PieBridge: Fast and Parameter-Efficient On-Device Training via Proxy Networks

Wangsong Yin[1], Daliang Xu[1], Gang Huang[1,2], Ying Zhang[1]
Shiyun Wei[3], Mengwei Xu[4#], Xuanzhe Liu[1]
[1]School of Computer Science, Peking University, Beijing, China
[2]National Key Laboratory of Data Space Technology and System, Beijing China
[3]Zhongguancun Laboratory, Beijing, China
[4]Beijing University of Posts and Telecommunications, Beijing, China.
yws@stu.pku.edu.cn
{xudaliang,hg,zhang.ying,weishiyun,liuxuanzhe}@pku.edu.cn
mwx@bupt.edu.cn

ABSTRACT

On-device training Neural Networks (NNs) has been a crucial catalyst towards privacy-preserving and personalized mobile intelligence. Recently, a novel training paradigm, namely Parameter-Efficient Training (PET), is attracting attention in both the machine learning and system community. In our preliminary measurements, we find PET well-suited for on-device scenarios; yet, its parameter efficiency does not translate coequal to time efficiency on resource-constrained devices, as the training time is dominated by the frozen layers.

To this end, this work presents `PieBridge`, an on-device training framework with both time and parameter efficiency. Its key idea is to dynamically approximate the frozen layers to cheaper ones (subnets) with data awareness during PET. To achieve effective and efficient approximate training, we introduce (1) a pre-training-assisted on-cloud subnets generation method and (2) an edge-friendly on-device data-aware subnets routing method. The subnets generation method performs fine-grained pruning and latent space alignment to generate a series of high-quality proxy subnets with varying speed-accuracy trade-offs for the deployment-ready NN. The subnets routing method perceives data diversity from two unique perspectives (referred to as importance and difficulty). The routing strategy is provided by an offline-learning and online-estimation fusion, which is accurate, end-to-end and cost-effective on devices. Through extensive experiments, we show that `PieBridge` exhibits up to 2.5× training speedup compared to state-of-the-art PET methods, and up to 6.6× speedup compared to traditional full model training and other on-device training frameworks, without compromising parameter efficiency and accuracy.

[#]Corresponding author.

1 INTRODUCTION

On-device training Neural Networks (NNs) has been a crucial catalyst towards privacy-preserving and personalized mobile intelligence. There exist numerous applications: input prediction personalization [94], sequential recommendation system [43], video analytics [25], voice assistant wake-up service [5], etc. Due to its huge resource demand, substantial prior literature [11, 16, 17, 55, 73, 92, 102, 103, 106] have attempted to alleviate its resource tension on mobile devices.

Recently, Parameter-Efficient Training (PET) methods such as adapter [50, 80] and LoRA [52] have been widely recognized as an effective training paradigm. Freezing most of the pre-trained weights, PET achieves competitive accuracy by adjusting only a tiny portion (e.g., ≤1%) of parameters compared to fully fine-tuning the model. Thereby, PET can efficiently deploy a shared foundation NN [96, 99] to multiple downstream tasks on devices, with only a few plug-and-play parameters. As will be shown in §2.2, it also reduces the training overhead including memory footprint and data labels — two scarce resources on mobile devices. It could even offer enhanced data security by leveraging TEE to protect the plugged parameters [104].

However, such parameter efficiency does not translate coequal to time efficiency. For instance, when fine-tuning ResNet50 [47] on a MI 10 smartphone, PET methods with less than 1% trainable parameters still consume up to 96% and on average 65% of the training time compared to traditional full-model fine-tuning (§2.2,

Figure 2b). Such long training time results in more energy consumption, compromised user experience and delayed personalization service. Diving into the reason, we find out that the frozen layers dominate the training time (both forward pass and backward pass). For instance, when training Vision Transformer (ViT_base) [33] on NVIDIA Jetson TX2 with PET methods, the frozen layers account for 96.34%—99.59% of the computation time (Table 2). This observation provides a key opportunity that, as the compute-intensive layers are frozen during training, we can approximate them with cheaper ones (dubbed as *Proxy sub-networks*, or *subnets* in short).

The main challenge in accelerating PET through NN approximation is that a *simple, static replacement strategy* could result in nontrivial degradation of convergence accuracy. For instance, approximating the frozen layers of a ViT_base with a structurally identical network that has significantly fewer FLOPs (e.g., < 50%) for fine-tuning on the DTD [3] dataset would result in a 29.11% decrease in accuracy (§5.2, Figure 13), and this number would further increase on more complex datasets [2, 8, 9]. This arises from two aspects: one is the error between the approximated network and the original network; the other is the compromised generalization ability due to the decreased model capacity.

In response, this work presents `PieBridge`, the first fast PET framework for mobile devices. Instead of statically approximating the frozen layers during PET, the key idea of `PieBridge` is **data-aware dynamic approximation and routing.** It's based on a key insight that the approximation of frozen layers must be more fine-grained, i.e., *be dynamic for different training samples* during on-device training due to the high diversity of real-world data [29, 31, 41, 45, 48, 54, 76, 100, 102]. More specifically, the training data exhibits opportunistic diversity in two unique aspects: *difficulty* and *importance*. Difficulty is that some training samples are intrinsically easier to extract accurate and informative features from. Importance is that some non-trivial training samples can facilitate faster convergence and better generalization. We define them in our system model and show their ubiquitous presence in §2.3.

Realizing the above design needs to tackle two crucial challenges. First, training-oriented subnets are hard to acquire: they must be with low approximation-induced accuracy loss, fast on-device execution speed, and a vast latency-accuracy trade-off space. Second, as will be shown in §2.3, perceiving data diversity is non-trivial: data importance is stateful, and data difficulty is stateless but posterior. Online perceiving data diversity brute-force and forming routing strategy introduce undesirable extra time overhead.

`PieBridge` incorporates a two-stage design akin to prior literature [34, 87, 89]: on cloud (offline), it pre-generates the approximated subnets through public data; on devices (online), it finetunes the models by judiciously routing the data to proper subnets. Specifically, the aforementioned challenges are tackled in the following two stages: the on-cloud pre-approximation and the on-device data-aware routing.

In the first stage, `PieBridge` employs a pruning-based fine-grained approximation method. By gradually and structurally removing various components from the original NN, this method generates subnets with faster on-device execution speed. The design differs significantly from prior inference-oriented pruning-based NN approximation [18, 60], since: (1) It redesigns the pruning

specification for PET, ensuring that the pruned NN structure is compatible with the parameter efficiency of the original NN. (2) It recognizes that the requirements of NN approximation for training are more stringent (i.e., latent space alignment instead of trivial label alignment), thereby proposes a lightweight on-cloud retraining mechanism for more effective NN approximation.

In the second stage, `PieBridge` proposes an online-offline fused data perceiving and subnets routing method. For data difficulty, it offline trains a transferable tiny policy model with public data as a substitute for online traversing all the subnets on devices; For data importance, it builds an online monitor to employ observed training loss for zero-overhead estimation. Specifically, the key insight of the policy model is that system-level compute-expensive estimation can be simplified using a data-driven black-box model. It perceives data difficulty through a novel joint training mechanism with subnets. Our method orchestrates the above perception (difficulty and importance) into subnets routing strategy during on-device training. Through these algorithm-system co-design, our method is accurate, end-to-end and cost-effective on devices.

