Hydro: Surrogate-Based Hyperparameter Tuning Service in Datacenters

Qinghao Hu, *Nanyang Technological University, S-Lab, NTU, and Shanghai AI Laboratory;* Zhisheng Ye, *Shanghai AI Laboratory and Peking University;* Meng Zhang, *Nanyang Technological University, S-Lab, NTU, and Shanghai AI Laboratory;* Qiaoling Chen, *Shanghai AI Laboratory and National University of Singapore;* Peng Sun, *Shanghai AI Laboratory and SenseTime Research;* Yonggang Wen and Tianwei Zhang, *Nanyang Technological University*

https://www.usenix.org/conference/osdi23/presentation/hu

This paper is included in the Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-34-2

Open access to the Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation is sponsored by

Hydro: Surrogate-based Hyperparameter Tuning Service in Datacenters

Qinghao Hu[1,2,3] Zhisheng Ye[3,4] Meng Zhang[1,2,3] Qiaoling Chen[3,5]
Peng Sun[3,6] Yonggang Wen[1] Tianwei Zhang[1]

[1]*Nanyang Technological University* [2]*S-Lab, NTU* [3]*Shanghai AI Laboratory*
[4]*Peking University* [5]*National University of Singapore* [6]*SenseTime Research*

Abstract

Hyperparameter tuning is an essential step in deep learning model development that provides better model performance at the cost of substantial resources. While existing systems can improve tuning efficiency, they still fail to handle large models with billions of parameters and efficiently leverage cluster resources. Motivated by these deficiencies, we introduce Hydro, a surrogate-based hyperparameter tuning service that optimizes tuning workloads in both the job-level and cluster-level granularities. Specifically, it consists of two key components: (1) *Hydro Tuner* automatically generates and optimizes surrogate models via scaling, parametrization and fusion; (2) *Hydro Coordinator* improves tuning efficiency and cluster-wide resource utilization by adaptively leveraging ephemeral and heterogeneous resources. Our comprehensive experiments on two tuning algorithms across six models show that Hydro Tuner can dramatically reduce tuning makespan by up to 78.5× compared with Ray Tune and no reduction in tuning quality. Hydro's source code is publicly available at https://github.com/S-Lab-System-Group/Hydro.

1 Introduction

Over the years, we have witnessed the incredible performance and rapid popularity of Deep Learning (DL) across many domains, such as vision and speech. However, it is non-trivial to acquire a qualified DL model because its performance is highly sensitive to the *hyperparameters*, which control the training process and require to be set before training [71]. Poor hyperparameters result in training instability and inferior model quality. Conversely, well-tuned hyperparameters can significantly improve model performance. For instance, PyTorch [91] recently applies a new hyperparameter recipe on ResNet-50 [41] and achieves 80.9% ImageNet classification accuracy [18], which is 4.8% higher than the former version (76.1%). Besides, RoBERTa [75] also demonstrates the critical impact of hyperparameters on the performance of large language models. Accordingly, hyperparameter tuning becomes a common practice during DL model development.

Due to the high dimensionality of the search space, a hyperparameter tuning job typically contains a large group of *trials*, each with a unique configuration [125]. To accelerate the tuning process, tech companies and researchers build hyperparameter tuning systems as cloud services [1, 8, 39, 92] or standalone frameworks [32, 71, 72, 82, 125, 127] (Table 1). However, we argue that state-of-the-art tuning systems are still expensive and inefficient in practice, as they suffer from several fundamental problems:

- **Unacceptable cost of tuning large models**. The extraordinary performance of large foundation models (e.g., BERT [30], GPT-3 [24]) attracts wide downstream applications [3, 4, 6]. Meanwhile, the hyperparameter tuning demand for these models increases rapidly. However, all of the existing tuning systems require training multiple trials using several times of resources, which is unaffordable for large models with billions of parameters. For example, training a SOTA language model PaLM [27] of Google takes over 6,000 TPU-v4 [59] for around 2 months. Performing a hyperparameter sweep on such model is intractable [23]. Consequently, hyperparameters of most large models are not well-tuned and can lead to subpar performance [75].
- **Inefficient hardware utilization**. Recent scheduling works [46, 114, 115, 124] report that GPUs are commonly underutilized in DL clusters due to massive training jobs involving mid- or small-scale models. Moreover, despite the growing trend of foundation models being employed in clusters, large-scale models often fail to fully utilize hardware resources due to the huge communication overhead and the presence of bubbles in the pipeline parallelism [106]. To improve resource utilization, some novel tuning systems incorporate features such as elastic training [32, 71, 82], GPU sharing [125], and inter-trial fusion [110]. However, these systems have certain limitations (§8) and often require substantial resources to explore trivial trials, which results in limited resources being contributed to the final model.
- **Agnostic to cluster-wide resources**. Hyperparameter tuning jobs are pervasive and occupy enormous resources in GPU clusters. As reported by Microsoft [50, 78], "approximately 90% of models require hyperparameter tuning, with each tuning job containing 75 trials in median." However, existing tuning systems only manage trials over the requested resources and lack interaction with cluster schedulers. Meanwhile, DL schedulers [36, 40, 46, 87, 94, 114, 123] also overlook the distinct characteristic of gradually diminishing hardware demand inherent in hyperparameter tuning jobs [71]. Consequently, the cluster encounters imbalanced resource problem: the active tuning jobs consistently occupy static resources,

leaving some of them vacant, while the queued jobs are unable to request these idle resources from the scheduler. This leads to severe queuing delay, which is exacerbated when long-term large-scale model training jobs coexist and they occupy the majority of cluster resources.

To bridge these gaps, we design Hydro, a surrogate-based hyperparameter tuning service that optimizes tuning jobs in both the job-level and cluster-level granularities via automated model scaling, fusion and interleaving. The core design of Hydro derives from the following three insights. First, *it is feasible to search hyperparameters with a smaller model*. Instead of tuning hyperparameters directly on the target model, we find it is possible to tune a model with a much smaller surrogate model by applying a novel hyperparameter transfer theory [117, 121]. Second, *cross-model fusion can be used to improve resource utilization*. Since the scaled surrogate model is prone to incur GPU underutilization, we can utilize the model architecture consistency of different trials to fuse them into a single one, achieving much higher GPU utilization and training throughput. Third, *ephemeral bubble resources in the datacenter can be leveraged for tuning*. Large model training jobs exist in the long term and occupy the majority of resources, which incurs the starvation of other jobs. We can leverage pipeline bubbles of large models to greatly extend the tuning job resources in an interleaving execution way, without hurting the training throughout of large models.

Incorporating the above insights, we build Hydro service to minimize the makespan of tuning workloads and improve the cluster-wide resource utilization. It consists of two key system components: (1) **Hydro Tuner** is the user interface that automatically generates surrogate models by *scaling* and *parametrization*. It optimizes tuning efficiency via *inter-trial* and *intra-trial fusion*, which involve combining multiple models into a single entity and subsequently performing compiler-based optimization. Besides, it efficiently orchestrates the tuning process with *adaptive fusion* and *eager transfer* mechanisms. (2) **Hydro Coordinator** is the datacenter interface that interacts with the scheduler to dynamically allocate resources and execute trials. It extends tuning resources by *interleaving training* with pipeline-enabled large model training tasks, effectively utilizing idle time intervals on each node known as *bubbles*, which are caused by the gaps between the forward and backward processing of microbatches [106]. Besides, it improves resource utilization and cluster-wide performance by *heterogeneity-aware* allocation.

To extensively assess the performance of Hydro, we conduct evaluations across 6 models, such as GPT-3 XL [24] and ResNet [41]. Experiments on Hydro Tuner show that it substantially outperforms Ray by 8.7∼78.5× on makespan reduction with single-fidelity tuning algorithm, while obtaining better final model quality. Besides, our experiments on Hydro Coordinator demonstrate that interleaving with a large pipelined model can further extend the resource of tuning workload, without sacrificing the throughput of the large model.

Features	Cloud Services		HPO Frameworks		Hydro
	Vizier	SageMaker	NNI	Ray	
Distributed Environment	✔	✔	✔	✔	✔
Elastic Training	♦	♦	♦	✔	✔
Auto Model Scaling	✘	✘	✘	✘	✔
Surrogate HP Transfer	✘	✘	✘	✘	✔
Inter-Trial Fusion	✘	✘	♦	✘	✔
Intra-Trial Fusion	✘	✘	✘	✘	✔
Heterogeneity Awareness	✘	✘	✘	✘	✔
Interleaving Training	✘	✘	✘	✘	✔

Table 1: Comparison between Hydro and existing popular HPO systems: Google Vizier [39, 105], Amazon SageMaker [28, 92], Microsoft NNI [9, 127] and Anyscale Ray [72, 84]. ♦ denotes system cannot support the feature for many cases.

Table 1 compares Hydro with existing tuning systems. To summarize, we make the following contributions:

★ We build a holistic system that automatically applies the novel hyperparameter transfer theory together with multiple system techniques to jointly improve the tuning efficiency.

★ We identify the opportunities for cluster-wide optimization in the datacenter, including squeezing bubble resources with interleaving and heterogeneity-aware trial allocation.

★ We demonstrate the excellent performance of surrogate-based hyperparameter tuning across general models.

2 Background and Motivation

2.1 Hyperparameter Tuning

Hyperparameter Tuning (i.e., Hyperparameter Optimization, HPO) aims to identify the optimal hyperparameters via massive configuration exploration [71, 82]. In the general workflow of an HPO job: (1) the user designates a *search space* of hyperparameters to explore; (2) the tuning algorithm creates a set of training *trials* and each trial contains one unique hyperparameter configuration sampled from the search space; (3) the HPO system coordinates trials execution until the best hyperparameter configuration is found.

Existing research works typically optimize HPO efficiency from the tuning algorithm [33, 47, 63, 64, 67, 68, 70, 79, 104] or system [32, 60, 69, 71, 82, 110, 125, 127] aspects:

Algorithm taxonomy. Depending on whether to enable early stopping, tuning algorithms can be divided into two categories [100]: (1) *single-fidelity* (e.g, Random [22], Bayes [104]) algorithms require each trial to be fully trained, which is accurate but inefficient; (2) *multi-fidelity* (e.g., ASHA [63], BOHB [33]) algorithms stop unpromising trials via successive halving [53] or curve fitting [31] strategies. They are efficient but may miss the best hyperparameter configuration due to the use of "low-fidelity" evaluations. Hydro well supports both the single- and multi-fidelity algorithms.

System optimization. To further improve the tuning efficiency and resource utilization, there are two advanced techniques applied in state-of-the-art HPO systems: (1) *elastic training* dynamically allocates more GPU resources to the top

performing trials [71] and further adjusts the entire requested resources [32, 82, 94]. (2) *GPU sharing* (i.e., trial packing) allows multiple trials to share the GPU using the NVIDIA MPS [13] or MIG [12] technologies to achieve higher utilization [125]. Different from them, Hydro combines scaling, fusion and interleaving for ultimate efficiency.

