Egeria: Efficient DNN Training with Knowledge-Guided Layer Freezing

Yiding Wang[1], Decang Sun[1], Kai Chen[1], Fan Lai[2], Mosharaf Chowdhury[2]
[1]iSING Lab, Hong Kong University of Science and Technology
[2]University of Michigan

Abstract

Training deep neural networks (DNNs) is time-consuming. While most existing solutions try to overlap/schedule computation and communication for efficient training, this paper goes one step further by *skipping* computing and communication through DNN layer freezing. Our key insight is that the training progress of internal DNN layers differs significantly, and front layers often become well-trained much earlier than deep layers. To explore this, we first introduce the notion of *training plasticity* to quantify the training progress of internal DNN layers. Then we design Egeria, a knowledge-guided DNN training system that employs semantic knowledge from a reference model to accurately evaluate individual layers' training plasticity and safely freeze the converged ones, saving their corresponding backward computation and communication. Our reference model is generated on the fly using quantization techniques and runs forward operations asynchronously on available CPUs to minimize the overhead. In addition, Egeria caches the intermediate outputs of the frozen layers with prefetching to further skip the forward computation. Our implementation and testbed experiments with popular vision and language models show that Egeria achieves 19%-43% training speedup w.r.t. the state-of-the-art without sacrificing accuracy.

CCS Concepts: • **Computing methodologies** → **Neural networks**.

Keywords: machine learning training, layer freezing

ACM Reference Format:
Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. 2023. Egeria: Efficient DNN Training with Knowledge-Guided Layer Freezing. In *Eighteenth European Conference on Computer Systems (EuroSys '23), May 8–12, 2023, Rome, Italy.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3552326.3587451

EuroSys '23, May 8–12, 2023, Rome, Italy

ACM ISBN 978-1-4503-9487-1/23/05…$15.00
https://doi.org/10.1145/3552326.3587451

1 Introduction

Recent advances in deep learning (DL) benefit significantly from training larger deep neural networks (DNNs) on larger datasets. Due to growing model sizes and large volumes of data, DNNs have become more computationally expensive to train, raising the cost and carbon emission of large-scale training [65]. Many recent DL research works focus on improving parallelism and pipelining via sophisticated computation-communication overlapping or scheduling to build more efficient systems and reduce training time [39, 61, 67]. Nevertheless, while approaching linear scalability can reduce the time to train a model, the total amount of computation requirement remains the same.

In this paper, we move one step further to explore: *can we reduce the total computation (and communication) in large DNN training?* We propose a *knowledge-guided* training system, Egeria, to accelerate DNN training via computation-communication freezing while still maintaining accuracy. Our key insight is that the training progress of internal DNN layers differs significantly, and front layers can become well-trained much earlier than deep layers. This is because DNN features transition from being task-agnostic to task-specific from the first to the last layer [95, 96]. Thus, the front layers of a DNN often converge quickly, while the deep layers take a much longer time to train, as generally observed in both vision and language models [75, 79], experimentally validated in §2.3. Egeria can safely freeze these converged DNN layers earlier, saving their corresponding computation and communication expenses without hurting model accuracy.

While freezing layers can reduce training cost, prematurely freezing under-trained layers will hurt the final accuracy. We observe that in transfer learning, freezing layers is mainly used for solving the overfitting problem [20]. While techniques such as static freezing [46] and cosine annealing [11] can reduce backward computation cost, accuracy loss is a common side effect. Thus, the main challenge of extending layer freezing to general DNN training is how to maintain accuracy by only freezing the converged layers.

To address this challenge, Egeria introduces the notion of *training plasticity*[1] to quantify a layer's training progress and safely detect the converged DNN layers to avoid premature freezing. To this end, Egeria uses a *reference model*, which is the proxy for semantic knowledge, to evaluate DNN layers' plasticity. The reference model, in essence, is a trained compressed DNN with the same architecture as the model being trained to understand layer-wise performance (details in §4.1). We compare the intermediate activations (internal outputs) between the training model and the reference model elicited by the same data batch to measure the plasticity. When the plasticity becomes stationary, it implies that the layer is converged and can be frozen safely (§4.2). In addition, Egeria can unfreeze the frozen layers to continue training with learning rate decay. Our approach is informed by recent advances in knowledge distillation research that suggest the same input data (images and word vectors) will elicit similar *pair-wise* activation patterns in *trained models* [4, 64, 83]. Compared to the straightforward metric of gradient's granularity or norm against a hard label, intermediate activation as soft distribution is proved to be more semantically meaningful, and thus more accurate by ML research (§4.2.1).

Egeria adaptively generates the reference model by instantly compressing a snapshot of the training model via quantization [24] on CPUs after the *bootstrapping stage* [2, 3] (early iterations during which the training model converges quickly). Large DNNs are robust to quantization, according to our evaluation and ML literature [48]. Egeria also profiles in the background to make sure the CPU-efficient reference model can provide accurate plasticity evaluation. The reference model exploits available CPU cores during GPU-heavy training, running forward operations parallel to the GPU training using the same input data in a non-blocking and asynchronous fashion. Hence, the system overhead is minimal and can be well hidden. In the remaining training process, Egeria updates the reference model using the latest snapshots to stabilize the plasticity curve. Essentially, we trade off small CPU resources for maintaining accuracy when freezing layers.

Freezing the front layers can save the backward computation and parameter synchronization. Nevertheless, we find that the forward pass still takes up to 35% of the time of an iteration. We observe that, in DNN training, the frozen front layers will produce the same forward output given the same input. Prior work on inference also shows that caching forward results can improve performance [8]. To take advantage of this, Egeria saves the intermediate activations of the frozen layers to the disk, *prefetches* the saved activation tensors to the GPU memory, and continues training the remaining layers from the cached activations in the following epochs. As the data loader knows the future data sequence, it is possible to prefetch relevant activations without stalling. Caching and prefetching are also compatible with random data augmentation. Therefore, we further save the frozen layers' forward computation without altering the training data sequence (§4.3).

We implement Egeria as a framework-independent Python library (§5). Existing code can work with Egeria with minimal changes. We evaluate Egeria using seven popular vision and language models on five datasets (§6). It achieves 19%-43% training speedup than the state-of-the-art frameworks and can reach the target accuracy.

To summarize, the key contributions of Egeria include: (1) leveraging semantic knowledge to save the backward computation and communication via DNN layer freezing while maintaining accuracy; (2) building an efficient system to implement the idea of knowledge-guided training; and (3) caching the intermediate results with prefetching to further save the forward pass of the frozen layers with negligible overheads.

2 Background and Motivation

2.1 DNN Training

Modern DNNs consist of dozens or hundreds of layers that conduct mathematical operations. Each layer takes an input tensor of features and outputs corresponding activations. We train a DNN by iterating over a large dataset many times and minimizing a loss function. The dataset is partitioned into *mini-batches*, and a pass through the full dataset is called an *epoch*. A DNN training iteration includes three steps: (1) forward pass, (2) backward pass, and (3) parameter synchronization. The forward and backward passes require GPU computation. In each iteration, the *forward pass* (FP) takes a mini-batch and goes through the model layer-by-layer to calculate the loss regarding the target labels and the loss function. In the *backward pass* (BP), we calculate the parameter gradients from the last layer to the first layer based on the chain rule of derivatives regarding the loss [56]. At the end of each iteration, we update the model parameters with an optimization algorithm, such as stochastic gradient descent (SGD) [10]. In data parallel distributed training, independently computed gradients from all workers are aggregated over the network to update the shared model.

2.2 Existing Optimizations for DNN Training

Training large DNNs is computation- (and communication-) intensive due to the ever-growing data volumes and model size [36, 61, 67]. One important direction for training acceleration from the system perspective closely related to Egeria is *computation-communication overlapping and scheduling*. Baseline training frameworks (e.g., TensorFlow, PyTorch, and Poseidon [99]) optimize distributed performance by issuing the gradient transmission once a layer finishes its

[1] *Plasticity* quantifies a layer's training progress toward convergence, which is borrowed from *neuroplasticity* in neural science and child development [15]. Basically, a DNN layer's training plasticity will gradually decrease and become stable as it converges.