We implemented `PieBridge` on multiple mobile/embedded devices including Nvidia Jetson TX2 [12], Raspberry Pi 4B [14], MI 10 smartphone [10] and an NPU-empowered Commercial Off-The-Shelf (COTS) device Huawei Mate 30 [7]. We then evaluate its performance on four popular datasets [1–4] and three representative NN architectures [33, 47, 51]. The results show that `PieBridge` consistently outperforms baselines in terms of time-to-accuracy, achieving up to 6.6× on-device speedup compared to traditional full model training and other competitive on-device training frameworks [55, 73], and 1.5×–2.5× speedup compared to state-of-the-art PET methods. Meanwhile, `PieBridge` does not compromise convergence accuracy and parameter efficiency. Moreover, with the assistance of NPUs [6], `PieBridge` makes on-device training more practical on COTS devices: a medium-sized workload (ResNet50, Caltech-101) requires only 0.21 hours and 0.91 kJ of energy on Huawei Mate 30.

The major contributions are summarized below.

- We thoroughly explore a key opportunity for tapping into the potential of parameter efficiency: dynamic frozen-layer approximation, and propose the first on-device training framework with both time and parameter efficiency.

- We introduce a pre-training-assisted on-cloud approximation method with high-quality speed-accuracy trade-offs, as well as an edge-tailored data perceiving and routing method that effectively and fully exploits the benefits brought by frozen-layer approximation.

- We implement `PieBridge` on diverse mobile/edge devices and demonstrate its effectiveness through extensive experiments. `PieBridge` makes on-device NN adaptation and deployment much more practical for COTS devices in the era of deep learning.
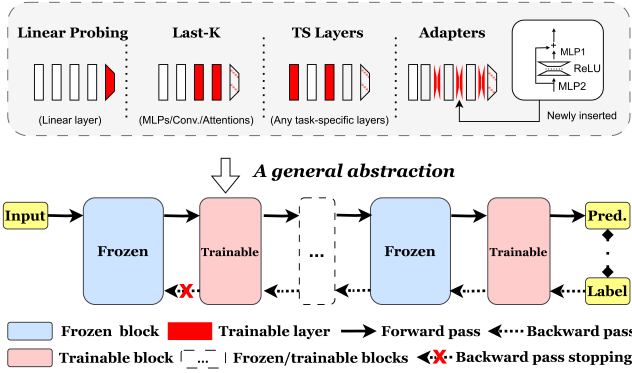
2 BACKGROUND AND MOTIVATION

2.1 Parameter-Efficient Training

Parameter-Efficient Training (PET) is a class of training techniques that aims to achieve performance comparable to full model training while freezing most pre-trained parameters. Its parameter efficiency exhibits the following two pivotal characteristics: (1) ***Few***

PET Method	Trainable layers	Formula	Trainable Parameters
Linear Probing	The last linear layer	$y = x \cdot W + b$	0.69%
Last-K	The last K layers	N/A	≥8.3%
TS Layers	Task-specific layers (Norm. layers here)	N/A	0.87%–2.80%
Adapters	The inserted adapters	$y = x + \sigma(x \cdot W_{down}) \cdot W_{up}$	0.71%

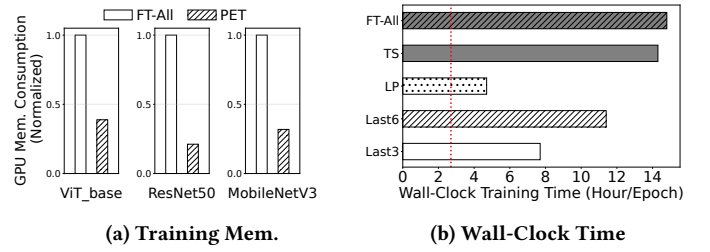Table 1: Comparisons of practical on-device PET methods. Model: ViT_base.

Figure 1: A layer-level general abstraction of PET.

Figure 2: The preliminary measurement results of PET. (a) Theoretical memory consumption of fine-tuning all parameters (FT-All) and PET (ViT_base: Adapters; ResNet50: Linear Probing; MobileNetV3: Last-3; mini-batch size=4). (b) Wall-clock training time per epoch (ResNet50 + Caltech-256 + MI10, LP: Linear Probing). Red dotted line: the average daily training-available time on smartphones (2.7 hours) [94].

adjustable parameters. Table 1 provides a detailed presentation. (2) *Statically predefined trainable components.* Trainable layers remain fixed during PET.

The rationale behind PET is that, through pre-training on large-scale datasets, the NN has learned a general inductive bias that is close to the downstream task (e.g., object contours in CV tasks). Therefore, the expression ability brought by only adjusting a small number of parameters can adapt this bias to the downstream task. PET is generally orthogonal to the deep model design and the training optimizers, and has gained lots of attentions in both academia and industry with the rise of large pre-trained models.

We summarize widely used on-device PET methods in Table 1. (1) "Linear Probing" means only fine-tuning the output layer and maximizing the retention of pre-trained knowledge.It can achieve accuracy close to or even better than fine-tuning all parameters according to recent studies, especially when data domains are close [59, 61, 68, 81]. (2) "Last-K" method tunes the last K layers based on the rationale that layers close to the NN input learn generic transferable features [30, 42, 90]. (3) "TS layers" means fine-tuning task-specific layers for specific downstream tasks, e.g., normalization layers for domain adaptation, prefix layers for tuning language models [67], or text-embedding layers for adding new concepts into text-to-image diffusion models [38]. (4) "Adapters" method proposes to inject trainable, task-specific "adapter" modules between layers of the pre-trained model [50, 80]. There are other advanced PET techniques such as LoRA [53] and p-tuning [71], yet they are mainly designed for very large models (e.g., LLMs) and are not readily well deployed on devices.

A unified abstraction of PET. We unify the above PET methods into a general abstraction for the design of PieBridge. As shown in Figure 1, the "frozen block" remains unchanged after pre-training, while the "trainable block" is optimized during downstream fine-tuning. Each "block" contains as many consecutive frozen or trainable layers as possible. In a PET training iteration, both the frozen and trainable blocks need computing the forward pass. During the backward pass phase, the frozen blocks only need to compute the error gradients, while the trainable blocks not only compute the error gradients but also perform weight updates. Note that when all the components preceding a specific trainable block are frozen, there is no need to continue computing the backward pass further.

2.2 Preliminaries

Observation#1: PET is friendly to the on-device deployment of foundation NNs. With the scaling of NN size and advancements in pre-training techniques [33, 46], on-device NNs are converging from various architectures and weights to one generalized foundation NN (e.g., M4 [99] and EdgeFM [96]). The parameter efficiency of PET enables the foundation NNs to adapt to diverse tasks through plug-and-play additional parameters, with high scalability and lightweight memory/storage/task-switching overhead. Furthermore, recent research on critical parameter protection on devices [104] indicates that PET's parameter efficiency provides enhanced security and privacy, as these few parameters can be executed within the Trusted Execution Environments (TEEs) like ARM TrustZone.

Observation#2: PET is less prone to the device memory wall. Since most parameters are frozen, PET does not need to keep their gradients and the intermediate activation (at least linear layers and bottom frozen layers) in memory [27]. Figure 2a shows that PET only consumes 21.19%–38% memory compared to full model training.