2.2 Hyperparameter Transfer Theory

Recently, the remarkable success of foundation models has ignited a keen interest in exploring the relationship between model size and its optimal hyperparameter. *Scaling Laws* [42, 43,52] empirically study the power-law functions of batch size and learning rate across varying model sizes. Nevertheless, the authors [52] candidly admit that only limited configurations are tested and the rule-of-thumb formulas break down when dealing with models that exceed one billion parameters.

Beyond heuristic exploration, some novel hyperparameter transfer strategies [49, 117, 121] are proposed by DL theorists. For simplicity, we call them ***parametrization***, the rule of how to adjust hyperparameters accordingly when models grow/shrink in both the width and depth. Different from existing HPO systems, Hydro enables automatic hyperparameter transfer based on parametrization. To make the obscure theory more accessible, we present a concise background overview of the underlying theory. [116] systematically builds a coherent theoretical framework for parameterization: the feature learning effect γ of a MLP model is proportional to

$$\gamma \equiv \frac{L}{w^{1-p}}, \quad p \in [0,1] \tag{1}$$

where w, L indicates the width and depth of the neural network respectively. For the purpose of simplicity, we assume that the numbers of the hidden-layer neurons are all of similar order, $w_1, w_2, ...w_{L-1} \sim w$. p is a metaparameter that interpolates different parametrization strategies into a unified framework, which is determined by inherent strategy. The objective is twofold: first, to maintain a fixed γ that allows hyperparameters transfer across different model sizes, and second, to strive for a larger γ that facilitates better feature learning. To this end, there are two directions of parametrization:

(1) *Neural Tangent (NT) parametrization* ($p = 0$) [49]. It naturally arose from the study of infinite-wide neural network as Neural Tangent Kernel (NTK) [49, 89], which can keep γ fixed by scaling the depth along with the width as $L \sim w$. NTK is a kernel method to explain the evolution of neural networks during training, which is derived by applying the first-order Taylor expansion to linearized models. It belongs to the *lazy training* regime where the weights move very little [121], so that linearization approximately holds around the initial parameters and does not learn features, which is a fatal weakness of the NTK theory in practice. Moreover, NT parametrization does not make sense since the wider model does not always perform better in this context [117], which conflicts with common observations [43, 52].

(2) *Maximal Update (MU) parametrization* ($p = 1$) [121].

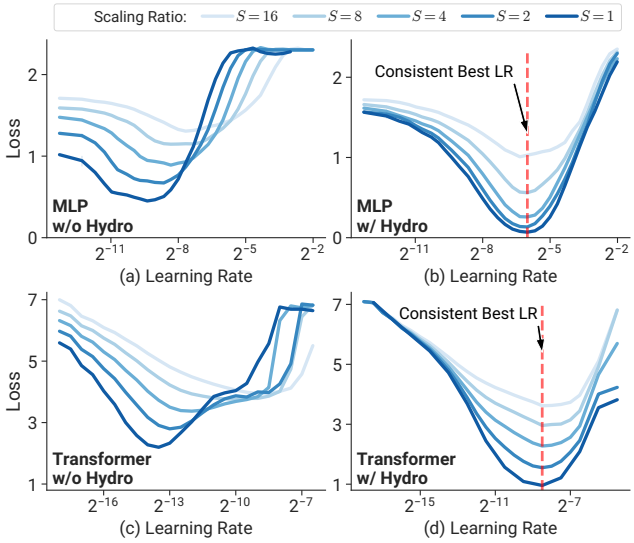

Figure 1: Effect of Hydro parametrization. The training loss against the learning rate on MLP (a, b) and Transformer (c, d) with different widths. S denotes the model scaling ratio.

It generalizes the mean-field limit of the 1-hidden-layer case [25, 80] and should be the unique parametrization that retains the representation-learning capability (non-rigorously referred to *active training*, in contrast to lazy training of NT parametrization) for a large-scale neural network, which means training does not become trivial or stuck at the initialization in the large width limit. Colloquially, it is designed to solve the issue that the input layer is updated much more slowly than the output layer, and make all hidden activations update with the same speed in terms of width [117].

Hydro adopts the MU parametrization, which will be further elaborated in §4.1 and we refer readers to [98, 117–122] for a comprehensive review of the theory.

2.3 Opportunities for Efficient Tuning

Lightweight surrogate-based tuning. Current HPO systems search hyperparameters directly on the *target model*, which is intuitive but inefficient. In contrast, Hydro makes it possible to tune a model with a much smaller *surrogate model* via applying a novel hyperparameter transfer technique (aforementioned in §2.2). For a clearer illustration of the surrogate-based tuning effect, we employ Hydro parametrization on two toy models and plot their converged training loss against a range of learning rates as shown in Figure 1. Specifically, the target MLP model contains two hidden layers (width=4096) and we train it with SGD on CIFAR-10. Similarly, the target Transformer model contains two TransformerEncoder layers (width=4096, i.e., d_{model}) and we train it with Adam on WikiText-2. Besides, we generate surrogate models with different scaling (shrinking) ratios S, and the smaller model is depicted by the lighter blue line. For instance, S=2 represents the model with width=2048. Obviously, the conventional training paradigm (Figure 1 (a, c)) cannot share the best hy-

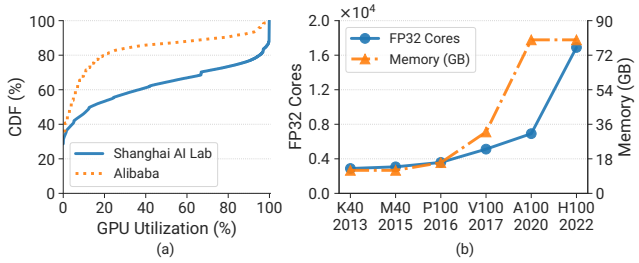

Figure 2: (a) GPU utilization distribution of one partition in our cluster and a GPU production cluster in Alibaba [115]. (b) Exponential growth of NVIDIA datacenter GPU capability. x-axis: GPU model name & release year.

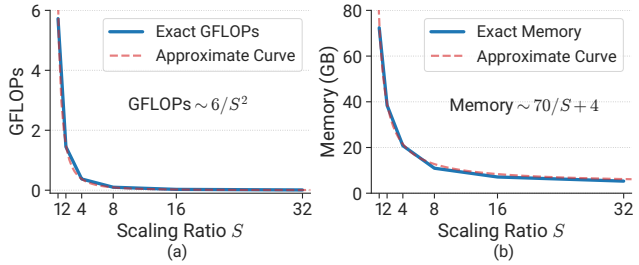

Figure 3: Model scaling effect of WideResNet-50. (a) Model GFLOPs (Giga Floating Point Operations). (b) GPU memory.

perparameter across different sizes of models and there are orders of magnitude optimal learning rate shifts. However, Hydro parametrization (Figure 1 (b, d)) makes surrogate models stay approximately the same optimal learning rate as the target model, which implies the feasibility of surrogate-based tuning. Furthermore, Hydro parametrization can deliver better performance since both tuned MLP and Transformer achieve lower training loss than their counterparts. An intuitive explanation is that the learning rate of the conventional paradigm must tame logits' surge, but preceding layers do not learn appreciably. We perform a comprehensive evaluation of several models in §6.3 and demonstrate the superiority of Hydro.

Fusion of numerous repetitive models. GPUs are commonly underutilized in DL clusters [45, 46, 115, 124, 125]. Figure 2 (a) plots the Cumulative Distribution Function (CDF) of one-week GPU utilization in one partition of our cluster, as well as an Alibaba trace [115] for reference. We find there are only 16% and 35% of GPUs achieving higher than 50% GPU utilization in Alibaba and Shanghai AI Laboratory respectively. This issue will be exacerbated if the Hydro surrogate-based tuning technique is applied. For instance, Figure 3 presents the model scaling effect of training WideResNet-50 on ImageNet, where GFLOPs follows approximately inverse-square (c_1/S^2) trend drop and memory footprint follows roughly $c_1/S + c_2$ trend decrease (c_i indicates constant). This implies model scaling can significantly reduce the computation overhead, but resources are more prone to be underutilized. To this end, inspired by JAX `vmap` function [35, 112], Hydro implements an *inter-trial fusion* mechanism to automatically combine multiple models into one. Operators of multiple trials can be fused owing to the property of HPO tasks: essential is a set of identical models (or with minor mutation). Compared with the conventional GPU sharing mechanism (e.g., MPS), Hydro can achieve higher training throughput, GPU utilization and lower memory footprint (Figure 8).

Cluster resource awareness. Although HPO jobs are pervasive in GPU datacenters, cluster schedulers typically regard them as general training workloads without any specific design. On the other hand, HPO systems [9, 72, 84] are cluster resource agnostic. This causes cluster-level inefficiency, such as long job queuing delay and low GPU utilization. However, the unique features of HPO jobs bring opportunities for more efficient tuning. (1) *Trial throughput insensitivity*. Unlike general DL jobs, HPO jobs are more tolerant to throughput slowdown of partial trials. Therefore, we can run more trials by leveraging ephemeral bubble resources of large language model training jobs, which are long-term existing in our datacenter (§5.1). (2) *Diminishing resource requirements*. Multi-fidelity HPO jobs usually explore plenty of trials at the beginning and gradually decrease the search concurrency [32, 71, 82]. At the final stage, only a few trials are exploited. Therefore, we can not only reduce the total resource amount progressively, but also properly leverage the heterogeneous GPU resource (§5.2). With the rapid evolution of GPU computing capability as shown in Figure 2 (b), they become more prone to be underutilized for most small-scale trials [87]. Allocating trials to appropriate GPUs can significantly improve cluster-wide efficiency without hurting a single HPO job makespan.

3 Hydro Overview

Design principles & goals. For practical adoptions, Hydro follows three design principles: (a) *Automatic and simple*. Manually converting surrogate models is tedious and error-prone. Hence, the whole tuning workflow should be automated and easy to use, which requires minimum user code modification. (b) *Incentive and interference-free*. Although our system focuses on optimizing HPO jobs, it does not sacrifice other workload performance. Instead, it is altruistic and requires fewer resources than conventional systems, which benefits all cluster users. (c) *Modular and extensible*. Each component in Hydro can work independently to support more scenarios (e.g., cloud). Moreover, Hydro can be applied to general HPO tasks, and more tuning algorithms can be easily integrated. In addition, Hydro has two primary objectives: (1) minimizing the makespan of HPO workloads; (2) improving the cluster-wide resource utilization.