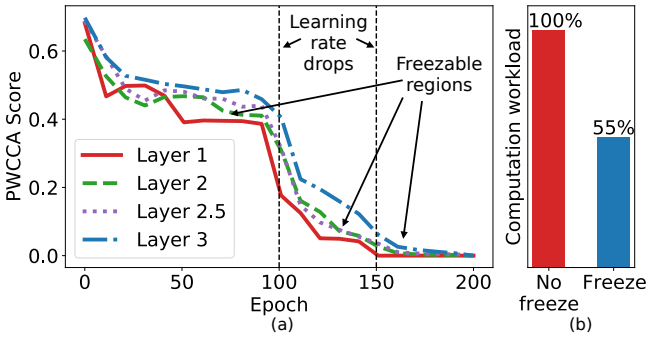


Figure 1. Post hoc layer convergence analysis with PWCCA. A lower score means the layer converges better. Layer module 2.5 is the first half of Layer module 3. The learning rate changes at 100th and 150th epochs and reboosts training. We can freeze layers when they are stationary and unfreeze them when the learning rate decreases extremely.

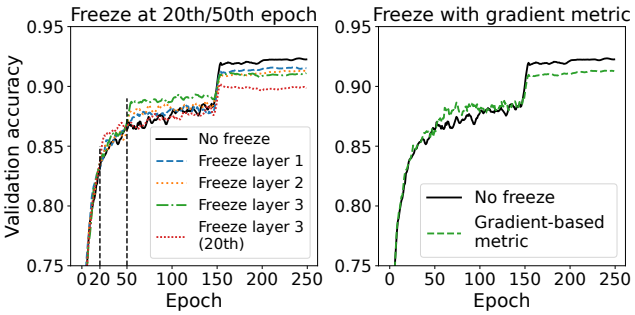


Figure 2. Prematurely freezing layers with transfer learning techniques can hurt the final accuracy in general training.

backward computation so that the deeper layers can overlap their communication with the front layer's BP. Priority-based communication scheduling systems (e.g., ByteScheduler [67], P3 [36] and TicTac [25]) leverage the layer-wise structure information to prioritize the front layers in communication which try to overlap the communication with FP. Pipelining solutions [33, 61, 94] add inter-batch pipelining to intra-batch parallelism to further improve parallel training throughput. While all these solutions can optimize computation-communication efficiency, the total computational cost remains the same.

Other methods. There are other optimizations, such as gradient sparsification [50] and quantization [88], to reduce communication volumes. These methods are largely orthogonal to Egeria, and we will overview them in §7.

2.3 Opportunities of DNN Layer Freezing

In this paper, we explore the idea of *reducing computation and communication costs* through DNN layer freezing. In the following, we first show the idea and its potential, and then lay out the challenges, motivating the design of Egeria.

Motivation for layer freezing. Recent efforts have shown that the front layers primarily extract general features of the raw data (e.g., the shape of objects in an image) and often become well-trained much earlier, while deeper layers are more task-specific and capture complicated features output from front layers [73, 95]. Our work is also inspired by transfer learning [31, 41, 52, 71]. When fine-tuning a pre-trained model on a new task, we find that ML practitioners can freeze (i.e., fix layer's weights) the front layers or only fine-tune them for a few iterations and focus on training the deep layers on the new dataset.

To demonstrate the potential in general training (i.e., not fine-tuning), we use PWCCA [60], a *post hoc* layer convergence analysis tool, to track the training progress of different layers of ResNet-56 [27] as an example. ResNet-56 is a popular model for image classification on the CIFAR-10 dataset, and it consists of three layer blocks (refereed to as modules or stages), where each module has 18 basic blocks of successive layers. PWCCA compares the intermediate activation (i.e., the output feature map produced by a DNN layer) with a *fully-trained* model. A low PWCCA score (0–1 range) suggests the layer is converged to the final state. We can clearly find some freezable regions in Figure 1: e.g., for layer 1, during the 50th–90th, 120th–140th, and after the 160th epoch, its score becomes stable, meaning it is temporarily converged; other layers also show some relatively stable regions. The scores drop at the 100th and 150th epochs because the learning rate decreases as scheduled; after that, they soon converge again. These patterns reveal a natural strategy: *Freeze the layers when their performance is stable and unfreeze them when the learning rate decreases.*

We find that if we freeze the layers in their freezable regions, by summing up the #parameters when a layer's PWCCA is stable in Figure 1, we can reduce the computation costs by 45% in theory! For natural language processing (NLP) models, this potential can be even larger because the front layers usually contain more parameters than CNNs.

Challenges in layer freezing. Although this opportunity has been pointed out by ML community, it is not feasible to run the hypothetical post hoc analysis (e.g., PWCCA) in practice. Quantifying the training progress of a layer is difficult due to the lack of prior knowledge (e.g., a trained model). Furthermore, we find that prematurely freezing layers using transfer learning techniques can substantially hurt a model's accuracy. To demonstrate this, we investigate the impact of static freezing [46] and a gradient-based metric [51], both from fine-tuning pre-trained models, on the final accuracy when training ResNet-56. In Figure 2, we first fix the parameters of each layer module at the 20th/50th epoch and show

their validation accuracies alongside the baseline. The degraded accuracies indicate that freezing layers prematurely can hurt accuracy by nearly 2% which is huge for such models. We test freezing with a gradient-based metric [51] to reach a similar 20% speedup, but constantly find ~1% non-negligible accuracy loss. According to the benchmark [1], proposing a DNN architecture or training method usually improves the accuracy by less than one percent, so the such loss could offset the benefit of a new ML technique.[2] This motivates an accurate way to freeze layers in general training beyond fine-tuning. We further compare Egeria with existing freezing proposals in §7.

3 Egeria Overview

We propose Egeria, an efficient DNN training system that detects and freezes the converged layers in a practical manner. Lacking prior knowledge of the hypothetical fully-trained model, Egeria introduces a self-generated *reference model* during training to provide semantic knowledge for evaluating a layer's convergence with minimal system overhead (§4.1). The reference model is essentially an accompanying lightweight DNN with the same architecture as the model being trained to match their internal layers and understand layer-wise performance.

To quantify the training progress, Egeria defines a system metric of *plasticity*. A layer's plasticity is formulated as the difference between the intermediate activation tensors of the training model and its reference model given the same mini-batch input.[3] The plasticity changes as the model evolves over training, and Egeria considers the layers with stable plasticity values to be converged, whereby Egeria freezes these layers without hurting the accuracy (§4.2).

To validate the effectiveness of plasticity in capturing the training progress, we use ResNet-56 and generate a reference model with the same architecture but pre-trained for only 50 epochs. We measure the plasticity of ResNet-56's first three layer modules during training. In Figure 4a, the top black curve shows the validation accuracy, and the other three indicate the derived plasticity for each layer module. We find that, in the first ~30 epochs, the plasticity of the first two modules converges quickly to a low level while the plasticity of layer module 3 is much higher and unstable. These layers show different trends of plasticity, more clearly after normalization in Figure 4b. We find similar freezable regions when using the post hoc analysis (Figure 1), e.g., layer 1 converges near the 50th epoch, while layer 3 only converges at the last epochs.

[2]Since AutoFreeze's implementation is deeply coupled with Transformers, we optimize the performance on ResNet training to the best of our ability.
[3]We measure the difference using the Similarity-Preserving loss (SP loss) [83], a novel loss function recently developed to compare two activation tensors for CV tasks and also echoed in NLP [64]. Unlike PWCCA [60] for post hoc convergence analysis, SP loss focuses on capturing the semantic difference for DNN training, making it a perfect fit for plasticity evaluation.

Plasticity requires no prior knowledge as PWCCA, only a training-in-progress model for reference, and accurately captures the trend of layer convergence. We conduct correctness analysis and find that plasticity shows similar patterns of layers' convergence as PWCCA but has ~10× lower overhead and optimized as a loss rather than for visualization (details deferred to the full version). The algorithm behind plasticity is tested performant for various tasks compared to traditional gradients and other activation-based metrics [64, 83].

Training life cycle with Egeria. Figure 3 describes the high-level workflow of Egeria in two stages.

(1) *Bootstrapping stage*: When a job is submitted, Egeria starts to monitor the job. The bootstrapping stage is a *critical period* of training, during which the DNN is sensitive and no parameter is eligible for freezing, according to recent research [2]. Egeria monitors the changing rate of the training loss (in line with the later plasticity monitoring) and moves to the next stage as the DNN moves out of the critical period.

(2) *Knowledge-guided training stage*: Egeria generates the reference model on the CPU using the latest snapshot of the training model. Afterwards, Egeria collects the intermediate activation of the frontmost non-frozen layer of the full model and its reference model for plasticity evaluation (§4.1), freezes the layer once it reaches the convergence criteria (§4.2), and moves to the next active layer. Egeria excludes the frozen layers during BP (and parameter synchronization in case of distributed training) to accelerate training. Meanwhile, Egeria caches the frozen layer's activations to the disk, so that we can also skip the FP computation by prefetching the intermediate results for the same input (§4.3).