Observation#3: However, parameter efficiency does not translate to time efficiency. Although >90% parameters are frozen, PET

Model	PET Config.	Upd. Paras.	Frozen Layers		Trainable Layers	
			Time (Sec.)	FLOPs (G)	Time (Sec.)	FLOPs (G)
ViT_base	Linear Probing	0.69%	0.46	67.44	1.88×10^{-3}	0.36
	Adapters (6)	1.75%	0.64	134.86	5.44×10^{-3}	0.54
	Adapters (12)	2.80%	0.84	202.31	1.24×10^{-2}	0.71
ResNet50	Linear Probing	0.81%	0.06	16.53	4.36×10^{-4}	4×10^{-2}
	Last-K (K=3)	14.3%	0.07	15.59	2.56×10^{-3}	2.15

Table 2: Breakdown of different parts in a PET iteration. mini-batch size=4; Device: Jetson TX2.

is still time-consuming with only 35.82% training speedup on average in Figure 2b. Fine-tuning ResNet50 1 epoch with Caltech-256 on a MI10 smartphone takes 14.8 hours, and even with the most aggressive PET strategy that only performs linear probing, it still takes 4.7 hours, which is significantly longer than the average daily training-available time on smartphones (2.7 hours, red dotted line) [94].

Observation#4: Frozen layers are significantly more computationally expensive than trainable layers in a PET iteration. Table 2 shows the measured time consumption of frozen and trainable layers in a PET iteration (forward + backward). The majority of the computation time is attributed to the frozen parts (96.34%–99.59%).

Implications Parameter-efficient training is a good fit to on-device training. However, even though most parameters are frozen, it is still time-consuming. To make on-device PET practical and efficient, system level support is needed to re-architect the model structure and training paradigm. Given the predominance of frozen layers, it is promising to focus on reducing their computational cost.
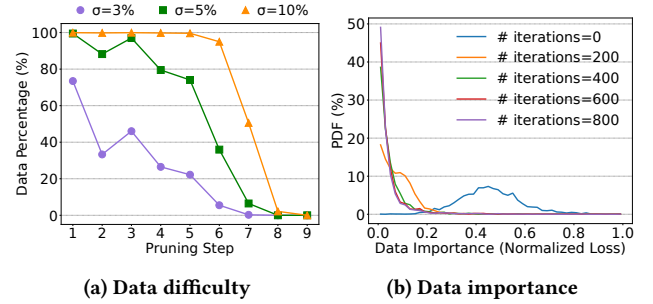
2.3 Training Data Diversity

To reduce the forward/backward training time on the frozen layers, this work proposes to opportunistically replace them with more lightweight ones. Yet, training is known to be sensitive to weights precision [22, 40, 83, 93] and simply downsampling the frozen layers could degrade the convergence accuracy significantly (§5.2, Figure 13).

To speedup on-device PET while not compromising the convergence accuracy, we exploit a known opportunity: *data diversity*. Existing studies have provided evidence for the presence of diversity in real-word data [29, 31, 41, 45, 48, 54, 76, 100, 102]. From our observation, data in deep learning presents two aspects: (1) Samples vary in their difficulty level, where some are intrinsically easier to extract accurate and informative features from, i.e., *data difficulty*. (2) Samples also differ in their importance level to training tasks, as certain nontrivial examples can facilitate faster model convergence and better generalization, i.e., *data importance*.

We provide an incomplete but intuitive example of data diversity in Figure 3. We select representative training samples from a classic transfer learning benchmark dataset Dogs vs. Cats [4]. Some training samples are more difficult to extract features from, because they are intrinsically blurry, obstructed, or complicated, etc. When the NN has learned plenty of tabby cats, a non-trivial British Shorthair can help the NN to better generalize to the broader concept of "cats", thus contributing more to convergence.

Figure 3: An intuitive example of data diversity. Dataset: Dogs vs. Cats; Class: cats.

(a) Data difficulty   (b) Data importance

Figure 4: The diversity of training data from a statistical perspective. Model: ResNet50; Dataset: Caltech-101; Pruning step in (a): about 7%–10% FLOPs.

Here, we provide more formal definitions, and demonstrate the training data diversity from a statistical perspective.

Definition#1: data difficulty is measured by feature quality. `PieBridge` provides multi-level approximation for a network. Specifically, given a pre-trained NN $\mathcal{F}_0$, we approximate it with a series of subnets $\{\mathcal{F}_n\}, n \in \{1, 2, \cdots, N\}$ (pruned and fine-tuned in §3.2; the larger n, the smaller capacity). Based on the rationale that NNs with larger capacity exhibit stronger capability, the difficulty of a data sample x can be reflected by the minimal subnet that captures feature that on-par with the non-approximated network. We define data difficulty $\phi(x)$ by

$$\phi(x) = max(n \in \{0, 1, \cdots, N\}),$$
$$s.t. \quad \mathcal{F}_n(x) \simeq \mathcal{F}_0(x). \tag{1}$$

where $\mathcal{F}_n(x)$ is the feature extracted by NN $\mathcal{F}_n$, $\simeq$ means the features are equivalent with each other in PET. The larger $\phi(x)$ is, the simpler the data sample x becomes.

Definition#2: data importance is measured by its contribution to the model convergence. Given a data sample x, the gradient norm $\|\nabla l(x; w_t)\|$ reflects the degree of contribution that x makes to the model convergence at time t [58, 102]. In practice, the feed-forward loss $l(x; w_t)$ is a more straightforward estimation for the gradient norm, and it is proven to possess a relatively tight upper bound [58]. Thus, we define data importance $\psi(x, t)$ by

$$\psi(x, t) = l(x; w_t) = \mathcal{L}[\mathcal{F}(x; w_t), y], \tag{2}$$

where w_t is the state of NN $\mathcal{F}$ at time (i.e., training iteration) t, y is the label of data sample x, $\mathcal{L}$ is the function to calculate feed-forward loss. Notably, data importance $\psi(x, t)$ is with respect to both the data sample x itself and the current training time step t, being stateful. This is owing to the data importance is influenced by the knowledge that has been learned by the NN in the training process.

Training data is diverse in difficulty and importance. After defining data difficulty/importance, we further show that a realistic training dataset can exhibit highly diversified difficulty/importance, which sheds light on assigning various subnets for various data samples.

For data difficulty, to concretely show its diversity, here we temporarily use a straightforward cosine similarity[1] for modeling the closeness of the extracted features. Specifically, $\mathcal{F}_n(x) \simeq \mathcal{F}_0(x)$ is measured by $\mathcal{F}_n(x) \otimes \mathcal{F}_0(x) > 1 - \sigma$, where $\otimes$ is the operator to compute cosine similarity, and σ is a threshold. Figure 4a shows the percentage of training data with feature cosine similarity greater than $1 - \sigma$ as NN is pruned. With the threshold σ set at 3%, more than 70% of the training data is capable of being transferred to lighter sub-networks, and in excess of 20% can be migrated to a sub-network demanding just half the computational power of original NN $\mathcal{F}_0$. With this inclined curve, we can see that data difficulty is diversified in a collection of data. The diversity consistently exists with varied σ in Figure 4a.

For data importance, in Figure 4b we observe that various samples also demonstrate diversity in importance; and the importance varies with time step (# iterations). For example, at the beginning of training, the data importance obeys a Gaussian distribution; post 200 iterations, the distribution rapidly consolidates to a minor fraction of data samples, with the remainder having a minimal impact on model training.

Remarks. So far, we have defined data difficulty and importance for PET, and have shown that a collection of training data is diverse in these aspects. This inspires `PieBridge` to dynamically approximate the frozen layers with the awareness of data characteristic during training. Notably, while training data has diversity is a known property [23, 86], we are the first to present a tailored and rigorous definition of data diversity in the context of PET, and we are also the first to identify the unique key opportunity that it can work with the approximation of static frozen layers to accelerate on-device fine-tuning. A further discussion can be found in §7.

3 PIEBRIDGE DESIGN

3.1 Overview

System design goal. `PieBridge` is a cloud-device collaborated framework that aims to speed up the parameter-efficient on-device fine-tuning while not compromising the convergence accuracy (i.e., better time-to-accuracy) and the parameter efficiency.

