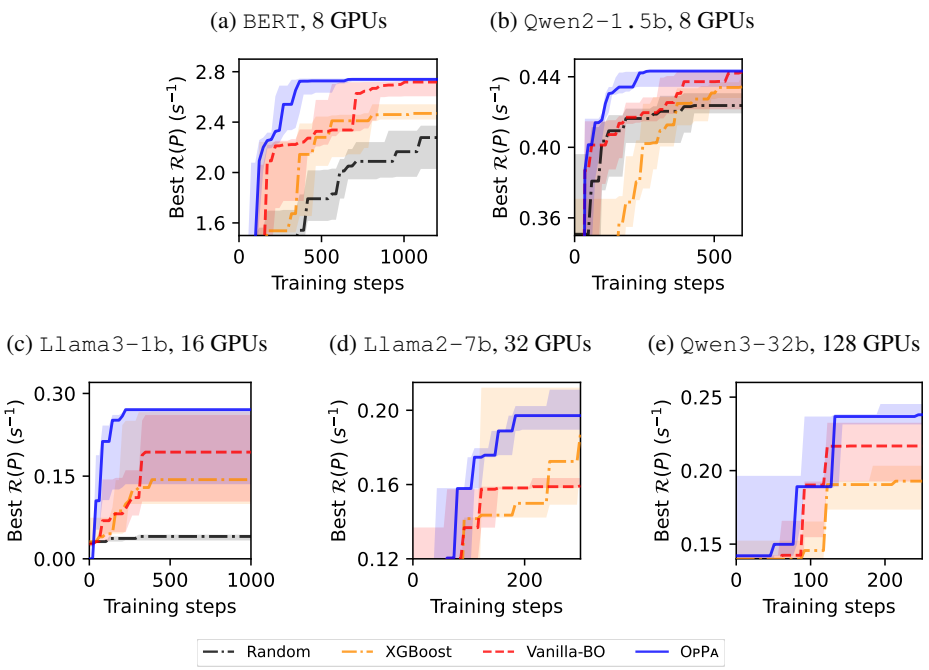


Figure 18: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across (a,b) 10 and (c,d,e) 5 repeated experiments with error bands showing lower and upper quartiles) vs. total number of training steps being run by the method for parallel NN training on (a,b) single- and (c,d,e) multi-node setups using COLOSSAL-AI framework (Li et al., 2023a). These results are from the same experiments 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 Tab. 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

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.

J.4 PC OPTIMIZATION FOR OTHER NN ARCHITECTURES

In Fig. 21, we present the results for PC optimization for ViT model (Dosovitskiy et al., 2021). 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 Shazeer et al. (2017), which allows for expert parallelism (EP) to be utilized in addition to the existing DP, TP, and PP. In the results, we again see that OPPA consistently outperforms