4 Design

Next we dive into the details on how to capture the semantic knowledge during training (§4.1), with which Egeria optimizes the computation and communication in the backward pass (§4.2), as well as the forward pass (§4.3) on the fly.

4.1 Egeria Architecture

Directly running another full DNN to measure the internal layers' plasticity can greatly slow down the training. Instead, Egeria decouples the control logic and the training logic with a controller-worker abstraction (§4.1.1), and asynchronously performs plasticity evaluation (§4.1.2). Besides, Egeria generates and continuously updates the reference model by fast quantization (§4.1.3).

4.1.1 Controller-Worker Framework. Figure 5 illustrates the controller-worker framework of Egeria, which primarily consists of a logically centralized controller and workers:

- *Controller*: It manages the life cycle of the reference model, including its generation and execution, gathering data for plasticity evaluation, and making layer freezing/unfreezing decisions for workers. It makes

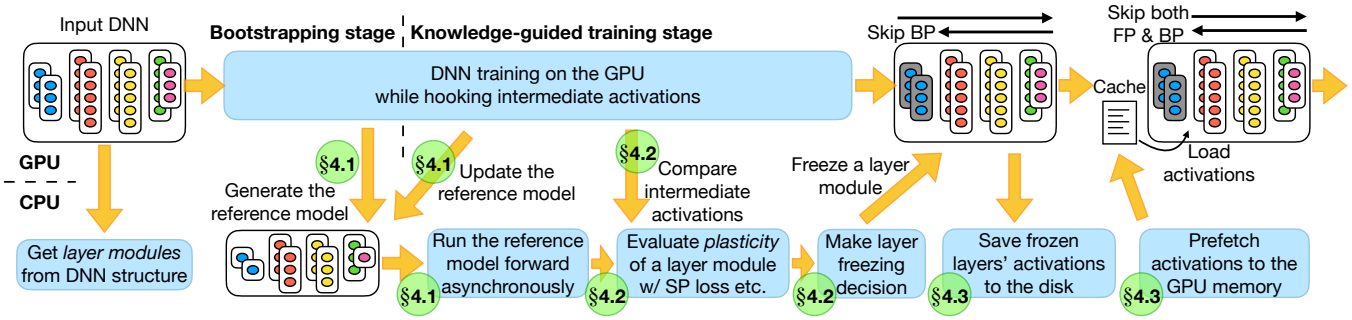
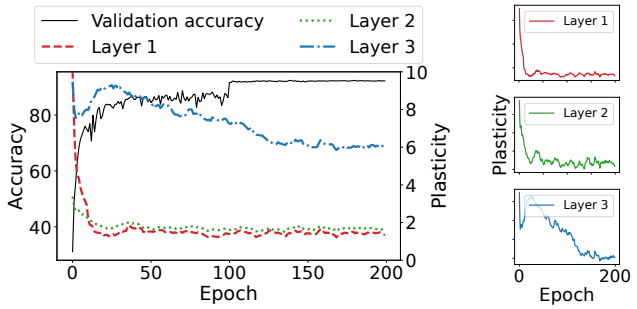


Figure 3. EGERIA overview. EGERIA covers two training stages (bootstrapping and knowledge-guided stages) and has three major design components (generating and executing the reference model, layer freezing with plasticity evaluation, and skipping FP with prefetching).



(a) Overall plasticity and accuracy **(b)** Trends

Figure 4. The plasticity of the front layers drops quickly, and they will produce semantically similar activations for most training iterations.

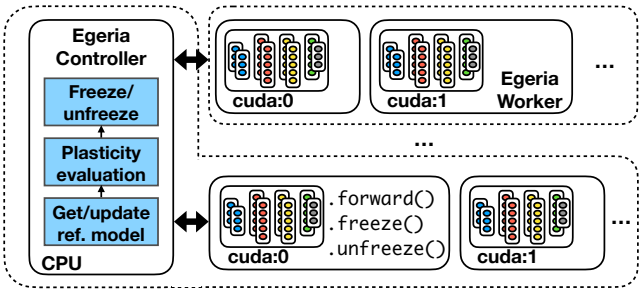


Figure 5. EGERIA uses a controller-worker framework. The controller co-locates on a training node. Solid lines denote device and logical boundaries; dashed lines denote machines.

plasticity evaluations at one place to reduce the overall computation overhead in case of distributed training, as multiple controllers only change the sample size.

- *Worker*: Each training worker has an EGERIA worker process. In addition to the original training operations, it performs EGERIA tasks, including transmitting data and handling controller decisions. The updated `forward()` method uses hooks to obtain the intermediate activation tensors. The `freeze()` and `unfreeze()` methods will be called by the controller and apply on target layers.

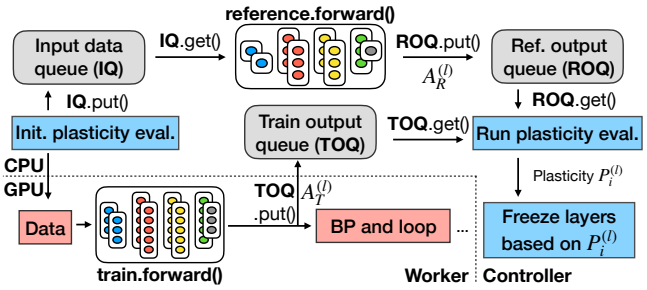


Figure 6. Red and blue blocks are worker and controller operations, respectively. They interact asynchronously through multiprocessing queues (gray blocks).

4.1.2 Non-Blocking Plasticity Evaluation.

To avoid slowing down the training, the controller runs an efficient reference model on CPUs in a *non-blocking and asynchronous* fashion, as shown in Figure 6. ML training servers typically have a high CPU-to-GPU ratio (e.g., 6:1 [5]) since GPUs are the scarcest resource. In addition, ML system research suggests that GPUs can be fully utilized by exploiting abundant resources like CPUs [37, 39], storage [58], and networking [67]. Similarly, EGERIA trades limited CPU cycles which are optimized for int8 inference-only operation [34] for reduced GPU workload. Since the plasticity evaluation runs periodically (e.g., every ~10 to ~100 iterations) and asynchronously, this non-time-critical process will not interfere with other CPU operations and can be well hidden behind GPU computation, as tested in our evaluation.

We implement the asynchronous plasticity evaluation using three *single-producer/single-consumer queues*: (1) When EGERIA initiates a plasticity evaluation, the co-located EGERIA worker puts the data batch in the input queue (IQ). (2a) The controller process polls IQ, runs a forward pass on the

reference model, and puts the hooked intermediate activation $A_R^{(l)}$ to the reference output queue (ROQ). The controller only executes the forward pass at low CPU load (e.g., <50%) to avoid interference to other CPU-based auxiliary operations, thanks to the non-blocking framework. (2b) The co-located Egeria worker puts the hooked intermediate activation $A_T^{(l)}$ to the training output queue (TOQ) and continues the training loop without blocking. (3) The controller polls ROQ and TOQ and calculates the plasticity of the frontmost active layer modules to make freezing decisions (§4.2).

4.1.3 Generating and Updating the Reference Model.

An ideal reference model should execute fast for diverse models and provide semantically meaningful activations. To this end, Egeria generates the reference model for the full model being trained on the fly.

There are several techniques to generate a light-weight version for a large model. For example, neural architecture search (NAS) [101] and knowledge distillation (KD) [29] can compress the model but have prohibitively large computation overhead. Besides, they may produce different architectures that do not match the internal layers for plasticity evaluation. Therefore, Egeria adopts *post-training quantization* [24] to instantly generate a reference model with the same structure.

Quantization is a popular model compression technique to accelerate inference on CPUs [32, 90, 92]. It reduces the precision of model's parameters (e.g., from 32-bit floating-point to 8-bit integers). By default, Egeria quantizes the reference model using 8-bit integers. This can reduce the reference memory footprint by 3× to 4× and accelerate the forward pass by 2× on CPUs; meanwhile, a lower precision (e.g., int4 or int2) cannot further improve the performance due to the CPU instruction set [49]. In our test, int8 quantization reaches a sweet spot to achieve both fast and accurate semantic reference (§6.4). Egeria can fall back to full-precision reference if the training DNN is extremely sensitive or running the reference on GPU in case the CPU resources are scarce, adapting to various environments. In addition, we design the freezing workflow (Algorithm 1) with robustness kept in mind.