Workflow. Figure 5 illustrates the two-stage simplified workflow of `PieBridge`.

At cloud offline stage, `PieBridge` generates proxy networks and prepares for the on-device training. ① It first employs a PET

oriented structured pruning to reduce the size of a given well-pre-trained original neural network. ② In order for the proxy sub-networks to participate in the training of the original network, a lightweight retraining mechanism is introduced to enable them to share the same latent space. ③ `PieBridge` profiles or estimates their actual on-device execution cost (including the original network) and records it in a latency score table. ④ Data difficulty is learned offline by an extremely lightweight end-to-end policy model. `PieBridge` jointly trains it with the original network, proxy sub-networks, and on-device cost information using general public data on cloud. ⑤ Finally, the original network, proxy sub-networks, and the policy model are all deployed together onto the device.

At device online stage, like other on-device continuous machine learning systems, `PieBridge` consists of two loosely-coupled runtimes, i.e., training runtime and inference runtime. At training runtime, the plug-and-play trainable parameters are fine-tuned with device-collected data to adapt it to the drifted task or domain. At inference runtime, the deployed NN serves as a one-size-fits-all foundation model shared by various applications. `PieBridge` focuses on training runtime. It dynamically routes training samples to different networks (the original network or the proxy sub-networks) according to an importance-and-difficulty-aware strategy. This routing strategy for certain data sample x is fused from two sources: (1) The difficulty-based routing strategy from the policy model; (2) The importance $\psi(x, t)$ monitored in near real-time.

3.2 On-Cloud Pre-Approximation

Independent-Dimension Constrained (IDC) pruning. There are many ways to obtain a lightweight approximation of the original network, such as pruning [18, 54], quantization [37], distillation [21], and early exit [91]. Intuitively, magnitude-based pruning provides the best trade-off between accuracy and computation among the above methods, since it is the most fine-grained. Given a neural network with weights denoted as W, we can rank its element w_i^j by the magnitude $\left\| \nabla l(x; w_i^j) \cdot w_i^j \right\|$ on calibration samples x and prune it. Akin to prior arts [35, 74], `PieBridge` derives calibration samples from public general dataset (100 samples from ImageNet, by default).

To achieve actual speedup on device processors (e.g., ARM CPUs [7, 10] or NVIDIA edge GPUs [12]) which typically do not support sparsity, the pruning must be structured. However, traditional on-device acceleration oriented structured pruning [13, 18] lacks additional structural constraints, resulting in the approximated network being unable to participate in PET alongside the original network. Figure 6 shows an example of removing 20% parameters from a PET structure in CNN. A tuple (128, 128) represents the input/output dimensions of a layer. If we arbitrarily prune any dimensions of a frozen layer (Figure 6(a)), the generated subnets cannot replace each other due to shape mismatch.

To this end, `PieBridge` employs a dedicated pruning specification for PET, named Independent-Dimension Constrained (IDC) pruning (Figure 6(b)). Specifically, Given a PET network structure, IDC pruning identifies dimensions that are independent to its trainable blocks. For a CNN block $y = C_r(\cdots C_1(x))$ where $C_r(x)$ represents the r_{th} convolutional layer, the independent dimensions are (1) $C_1^{out}, C_2^{in}, \cdots, C_{r-1}^{in}$, if r is odd; (2) $C_1^{out}, C_2^{in}, \cdots, C_r^{in}$, if r is even.

---

[1] Cosine similarity reflects the similarity of two tensors a and b by calculating $\frac{a \cdot b}{\|a\| \cdot \|b\|}$. It is widely used in ML techniques such as knowledge distillation [20, 49] and vector database [44].
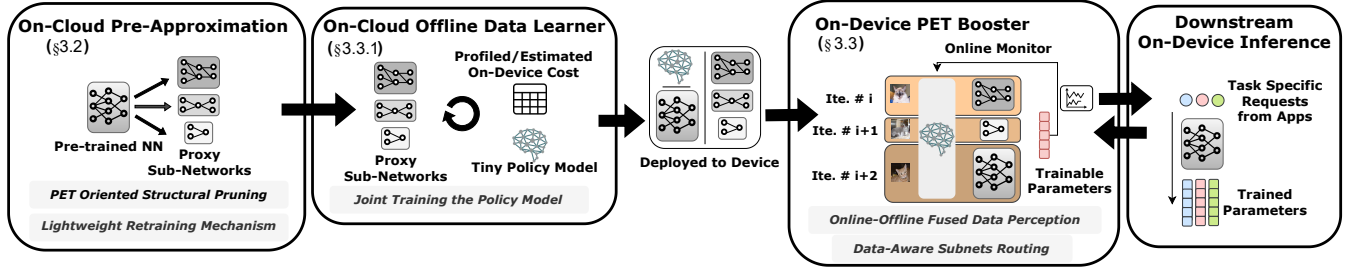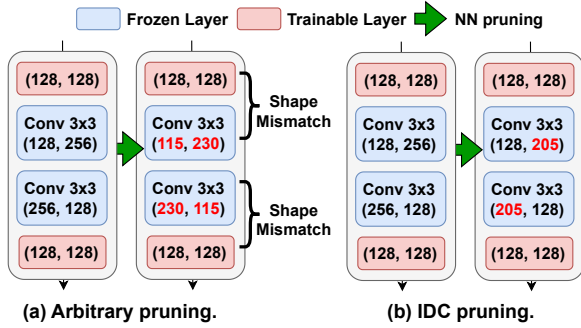
Figure 5: An overview of `PieBridge`.

Figure 6: `PieBridge` only prunes independent dimensions of NNs. In doing so, proxy sub-networks are compatible with each other.
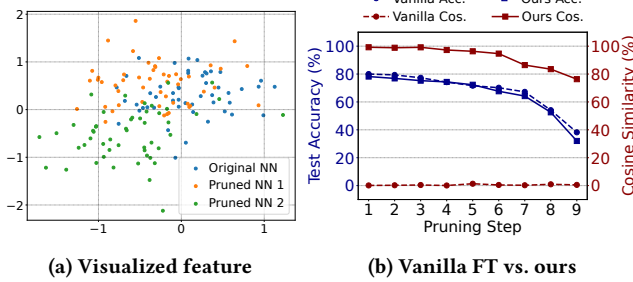

Figure 7: The latent space shift. (a) The visualized feature extracted by the original NN and 2 pruned subnets suffers a shift. (b) Test Accuracy and average cosine similarity between pruned and original NN.

For a Transformer block y = $\mathcal{W}_{down}(\mathcal{W}_{up}(\mathcal{W}_O(\mathcal{W}_Q(x)\mathcal{W}_K(x)\mathcal{W}_V(x))))$, the independent dimensions are $\mathcal{W}_Q^{out}, \mathcal{W}_K^{out}, \mathcal{W}_V^{out}, \mathcal{W}_O^{out}, \mathcal{W}_{up}^{out}, \mathcal{W}_{down}^{in}$. Following the above constraints, `PieBridge` performs magnitude-based pruning at a step size d (by default 10%, varied in §5.2). For instance, proxy sub-network $\mathcal{F}_1/\mathcal{F}_5$ removes 10%/50% parameters compared to the original network $\mathcal{F}_0$ (and FLOPs are reduced in proportion).

In a nutshell, `PieBridge`'s IDC pruning effectively provides approximations that achieve on-device speedup, and enables the generated proxy subnets to work seamlessly with each other.