System architecture. Figure 4 depicts the architecture of the Hydro service. It consists of two key system components: ***Hydro Tuner*** (blue block) as a user interface to automatically generate surrogate models and optimize tuning trials, and ***Hydro Coordinator*** (purple block) for improving tuning efficiency and datacenter-level resource utilization. Each component contains several modules for different purposes. Specifically, there are three main modules in Hydro Tuner:

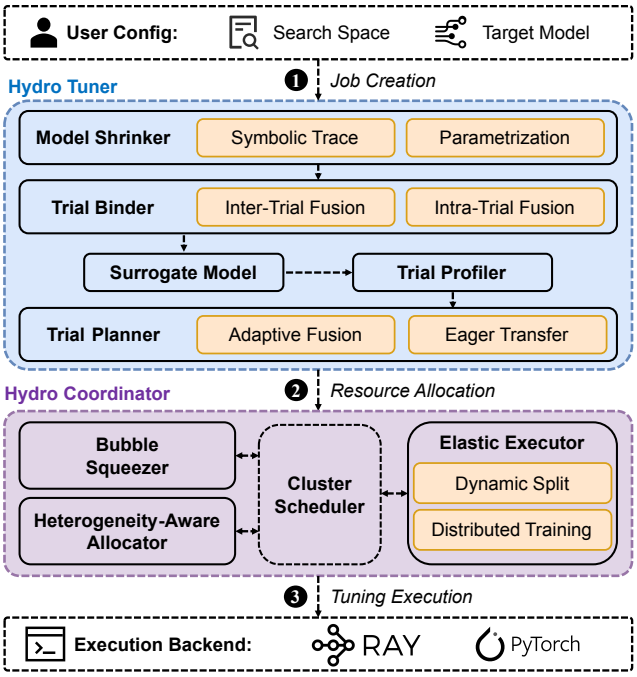

Figure 4: Overview of Hydro architecture and workflow.

- *Model Shrinker*: to obtain surrogate models by automatically tracing, scaling and parametrization.
- *Trial Binder*: to better utilize accelerators by binding multiple trials and fusing internal operators.
- *Trial Planner*: to adaptively determine the tuning strategy based on the profiling information and intermediate results.

Additionally, Hydro Coordinator also includes three modules:

- *Bubble Squeezer*: to extend tuning workload resources by interleaving training with a pipeline-enabled large model.
- *Heterogeneity-Aware Allocator*: to improve resource utilization and cluster-wide performance by allocating proper accelerators on different tuning stages.
- *Elastic Executor*: to dynamically execute trials by splitting fused trials and enabling distributed training.

API Design. Hydro enables high-efficient surrogate-based hyperparameter tuning with a few lines in the developer's code, as shown in Figure 5. It follows the Ray Tune [72] API to define the search space and invoke the `fit()` function. To support Hydro functions, developers only require to wrap their model, dataloader and optimizer with the `prepare_xxx()` API (lines 6∼8). Hydro traces the whole function to control the trial execution, convert surrogate model, enable model fusion and elastic training.

Tuning Workflow. The system workflow of Hydro is presented by black arrows in Figure 4. Specifically, when a developer wants to tune a model, she only needs to define the search space and invoke the Hydro APIs (❶). After job creation, Hydro Tuner automatically generates and optimizes surrogate models by scaling and fusion. Furthermore, it adopts *Trial Planner* to efficiently orchestrate the tuning process. Then Hydro Coordinator is responsible for contacting the cluster

```
1  import ray, hydro
2  import hydro.train as ht
3
4  def train_func(config):
5      # Wrap model, dataloader and optimizer
6      model = ht.prepare_model(model)
7      data_loader = ht.prepare_data_loader(data_loader)
8      optimizer = ht.prepare_optimizer(SGD, lr=config["lr"])
9      for _ in range(1):  # User defined training loop
10         train_epoch(...)
11         result = validate_epoch(...)
12         ray.session.report(result)
13
14 search_space = {"lr": ray.qloguniform(1e-4, 1, 1e-4)}
15 trainer = hydro.Trainer(train_func)
16 tuner = hydro.Tuner(trainer, search_space, scaling_num=8)
17 results = tuner.fit()
```

Figure 5: A code example of how to use Hydro APIs to define the search space and perform hyperparameter tuning.

scheduler to dynamically allocate resources and execute trials (❷). It supports two novel mechanisms, which leverage ephemeral bubbles and heterogeneous resources to further improve datacenter efficiency. Finally, the tuning job is successfully scheduled and starts running, where Ray [84] and PyTorch [91] serve as the execution backend (❸). More details are introduced in the following sections (§4 & §5).

4 Hydro Tuner

Hydro Tuner is a core component of the Hydro service for job-level optimization. It consists of three modules: Model Shrinker, Trial Binder and Trial Planner.

4.1 Model Shrinker

Model Shrinker aims to obtain surrogate models by automatically tracing, scaling and parametrizing the target model. The upper part of Figure 6 depicts its workflow. It first traces the target model and edits each layer's configuration to build a scaled model (①). To enable hyperparameter transfer, it then automatically parametrizes the scaled model by reinitializing the weight and adjusting the learning rate of each layer accordingly (②). Below we first summarize the MU parametrization theory that Hydro parametrization relies on, and then introduce how Hydro brings it into practice.

Maximal Update parametrization. As introduced in §2.2, Hydro employs the MU parametrization theory [117, 121] to search hyperparameters on a small surrogate model and transfer them to the large target model. The theory is built atop Tensor Programs [117–122], a unified theoretical framework that formulates the computation of common neural networks components as Gaussian Processes (GPs), including multilayer perceptrons (MLPs), recurrent neural networks (RNNs) (e.g., Long-Short Term Memory (LSTM) [21]), skip connections [41], convolutions [62] or graph convolutions [55], pooling [62], batch normalization [48], layer normalization [20],

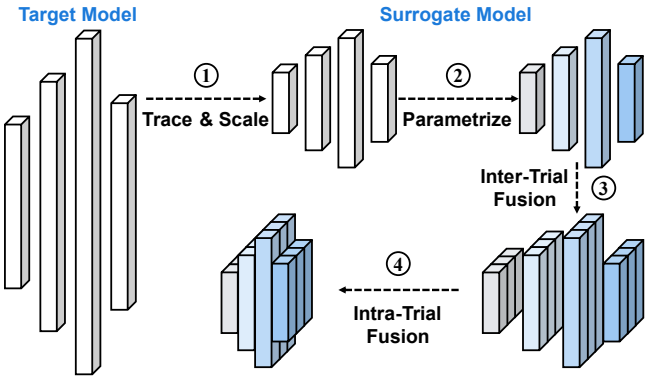

Figure 6: Illustration of Model Shrinker (①, ②) and Trial Binder (③, ④). The length of each bar represents layer width.

and attention [108]. As a result, many practical models like ResNet [41] and Transformer [108] can be expressed as GPs and apply MU parametrization, since they inherently consist of these basic components.

➤ ***Theory assumption***. In contrast to prior works such as NTK [49] that necessitate unnatural conditions, MU parametrization only requires standard Gaussian initialization for the model, which is easily achievable in practice. In terms of data, MU parametrization requires i.i.d. samples, which is typically present in the same dataset. However, this requirement also limits its ability to support fine-tuning (§7).

➤ ***Key insight and mechanism***. The main idea of MU parametrization is: every activation vector has roughly i.i.d. coordinates at **any time** during training neural networks in the infinite-width limit. It aims to overcome the imbalanced per-layer learning speed issue in practice. To this end, MU parametrization performs layer-wise fine-grained adjustment, including per-layer initialization variance, learning rate and other optimizer-related hyperparameters (e.g., SGD momentum, Adam beta). Specifically, since the output layer is updated much faster than the input layer, MU parametrization suppresses the learning rate and initialization variance of output weights by w (width) times. In addition, for SGD-like optimizers (linear tensor update), the learning rate of input weights and all biases is multiplied by w. For Adam-like optimizers (non-linear tensor update, normalizes the gradient coordinate-wise), the learning rate of hidden weights is divided by w. Hence, MU parametrization ensures consistent magnitude updates for each layer during training regardless of its width so that hyperparameters can be transferred across models with different widths at any time (i.e., same converge speed across scaled models).

To summarize, in the large width limit, MU parametrization reveals that hyperparameters yielding lower training losses for narrower models also result in better performance for wider models through a specific transfer mechanism. Hydro leverages this effect to obtain better test accuracy efficiently via surrogate-based tuning, albeit without a rigorous theoretical guarantee for every model.

➤ ***Instructive example***. To provide a clearer explanation of why parametrization is necessary and how it operates, we recapitulate the key insights of [121] with an instructive example [117]. Consider a 1-hidden-layer linear model $f(x) = V^\top Ux$ with scalar inputs and outputs, as well as w-width layer weights $V, U \in \mathbb{R}^{w\times 1}$. In common practice (e.g., Xavier initialization [37]), we initialize them with $V \sim \mathcal{N}(0, 1/w)$ and $U \sim \mathcal{N}(0,1)$, which ensures $f(x) = \Theta(|x|)$ at initialization ($\Theta(\cdot)$ indicates asymptotically tight bound). After one step of SGD with learning rate 1, the new weights are $V' \leftarrow V + \theta U$ and $U' \leftarrow U + \theta V$, where θ is some scalar of size $\Theta(1)$ depending on the inputs, labels, and loss function. Then

$$\begin{aligned} f(x) &= V'^\top U' x \\ &= \left(V^\top U + \theta U^\top U + \theta V^\top V + \theta^2 U^\top V\right) x \end{aligned} \tag{2}$$

which will blow up with width w in the infinite limit because $U^\top U = \Theta(w)$ by Law of Large Numbers. In other word, it only allows $O(1/w)$ learning rate so as to avoid float overflow, and yield kernel limits (§2.2). Consequently, it fails to perform feature learning (learning rate $\to 0$) to update weights after random initialization.

However, by applying maximal update parametrization, we have $V \sim \mathcal{N}\left(0, 1/w^2\right)$, $U \sim \mathcal{N}(0,1)$, learning rates $\eta_V = 1/w$ and $\eta_U = w$. After one step of SGD, now we have

$$f(x) = \left(V^\top U + \theta w^{-1} U^\top U + \theta w V^\top V + \theta^2 U^\top V\right) x \tag{3}$$

and one can verify this is $\Theta(1)$ and remains bounded. In contrast to common practice, MU parametrization has $\Theta(1)$ learning rate and admits feature learning *maximally*, which allows every parameter to be updated maximally (in terms of scaling with width) without leading to float overflow.

➤ ***Heuristic adaptation***. While Tensor Programs support more versatile model components (e.g., convolution), obtaining a closed-form solution for arbitrary models is infeasible. The efficacy of the MU parametrization has been rigorously demonstrated on a 2-hidden-layer MLP trained with SGD for multiple steps, and the proof can be readily extended to deeper MLPs [121]. For more general models in practice, some heuristic tricks are adopted to enhance their hyperparameter transferability. For example, Transformer [108] models require two additional operations in the self-attention: (1) scaling the attention logit by $1/d_k$ rather than $1/\sqrt{d_k}$, where d_k is the attention head size; (2) zero initialization on query layer q. We also empirically find that using a larger sequence length provides a better transfer effect for Transformer models. For models with some special components or architecture (e.g., MoE [101]), hyperparameters may not well transfer with MU parametrization alone. Hence, additional analysis and tailored adjustments may be required.