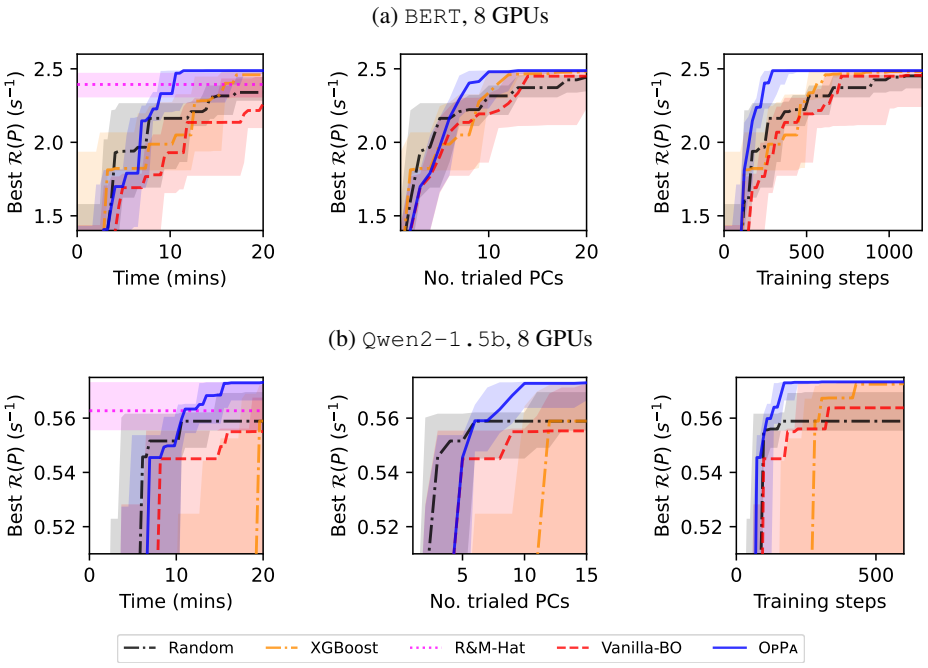


Figure 19: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across 10 repeated experiments with error bands showing lower and upper quartiles) vs. its incurred time for parallel NN training on a single-node setup using the NEMO framework (Kuchaiev et al., 2019). These results are based on similar experiments as that in Figs. 4a and 4b, except that parallel NN training is performed using the NEMO framework.

the other tested methods. Specifically, even if OPPA may eventually lead to similar results as vanilla BO at the end of the 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.

J.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

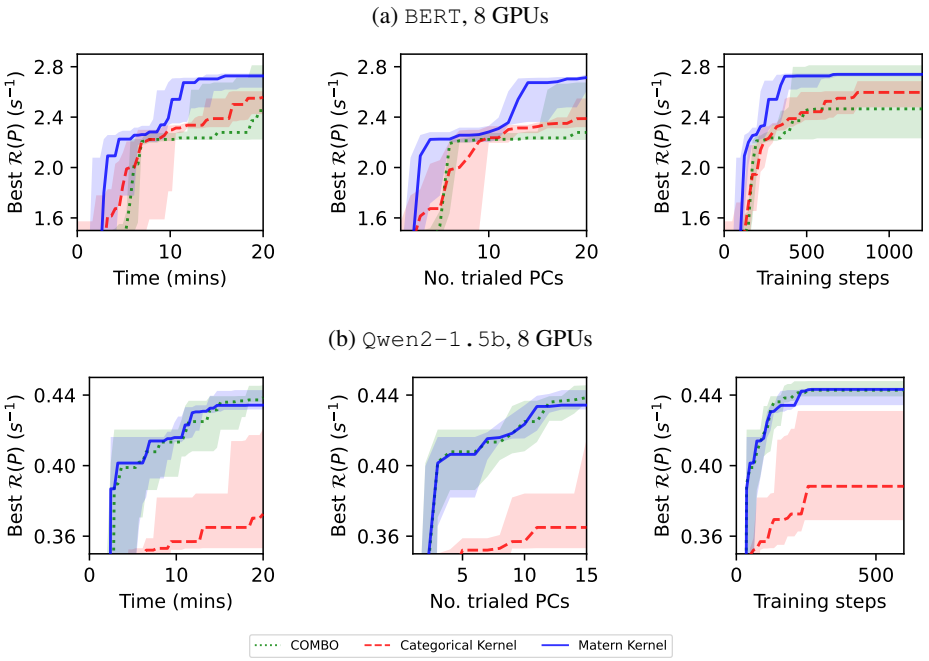


Figure 20: Graphs of best (estimated) measured throughput (higher is better) achieved by OPPA using Matérn vs. categorical kernels Couto (2005) (i.e., plotted as medians across 10 repeated experiments with error bands showing lower and upper quartiles) vs. its incurred time for parallel NN training on a single-node setup using the COLOSSAL-AI framework (Li et al., 2023a). The results for the Matérn kernel are exactly the same as that in Figs. 4a and 4b.

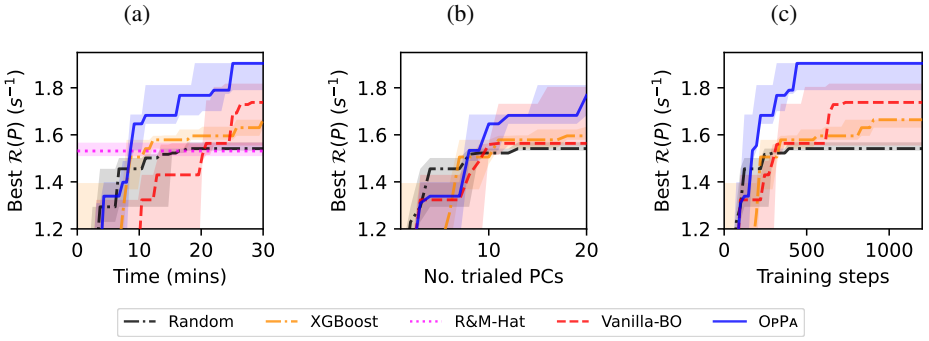


Figure 21: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across 5 repeated experiments with error bands showing lower and upper quartiles) vs. (a) its incurred time, (b) the number of PCs it has trialed, and (c) the total number of training steps being run by the method for parallel training of ViT model (Dosovitskiy et al., 2021) on a single node of 8 GPUs using the COLOSSAL-AI framework (Li et al., 2023a).