Latent space alignment through lightweight retraining. Unfortunately, the proxy sub-network pruned by IDC pruning is still not a good (accurate) approximation. We dive into the reason and

find out that the requirements of NN approximation for training are more stringent. In Figure 7a. we reduce the dimensionality of the latent space through PCA (Principal Component Analysis) and visualize it. The latent spaces exhibit a gap. It means that, though the pruned network may retain the ability to do inference (e.g., predict a dog as "dog"), the feature feed to the shared trainable block may be tortured and thus noisy for PET. To this end, we propose an enhanced alignment method with a dedicated regularization term. For each proxy sub-network $\mathcal{F}_n$, $n \in \{1, 2, \cdots, N\}$, we respectively minimize

$$\mathcal{L} = \mathcal{L}' + \sum_{r=1}^{R} \lambda_r (h_0^r \otimes h_n^r), \qquad (3)$$

where $\mathcal{L}'$ is vanilla fine-tuning loss (e.g. cross entropy loss), h_n^r is the hidden state before the r_{th} trainable block (i.e., the input), R is the number of trainable blocks, $\otimes$ is an operator to compute similarity, and λ_r is the scale factor (1.0 by default). **Effectiveness.** Figure 7b reports the test accuracy and latent space alignment of vanilla fine-tuning and our method. After vanilla fine-tuning, the features can only maintain a good test accuracy but compromise the alignment, while our method achieves both. We integrate our alignment method into a lightweight retraining mechanism triggered after pruning. We also conduct experiments on significance of key designs in §5.2.

Overhead analysis. Firstly, it is worth noting that the pre-approximation process takes place on cloud, which is relatively resource-abundant. Secondly, compared to the resource-intensive pre-training process, the cost of our pruning and retraining is much lower. We only perform 3 epoch of retraining for each sub-network. Furthermore, this process is executed one-shot: these approximated public proxy sub-networks can be shared by various devices.

3.3 On-Device Data-Aware Routing

Profiling data diversity: undesirable system overhead. According to equation 1, acquiring the difficulty of a training sample is "posterior": it demands up to N forward passes in the worst case (whereas the training itself involves just one forward/backward pass), which is unacceptable on devices. According to equation 2, data importance is "stateful": it is influenced by the current training state t, which incurs additional forward computations [102] for accurate perception because of the issue of "staleness" (Figure 8).

Online-offline fused data perception. Inspired by ML-based system optimization efforts, such as database query optimization [75,
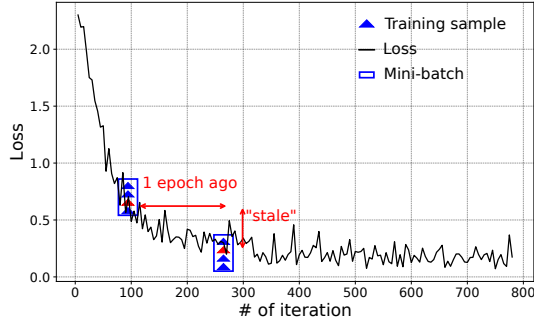
Figure 8: The precision of data importance estimation. The estimate of specific training sample (red triangle) from the previous forward pass becomes "stale" as the training progresses.

Algorithm 1: Data-aware on-device training.

```
input    : {𝓕ₙ}, n ∈ {0, 1, 2, · · · , N}; Ω; On-device dataset D;
output   : Fine-tuned 𝓕₀;
1  Function data_aware_training():
2     train_loader ← D, Policy_sampler(); θ ← −1; m ← 0.8;
3     for epoch in epochs do
4        for data in train_loader do
5           X, n ← data; // mini-batch and routing strategy
6           l ← X, 𝓕ₙ; // get loss by one sub-network 𝓕ₙ
7           /* Opportunistic data skipping                  */
8           for x in X do
9              if l[x] > θ then l[x].bp(); sampler.adjust(x, 0);
10             else sampler.adjust(x, 1);
11          /* Momentum-based filtering threshold            */
12          if θ==−1 then θ ← l; else θ ← mθ + (1 − m)l;
13          /* Synchronizing trainable blocks                */
14          for i in {0, 1, · · · , N} do 𝓕ᵢ.trainable ← 𝓕ₙ.trainable;
15    return 𝓕₀;
16 Function Policy_sampler(): // called once per epoch
17    P ← Ω, D; // a list, routing strategy n = P[index of x]
18    Function adjust(x, ε); // adjust x's routing strategy by ε
19    Q ← [0 : len(D)]; // a list storing sampling order of D
20    Q.sort().shuffle(); // sort by policy, shuffle in same policy
21    return sampler; // for train_loader to sample mini-batch
```

78] and matrix multiplication optimization [36], we decouple the perception of data difficulty to the on-cloud offline phase. Specifically, we design a data-driven black-box tiny model, referred as policy model Ω. The model is offline trained and performs online inference. The design and training details are shown in §3.3.1. As a stateful attribution, data importance is still perceived online. We use the current forward pass to get real-time data importance. At a certain iteration, the importance comes from the current network state by reusing the training loss, and the details are presented in the following Algorithm 1.

The above online-offline fused data perception incurs negligible online overhead. Most of the additional overhead comes from the online inference of the tiny policy model. For instance, on NVIDIA Jetson TX2, the per-sample perception time for ResNet50 is only 0.6 ms on average, which is less than 1% of an entire training iteration.
Data-aware subnet routing. Based on the perceived data diversity, `PieBridge` judiciously assigns training samples to proper subnets. In general, we obtain a difficulty-based routing strategy from the

policy model, and adjust this strategy with real-time data importance. This strategy is dynamic: during training, each mini-batch selects the most suitable sub-network for itself.

In detail, we demonstrate our routing strategy by describing the on-device training procedure of `PieBridge` in algorithm 1. It receives $\{\mathcal{F}_n\}$, Ω, and the on-device dataset D, and finally outputs the fine-tuned network $\mathcal{F}_0$. The routing strategy is maintained by function $Policy\_sampler()$, which samples mini-batches and ensures that samples within the same mini-batch share the same level of difficulty. After getting the output of Ω (line 17), we online adjust it by function $adjust()$ (line 18, line 8–10). Both Ω and $adjust()$ adhere to the principle of minimizing computation for easier or less important data. The main body of on-device training is in function $data\_aware\_training()$. During each iteration, we sample a mini-batch X, obtain its routing strategy n, and select a proxy sub-network $\mathcal{F}_n$ for training (line 5–6).

In line 7–12, we further filter unimportant data from back-propagation. Training data that has already well converged is all pain and no gain: it consumes computational resources while contributes little to convergence. We skip the backward pass stage of training sample x by comparing its data importance to a threshold θ. To perform stable and solid filtering, θ is dynamically updated based on momentum. Let θ_t be the threshold at iteration t, $m \in [0, 1)$ be the momentum coefficient, l be the loss of current iteration, then we get

$$\theta_{t+1} = m\theta_t + (1 - m)l. \tag{4}$$

Usually, θ is initialized as the loss from the first iteration, and m is relatively large (e.g., 0.8, our default). After filtering, we then feed the profiled importance back to routing strategy by function $adjust()$.

Asynchronous subnets loading. Notably, only *one* proxy sub-network is involved in a training iteration, so keeping all networks in device memory simultaneously is not necessary. Thereby, `PieBridge` asynchronously loads the subnet of next iteration during the current iteration. Such a loading is overhead-free in training time as computation time is much longer than loading. In doing so, the peak memory usage of our framework is nearly identical to PET, with only additional buffers for maintaining weights of a proxy sub-network of next iteration (detailed in §5.3).