Egeria will periodically update the reference model every W iterations (§4.2.2 discusses the frequency value) using the latest training snapshot to keep it up-to-date for plasticity evaluation. We empirically find that a stale reference model can amplify the inherent fluctuations in stochastic gradient descent (SGD) training [10], making the plasticity curve drastically changes, which is hard for Egeria to understand its trend. Updating the reference model can smooth the plasticity and better keep up with the baseline. We use a periodic update because we find that the frequency of reference update is quite insensitive: Frequently updating is unnecessary because learning is a slow process, while a changing rate-based method that slows down updating in the later training stage brings little gain since the overhead is low anyway.

4.2 Freezing Layers with Plasticity

Next, we elaborate how Egeria decides when to freeze the stable layers by comparing the intermediate activations of the model being trained with that of the reference model. During this stage, Egeria needs to address two challenges: (1) how to quantify the training plasticity of a layer module; and (2) how to accurately make the layer freezing decision.

4.2.1 Evaluating the Plasticity of a Layer Module.

The raw data we collect is the intermediate activation tensors of the training and reference models, which have been widely studied as a direct metric of layer performance in understanding and exploring new methods of DNN training [60, 95]. For example, knowledge distillation uses the difference of activations between the training model and a trained teacher model [76, 83] as a supervisory signal to improve accuracy because intermediate activation plays an important role in forming the decision boundaries for the partitioning of the feature space in each hidden layer [28, 63]. Rather than comparing the gradients of the two models calculated against a hard label, using intermediate activation as a "soft" label is proven more effective and accurate in guiding training in recent research [4, 83] since it is a more semantically meaningful indicator and provides *contextual* knowledge [14, 64]. We also test freezing with a gradient-based metric [51] from fine-tuning but find ~1% accuracy loss, which is undesirable for a general training system and echoes the KD research, as shown in Figure 2. To freeze layers accurately, we quantify the training plasticity by measuring the changes in the layer's intermediate activations.

We use the SP loss [83] that shows high efficacy in KD tasks to compare the intermediate tensors between the two models. The theory behind SP loss is that the same input data will elicit similar pair-wise activation patterns in trained models for both CV and NLP tasks [64, 83]. Different from gradient norm that is calculated against single-dimension hard labels (e.g., "cat" for image classification), SP loss calculated from two high-dimension activation tensors can better capture the semantic and contextual information, as discussed in recent KD research [4, 83] and tested in §6.2. Thus, Egeria can make freezing decisions more accurately, and our evaluation shows higher accuracy when achieving the same speedup. Compared to PWCCA [60] used in post hoc analysis, our empirical analysis finds that SP loss shows similar training progress of layers and freezing opportunities. We choose SP loss because (1) it is designed as a training loss to measure the performance difference *in the actual scale* for direct model updating, while PWCCA is a visualization and analysis tool that uses *weighted* projection to fit largely distinct values into the scale of 0–1, showing performance

gaps differently in different training stages; (2) we find that PWCCA is more computation-intensive for its projection operation.

We focus on the performance of a layer module that contains consecutive layers defined together. Layers in a module are closely related to performing a sequence of transformations for a certain goal [47] and have similar training progress. Meanwhile, individual layers with fewer parameters (e.g., linear layers) are less stable in SGD training. Though it's rare, even a few individual front layers might not converge in strict order (as observed in [98]), our module-based freezing can mitigate this and revisit them in the future unfreezing stage (§4.2.2). Egeria provides configuration options to customize the granularity of layer module through regular expression, e.g., evaluating every convolutional layer.

Given the input data of batch size b, we denote the activation tensors of the training and reference models at a layer l as $A_T^{(l)}$ and $A_R^{(l)}$. For the image data, the activation tensors $A_T^{(l)}, A_R^{(l)} \in \mathbb{R}^{b\times c\times h\times w}$, where c, h, and w are channel number, height, and width; similar for the word embeddings. Then SP loss will align $A_T^{(l)}$ and $A_R^{(l)}$ to $b\times b$-shaped matrices, which encode the pair-wise similarity in the activation tensors that are elicited by the input mini-batch. We then denote the *training plasticity* $P_i^{(l)}$ of layer l at an iteration i using the SP loss between the two matrices, representing the semantic difference compared to the reference model, as shown in Equation 1. The lower and more stable the plasticity, the DNN layers are closer to convergence.

$$P_i^{(l)} = SP\_loss(A_T^{(l)}, A_R^{(l)}) \tag{1}$$

4.2.2 How to Decide Layer Freezing. During the knowledge-guided training stage, Egeria will periodically run the plasticity evaluation every n iterations and decide whether to freeze the layer or not. The intuition of the freezing criterion is straightforward: If a layer's plasticity becomes stationary for some iterations W, Egeria considers its semantic performance stable and can safely freeze it.

When obtaining the plasticity $P_i^{(l)}$, we first smooth it with the moving average of its recent values (using W or the max span as the history buffer size), as shown in Equation 2.

$$\overline{P_i^{(l)}} = \begin{cases} \frac{P_{i-W}^{(l)}+\cdots+P_i^{(l)}}{W}, & i \geq W \\ \frac{P_0^{(l)}+\cdots+P_i^{(l)}}{i}, & i < W \end{cases} \tag{2}$$

To determine whether the curve has become stable, we fit $\overline{P_i^{(l)}}$ with linear least-squares regression to a straight line and analyze its slope (0 means no change at all). This method can filter out the drastic fluctuation in SGD training and provide a recent history context than simply evaluating the delta. If the plasticity slope has been less than the tolerance T for W evaluations, we consider the layer converged, freeze it, and move to the next layer, as detailed in Algorithm 1. This simple yet effective method has a similar intuition to the early stopping in DNN training [40, 78].

Egeria monitors the frontmost active layer module l to avoid a fragmented frozen model. According to the chain rule of automatic differentiation [56], only excluding the last link of backpropagation can reduce the workload. It is widely recognized that the front layers converge faster [75, 79, 95, 96], and Egeria can handle exceptions with the aforementioned module-based freezing and unfreezing mechanism.

Unfreezing. Learning rate (LR) scheduling can influence the convergence of all layers [2] (e.g., the PWCCA score and accuracy boost in Figure 1 and 2) and is an external factor to the model. LR annealing [19], the most commonly employed scheduling technique, recommends gradually lowering the LR during training with step decay or exponential decay. Given this, Egeria will restart training all the frozen layers if the LR has dropped over a factor of 10 since the frontmost layers' freeze and halve the counter and history buffer W for refreezing, which we find effective for different models. Another type of LR scheduling is periodically increasing and decreasing the LR, e.g., cosine annealing [53] and cyclical LR [80]. Due to its complexity, Egeria lets the user customize the unfreezing and refreezing criteria, e.g., training for a few iterations in each cycle after freezing a layer.

Hyperparameters guideline. We use three hyperparameters: n (plasticity evaluation and bootstrapping stage monitoring interval), T (the tolerance of plasticity slope), and W (number of low slope evaluations to freeze layers and history buffer). They are highly related and can be automatically set with some task knowledge. Egeria sets T for each layer module as the 20% of the maximal plasticity slope in its initial 3 readings; the rationale is that layers move differently and thus should have per-layer thresholds. We recommend setting n as a moderate frequency value that can cover the evaluation of all layers. For example, for training ResNet-56 with 7 layer modules, LR scheduling, and W=10 for 200 epochs (~78k iterations), we set n to 300 iterations ($\approx 78k/(10*2)/7/(1+0.5+0.25)$ considering bootstrapping, smoothing delay, and window halving). Our extensive empirical analysis shows that we achieve consistently good performance across different parameters following our general guideline, and we conduct sensitivity analysis of W, n, and T in §6.4 to show their impact on performance with largely different values. The changing rate of ending the bootstrapping stage is permissively set to 10%.

4.3 Skipping Forward Pass with Caching and Prefetching

By freezing the converged front layers, we can exclude them from the backward pass and parameter update to reduce training cost. However, the forward pass is still necessary because the deep layers require the frozen layers' activations as input [56]. Naturally, we can cache the frozen layers'

Algorithm 1: Layer freezing algorithm.