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 Tab. 6, we show the results for the lengthscales of the Matérn kernel (as given in Sec. D.3) which maximizes the log-likelihood. We see that PC hyperparameters that have larger effects on the resulting throughput or 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

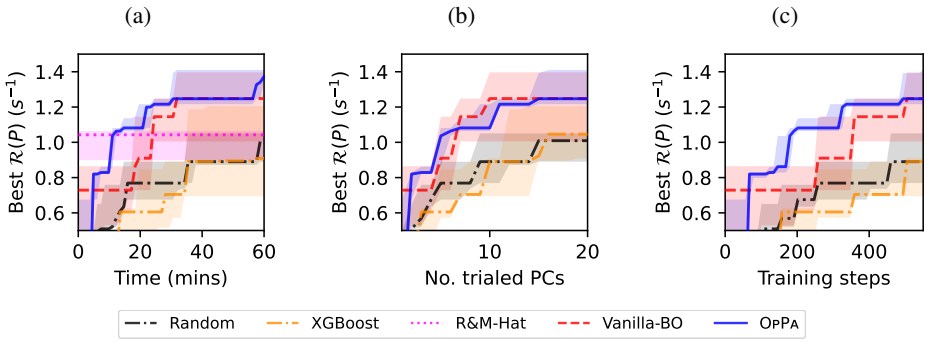


Figure 22: Graphs of best (estimated) measured throughput (higher is better) achieved by each method (i.e., plotted as medians across 5 repeated experiments with error bands showing lower and upper quartiles) vs. (a) its incurred time, (b) the number of PCs it has trialed, and (c) the total number of training steps being run by the method for parallel training of MoE (Shazeer et al., 2017) on a single node of 8 GPUs using the NeMo framework (Kuchaiev et al., 2019).

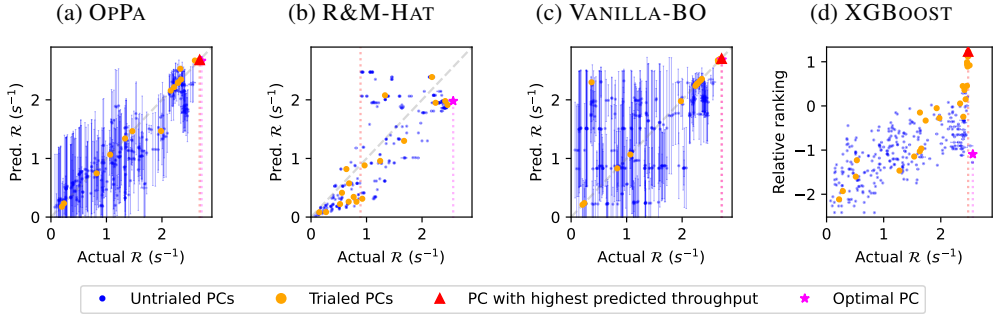


Figure 23: Graphs of throughputs predicted by different methods' surrogate models vs. actual measured throughputs after 20 NN training trials based on an experiment in Fig. 4a. Each untrialed PC in Figs. 23a and 23c is associated with error bars showing twice the GP posterior std. dev., which are not in Figs. 23b and 23d as $\widehat{R}$ gives deterministic predictions and the XGBOOST surrogate model deterministically predicts their relative rankings, as explained in Sec. I.2.

our intuition where the throughput would be more sensitive to the change in TP or PP dimension sizes.

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

J.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.