3.3.1 **Learned Data Difficulty.** Here we focus on the details of how data difficulty is obtained through offline-online collaboration. As discussed before, a straightforward way to perceive the data difficulty is to run all (pre-trained) proxy networks for a data sample to compare the features (recall Equation 1). On one hand, running all proxy networks is unacceptable on devices, and the constraint to determine whether $\mathcal{F}_n(x) \simeq \mathcal{F}_0(x)$ also involves ad-hoc designs (e.g., using cosine similarity; setting threshold σ, etc.). On the other hand, as an intrinsic characteristic standalone to training state, intuitively the data difficulty does not have to be perceived online.
"Learned data difficulty" through a lightweight policy model. `PieBridge`'s response is an end-to-end learning for data difficulty with implicitly determining $\mathcal{F}_n(x) \simeq \mathcal{F}_0(x)$ by the learning loss. We design a lightweight policy model Ω to end-to-end learn the data difficulty. The input of the policy model is a training sample x' downsampled from x (i.e., resizing the input using interpolation),
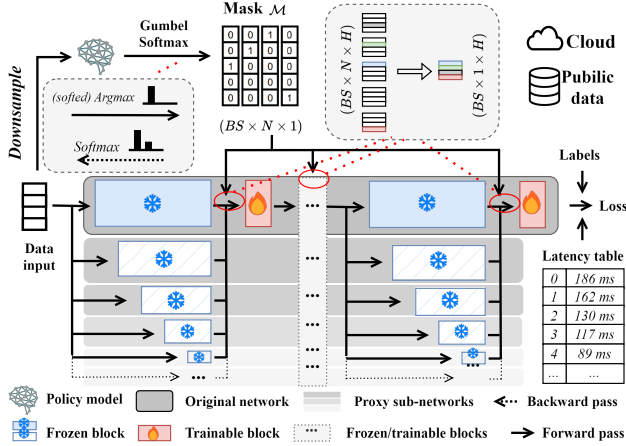
Figure 9: Learning difficulty through joint training.

and the output is a one-hot vector $\Omega(x') \in \mathbb{R}^{1 \times (N+1)}$. Ω is tiny because we only use a small fraction of the original network $\mathcal{F}_0$. For instance, we use 2 residual blocks for ResNet50, 2 basic blocks with 2 depthwise separable convolutions each for MobileNetV3, or 1 encoder layer for ViT base, and each module has a smaller size compared to the original network. The total parameter count is less than 5% of the original network.[2]

Joint training the policy model. The policy model is jointly trained with the original network and the proxy sub-networks (i.e., $\{\mathcal{F}_n\}, n \in \{0, 1, 2, \cdots, N\}$) on large-scale public cloud data. The training procedure is shown in Figure 9. We combine Ω and $\{\mathcal{F}_n\}$ together to create a joint end-to-end trainable network. The network structure before the r_{th} trainable block (i.e., the red circle in Figure 9) is

$$x_r = \sum_{dim=1} [(x_0^r | x_1^r | \cdots | x_n^r | \cdots | x_N^r) \circ \mathcal{M}] \tag{5}$$

where $x_n^r \in \mathbb{R}^{BS \times 1 \times H}$ is from the n_{th} network $\mathcal{F}_n$, $\circ$ is the Hadamard product, $\mathcal{M} \in \mathbb{R}^{BS \times N \times 1}$ is the mask from the policy model, BS is the mini-batch size, N is the number of proxy sub-networks, H is the hidden state size, $(x_0^r | x_1^r | \cdots | x_n^r | \cdots | x_N^r)$ represents concatenating the tensors sequentially at the 1_{st} dimension, $\sum_{dim=1}$ represents summarizing the tensors along the 1_{st} dimension, and finally we get the input tensor x_r of the r_{th} trainable block. In addition, since the output of the policy model Ω is discrete, which makes Ω non-differentiable, we borrow the Gumbel-Softmax [56] trick[3] to facilitate the backpropagation of the gradient to the policy model.

We update the parameters of the policy model and the trainable blocks while keeping the frozen blocks frozen again. We minimize a multi-task learning loss

$$\mathcal{L} = \mathcal{L}' + \lambda \frac{1}{N} \sum_* (\mathcal{M} \circ \mathcal{S}), \tag{6}$$

---

[2]Although increasing the model size may lead to better routing performance, the benefits diminish marginally, while the online inference overhead shows no marginal effect. Therefore, we keep it small enough.

[3]An open-source implementation of Gumbel-Softmax Sampling: https://github.com/gyhui14/spottune/blob/master/gumbel_softmax.py

Name	Processor	Software env.
Jetson TX2 [12]	Dual-Core NVIDIA Denver 2 64-Bit CPU, 256-core NVIDIA Pascal™ GPU.	Ubuntu 18.04 LTS, PyTorch 1.7.1.
RPI 4B [14]	Broadcom BCM2711B0 quad-core A72 64-bit @ 1.5GHz CPU.	Raspbian 11, PyTorch 1.7.1.
MI 10 [10]	2.84GHz Cortex-X1, 3× 2.4GHz Cortex A78, 4× 1.8GHz Cortex A55 CPU.	Android 10, MNN 2.0.0, ONNX 1.13.1.
Huawei Mate 30 [7]	2x 2.86 GHz ARM Cortex-A76, 2x 2.09 GHz ARM Cortex-A76, 4x 1.86 GHz ARM Cortex-A55 CPU, Kirin 990 NPU.	

Table 3: Details of devices used in the experiments.

where λ is the scale factor (1.0 by default), $\sum_*$ represents summing all elements in the tensor, $\mathcal{M} \in \mathbb{R}^{N \times 1}$ is the mask tensor from the policy model, $\mathcal{S} \in \mathbb{R}^{N \times 1}$ is the on-device profiled latency scores of networks $\{\mathcal{F}_n\}, n \in \{0, 1, 2, \cdots, N\}$. Equation 6 forces the policy model Ω to learn a strategy that minimizes computational complexity (the second term) while ensuring accuracy (the first term).

Training data. By default, the learning of policy model is on the large-scale general dataset used in the pre-training process (e.g. ImageNet). Although the policy model is deployed on devices for new fine-tuning data, it shows a strong transferablity without the need for retraining the policy model in most cases, since a large-scale general training dataset has covered a wide range of difficulty patterns. We make a detailed evaluation and discussion on the training data distribution issue in §5.2 and §7.

Training overhead. The joint training converges within a single epoch and also only requires one-shot execution like the on-cloud pre-approximation. In general, the entire training only incurs 5%–10% GPU-hours of the vanilla pre-training process.

4 IMPLEMENTATION AND METHODOLOGY

System implementation. We have fully implemented a `PieBridge` prototype with 4k LoC in Python and 3k LoC in C/C++ atop PyTorch and MNN [11]. All the techniques of `PieBridge` can work in other ML frameworks designed for different devices and runtime environments like TensorFlow [15] and TFLite [16]. At the cloud side, we use a GPU server with 4× NVIDIA A40. At the edge side, we use devices described in Table 3.

Models, datasets and metrics. We evaluate `PieBridge` with the following representative models: MobileNetV3-L [51], ResNet50 [47] and ViT_base [33]. We on-cloud pre-train these models with the ImageNet2012 [9] dataset that contains 1.2M images, and then on-device fine-tune them with the downstream datasets (Caltech-101 [1], Caltech-256 [2], Dogs vs. Cats [4], DTD [3]) which are more fine-grained. We mainly focus on the following metrics: *the wall-clock on-device training time* and *the best test accuracy*.