Input:
Intermediate activations of the training and reference models $A_T^{(l)}$, $A_R^{(l)}$, layer module l, training iteration i, counter and history buffer length W, tolerance T.
Output: The (updated) frontmost active layer l.

```
/* Initialize global variables. */
pList_l ← ∅ ; ▷ Plasticity evaluation history of l across iterations.
staleCounter ← 0 ;        ▷ Number of consecutive stale P_i^(l)‾.
Function checkPlasticity(A_T^(l), A_R^(l), l, i, T, W):
    assert l is not the last layer
    if staleCounter < W then
        P_i^(l) ← calculateSPLoss (A_T^(l), A_R^(l), l, i)
        /* Use moving average to mitigate outliers (Equation 2). */
        P_i^(l)‾ ← smoothPlasticity(P_i^(l), W)
        /* Update the time-series plasticity list. */
        pList_l ← pList_l ∪ P_i^(l)‾
        /* Calculate the slope of the linear-fitted plasticity. */
        s ← windowLinearFit(pList_l, W).slope
        /* If the fitting line is close to horizontal. */
        if s < T then
            staleCounter ← staleCounter + 1
        else
            staleCounter ← 0
        end if
    else
        freezeLayer (l)
        l ← l + 1
    end if
    /* Learning rate-based unfreezing mechanism. */
    if LR annealing and LR decreased by 90% then
        unfreezeAllLayers ()
        l ← l_0 ;                 ▷ Reset l.
    else
        if cyclical LR scheduling then
            customizedUnfreeze ()
        end if
    end if
    return l
```

intermediate activations to save the forward pass because they output the same activation given a certain input.

There are two challenges of caching computation results for a DNN training task. First, training a large model requires a large dataset (e.g., the training set of ImageNet is over 100 GB). The size of the intermediate activation tensor depends on the output shape of the last frozen layer. In our evaluation, the storage space of ResNet-50 intermediate activations ranges from 1.5× to 5.3× of the input data. It is not technically appropriate to cache a whole epoch's results

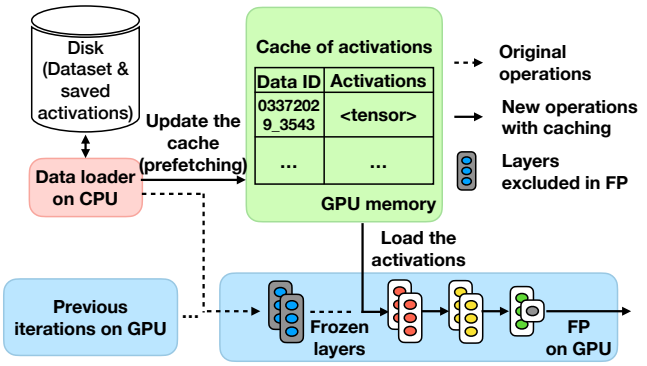


Figure 7. Egeria caches the intermediate activation and prefetches the tensors into GPU memory during the FP.

to the GPU/CPU memory. Second, given the memory limit, caching systems usually improve their hit rate by keeping the most frequent content using replacement policies like LRU (least-recently-used). However, in DNN training, the data loader randomly samples a mini-batch, meaning there is no popular data to prioritize in the cache.

To solve these challenges, we exploit a training workflow feature: Before an iteration, the *data loader* samples future mini-batches in advance, so unlike typical cache systems, we actually "know the future" (the incoming data indices)! Prefetching is an effective technique in ML applications [7, 69]. Egeria saves the forward computation results of the frozen layers to the disk and prefetches the relevant activations to the GPU memory so that the active deep layers can instantly read them as input. The cache only stores the recent five mini-batches for minimal memory usage. Users can set the storage limit for activations that are up to an epoch (see analysis in §6.5). At the early training stage, we disable prefetching if the forward pass of a few layers is faster. In this way, we can skip the forward computation of the frozen layers and efficiently overlap the slower disk access with prefetching.

Figure 7 illustrates a forward pass in Egeria. We maintain a hash table in the GPU memory. The key is the sample ID, and the value is the corresponding activation tensor. During training, the data loader will sample a data batch and prefetch their activations from the storage to the GPU memory in parallel to the GPU training. Egeria prefetches more than one mini-batch of the future activations, similar to the data loader [69], depending on the memory and CPU availability.

Egeria can cover different training techniques and cache their outputs. Egeria is compatible with stateless random operations, which are recommended for random data augmentation and dropout [81, 82]. Thus we can deterministically keep the randomly augmented images the same across epochs without ML performance penalty. Each worker machine maintains its own cache in distributed training to avoid

extra network overhead; besides, the random seed is device-dependent. For most layers, e.g., convolutional (for CNNs) and linear (for language models), the output activation only depends on the parameters given the same data, so caching can work naturally. For batch normalization layers, the activation also depends on the specific data batch. Egeria handles this case using the practice in transfer learning [41]: we set these layers to the inference mode, using the dataset statistics to normalize the input rather than the specific batch.

5 Implementation

Egeria is independent of DNN training frameworks. In this paper, we implement and evaluate Egeria using PyTorch and Huggingface Transformers [89]. All the technical dependencies of Egeria (e.g., quantization and asynchronous computation) can work in other ML frameworks like TensorFlow and MXNet. Egeria obtains the layer modules by parsing the model definition and adds forward hooks to obtain intermediate activations.

Reference model. When Egeria controller generates or updates the reference model, it directly moves a snapshot of the training model from GPU to CPU and runs int8 quantization using PyTorch's built-in library. We use dynamic quantization for NLP models and static quantization for convolutional networks, which add little overhead in the background. We add the same forward hooks to the reference model to match the training model.

Knowledge-guided training. Our high-level API allows us to take advantage of the framework engines to execute all the DNN computation operations. To freeze a layer, we essentially set the `requires_grad` flag of all its parameters to false to exclude the subgraph from gradient computation [68]. Distributed training requires rebuilding the communication buffer. For caching, we use the dictionary data structure with $O(1)$ lookups.

6 Evaluation

In this section, we evaluate the effectiveness of Egeria, namely accelerating DNN training while maintaining accuracy, for different tasks and models using single or multiple machines. The main takeaways are:

- Egeria can work for different CV and NLP models;
- Egeria accelerates general training by 19%-43% without hurting accuracy; and
- Egeria minimizes the system overhead while accurately freezing layers.

6.1 Methodology

Testbed setup. We evaluate Egeria using two testbed configurations: a cluster of 5 machines and a multi-GPU machine. In the 5-node cluster, each machine has 2 NVIDIA V100 GPUs (32 GB), 40 CPU cores, 128 GB memory, and 2 Mellanox CX-5 NICs. To match the CPU-to-GPU ratio of the cloud training instance [5], we use 12 CPU cores (equivalent to 24 vCPUs) with `taskset` [54]. The testbed has a leaf-spine topology with two core and two top-of-rack (ToR) switches; each ToR switch is connected to 5 servers using 40 Gbps and two core switches using 100 Gbps links. The single node has 8 NVIDIA RTX 2080 Ti and 64 CPU cores.

Tasks, models, and datasets. We evaluate two CV and two NLP tasks: image classification, semantic segmentation, machine translation, and question answering; the corresponding 7 models and 5 datasets are listed in Table 1. We follow the recommended learning rate and batch size settings [62, 70] and the learning rate schedulers are step decay LR schedule for CV training, inverse square root schedule for Transformer training, and linear schedule for fine-tuning BERT. We use the all-reduce parameter synchronization scheme for data parallel distributed training with multiple GPUs or machines and allocate one GPU per process.

Metrics and baselines. The training performance metric is the time taken to a converged validation accuracy (TTA), as listed in Table 1. We compare Egeria with the vanilla training framework, PyTorch, and a communication scheduling system, ByteScheduler [67], in multi-node distributed training using its default configuration. ByteScheduler achieves the theoretically optimal scheduling without skipping any parameter and full accuracy. We use scheduling/pipelining systems as the main baseline since maintaining accuracy is our major goal. We also compare Egeria to a recent gradient-based layer freezing system, AutoFreeze [51], and to using the metric of Skip-Conv [23] as an alternative to plasticity. We use the input-norm gate of Skip-Conv, which applies to intermediate activation rather than convolution-specific.

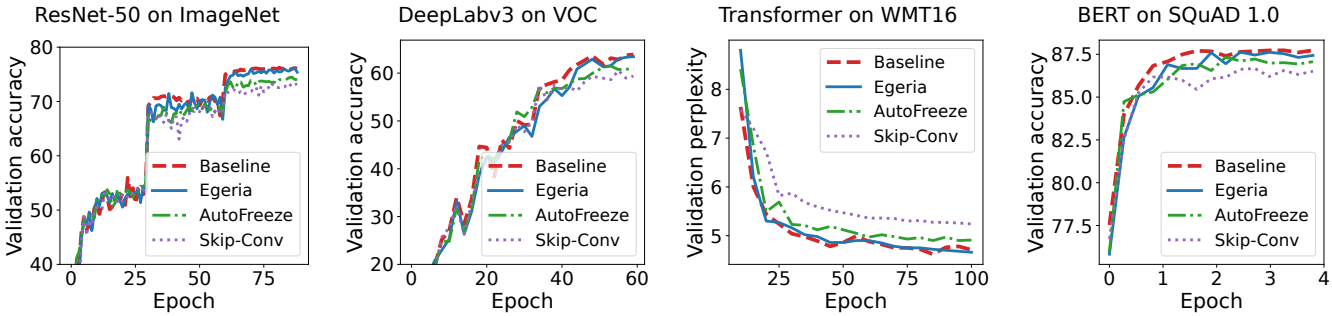
6.2 End-to-End Training Performance

We use Egeria to train different models to reach the target accuracies with largely reduced training time. Table 1 summarizes the evaluation results and the time-to-accuracy (TTA) speedups compared to the baseline training system. We tune AutoFreeze and Skip-Conv to achieve a similar training time as Egeria and compare their final accuracy to Egeria and full training in Figure 8.