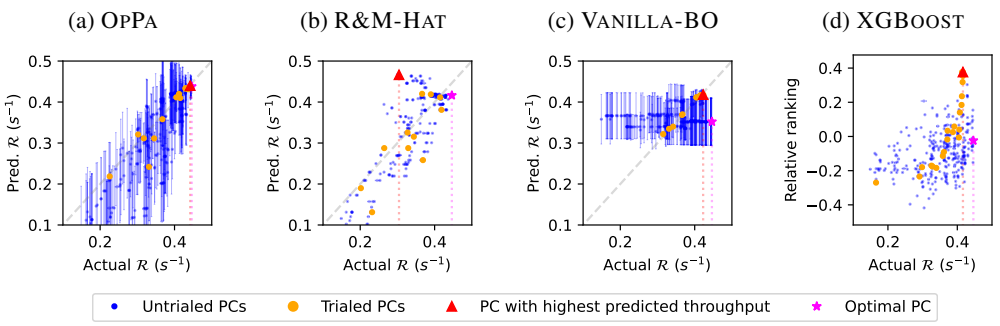


Figure 24: Graphs of throughputs predicted by different methods' surrogate models vs. actual measured throughputs after 20 NN training trials based on an experiment in Fig. 4b. Each untrialed PC in Figs. 24a and 24c is associated with error bars showing twice the GP posterior std. dev., which are not in Figs. 24b and 24d as $\widehat{R}$ gives deterministic predictions and the XGBOOST surrogate model deterministically predicts their relative rankings, as explained in Sec. I.2.

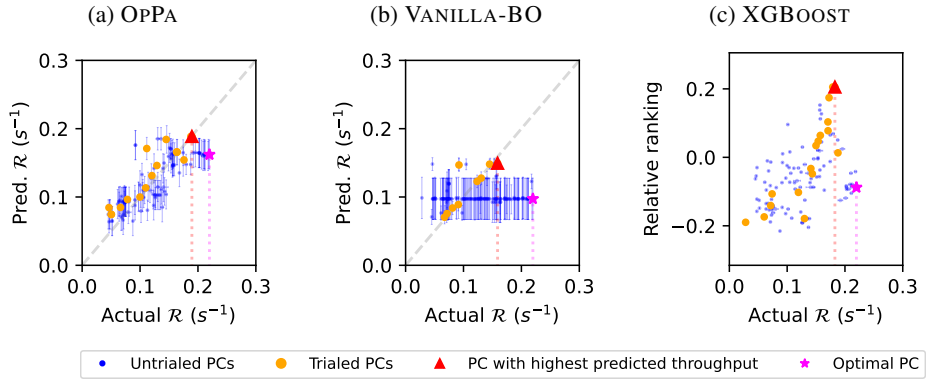
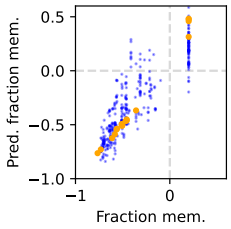


Figure 25: Graphs of throughputs predicted by different methods' surrogate models vs. actual measured throughputs after (a) 20 and (b, c) 10 NN training trials for parallel training of `Llama2-7b` model on 8 nodes of 4 GPUs each (i.e., a total of 32 GPUs) using the COLOSSAL-AI framework. Each untrialed PC in Figs. 25a and 25b is associated with error bars showing twice the GP posterior std. dev., which is not in Fig. 25c as the XGBOOST surrogate model deterministically predicts their relative rankings, as explained in Sec. I.2. Among the three methods, only OPPA can run up to 20 NN training trials with the alloted time budget (Tab. 2).

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 Tab. 7. We see that even when $\widehat{R}$ and $\widehat{M}$ are adversarially perturbed, we are still 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.

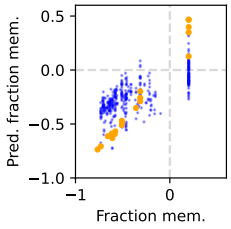
In practice, prior misspecification is typically due to the parallelism-informed prior belief not being sufficiently complex to match the actual parallel NN training dynamics arising from incomplete or inaccurate domain knowledge about the actual system, rather than due to an adversarial construction of $\widehat{R}$ and $\widehat{M}$. 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.

(a) OPPA



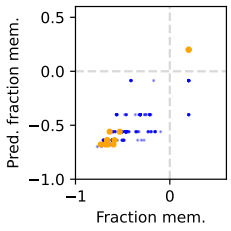
	Predicted Feasible	Predicted OOM
Actual Feasible	296	16
Actual OOM	4	84

(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.

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

J.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.