Baselines. We compare `PieBridge` to these alternatives: Fine-tuning all parameters (`FT-All`) always fine-tunes the whole pre-trained NN. This is the default fine-tuning methodology used in most pre-training-and-fine-tuning paradigms. Parameter-Efficient Training (PET) freezes most layers of the pre-trained NN and trains only a small portion. We evaluate various PET methods: Linear Probing [61] for ResNet50, Last-3 [42, 90] for MobileNetV3 and
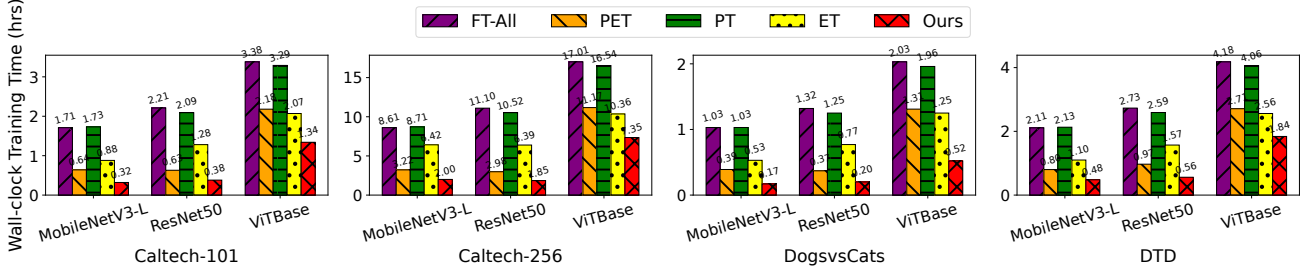
Figure 10: Wall-clock on-device training time. Device: Jetson TX2.

	MobileNetV3-L					ResNet50					ViT_base				
	FT-All	PET	PT	ET	*Ours*	FT-All	PET	PT	ET	*Ours*	FT-All	PET	PT	ET	*Ours*
C-101	87.66	84.27	87.76	89.20	*89.82*	91.96	91.13	91.23	91.09	*92.03*	94.53	96.00	93.98	95.64	*96.12*
C-256	72.00	72.50	73.08	71.72	*73.51*	83.93	82.38	82.94	80.81	*82.89*	85.61	84.93	81.39	81.83	*84.91*
DVC	95.35	95.50	95.15	95.20	*94.60*	95.55	95.25	95.25	95.55	*95.50*	95.40	95.96	95.05	95.50	*95.00*
DTD	53.19	50.27	50.48	50.69	*52.87*	60.37	62.93	60.48	60.16	*62.23*	65.05	66.12	62.55	66.54	*67.71*

Table 4: Accuracy of `PieBridge` and baselines. C-101: Caltech-101; C-256: Caltech-256; DVC: Dogs vs. Cats.
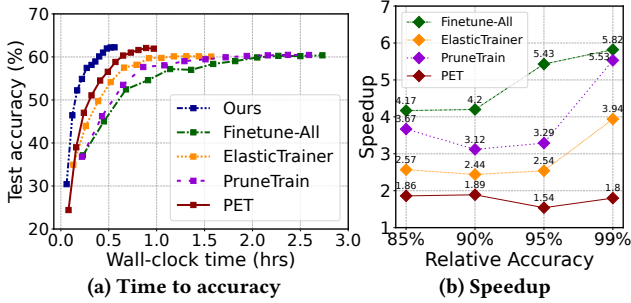

(a) Time to accuracy   (b) Speedup

Figure 11: `PieBridge` achieves significant speedup across different accuracy targets. Model: ResNet50; Dataset: DTD.

Adapters(12) [50, 80] for ViT_base. PruneTrain (PT) [73] is a framework that dynamically prunes the NN during training. ElasticTrainer (ET) [55] is one of the state-of-the-art on-device training frameworks that dynamically determines trainable layers by estimating the importance of NN tensors.

Hyper-parameters. The momentum coefficient m of PruneTrain is 0.8. The L1 penalty coefficient of PruneTrain is set to 1e-4. The ratio ρ of ElasticTrainer is set to 0.4. To provide big-enough search space, we on-cloud generate $N = 10$ proxy sub-networks with a pruning step size $d = 7\%–10\%$. Limited by the memory constraint of edge devices, the mini-batch size is set to 4. Training epochs are set to 12 following prior work [55]. Training samples are resized to a 224×224 RGB image with standard data augmentation strategies such as random cropping and flipping.

5 EVALUATION

5.1 End-to-End Performance

Overall performance. We begin with comparing the overall performace of `PieBridge` to its baselines. We conduct the experiments on 3 various NN architectures and 4 various downstream datasets. The details are mentioned in §4. We record the test accuracy every epoch and report the best test accuracy and corresponding

wall-clock training time. The computing cost on testing is excluded. Table 4 shows the best test accuracy. `PieBridge` does not compromise test accuracy compared to its baselines. As is shown in Figure 10, by dynamically routing training data to lightweight subnets, `PieBridge` achieves up to 6.60× speedup over all baselines, and 1.47×–2.52× speedup compared to PET. Here we perform a detailed analysis:

- *Compared to* FT-All. `PieBridge` significantly outperforms FT-All by up to 6×. This comes from both the parameter efficiency and our key designs.
- *Compared to* PET. `PieBridge` consistently outperforms PET by up to 2.5×. This is attributed to the full utilization of data diversity.
- *Compared to other strong baselines (*PT/ET*).* `PieBridge` also outperforms these strong baselines. PT's NN approximation only reduces training FLOPs, which does not translate to wall-clock time reduction on general-purpose processors. ET only performs tensor-level non-deterministic training with evaluated importance. It cannot fully utilize the potential of NN approximation and data diversity. Another significant drawback is that they compromise parameter efficiency, preventing them from enjoying various benefits of PET in terms of both training and deployment mentioned in §2.1.

Performance under various accuracy budgets. In practice, on-device training tasks often do not choose to reach the highest test accuracy because of the marginal effect. Thus, we also explore the wall-clock training time for different accuracy targets. Figure 11a shows the time-to-accuracy curve of `PieBridge` and its baselines. We observe that, thanks to our dynamic routing design, `PieBridge` achieves shorter per-epoch computation time and faster end-to-end convergence speed. Further, we report the speedup achieved by `PieBridge` when reaching 85%/90%/95%/99% of the baseline's best accuracy. As is shown in Figure 11b, `PieBridge` always achieves significant speedup (1.86×/1.89×/1.54×/1.80× to PET, and 4.17×/4.20×/5.43×/5.82× to FT-All) under different accuracy budgets. Specifically, there is a trend that the higher the

	FT-All			PieBridge		
	5-shot	10-shot	PL	5-shot	10-shot	PL
Time (min.)	7.92	16.38	138.21	3.60	5.97	35.53
Accu. (%)	12.92	27.28	55.32	18.13	32.71	60.15

Table 5: Performance in extremely few-shot scenarios. PL: Pseudo Labeling; Model: ResNet50; Dataset: DTD Device: Jetson TX2.
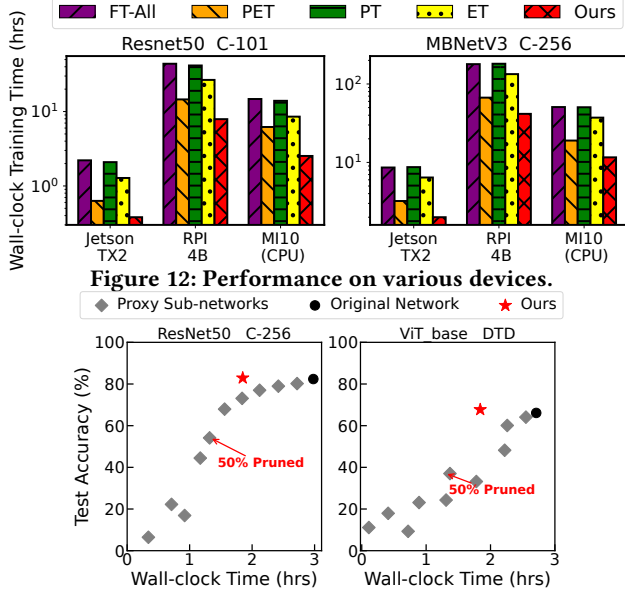
Figure 12: Performance on various devices.