Image classification. ResNet-50 for ImageNet is a popular CNN benchmarking model. It consists of 48 layer modules, grouped into four stages, and the deep stages of layers have more parameters than the front stages (similar to the ResNet-56 structure in Figure 11). Figure 8a shows the validation accuracy curves of Egeria and the baselines. Within 90 epochs of training, Egeria reaches the target accuracy while AutoFreeze and Skip-Conv each loses 1.5% and 2.6% when achieving the same time speedup of 28%. During these critical stages, the unfreezing mechanism of Egeria (§4.2.2)

Task	Model	Dataset	Accuracy target	# Servers × # GPUs/server	# Building layer modules	TTA speedup
Image classification	ResNet-50 [27]	ImageNet [16]	Top 1 75.9%	1×2 2×2 - 5×2	48 (residual blocks)	28% 27%-33%
	MobileNet V2 [77]		71.2%	1×2	17 (inverted residual blocks)	22%
	ResNet-56 [27]	CIFAR-10 [42]	92.1%	1×2	54 (residual blocks)	23%
Semantic segmentation	DeepLabv3 [13]	VOC [18]	mIoU 63.3%	1×2	49 (residual blocks and DeepLab head)	21%
Machine translation	Transformer-Base [84]	WMT16 EN-DE [9]	Perplexity 4.7	4×2 2×2 - 5×2	12 (6 encoders & 6 decoders)	43% 33%-43%
	Transformer-Tiny		53.3	1×8	4 (2 & 2)	19%
Question answering	BERT-Base [17] (fine-tuning)	SQuAD 1.0 [74]	F1 score 87.6	1×2	12 (Transformer blocks)	41%

Table 1. Summary of evaluation tasks. Accuracy targets are the converged accuracy in baseline training. Egeria accelerates different models by 19%-43% to reach the target accuracy.



(a) ResNet-50 training achieves 1.5%+ higher accuracy.

(b) DeepLabv3 training achieves a 21% speedup and full accuracy.

(c) Transformer-Base for machine translation.

(d) Fine-tuning speedups by 41% while smaller accuracy gap.

Figure 8. Egeria can accelerate training for different tasks without sacrificing accuracy compared to the full training baseline, while previous freezing techniques suffer from accuracy loss when reaching the same speedup (except for fine-tuning).

restarts the frozen layers and achieves the same level of accuracy boost. The performance improvement primarily comes from later training stages when Egeria freezes the deeper layer modules with more parameters. Egeria can accelerate lightweight models (e.g., MobileNet V2) on smaller datasets (e.g., CIFAR-10) with 22% and 23% speedups.

Semantic segmentation. We use the DeepLabv3 model with a ResNet-50 backbone for semantic segmentation training. The structure of DeepLabV3 includes a backbone module for feature computation and extraction plus a classifier module that takes the output of the backbone and returns a dense prediction. DeepLabv3 uses a Lambda LR scheduler that changes along with the training procedure, which will trigger the unfreezing mechanism of Egeria at the 45th epoch. Figure 8b shows that, compared to the full baseline, Egeria can reach the target accuracy (mIoU of 63.3%) 21% faster and quickly improve accuracy at the later training stage when the LR scheduler significantly decreases; while the other freezing baselines lose accuracy by 2.1% and 3%.

Machine translation. Egeria not only works for CV models but also for language models. A low perplexity means high accuracy for translation tasks. In Figure 8c, the model quickly reaches a low level of perplexity then continues to improve slowly. Egeria brings a 43% speedup by freezing the front encoders. Unlike CNN models that usually have heavy deep layers, Transformer has a balanced structure, so skipping front layers can bring a considerable speedup. The other freezing baselines each loses perplexity by 0.3 and 0.62. We also evaluate Transformer-Tiny using an 8-GPU machine and achieve a 19% speedup.

Question answering. Training a question answering model is different from the other tasks because we fine-tune a pre-trained general-purpose language model BERT for a new task on a new dataset, rather than training from scratch [17]. Fine-tuning a pre-trained language model (e.g., BERT [17] and GPT-2 [72]) is a popular training technique for NLP tasks because it can save computation overhead and achieve state-of-the-art results for many tasks, e.g., sequence classification and sentiment analysis. The freezing technique was also first used in fine-tuning/transfer learning (see §7). Figure 8d shows the results of fine-tuning BERT on the SQuAD 1.0 dataset. The metric for question answering is the F1 score. Egeria accelerates the baseline by 41% to reach the target accuracy, while AutoFreeze achieves a close performance compared to Egeria as it is design for fine-tuning language models. Since fine-tuning converges faster than training from scratch, Egeria does not freeze many deep layers before achieving the target, but the frozen front layers can still provide a good speedup. During the training, the learning rate scheduler does not trigger the unfreezing mechanism.

Compared to freezing alternatives. Egeria is motivated by the accuracy loss of training from scratch with freezing techniques designed for transfer learning, which is evaluated in Figure 8. AutoFreeze performs well in fine-tuning BERT but loses non-negligible final accuracy in other tasks, while Egeria achieves the target accuracy in all tasks. Using the metric of Skip-Conv loses more accuracy, which was designed for identifying differences in consecutive frames [23]. When comparing models' intermediate results, Skip-Conv metric works similarly to an early KD research, FitNets [76], by directly subtracting two tensors. Recent ML research suggests that, compared to such loss metric, SP loss can better capture the high-level similarity between activations [83].

6.3 Performance Breakdown

FP caching benefits. For single-node training, the performance speedup comes from the BP computation of the frozen layers and prefetching the cached FP results. Figure 9 shows that caching FP generally contributes more for CNN models than language models but are all less than 10%. If there are few frozen layers or the front layers have fewer parameters, FP caching will be disabled.

Distributed training. Egeria accelerates multi-node data parallelism training as shown in Figure 10. Egeria can also work together with communication optimizations like ByteScheduler [67]. Other distributed methods, e.g., pipeline parallelism, can be explored in the future.

ResNet-50 and Transformer are both computation-intensive models, just like most of the recent DNN architectures, so the performance improvement of ByteScheduler is limited here. A slight throughput drop when communication is not the bottleneck is normal for ByteScheduler with the default configuration [66]. While the benefits of Egeria mostly come from computation saving, since frozen layers are not required for parameter synchronization, the reduced communication traffic can speedup the training by up to 5% for ResNet-50, which can benefit the linear scalability of large scale training.

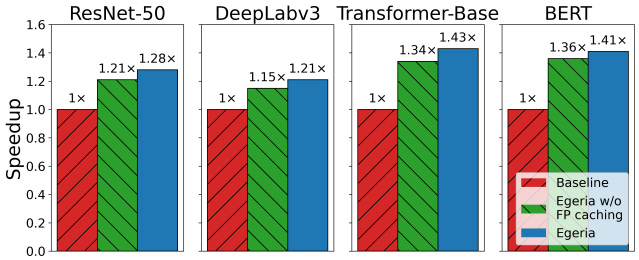


Figure 9. Performance breakdown of using layer freezing (middle) and prefetching FP pass (right).

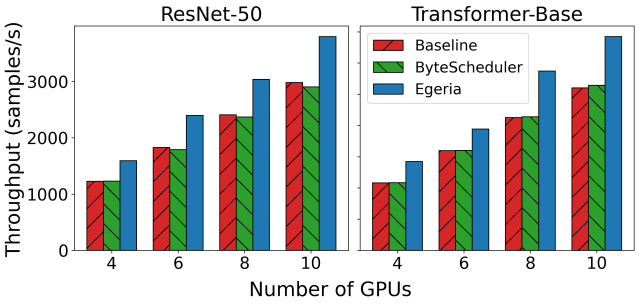


Figure 10. Distributed training performance. Egeria freezes layers to exclude them from parameter synchronization.