Hydro parametrization. It is arduous and error-prone to implement MU parametrization manually to generate a surrogate model. Developers are required to not only thoroughly understand the MU parametrization theory, but also manually

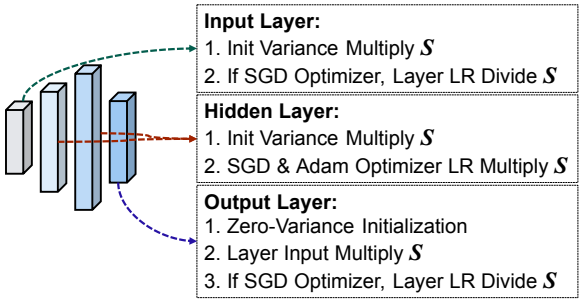

Figure 7: Hydro parametrization implementation. Illustration on a simple 4-layer model with SGD or Adam-like optimizer.

adjust the model width, initialization function and learning rate layer by layer. Any incorrect adjustment may directly incur hyperparameter transfer failures. To this end, we implement Hydro parametrization, an automated and simplified parametrization strategy based on MU parametrization. We demonstrate the excellent effect of Hydro parametrization with visualized results in Figures 1 and 10.

For a clearer illustration, we present the Hydro parametrization process in Figure 7, which applies different strategies to the input, hidden and output layers. Developers only need to specify their desired *scaling ratio* S ($S = 8$ by default) and then Hydro will parametrize the model accordingly. Concretely, at the model initialization stage, we apply zero-variance initialization to the output layer instead of $1/w^2$, which will not be detrimental to performance and can remove this mismatch issue between the surrogate model and target model in the initial Gaussian process [117]. Moreover, we apply zero initialization to all biases, and weights as well as learning rate scaling strategies are annotated in the figure, which is invoked by the `prepare_optimizer` API to build a `hydro_optimizer`. Besides, we insert a `Multiply` layer in front of the output layer to scale its input by S.

Applicable Scope: Hydro parametrization works well for most ubiquitous hyperparameters that control model initialization and training, including learning rate, batchsize, lr_scheduler, momentum, etc. However, it has limited support on regularization-related hyperparameters, such as weight decay and dropout, because they naturally depend on both the model size and data size. Although parametrization cannot be applied to all hyperparameters, it is sufficient to achieve qualified performance in most cases. After most hyperparameters are tuned with the surrogate model, developers can further tune the regularization hyperparameters within a much smaller search space on the target model if needed. Moreover, we provide a comprehensive summary of additional limitations associated with Hydro parametrization in Section 7.

Trace and scale. Before performing the above parametrization, we need to first trace the target model and build a scaled model. Since there are various model definition styles in the PyTorch ecosystem, it is necessary to obtain a uniform and equivalent modality from disparate community model codes. We implement `HydroTracer` based on *torch.fx* [97], which allows developers to trace and edit the model. Specifically, we replace *call_function* nodes (e.g., *torch.nn.functional*) with the corresponding *call_module* nodes (e.g., *torch.nn*) for subsequent layer scaling and fusion (§4.2). We apply different scaling rules to the input, output and hidden layers. For instance, we parse *nn.Linear* kwargs and modify both the *in_features* and *out_features* values by dividing S for hidden layers. In addition, we only scale the *out_features* of input and *in_features* value of output layers. To handle the data-dependent control-flow, we use proxy nodes along with developer-provided concrete values to determine the execution flow [61]. According to our evaluation of notable models, including TorchVision [18] (e.g., ResNet [41], MobileNet [44], VGG [103]) and HuggingFace Transformers [113] (e.g., BERT [30], GPT [95], Swin [76]), developers can trace and scale these models with Hydro without modifying the code.

Correctness check. While Hydro has achieved automatic parameterization, there are still potential failures due to certain special model components that require heuristic adaptation as previously mentioned, as well as other corner cases that have not been considered. To this end, we further implement a safeguard mechanism to check the correctness of the parametrization and notify users whether they should use Hydro to prevent misleading hyperparameters and resource wastage. Firstly, Hydro performs a simple per-layer width check when scaling to avoid too narrow layers (e.g., only 1 neuron width for a Linear layer). Additionally, taking inspiration from gradient checking as a simple method for verifying the correctness of an autograd implementation, Hydro has a quick parameterization profiling stage that checks whether the average size (L1 value) of each activation vector is bounded to avoid possible parameterization failure based on [117]. It only lasts for very few steps at the beginning of the HPO job.

4.2 Trial Binder

Although Model Shrinker dramatically reduces the computation of each trial (Figure 3), it inevitably incurs the resource underutilization issue, which deteriorates small- or mid-size target models (e.g., deployed on edge devices). To address this problem, Trial Binder further optimizes surrogate models by binding multiple trials and fuses internal operators to better utilize accelerators. We illustrate its mechanism in the bottom part of Figure 6. It merges a set of fusible trials into a *HydroTrial* with grouped operators and optimizer (③). To further accelerate training, we automatically just-in-time (JIT) compile the fused (*inter*-) surrogate model to obtain fast and flexible fusion (*intra*-) kernels (④). Note that the last model with closer layer distance represents the reduced memory-bounded operations through intra-trial fusion.

Inter-trial fusion. There are plenty of trials with the same or similar model structure in a HPO job. Inspired by JAX `vmap` [35, 112], which returns a batched version of the target function by vectorizing each input along the axis specified, we can batch multiple trials into a single one by fusing their opera-

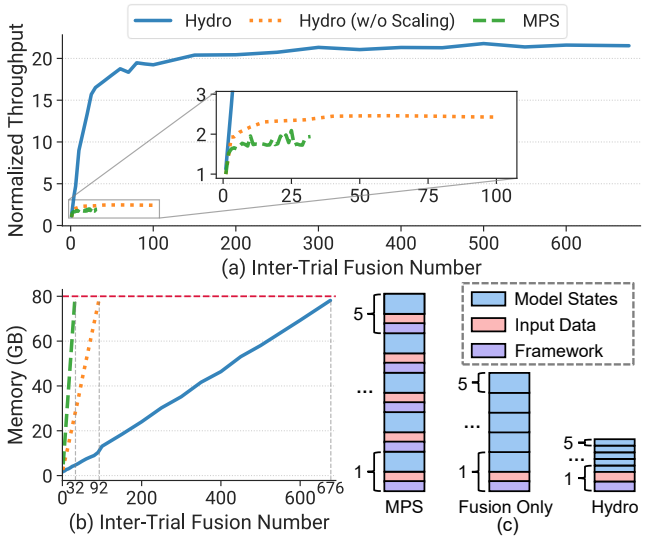

Figure 8: Inter-trial fusion effect on ResNet-18. (a) Accumulated throughout of fused surrogate model w.r.t the target model. (b) GPU memory footprint of different fusion counts. Red horizontal line denotes the A100 memory bound. (c) Schematic diagram of memory occupation detail of 5 models GPU sharing with MPS, Hydro and Fusion (w/o Scaling).

tors. Hydro implements an *inter-trial fusion* mechanism to automatically bind surrogate models. Specifically, Trial Binder traverses the traced surrogate model and replaces the *torch.nn* operators with grouped `hydro.nn` operators according to the predefined fusion rule and fusion count F determined by Trial Planner. `hydro.nn` provides mathematically equivalent implementations of batched original PyTorch operators based on HFTA [110]. For instance, `hydro.nn.Linear` is implemented atop *torch.baddbmm* (i.e., batch matrix-matrix product and add), which adds an additional dimension batch (i.e., F) compared with *torch.nn.Linear (addmm)*. Besides, for each `hydro.nn` operator, we reimplement the initialization function to support independent model-wise Hydro parametrization and realize the defusion mechanism to extract a specific sub-model. Additionally, `hydro_optimizer` and `hydro_lr_scheduler` are designed to support both the model fusion and parametrization simultaneously. These are performed automatically, and developers typically do not need to understand the rationale and modify codes.

Figure 8 plots the extraordinary effect of integrating model scaling with inter-trial fusion on ResNet-18 ($S = 8$), tested on CIFAR-10 with batchsize=256. It is evident that Hydro is capable of concurrently training impressive 676 models on a single A100 GPU. Compared with the conventional GPU sharing mechanism MPS [13] (MIG [12] has similar performance), Hydro achieves over 10× training throughput improvement and over 20× GPU memory conservation. If we directly apply inter-trial fusion to the target model (without scaling), the throughput improvement is relatively much limited. Furthermore, we provide an intuitive interpretation of how memory footprint reduction occurs in Figure 8 (c). The model states (blue blocks) encompass all aspects associated with model training such as model weights, gradients, activations, and optimizer states [96]. MPS has repetitive memory overheads incurred by CUDA context of DL framework (purple blocks) and independent data loading (pink blocks). In contrast, Hydro avoid such redundancy and further reduce model-related memory footprint. Note that here we only compare with vanilla training paradigm without considering more advanced memory optimization techniques like Salus [124]. Moreover, beyond the better GPU utilization and higher throughput, inter-trial fusion also alleviates the I/O pressure owing to the accompanied data-loading fusion.

Lazy intra-trial fusion. Hydro supports automatic model fusion to further accelerate training based on the `nvFuser` [10] compiler backend. Although plenty of previous works [51, 107, 129] demonstrate that operator fusion can improve training throughput via better memory locality, it does not always bring benefits to HPO workloads due to its high compiling overhead. For instance, `nvFuser` [10] takes approximately 2-epoch time to compile a ResNet-18 model to deliver around 10% speedup per epoch, which means a trial needs to run at least 20 epochs to avoid slowdown. However, most trials will end up in a few epochs for multi-fidelity tuning algorithms. To this end, Hydro apathetically adopts the intra-trial fusion. For simplicity, Hydro currently only applies to trials with *deterministic* training steps, such as all `HydroTrials` when applying single-fidelity tuning algorithms and the trial that trains the target model with transferred hyperparameters.

4.3 Trial Planner

Trial Planner is the key module that interacts with the tuning algorithm and trial executor. We introduce two mechanisms that improve the surrogate-based tuning efficiency.

Adaptive fusion. The trial count and resource amount vary significantly across different HPO jobs. Hence, the fusion count F of each `HydroTrial` should be adaptively determined to achieve the desired performance. Hydro contains the following steps to fuse trials and assign GPUs: (1) Trial Planner invokes the tuning algorithm to generate a set of hyperparameter configurations (trials). (2) Since inter-trial fusion requires trials with the same operator shapes, we split them into different *trial groups* according to their batchsizes. (3) Based on the linear growth of GPU memory shown in Figure 8 (b), we can profile the trials with $F = 1$ and $F = 2$ for each *trial group* and estimate the upper bound of the fusion count F_{max}. (4) Hydro assigns all available GPUs to each *trial group* according to group's weight, which equals to $B \times N$ (denoted as the product of batchsize and trial count of the group). (5) Each *trial group* evenly distributes trials based on the group GPU amount and F_{max}, and Hydro fuses them as a `HydroTrial` on each GPU. In this way, Hydro can leverage as many GPUs as possible and achieve the optimal global throughput.