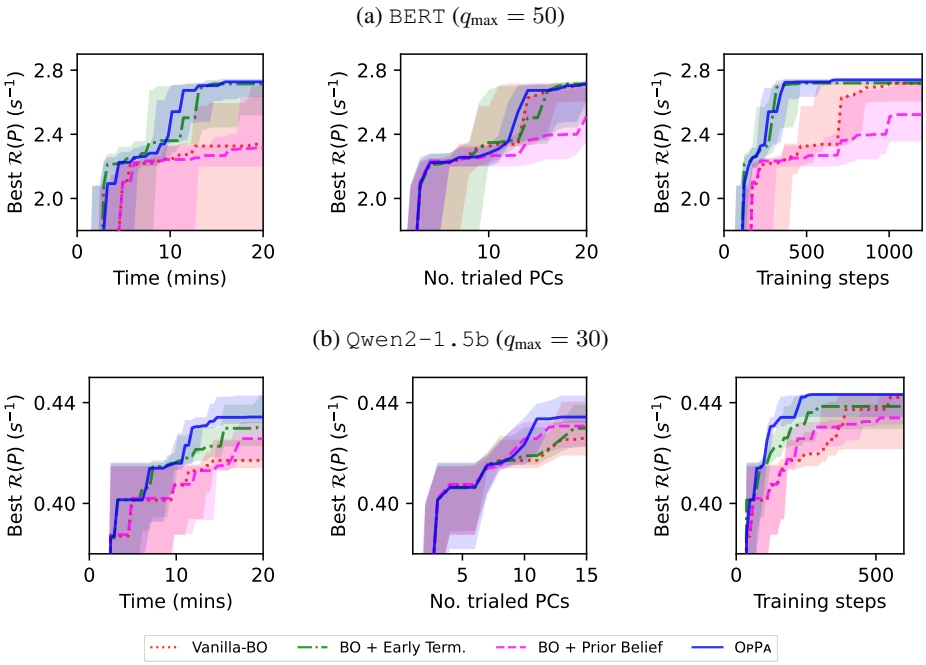


Figure 27: Effects of different efficiency tricks in OPPA on the optimal PC found. We present the measured throughput vs., from left to right, the time the method has been run for, the number of unique PCs trialed, and the number of training steps that have been run across all of the trials.

J.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, we obtain the least optimal PC in the case where $q_{min} = q_{max}$. For a smaller q_{min}, it can be observed that there is a smaller difference in the best measured throughput. This may be attributed to the fact that at this stage we are still able to get an accurate enough estimate of the throughput even with early termination, and also that 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 too small, likely because the throughput estimate is now too noisy to give good information. For the Qwen2-1.5b case, similar trends can be observed 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.

K 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 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.

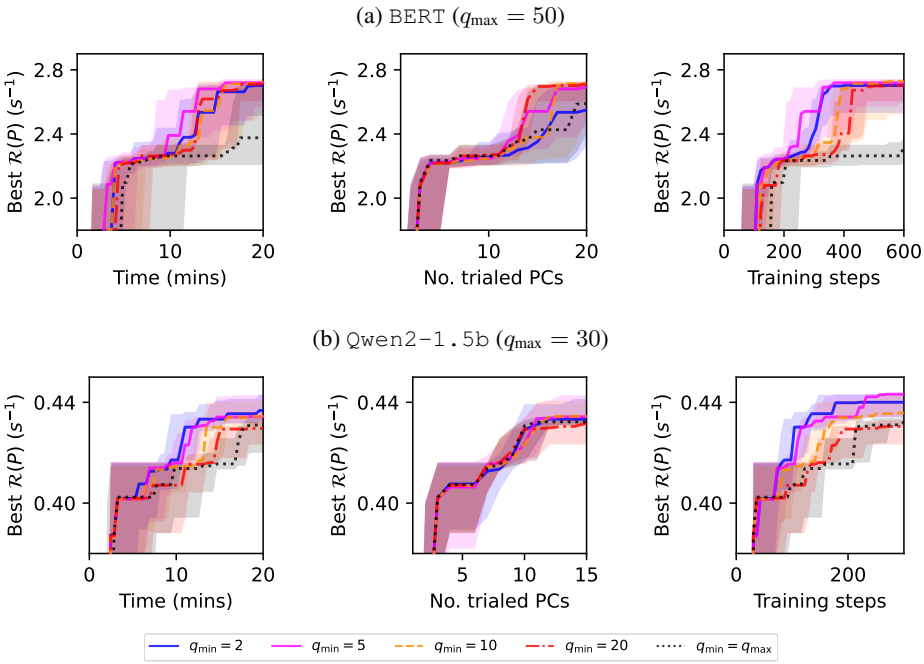


Figure 28: Effect of q_{min} on the optimal PC found. In each row, we present the results in a certain training setup, where we present the measured throughput vs., from left to right, the time the method has been run for, the number of unique PCs trialed, and the number of training steps that has been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