Freezing & unfreezing decisions. We take a closer look at one of our evaluations, training ResNet-56, to understand the decisions made by Egeria in Figure 11. The bottom-up DNN consists of layer 1.0–1.8, 2.0–2.8, 3.0–3.8, and input/output layers adjacent to layer 1.0 and 3.8. Egeria parses the model based on its structure and the size of each layer, so that layer 3 (75% of the total parameters), which is significantly larger than layer 2 (20%), is split finer-grained into similar-sized modules; while layer 1 (5%) and layer 2 are evaluated as a whole. Layer 3.7–3.8 (17%) is further split because it is the last module. Egeria gradually freezes layers and remarkably reduces the training cost (the blanks) without hurting accuracy. Refreezing after the 100th and 150th epochs' unfreezing takes much less time because of the relaxed criteria (§4.2.2).

6.4 Sensitivity Analysis

Impact of the reference model's precision on accuracy. Egeria generates the reference model using int8 quantization by default for CPU execution efficiency (§4.1). We evaluate using higher precisions for the reference model, including float16 and float32 (full-precision), in ResNet-56 training on CIFAR-10, as shown in Table 2. We find using the int8-quantized reference model, which averagely has a

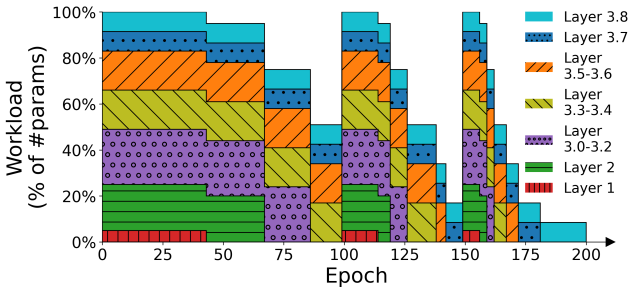


Figure 11. Freezing and unfreezing decisions breakdown through the ResNet-56's 200-epoch training. Y-axis shows the percentage of the active layers' parameters (their sizes).

Performance	int8	float16	float32
Final accuracy	92.1%	92.0%	92.2%
CPU inference speed	3.59×	1.69×	1×
Reference acc. gap	-0.6%	-0.2%	0

Table 2. Using difference precisions for the reference model. Egeria hits the sweet spot between efficiency and accuracy.

0.6% lower accuracy, will not degrade the final accuracy and can largely improve the inference speed to obtain the intermediate activation. Besides, Egeria can switch to higher-precision if int8 fails. Other tasks show similar results.

Impact of hyperparameters on performance. Egeria evaluates plasticity in every n iterations and uses the slope of linear fitting on a moving window W to filter out the drastic fluctuation and provide a recent context. If the plasticity slope has been considerably lower compared to itself during the early fast training stage (i.e., $s < T$) for W evaluations, we freeze the layer. We find that the hyperparameters are tolerant in general when following our guidelines, while drastically changing them could result in performance penalties. As shown in Figure 12, halving W from 10 to 5 or doubling T's coefficient from 20% to 40% would eagerly freeze unconverged layers, hurt the accuracy, but only make training slightly faster, while doubling W to 20 or evaluation interval n from 300 to 600 would lead to longer training time without accuracy gain. Halving T's coefficient to 10% virtually disables freezing. Making frequent evaluations (n=150) brings no extra speedup, while further reducing n could consume more CPUs and potentially slow down the training.

6.5 System Overhead and Discussion

Egeria leverages CPUs to freeze layers accurately while maintaining accuracy; it also uses disk storage to reduce forward computation overhead. As a result, Egeria reduces the training time by 19%-43%. Through careful system designs, we minimize the extra overhead of Egeria.

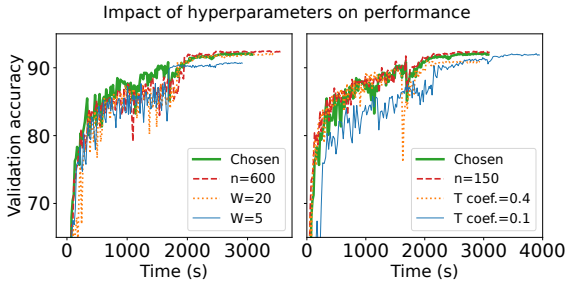


Figure 12. Following our hyperparameter guideline can balance accuracy and speedup (ResNet-56).

The reference model. Using the reference model for plasticity evaluation involves the model generation and execution. We find that generating and updating the reference model through dynamic and static quantization on CPU take 0.5s–1.5s for each time, thus bring no noticeable slowdown. Running the reference model on CPU could introduce up to 1.5% time overhead to the overall training process on standard training server configurations, which is worthwhile compared to the saving from layer freezing. If CPU resources are limited (e.g., on shared machines or using CPU-based optimizations), we support GPU execution for the reference model.

Caching and prefetching. We store the serialized intermediate tensors of the frozen layers to the disk for prefetching. The storage usage depends on the DNN architecture. For example, we need 1.5× to 5.3× compared to the input for ResNet-50, which is generally viable. For language models, since the text data volume is smaller, the overall space usage is limited. Since we only keep the relevant tensors in memory, the overhead is small compared to the regular utilization. It takes hundreds of MB of GPU memory, which is a small fraction of device memory for modern GPUs.

Generalization. We design Egeria as a general system. Users can adjust the usage of CPU and storage in plasticity evaluation and caching to meet their needs. Future research can study how Egeria collaborates with other CPU-based (e.g., BytePS [39]) or storage-based (e.g., CoorDL [59]) optimizations and on different hardware.

7 Related Work

Efficient training systems. Accelerating DNN training is a key goal of ML systems. To optimize the computation, they may optimize the computation graph to maximize the degree of parallelism [38], or deploy advanced scheduling to distribute the computation across multiple machines [21, 57]. To maximize the communication efficiency, priority-based communication scheduling systems [36, 67] use the layered structural information to prioritize the front layers and avoid blocking layers with high priority. BytePS [39] combines the

benefits of parameter server and all-reduce and transmits the gradients among workers or between workers and servers. Some efforts [35, 50] measure the importance of gradient updates in terms of the magnitude of difference w.r.t. the last update, and then filter out trivial parameters before shipping the updates. EGERIA aims to reduce the total training workload, thus should be compatible with them. Additionally, there are a wide range of networking solutions that can help in distributed DNN training [55, 85–87, 97]. ModelKeeper [43] accelerates training by repurposing previously-trained models in a shared cluster. Oort [44, 45] accelerates federated training with guided participant selection.

Using an assistant model in training. EGERIA echoes the broad idea of using another DNN to assist training. Knowledge distillation trains a small student model to mimic the probability distribution of a pre-trained large model [29]. Co-distillation [6] trains multiple tweaked copies of the model in a distributed manner and encourages one model to agree with others' predictions. AutoAssistant [100] trains a light-weight assistant model to identify the hard-to-classify examples and feed them to the training model to improve its performance fast. Infer2Train [30] runs a copy of the training model on the additional hardware accelerator and finds the difficult examples to prioritize in the following iterations.

Freezing parameters and caching DNN results. Existing proposals on freezing are limited to fine-tuning certain models [22, 26, 51] or reducing communication only [12]; otherwise, considerable accuracy loss would nullify any improvements in training speed [11, 46]. An early work FreezeOut [11] explores the freezing technique in general training with heuristics but reports large accuracy loss on many models; nevertheless, it shows that freezing can trade off accuracy for speed. A concurrent work AutoFreeze [51] focuses on fine-tuning pre-trained Transformer-based models; it falls into the original use of transfer learning rather than general training and we found that fine-tuning suffers less from accuracy loss than training from scratch (discussed in §2.3 and §6.2). PipeTransformer [26] also applies freezing in fine-tuning Transformers with pipeline parallelism using a gradient-based importance metric [91]; still, it novelly explores opportunities in pipeline parallelism. We discuss the accuracy performance of gradient-based metrics in §4.2.1. APF [12] excludes stable parameters from synchronization in federated learning; it suggests that model snapshots can best capture the performance and implements a workaround. GATI [8] accelerates DNN inference by caching the intermediate results and skipping the rest of the forward pass.

8 Conclusion

We introduce a novel system EGERIA to accelerate DNN training while maintaining accuracy by accurately freezing the converged layers. To avoid the limitations of existing work, we employ a reference model and use semantic knowledge to evaluate the *plasticity* of internal layers efficiently during training. EGERIA excludes the frozen layers from the backward pass and parameter synchronization. Furthermore, we cache the frozen layers' intermediate computation with prefetching to skip the forward pass. We evaluate EGERIA using several CV and language models and find that EGERIA can accelerate training by 19%-43% without hurting accuracy.