Eager transfer. As the HPO job progresses, more and more

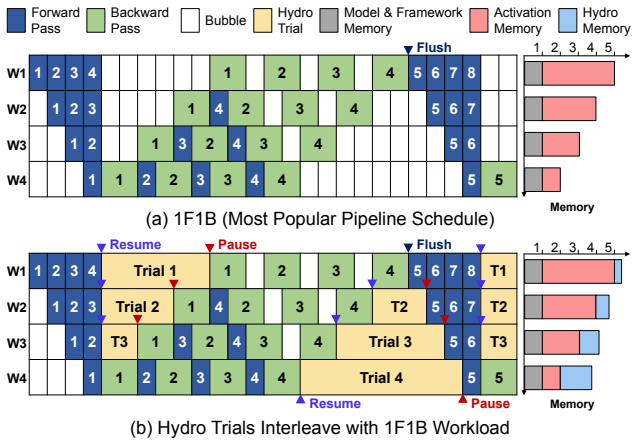

Figure 9: Illustration of (a) 1F1B Pipeline and (b) Hydro Bubble Squeezer, with four pipeline stages and four micro-batches. Note the right-side memory diagrams can only reflect the relative relation of the *same* color blocks across workers.

trials terminate and the degree of the parallelism gradually decreases, resulting in underutilized or idle resources. On the other hand, the best hyperparameter configuration sometimes appears in the early stage. Therefore, instead of training the target model after all the surrogate-based tuning trials are done, we can eagerly transfer the intermediate best hyperparameters and leverage vacated resources to validate the configuration on the target model. Hydro records all evaluated hyperparameters and schedules a *TargetTrial* for the target model training when 50% (customizable) of the surrogate-based tuning trials are done and there exist idle resources. If a better hyperparameter is searched, Hydro terminates the on-going *TargetTrial* or starts a new *TargetTrial* depending on the resource utilization. This mechanism efficiently shortens the job makespan and improves the resource utilization.

5 Hydro Coordinator

Hydro Coordinator focuses on cluster-level optimization. It consists of three modules: Bubble Squeezer, Heterogeneity-Aware Allocator and Elastic Executor. It is important to highlight that the first two modules are tailored for specific cluster scenarios. Specifically, Bubble Squeezer can only be activated when a pipeline-enabled foundation model pretraining job is running within the cluster. The Heterogeneity-Aware Allocator is meticulously designed to better leverage multiple generations of GPUs coexisting in the cluster.

5.1 Bubble Squeezer

In addition to HPO jobs, there are many kinds of workloads that coexist in the GPU datacenter, such as inference, debugging and large-scale distributed training jobs [45, 50, 111]. With the rapid popularity of foundation models (e.g., GPT-3 [24]) in recent years, some large model pretraining workloads exist in our datacenter in the long term. As complained by many users, the majority of machines are occupied by large model training jobs that usually last for days to weeks, which incurs the starvation of other jobs. Additionally, the pipeline parallelism [85, 88] is usually adopted to support a larger model by splitting it into several stages and placing them across multiple workers. However, bubbles inherently exist in the synchronous pipeline parallelism [106], such as the commonly used 1F1B [34, 86] strategy. Besides, the imbalance peak memory issue (Figure 9) between different pipeline stages further exacerbates the resource inefficiency [65].

Hydro designs Bubble Squeezer, which leverages bubbles to greatly extend the tuning job resources in an interleaving execution way, almost without hurting the training throughout of large models. `HydroTrials` are perfectly suitable for the bubble interleaving execution due to the following unique features: (1) *Throughput insensitivity*. Unlike general DL training jobs, tuning jobs are more tolerant of the slowdown of partial trials. This inspires us to squeeze the spare resources of the bubbles and execute trials in a pause-and-resume way. (2) *Deterministic resource pattern*. General small-scale workloads (e.g., debugging) have unknown and unpredictable resource requirements. However, Hydro profiles and records the resource consumption of `HydroTrials`, mitigating the potential out-of-memory (OOM) issues if they are colocated with large models. (3) *Elastic trial size*. Based on Model Shrinker, the scaled model has a much smaller memory footprint (Figure 3) than the original model, which means we typically do not need to swap out its GPU memory during colocation. Besides, we can dynamically adjust the trial fusion count according to the remaining GPU memory with Trial Binder.

To clearly illustrate how Bubble Squeezer works, we first introduce the 1F1B pipeline parallelism in Figure 9 (a). It transfers intermediate activations of the partial model at the forward and backward passes between different workers using point-to-point communication [130], thus each worker cannot continuously utilize the GPU. For Worker 1, after the forward pass of the last micro-batch (blue block 4), it has to wait for the backward pass of the first micro-batch (green block 1), leaving GPU idle for a long time. Other workers also present similar bubble patterns but occupy less GPU memory since fewer activations of micro-batch needed to store.

In Figure 9 (b), Hydro interleaves four `HydroTrials` of different sizes with the large model training workload. Each trial executes in a pause-and-resume paradigm to squeeze the bubbles. Since Hydro Tuner has traced and canonicated each layer with `hydro.nn`, we further register hooks on each module of the trial to support on-demand pause and resumption in the forward and backward passes of each layer. When a large model training job exists, Hydro coordinates with datacenter scheduler to acquire more GPUs from this model and tags them as *ephemeral* resources. For the large model, we also implement a corresponding hook inside its training framework (i.e., DeepSpeed [96]) to report its training progress and resource consumption. Each worker executes its corresponding pipeline under DeepSpeed's pipeline parallelism. Therefore, we implement a fine-grained synchronization mechanism to

guarantee that HydroTrials only could be executed within the bubbles, by intercepting the status of the CUDA streams of the NCCL kernels. Hydro can further adjust the fusion count to adaptively fit in the remaining memory and improve GPU utilization. At the beginning and end of the bubble of large model training, we control the resumption and pause of trial model training by Linux signals. The fine-grained suspend-resume control eliminates the performance interference caused by CUDA kernels running simultaneously.

In general, the effectiveness of Bubble Squeezer varies depending on multiple factors, and we present the scenarios where it works best. Regarding the HPO job aspect, Hydro is more effective when using (1) *multi-fidelity tuning algorithms* because they allow most trials to be terminated in a few epochs using the ephemeral resources and execute immediate top trials on exclusive resources to avoid possible blocking caused by interleaving slowdown. In addition, (2) *models with fewer layers* are preferred as they are prone to complete the entire iteration within the bubble time and require relatively less memory to support a higher fusion number. As for pipelined large model aspect, Hydro can achieve better performance when the pretraining job has (3) *more pipeline stages across more servers*, which implies a higher bubble ratio and more ephemeral resources. A large model pretraining job typically can support multiple different HPO jobs interleaving simultaneously and accelerate dozens, even hundreds of HPO jobs (depending on its resources and duration scale) during its pretraining process. In addition, there may be cases where some scaled models are still too large to be allocated on any GPU of the pretraining model. Due to the high memory swap overhead in our scenario, Hydro does not support offloading techniques like Bamboo [106]. As a result, Bubble Squeezer is unable to support such models.

5.2 Heterogeneity-Aware Allocator

HPO workloads generally have diminishing resource requirements [71]. They usually explore plenty of trials at the beginning and gradually decrease the search concurrency. At the final stage, only a few trials are exploited. Hence, tuning with fixed GPU resources can lead to underutilization. Existing HPO systems [32, 82] support autoscaling to dynamically adjust the tuning resources. However, they do not consider the GPU heterogeneity in the datacenter.

Inspired by Gavel [87], a novel heterogeneity-aware cluster scheduler for general DL jobs, we design a resource allocator to allocate appropriate GPUs to trials, which can improve the cluster-wide efficiency without sacrificing the job makespan. Hydro supports both resource autoscaling and heterogeneity-aware allocation. Specifically, if there is any node or GPU idle for over 1 minute (customizable), Hydro will interact with the cluster scheduler to release the resource. Other affiliated resources like CPU will also be released as a bundle. Additionally, Hydro creates *TargetTrial* with the eager transfer mechanism and makes the target model training process well hidden inside the tuning time. Since the *TargetTrial* typically trains alone without fusion, it may not be able to fully utilize the GPU resources. So Hydro will place it on an GPU of old version (e.g., V100) if its `SM Activity` rate (measured by NVIDIA DCGM [11]) is lower than 50% (customizable). Similar action will be applied to surrogate models if their allocated resources are underutilized and there exist other HPO jobs pending in our service queue.

5.3 Elastic Executor

Elastic Executor is designed to improve the job efficiency by leveraging all assigned GPU resources. It supports two elastic mechanisms: (1) dynamic split and (2) automated distributed training. Specifically, when an idle GPU emerges, the *fused* HydroTrial will not directly increase its GPU count by conventional distributed training. Instead, Hydro will evenly split this HydroTrial into multiple HydroTrials and exclusively place them on the idle GPUs to reduce the communication overhead. Furthermore, since the memory footprint of some large models is high even though scaled, Hydro supports two types of elastic strategies for *unfused* surrogate models: (a) *Evenly distribute*: allocating idle GPU resources to all unfused surrogate models evenly. (b) *Performance-aware* (default): allocating idle GPU resources to the top performing trial. For the target model, Hydro automatically increases the number of workers to enable distributed training.

6 Evaluation

Hydro is implemented on top of Ray [72, 84] with about 12K LoC. For Hydro Tuner, Model Shrinker relies on torch.fx [97] and mup [117], while Trial Binder is built with HFTA [110] and nvFuser [10]. As for Hydro Coordinator, we modify DeepSpeed [96] to further support Bubble Squeezer and validate the interleaving execution as a prototype. And the Elastic Executor based on Ray Train as well as PyTorch FSDP [17].

We evaluate Hydro Tuner and Hydro Coordinator independently for a fair comparison. Our experiment search space does not include weight decay because Hydro is unable to transfer regularization hyperparameters, but it is sufficient to achieve qualified performance without tuning it.

6.1 Experiment Setup

Testbed. We conduct our experiments on a GPU datacenter of Shanghai AI Laboratory. Each node has 8 NVIDIA A100 80GB GPUs, 2 AMD EPYC 7742 CPUs (128 cores) [2] and 1TB memory. GPUs are interconnected to each other by NVLink and NVSwitch [14], and inter-node communication is achieved via NVIDIA Mellanox 200Gbps HDR InfiniBand [7]. All the experiments are conducted on A100 GPUs, unless explicitly stated in §6.5.

Workloads and search spaces. We evaluate Hydro tuning performance over six popular CV/NLP models, as listed in Table 2. Specifically, *GPT-3 XL* is a large language model architecture belonging to GPT-3 family. It contains 1.3B parameters and we use an open source implementation by GPT-

Task	Search Space	Model	Dataset	Optimizer	# of GPU	# of Trial	Avg. Time Reduction	Avg. Quality Difference	Size
Language Modeling	lr: $\mathbf{U}_{Q\log}(10^{-5}, 10^{-1}, 10^{-5})$ gamma: $\mathbf{U}_{Q}(0.01, 0.9, 0.01)$	GPT-3 XL [24]	OpenWebText [38]	AdamW	128	100	78.5 ×	−0.48 ppl	XL*
		Transformer [108]	WikiText-103 [81]	Adam	8	200	8.7 ×	−0.15 ppl	M
Image Classification	lr: $\mathbf{U}_{Q\log}(10^{-4}, 1.0, 10^{-4})$ momentum: $\mathbf{U}_{Q}(0.5, 0.999, 10^{-3})$ batchsize: [128, 256, 512] gamma: $\mathbf{U}_{Q}(0.01, 0.9, 0.01)$	WideResNet-50 [126]	ImageNet [29]	SGD	32	200	20.3 ×	+1.18% acc	XL*
		MobileNetV3 Large [44]	Flowers102 [90]	Adam	16	500	12.3 ×	+0.05% acc	L
		VGG-11 [103]	CIFAR-100 [57]	SGD	8	500	10.8 ×	+0.09% acc	M
		ResNet-18 [41]	CIFAR-10 [57]	SGD	8	1000	16.2 ×	+0.02% acc	M

Table 2: Summary of (1) workloads used in the evaluation and (2) single-fidelity tuning improvements over Ray. *Model Quality*: ppl indicates perplexity (the lower the better) and acc denotes accuracy (the higher the better). * For XL tasks, we estimate the time cost of Ray based on simulation and use the official hyperparameter setting as the model quality baseline.

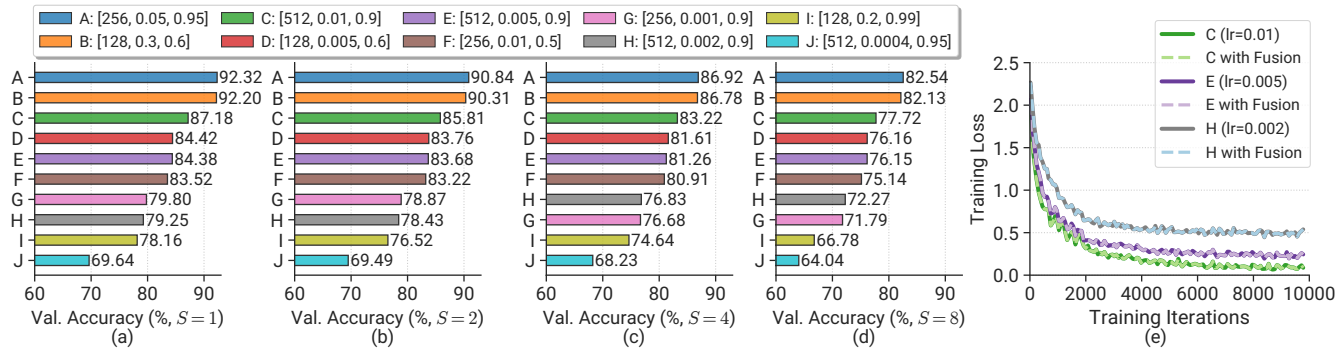

Figure 10: Hydro Tuner mechanisms validation. (a)∼(d) *Scaling validation*: randomly select 10 hyperparameter sets ([batchsize, lr, momentum]) to visualize the transfer effect of multi-dimensional hyperparameters across different scaling ratios $S = 1, 2, 4, 8$ on model ResNet-18. (e) *Fusion validation*: loss curves of the standard model (solid line) and inter-trial fused model (dash line).

Neo [5, 23]. We further enable mixed precision training for *WideResNet-50* and two language modeling tasks. For the dataset, we crop Flowers102 into 224×224 images, whose input size is the same as ImageNet. And we swap its train and test dataset split to get a larger training dataset to make it similar to more general jobs. Moreover, we denote single-node tasks as M-size, and distributed tuning tasks as L/XL-size.

We adopt three kinds of optimizers for above models, including SGD [99], Adam [54], and AdamW [77]. We use StepLR to decay the learning rate (lr) of each parameter group by gamma at every fixed step for all tasks. Additionally, we design two groups of search spaces for CV and NLP tasks respectively (Table 2), where $\mathbf{U}_Q(lower, upper, q)$ represents uniformly sampling a quantized (increment of q) float value between *lower* and *upper*. Similarly, $\mathbf{U}_{Q\log}$ uniformly samples in different orders of magnitude. Note that the search space of *MobileNetV3 Large* excludes momentum due to the incompatibility of Adam.

Tuning algorithms. Hydro supports multiple popular single-fidelity and multi-fidelity tuning algorithms, such as Random [22], HyperBand [64], ASHA [63]. Since our work focuses on system aspect optimization instead of tuning algorithms, we select two representative tuning algorithms in our evaluation: (1) *Random* (single-fidelity): fully evaluates each randomly generated trial; (2) *ASHA* (multi-fidelity): eliminates unpromising trials via asynchronous successive halving strategy. They are common hyperparameter tuning paradigms in practice. Besides, their asynchronous and prior-independent nature makes them more suitable for large-scale distributed tuning with numerous trials [71].

Baselines. We consider the following two systems as baseline: (1) *Ray* [72, 84]: performs HPO with the vanilla Ray Tune library; (2) *Ray+ES*: applies two advanced techniques in Ray Tune (*Elastic training* and *GPU Sharing*). Our implementation of *Ray+ES* refers to HyperSched [71] and Fluid [125]. Specifically, we place multiple trials on the same GPU using NVIDIA MPS [13] and allocate more GPU resources to the top performing trials if idle GPUs are available. We do not employ A100 MIG [12] sharing due to its similar performance with MPS but less flexibility [110]. Additionally, since existing popular HPO systems (Table 1) mainly differ in the application scenario and API design, and their system performance on the same tuning algorithm is similar, the Ray-based systems are sufficient for representing SOTA.

6.2 Surrogate-based Tuning Validation

Before performing end-to-end evaluations, we first give an intuitive experiment to validate the effect of surrogate-based tuning, which is the foundation of Hydro. As shown in Figure 10 (a)∼(d), we randomly choose 10 hyperparameter configurations (denoted as A∼J) on the ResNet-18 model and build

Model	# of GPU	# of Trial	Avg. Time Improvement	Avg. Quality Difference
GPT-3 XL	64	100	33.4 ×	−0.43 ppl
Transformer	4	200	5.8 ×	−0.09 ppl
WideResNet-50	16	200	9.7 ×	+0.87% acc
MobileNetV3 Large	8	500	8.0 ×	+0.08% acc
VGG-11	4	500	9.4 ×	+0.19% acc
ResNet-18	4	1000	14.5 ×	+0.05% acc

Table 3: Summary of multi-fidelity tuning improvements.

Deadline (s)	# of GPU	Model	Avg. Accuracy		
			Ray	Ray+ES	Hydro
900	4	VGG-11	65.42%	66.39%	**68.68%**
		ResNet-18	89.66%	90.71%	**91.32%**

Table 4: Summary of tuning performance with a deadline.

surrogate models with Hydro using different scaling ratios $S = 2, 4, 8$, where $S = 1$ represents the target model. We train each model for 100 epochs on the CIFAR-10 dataset with a fixed seed=1. Since the HPO job is essentially a ranking problem of hyperparameter configurations, we mainly care about whether the order is maintained especially for the top configurations, namely hyperparameter transfer effect. From the result, it is obvious that the performance ranking of hyperparameters transfers well across different scaling ratios. Admittedly, configurations G and H are swapped when $S \geq 4$, but it has no influence on the final tuning result since they perform poorly and top configurations keep a consistent ranking. Besides, the wider model always outperforms the narrower one under the same hyperparameters, which is inline with MU parametrization theory and demonstrates that surrogate model can effectively transfer multi-dimensional hyperparameters.

Additionally, we also validate the inter-trial fusion effect, which is another key mechanism of Hydro. Figure 10 (e) shows the training loss curves of trials C, E, H and their fused versions. We select these three trials because their `batchsize` and `momentum` are consistent and only differ in `lr`. As we can see, the convergence curves of the fused model well overlap with the original standalone training curves, which demonstrates that inter-trial fusion is a mathematically equivalent transformation and does not affect the model convergence.

6.3 End-to-End Performance of Hydro Tuner

To cover most hyperparameter tuning scenarios in practice, we conduct end-to-end experiments across 6 workloads with different settings and 3 common tuning paradigms (case I∼III). Note that Hydro Tuner adopts a fixed resource size (without enabling Hydro Coordinator) for fair comparisons.

Case I: single-fidelity tuning. When a user seeks for extremely excellent model performance with ample resources, single-fidelity tuning is applied to avoid missing the best hyperparameter configuration. Table 2 summarizes the Hydro

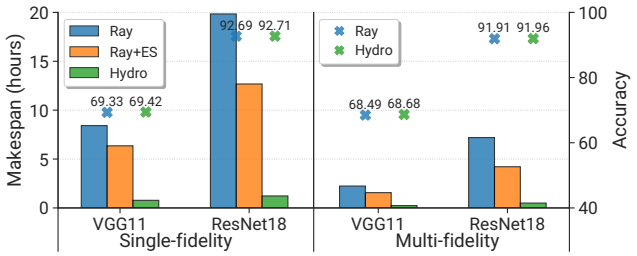

Figure 11: Summary of the end-to-end results. Bars indicate tuning makespan and points represent final model accuracy.

improvement on single-fidelity tuning over different sizes of workloads, where we apply $S = 16$ for XL models and $S = 8$ (default value) for other models. Since HPO jobs require completely training massive trials, we perform each experiment twice and report their average results on time reduction and tuned model quality over Ray. Besides, we obtain Ray tuning time of XL experiments based on simulation due to their unacceptable tuning cost, and adopt the official hyperparameter configurations [16, 24] to train the model as quality baselines. The target model training time is included in Hydro.

From the table, we can see that Hydro substantially outperforms Ray by 8.7∼78.5× in time reduction, while obtaining better final model quality. The time reduction mainly derives from two aspects: (1) *Less resource demand of trials*. For instance, the scaled GPT-3 XL trials do not require distributed training. For smaller models, Hydro further applies inter-trial fusion to improve trial concurrency and resource utilization. (2) *Smaller model trains faster*. Each trial has fewer FLOPs (Figure 3) to compute, which is more obvious on larger models. Additionally, we also observe that the effect of Hydro is more evident for larger models, with more intensive trials and fewer resources. This reflects Hydro is more suitable for large-scale HPO jobs with limited resources, which is hard to handle by existing systems.