Acknowledgment

We thank the anonymous EuroSys reviewers and our shepherd Dr. Jayashree Mohan for their constructive feedback and suggestions. This work is supported in part by the Key-Area R&D Program of Guangdong Province (2021B0101400001), the Hong Kong RGC TRS T41-603/20-R, GRF-16213621, ITF-ACCESS, the NSFC Grant 62062005, and the Turing AI Computing Cloud (TACC) [93]. Fan Lai and Mosharaf Chowdhury were partly supported by National Science Foundation grants (CNS-1900665, CNS-1909067, CNS-2106184). Kai Chen is the corresponding author.

References

[1] CIFAR-10 benchmark leaderboard. https://paperswithcode.com/sota/image-classification-on-cifar-10.

[2] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[3] Saurabh Agarwal, Hongyi Wang, Kangwook Lee, Shivaram Venkataraman, and Dimitris Papailiopoulos. Adaptive gradient communication via critical learning regime identification. *Proceedings of Machine Learning and Systems*, 3, 2021.

[4] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. In *AAAI*, pages 7350–7357, 2020.

[5] Amazon. Amazon EC2 P3dn instances. https://aws.amazon.com/ec2/instance-types/p3/.

[6] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. In *6th International Conference on Learning Representations, ICLR*, 2018.

[7] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. Pipeswitch: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 499–514, November 2020.

[8] Arjun Balasubramanian, Adarsh Kumar, Yuhan Liu, Han Cao, Shivaram Venkataraman, and Aditya Akella. Accelerating deep learning inference via learned caches. *arXiv preprint arXiv:2101.07344*, 2021.

[9] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, 2016.

[10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[11] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. FreezeOut: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*, 2017.

[12] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. Communication-efficient federated learning with adaptive parameter freezing. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 1–11. IEEE, 2021.

[13] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[14] Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda Viégas, and Martin Wattenberg. Visualizing and measuring the geometry of bert. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8594–8603, 2019.

[15] Moheb Costandi. *Neuroplasticity*. MIT Press, 2016.

[16] 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*, pages 248–255. Ieee, 2009.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[19] L. Fei-Fei, A. Karpathy, and J. Johnson. Cs231n: Convolutional neural networks for visual recognition: Annealing the learning rate.

[20] L. Fei-Fei, A. Karpathy, and J. Johnson. Cs231n: Convolutional neural networks for visual recognition: Transfer learning.

[21] 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)*, pages 485–500, Boston, MA, 2019.

[22] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: Transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] Amirhossein Habibian, Davide Abati, Taco S Cohen, and Babak Ehteshami Bejnordi. Skip-convolutions for efficient video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2695–2704, 2021.

[24] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[25] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H Campbell. TicTac: Accelerating distributed deep learning with communication scheduling. In *Proceedings of Machine Learning and Systems, MLSys*, 2019.

[26] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of large-scale models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4150–4159. PMLR, 18–24 Jul 2021.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[28] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3779–3787, 2019.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[30] Elad Hoffer, Berry Weinstein, Itay Hubara, Sergei Gofman, and Daniel Soudry. Infer2train: leveraging inference for better training of deep networks. In *NeurIPS 2018 Workshop on Systems for ML*, 2018.

[31] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, 2018.

[32] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. Starfish: resilient image compression for aiot cameras. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 395–408, 2020.

[33] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

[34] Intel. oneDNN library. https://01.org/oneDNN/.

[35] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems (NeurIPs)*, 2019.

[36] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed DNN training. In *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*, 2019.

[37] 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)*, pages 947–960, 2019.

[38] 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, page 47–62, 2019.

[39] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

[40] Keras. Earlystopping. https://keras.io/api/callbacks/early_stopping/.

[41] Keras. Transfer learning & fine-tuning. https://keras.io/guides/transfer_learning/.

[42] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[43] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. Modelkeeper: Accelerating dnn training via automated training warmup. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[44] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pages 11814–11827. PMLR, 2022.

[45] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *OSDI*, pages 19–35, 2021.

[46] Jaejun Lee, Raphael Tang, and Jimmy Lin. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*, 2019.

[47] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[48] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train large, then compress: Rethinking model size for efficient training and inference of transformers. In

Proceedings of the 37th International Conference on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 5958–5968. PMLR, 13–18 Jul 2020.

[49] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33, 2020.

[50] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

[51] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. *arXiv preprint arXiv:2102.01386*, 2021.

[52] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[53] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[54] Robert M. Love. taskset(1) — Linux manual page. https://man7.org/linux/man-pages/man1/taskset.1.html.

[55] Yiqing Ma, Hao Wang, Yiming Zhang, and Kai Chen. Autobyte: Automatic configuration for optimal communication scheduling in dnn training. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 760–769. IEEE, 2022.

[56] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, volume 238, page 5, 2015.

[57] 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)*, pages 289–304, Santa Clara, CA, 2020.

[58] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. *arXiv preprint arXiv:2007.06775*, 2020.

[59] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. *Proc. VLDB Endow.*, 14(5):771–784, January 2021.

[60] Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5732–5741, 2018.

[61] 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, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[62] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[63] Xingyuan Pan and Vivek Srikumar. Expressiveness of rectifier networks. In *International Conference on Machine Learning*, pages 2427–2435, 2016.

[64] Geondo Park, Gyeongman Kim, and Eunho Yang. Distilling linguistic context for language model compression. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021.

[65] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[66] Yanghua Peng and ByteScheduler team. ByteScheduler issues. https://github.com/bytedance/byteps/issues?q=label%3Abytescheduler.

[67] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 16–29, 2019.

[68] PyTorch. PyTorch autograd mechanics. https://pytorch.org/docs/stable/notes/autograd.html.

[69] PyTorch. PyTorch DataLoader. https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader.

[70] PyTorch. PyTorch Vision. https://github.com/pytorch/vision.

[71] PyTorch. Transfer learning for computer vision tutorial. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html.

[72] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[73] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep understanding and improvement. *network*, 200(200):200, 2017.

[74] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[75] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[76] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[77] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[78] Scikit-learn. Early stopping of stochastic gradient descent. https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_early_stopping.html.

[79] Sheng Shen, Alexei Baevski, Ari S Morcos, Kurt Keutzer, Michael Auli, and Douwe Kiela. Reservoir transformer. *arXiv preprint arXiv:2012.15045*, 2020.

[80] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[81] TensorFlow. TensorFlow stateless dropout. https://www.tensorflow.org/api_docs/python/tf/nn/experimental/stateless_dropout.

[82] TensorFlow. TensorFlow stateless random image transformations. https://www.tensorflow.org/tutorials/images/data_augmentation#random_transformations.

[83] Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[85] Xinchen Wan, Kai Chen, and Yiming Zhang. Dgs: Communication-efficient graph sampling for distributed gnn training. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2022.

[86] Hao Wang, Jingrong Chen, Xinchen Wan, Han Tian, Jiacheng Xia, Gaoxiong Zeng, Weiyan Wang, Kai Chen, Wei Bai, and Junchen Jiang. Domain-specific communication optimization for distributed DNN training. *arXiv preprint arXiv:2008.08445*, 2020.

[87] Weiyan Wang, Cengguang Zhang, Liu Yang, Kai Chen, and Kun Tan. Addressing network bottlenecks with divide-and-shuffle synchronization for distributed dnn training. In *IEEE INFOCOM 2022-IEEE*

Conference on Computer Communications, pages 320–329. IEEE, 2022.

[88] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1508–1518, 2017.

[89] 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*, pages 38–45. Association for Computational Linguistics, October 2020.

[90] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344. IEEE, 2019.

[91] Xueli Xiao, Thosini Bamunu Mudiyanselage, Chunyan Ji, Jie Hu, and Yi Pan. Fast deep learning training through intelligently freezing layers. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1225–1232. IEEE, 2019.

[92] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[93] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. Tacc: A full-stack cloud computing infrastructure for machine learning tasks. *arXiv preprint arXiv:2110.01556*, 2021.

[94] Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems*, 3, 2021.

[95] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014.

[96] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014.

[97] Chaoliang Zeng, Xiaodian Cheng, Han Tian, Hao Wang, and Kai Chen. Herald: An embedding scheduler for distributed embedding model training. *6th Asia-Pacific Workshop on Networking (APNet)*, 2022.

[98] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

[99] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 181–193, 2017.

[100] Jiong Zhang, Hsiang-Fu Yu, and Inderjit S Dhillon. Autoassist: A framework to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[101] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.