Case II: multi-fidelity tuning. When a user desires to obtain a good model with a relatively lower cost, multi-fidelity tuning is applied to search hyperparameters efficiently. Table 3 reports the Hydro performance on multi-fidelity tuning. We keep the same experiment settings as Case I, except using half GPU resources. Besides, we configure ASHA [63] with $bracket = 1, grace = 3, reduction = 3$. We observe that Hydro can achieve 5.8∼33.4× reduction over Ray. Hydro can further benefits ASHA due to its much higher concurrency, which prevents the inaccurate promotion issue of ASHA [66]. Furthermore, we find that Hydro can also slightly improve the final model quality, which is mainly due to the different model initialization and more balanced layer-wise training rate configuration by Hydro parametrization. The results are also in line with Figure 1 that Hydro delivers a lower loss.

Case III: tuning with a deadline. When a user wants to get a model as good as possible by a fixed deadline, budget-bounded ASHA is applied. We simply evaluate two models with a deadline of 15 minutes as shown in Table 4. Hydro

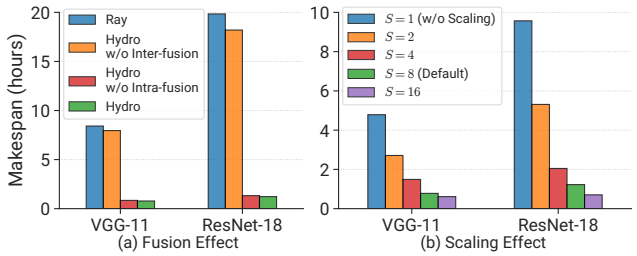

Figure 12: Ablation study. (a) Effect of inter- or intra-trial fusion. (b) Makespan of different scaling ratios.

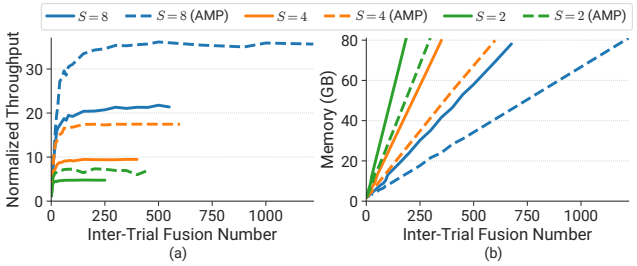

Figure 13: Sensitivity analysis of S and AMP on ResNet-18. (a) Accumulated throughout. (b) GPU memory footprint.

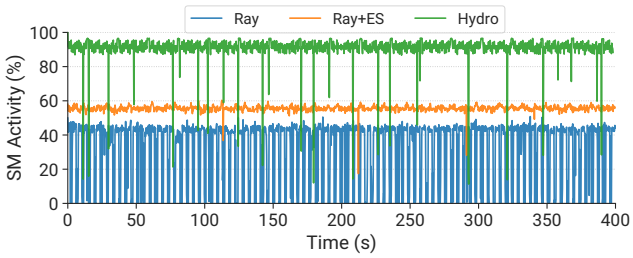

Figure 14: GPU utilization of HPO systems on ResNet-18.

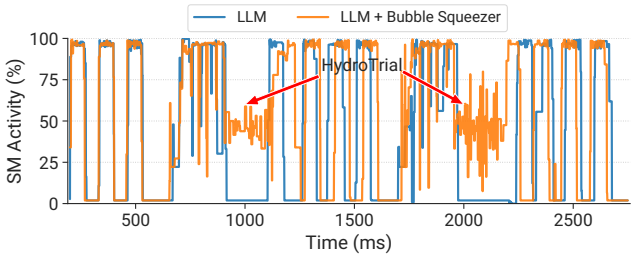

Figure 15: Visualizing Bubble Squeezer effect via DCGM. Two iterations of the first pipe stage are presented. The execution periods of the HydroTrial are highlighted by red arrows.

outperforms other baselines in final model accuracy within a limited time since it can well hide the target model training time inside the surrogate model tuning with Eager Transfer.

End-to-end result visualization. Figure 11 summarizes the makespan and accuracy of VGG-11 and ResNet-18 across different tuning algorithms and baselines. We note that Ray and Ray-ES share the same accuracy point since elastic and GPU sharing have no effect on the final model quality. The surrogate-based tuning (Hydro) can significantly reduce the search makespan without sacrificing the model accuracy. Due to the page limit, we only select these two models for presentation because of their relatively obvious efficacy of Ray-ES. Ray-ES has less improvement over Ray for larger models like WideResNet-50, since it cannot benefit from GPU sharing and the elastic improvement is limited (only for later stage).

6.4 More Evaluation on Hydro Tuner

Ablation study of fusion. Figure 12 (a) reveals an interesting observation that Hydro can only achieve very limited improvement over Ray if inter-trial fusion is disabled, even though we have scaled the model by 8×. This is because GPUs are underutilized for such small models and there is no evident training speedup although we scale the model. Hence, it is important to combine Model Shrinker and Trial Binder to achieve the desired performance. Additionally, we also evaluate the effect of intra-trial fusion. However, we find its improvement is limited on small models.

Sensitivity analysis of scaling. Figure 13 clearly presents the effect of the scaling ratio S on GPU memory and accumulated fused trial throughput, where the normalization base is the throughput of the target model. We find that the peak throughput increases linearly alone with S. GPU memory also shows a similar pattern. In Figure 12 (b), we further evaluate the effect of the scaling ratio S on the overall tuning time. Hydro can continuously obtain benefits from higher scaling ratios. Besides, the final model accuracy maintains stable.

Sensitivity analysis of AMP. Figure 13 analyzes the effect of mixed precision training (i.e., AMP [15]), where solid and dashed lines represent the settings without and with AMP, respectively. We can find that the peak throughput can be further improved via enabling AMP. Besides, its effect on memory is also obvious, improving nearly 2× maximum fusion count.

Impact on GPU utilization. Figure 14 plots the GPU utilization traces on one GPU for 300 seconds using different HPO systems. We employ NVIDIA DCGM [11] to record `SM Activity` as GPU utilization. It is obvious that Hydro can achieve much higher GPU utilization than other baselines owing to the superior capability of inter-trail fusion [110].

Overhead analysis. We perform the overhead analysis on the ResNet-18 multi-fidelity tuning workload. Its overhead mainly derives from two aspects: (1) *profiling* accounts for 0.8%; (2) *defusion* (including trial restart) accounts for 3.3%. The associated overhead is minor when weighed against the substantial enhancements in the tuning efficiency of Hydro.

6.5 Hydro Coordinator Evaluation

Bubble Squeezer. To evaluate the impact of Bubble Squeezer, we interleave `HydroTrials` with a large GPT model over 32 A100 GPUs containing 4 pipeline stages on 4 nodes, which is implemented based on DeepSpeed [96] along with MegatronLM [56, 88, 102]. We measured the SM activity with and without Bubble Squeezer in Figure 15. Two traces are collected separately and we align them at the beginning of the figure. For the original GPT training, since the only active

kernel in the bubble is NCCL kernel for communication, the SM activity is extremely poor (about 2%) during the bubble. Hydro utilizes the unused SMs and achieves a relatively high SM utilization at about 50%, with no evident slowdown to the GPT model training. Here the `HydroTrial` is ResNet-18 model with fusion count $F = 16$, obtaining around 15% of exclusive throughput. We also measure the throughput influence of direct colocation and find it causes unacceptable interference (about 12% slowdown for the large model). Additionally, we further simulate the end-to-end performance of Bubble Squeezer. Here we set that the Hydro tuning job can only apply 1 exclusive GPU since most resources are occupied by the large model. We find the makespan of the tuning job can be greatly reduced by 2.7× with the free lunch.

Heterogeneity-Aware Allocator. We create a tiny cluster partition with 2 A100 and 2 V100 nodes (32 GPUs in total) to evaluate the impact of Heterogeneity-Aware Allocator. Besides, we uniformly sample 20 middle-size HPO jobs from Table 3 and randomly generate their job arrival time within one hour. Compared to resource-agnostic allocation, we find Heterogeneity-Aware Allocator achieves approximately a 1.3x reduction in the average job completion time.

7 Discussion

Limitations. Despite the extraordinary performance, Hydro's surrogate-based tuning paradigm does have three limitations: (1) Hydro parametrization does not support regularization hyperparameters, such as weight decay and dropout, as elucidated in §4.1. (2) Hydro does not allow for any customized initialization techniques because Hydro implements its own automatic layer-wise re-initialization mechanism, which plays a crucial role in parameterization. (3) Hydro does not support fine-tuning since its theory is built atop i.i.d. samples (requiring the same dataset). Nevertheless, Hydro can deliver qualified models for most cases.

Future work. In the future, we plan to improve our work in following directions. (1) Supporting more DL frameworks like TensorFlow [19] and JAX [35]. (2) Considering more resource dimensions like CPU and network bandwidth besides GPU [83, 128], such as implementing the dataloader fusion of trials to further alleviate I/O contention. (3) Expanding the application scenarios such as cloud environments. It presents an opportunity for dynamic selection of heterogeneous spot instances, which can yield substantial cost savings [82, 106]. (4) Enabling partial model fusion across trails with minor architectural differences (e.g., add/remove/modify a few layers/blocks). Furthermore, Hydro can integrate model matching technique from ModelKeeper [60] to identify the models with similar architectures across jobs from different users and achieve cross-job level fusion, which can significantly improve cluster efficiency.

8 Related Work

AutoML systems. Automated Machine Learning (AutoML) refers to the process of automating the tasks associated with optimizing ML model performance. In general, AutoML comprises two essential components: HPO and Neural Architecture Search (NAS). NAS systems (e.g., Retiarii [127], ModularNAS [74]) aim to discover the optimal model architecture for a specific task. On the other hand, HPO focuses on optimizing the hyperparameters of a fixed architecture, usually separate from NAS. Our work primarily concentrates on HPO.

Prior HPO systems like HyperSched [71], Rubberband [82] and Seer [32] support elastic training to allocate more GPU resources to promising trials, which is also supported in Hydro. Elastic training can make use of idle GPUs but fails to improve single GPU utilization. On the other hand, Fluid [125] further leverages NVIDIA MPS [13] technique to allocate multiple trials on a single GPU. HFTA [110] achieves inter-trail fusion on a shared accelerator. They can improve hardware utilization but only work well on tiny models (e.g., AlexNet [58], PointNet [93]). Based on the unique surrogate-base tuning nature, Hydro significantly extends the fusion application scope via model scaling and achieves automatic model fusion with minimum manual effort.

Pipeline parallelism and interleaving execution. Recent studies exploit bubbles induced by pipeline parallelism from multiple angles. Bamboo [106] fills redundant computations into bubbles to provide resilience and fast recovery for preemptible cloud instances. EnvPipe [26] selectively lowers the SM frequency of bubble periods to save energy. Unlike them, Hydro leverages bubbles to train HPO trials via interleaving execution, which is inspired by some prior works. For instance, Wavelet [109] and Zico [73] reduce the GPU peak memory based on interleaving. Muri [128] supports multi-resource interleaving to reduce contention.

9 Conclusion

This paper presents Hydro, a surrogate-based hyperparameter tuning service that provide job and cluster level optimization via automated model scaling, fusion and interleaving. Our experiments show that Hydro can dramatically reduce the tuning makespan and improve the cluster resource utilization.

Acknowledgments

We sincerely thank our shepherd, Mathias Lécuyer, and the anonymous OSDI reviewers for their valuable comments on this paper. We also want to thank Greg Yang from Microsoft for the theory part support, Richard Liaw and Antoni Baum from Anyscale for the system development assistance, Shang Wang and Xin Li from UofT for their insightful discussion on inter-trial fusion, Shenggan Cheng and Shenggui Li from NUS for their constructive feedback on bubble squeezer. Additionally, we thank Li Ma and Shixin Yu for their technical support, as well as generous hardware resources from Shanghai AI Laboratory. This study is supported under the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s). Zhisheng Ye, Meng Zhang and Qiaoling Chen contribute equally to this work.

References

[1] Alibaba machine learning platform for ai. https://www.alibabacloud.com/product/machine-learning, 2023.

[2] Amd epyc 7742 cpu. https://www.amd.com/en/products/cpu/amd-epyc-7742, 2023.

[3] Chatgpt. https://openai.com/blog/chatgpt, 2023.

[4] Duolingo. https://www.duolingo.com/, 2023.

[5] Eleutherai gpt-neo 1.3b. https://huggingface.co/EleutherAI/gpt-neo-1.3B, 2023.

[6] Github copilot. https://github.com/features/copilot/, 2023.

[7] Infiniband networking. https://www.nvidia.com/en-us/networking/products/infiniband/, 2023.

[8] Microsoft azure automated machine learning. https://azure.microsoft.com/en-us/products/machine-learning/automatedml, 2023.

[9] Microsoft neural network intelligence. https://github.com/microsoft/nni, 2023.

[10] Nvfuser. https://github.com/pytorch/pytorch/projects/30, 2023.

[11] Nvidia data center gpu manager. https://developer.nvidia.com/dcgm, 2023.

[12] Nvidia multi-instance gpu. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/, 2023.

[13] Nvidia multi-process service. https://docs.nvidia.com/deploy/mps/index.html, 2023.

[14] Nvlink and nvswitch. https://www.nvidia.com/en-us/data-center/nvlink/, 2023.

[15] Pytorch automatic mixed precision training. https://pytorch.org/docs/stable/amp, 2023.

[16] Pytorch examples. https://github.com/pytorch/examples, 2023.

[17] Pytorch fullyshardeddataparallel. https://pytorch.org/docs/stable/fsdp, 2023.

[18] Torchvision new training recipe. https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/, 2023.

[19] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, 2016.

[20] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, 2016.

[21] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, ICLR '15, 2015.

[22] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

[23] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics — Workshop on Challenges & Perspectives in Creating Large Language Models*, ACL-BigScience '22, 2022.

[24] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, NeurIPS '20, 2020.

[25] Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in Neural Information Processing Systems*, NeurIPS '18, 2018.

[26] Sangjin Choi, Inhoe Koo, Jeongseob Ahn, Myeongjae Jeon, and Youngjin Kwon. Envpipe: Performance-preserving dnn training framework for saving en-

ergy. In *2023 USENIX Annual Technical Conference*, USENIX ATC '23, 2023.

[27] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, 2022.

[28] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, Wilton Wu, Aida Zolic, Huibin Shen, Amr Ahmed, Fela Winkelmolen, Miroslav Miladinovic, Cedric Archembeau, Alex Tang, Bhaskar Dutt, Patricia Grao, and Kumar Venkateswar. Amazon sagemaker autopilot: a white box automl solution at scale. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, DEEM '20, 2020.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '19, 2019.

[31] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI '15, 2015.

[32] Lisa Dunlap, Kirthevasan Kandasamy, Ujval Misra, Richard Liaw, Michael Jordan, Ion Stoica, and Joseph E. Gonzalez. Elastic hyperparameter tuning on the cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, 2021.

[33] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, ICML '18, 2018.

[34] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. Dapple: a pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21, 2021.

[35] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of Machine Learning and Systems*, MLSys '18, 2018.

[36] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision. *CoRR*, 2022.

[37] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ICML '10, 2010.

[38] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. https://skylion007.github.io/OpenWebTextCorpus/, 2023.

[39] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, 2017.

[40] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '19, 2019.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, 2016.

[42] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling. *CoRR*, 2020.

[43] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, NeurIPS '22, 2022.

[44] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV '19, 2019.

[45] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[46] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '23, 2023.

[47] Yimin Huang, Yujun Li, Hanrong Ye, Zhenguo Li, and Zhihua Zhang. Improving model training with multi-fidelity hyperparameter evaluation. In *Proceedings of Machine Learning and Systems*, MLSys '22, 2022.

[48] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, ICML '15, 2015.

[49] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, NeurIPS '18, 2018.

[50] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference*, USENIX ATC '19, 2019.

[51] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, 2019.

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

[53] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, ICML '13, 2013.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, ICLR '15, 2015.

[55] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, ICLR '17, 2017.

[56] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *CoRR*, 2022.

[57] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, NeurIPS '12, 2012.

[59] Sameer Kumar, Yu Wang, Cliff Young, James Bradbury, Naveen Kumar, Dehao Chen, and Andy Swing. Exploring the limits of concurrency in ml training on google tpus. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021.

[60] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. ModelKeeper: Accelerating DNN training via automated training warmup. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23, 2023.

[61] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *CoRR*, 2022.

[62] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[63] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, MLSys '20, 2020.

[64] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 2018.

[65] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[66] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Jixiang Li, Ji Liu, Ce Zhang, and Bin Cui. Hyper-tune: towards efficient hyper-parameter tuning at scale. *Proceedings of the VLDB Endowment*, 2022.

[67] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Zhi Yang, Ce Zhang, and Bin Cui. Transbo: Hyperparameter optimization via two-phase transfer learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, 2022.

[68] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. Mfes-hb: Efficient hyperband with multi-fidelity quality measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '21, 2021.

[69] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, Ce Zhang, and Bin Cui. Openbox: A generalized black-box optimization service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, 2021.

[70] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. Volcanoml: speeding up end-to-end automl via scalable search space decomposition. *Proceedings of the VLDB Endowment*, 2021.

[71] Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. Hypersched: Dynamic resource reallocation for model development on a deadline. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, 2019.

[72] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *CoRR*, 2018.

[73] Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient GPU memory sharing for concurrent DNN training. In *2021 USENIX Annual Technical Conference*, USENIX ATC '21, 2021.

[74] Yunfeng Lin, Guilin Li, Xing Zhang, Weinan Zhang, Bo Chen, Ruiming Tang, Zhenguo Li, Jiashi Feng, and Yong Yu. Modularnas: Towards modularized and reusable neural architecture search. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021.

[75] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *CoRR*, 2019.

[76] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE International Conference on Computer Vision*, ICCV '21, 2021.

[77] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, ICLR '19, 2019.

[78] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '20, 2020.

[79] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 2014.

[80] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 2018.

[81] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, ICLR '17, 2017.

[82] Ujval Misra, Richard Liaw, Lisa Dunlap, Romil Bhardwaj, Kirthevasan Kandasamy, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. Rubberband: Cloud-based hyperparameter tuning. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, 2021.

[83] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking beyond GPUs for DNN scheduling on Multi-Tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, 2022.

[84] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, 2018.

[85] 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 *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, 2019.

[86] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, 2021.

[87] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, 2020.

[88] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[89] Radford M. Neal. *Priors for Infinite Networks*. Springer New York, 1996.

[90] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[91] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, NeurIPS '19, 2019.

[92] Valerio Perrone, Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed, Tanya Bansal, Michele Donini, Fela Winkelmolen, Rodolphe Jenatton, Jean Baptiste Faddoul, Barbara Pogorzelska, Miroslav Miladinovic, Krishnaram Kenthapadi, Matthias Seeger, and Cédric Archambeau. Amazon sagemaker automatic model tuning: Scalable gradient-free optimization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, 2021.

[93] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '17, 2017.

[94] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '21, 2021.

[95] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[96] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, 2020.

[97] James Reed, Zachary DeVito, Horace He, Ansley Ussery, and Jason Ansel. torch.fx: Practical program capture and transformation for deep learning in python. In *Proceedings of Machine Learning and Systems*, MLSys '22, 2022.

[98] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022.

[99] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, 2017.

[100] David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *First Conference on Automated Machine Learning*, AutoML '22, 2022.

[101] 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 *International Conference on Learning Representations*, ICLR '17, 2017.

[102] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, 2020.

[103] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, ICLR '15, 2015.

[104] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NeurIPS '12, 2012.

[105] Xingyou Song, Sagi Perel, Chansoo Lee, Greg Kochanski, and Daniel Golovin. Open source vizier: Distributed infrastructure and api for reliable and flexible blackbox optimization. In *First Conference on Automated Machine Learning*, AutoML '22, 2022.

[106] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. Bamboo: Making preemptible instances resilient for affordable training of large dnns. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23, 2023.

[107] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, Xi Luo, Dheevatsa Mudigere, Jongsoo Park, Misha Smelyanskiy, and Alex Aiken. Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, 2022.

[108] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, NeurIPS '17, 2017.

[109] Guanhua Wang, Kehan Wang, Kenan Jiang, XIANGJUN LI, and Ion Stoica. Wavelet: Efficient dnn training with tick-tock scheduling. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021.

[110] Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. Horizontally fused training array: An effective hardware utilization squeezer for training novel deep learning models. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021.

[111] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '22, 2022.

[112] William F. Whitney. Parallelizing neural networks on one gpu with jax, 2023.

[113] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, EMNLP '20, 2020.

[114] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, 2018.

[115] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on*

Operating Systems Design and Implementation, OSDI '20, 2020.

[116] Sho Yaida. Meta-principled family of hyperparameter scaling strategies. *CoRR*, 2022.

[117] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, NeurIPS '21, 2021.

[118] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. In *Advances in Neural Information Processing Systems*, NeurIPS '19, 2019.

[119] Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *CoRR*, 2020.

[120] Greg Yang. Tensor programs iii: Neural matrix laws. *CoRR*, 2021.

[121] Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, 2021.

[122] Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, 2021.

[123] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. Astraea: A fair deep learning scheduler for multi-tenant gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[124] Peifeng Yu and Mosharaf Chowdhury. Fine-grained gpu sharing primitives for deep learning applications. In *Proceedings of Machine Learning and Systems*, MLSys '20, 2020.

[125] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. Fluid: Resource-aware hyperparameter tuning engine. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021.

[126] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference*, BMVC '16, 2016.

[127] Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. Retiarii: A deep learning Exploratory-Training framework. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, 2020.

[128] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '22, 2022.

[129] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, 2020.

[130] 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 *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, 2022.