Figure 13: Compared to static alternative proxy sub-networks. Our data-aware subnet routing is a pareto-improved solution.

target accuracy, the greater the speedup achieved by PieBridge. This is because, as training progresses, more and more data require fewer computational resources, allowing PieBridge to effectively mitigate the marginal effect of training.

Performance on various devices. We further explore the performance of PieBridge on 3 devices: Jetson TX2, Raspberry Pi 4B and MI 10 smartphone in Figure 12. PieBridge outperforms its baselines on all these devices by a minimum factor of 1.61×/1.63×/1.72×, and the improvements are particularly notable on both GPU-empowered devices (e.g., Jetson TX2) and resource-constrained devices (e.g., Raspberry Pi 4B and MI 10 smartphone).

Performance in extremely few-shot scenarios. We also explore the performance of PieBridge with extremely few human-labeled training data in Table 5. Our system consistently outperforms the baseline method under 5-shot/10-shot settings. We further apply a commonly used few-shot learning approach: pseudo-labeling [62]. We provide 100 pseudo-labeled data for each category. PieBridge significantly outperforms the baseline and achieves accuracy close to training with adequate human-labeled data.

5.2 Design Effectiveness and Sensitivity Analysis

In this section, our primary focus is on analyzing the effectiveness and sensitivity of PieBridge design.

Compared to static alternative proxy sub-networks. We compare our data-aware subnet routing strategy (§3.3) to statically
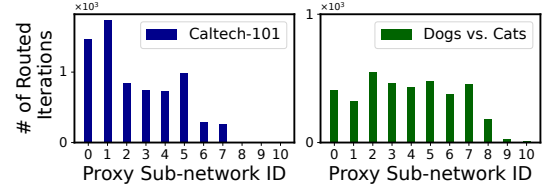
Figure 14: Behavior on datasets of different complexity.

routing all training samples to the same proxy sub-network. Figure 13 shows the best test accuracy and wall-clock training time of the original network (●), the proxy sub-networks (♦) and our data-aware subnet routing (★). We have the following observations:

(1) *A simple, static replacement strategy could result in nontrivial degradation of convergence accuracy.* After pruning the original network by 50%, ResNet50 experiences a 28.24% decrease in test accuracy on the Caltech-256 dataset, and ViT_base suffers a 29.11% decrease in test accuracy on the DTD dataset. There are two reasons behind this. One is that there is an error between the original network and the approximated subnet. The other is that, without a resource-intensive search, the pruned network structure may not be optimal.

(2) *Our data-aware subnet routing is a pareto-improved solution.* Compared to other static solutions, our approach not only reduces training time but also does not exhibit a noticeable decrease in accuracy.

Behavior on datasets of different complexity. As is shown in Figure 14, compared to the more complex dataset Caltech-101, PieBridge effectively routes more training samples to lighter subnets on Dogs vs. Cats dataset.

Number of proxy sub-networks. Recall that we globally choose a proxy number $N = 10$ (the pruning step size d is about $1/N$). Here we analysis the rationale and influence of choosing this factor. As shown in Figure 15(a), a larger N provides more fine-grained approximation tradeoffs, but its benefit diminishes when large enough (e.g., >=10). Determining it properly isn't a challenge: PieBridge can provide heuristic suggestions to a typical range (e.g., 10–15); or the cloud can automatically obtain it through profiling on public dataset.

From our observation, the number of subnets does not apparently influence wall-clock training time (Figure 15(b), ResNet50+Caltech-256+TX2). This is mainly because the size of the subnets already covers the entire trade-off space, regardless of the number N.

Policy model training data. Recall that we train our policy model Ω on large scale public dataset, i.e. ImageNet. Through the aforementioned end-to-end experiments, its effectiveness has been verified. Here we discuss PieBridge's sensitivity to the data for capturing data difficulty. We report fine-tuning ResNet50 on Caltech-101 dataset. The data difficulty learned from the large-scale general dataset ImageNet exhibits a strong *transferability*. Figure 15(c) shows that training on ImageNet achieves comparable accuracy compared to directly training on Caltech-101. When training policy model with 5% data (simulating crowd-sourcing user-permitted training data), the policy model still can work well. Only when the cloud training data completely mismatches with device data (e.g.,

	PieBridge	W/O IDC pruning	W/O latent space alignment	W/O policy model based routing
Training Time (h)	1.85	N/A (shape mismatch)	2.06	26.2
Test Acc.(%)	82.89	N/A (shape mismatch)	64.24	82.62

Table 6: Significance of key designs.

(a) of proxy subnets v.s. Acc.**(b)** # of proxy subnets v.s. time**(c)** Data to train policy model
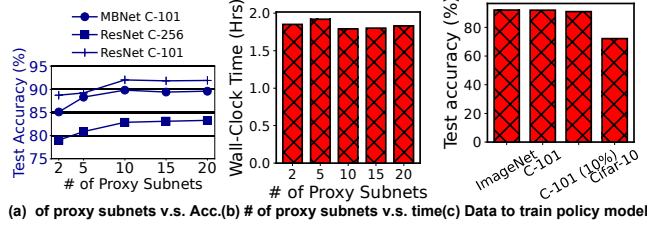
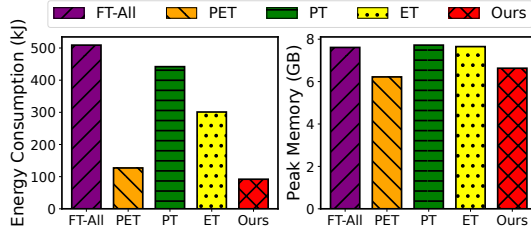**Figure 15: Sensitivity analysis.**

Figure 16: Energy and memory consumption.

using cifar-10 for training in Figure 15(c)), the policy model behaves biased.

Significance of key designs. In Table 6, we show the significance of PieBridge's key designs. We report fine-tuning ResNet50 on Caltech-256, Jetson TX2. The results show that without IDC pruning, the proxy subnets cannot replace the original network because of shape mismatch. Without latent space alignment, the pruned subnets conflict with each other, leading to a significantly dropped convergence accuracy. Without the lightweight policy model for perceiving the data difficulty, the training is slowed down by over 13×, because that PieBridge has to run each subnet to get and compare the extracted feature in each iteration.

5.3 Energy and Memory Consumption

In this section, we report the energy and memory consumption of PieBridge against its baselines. We train ResNet50 on Caltech-101 dataset with Jetson TX2. We obtain energy/memory consumption by jtop [19]. The results are shown in Figure 16.

Energy consumption. PieBridge reduces energy consumption by 5.53×/1.28× compared to FT-All and PET. This is attributed to PieBridge's time efficiency, since PieBridge achieves better time-to-accuracy. However, compared to PieBridge's training time reduction, the energy reduction is slightly lower. This is mainly due to PieBridge periodically loads proxy sub-networks from disk (§3.3), which is also energy consuming.

Memory consumption. We have the following observations. (1) Both PET and PieBridge consumes fewer memory than FT-All/PT/ET. The measured reduction is less than theoretical. This is mainly due to pytorch framework loads some runtime packages to memory, and does not prioritize memory saving operations such as in-place
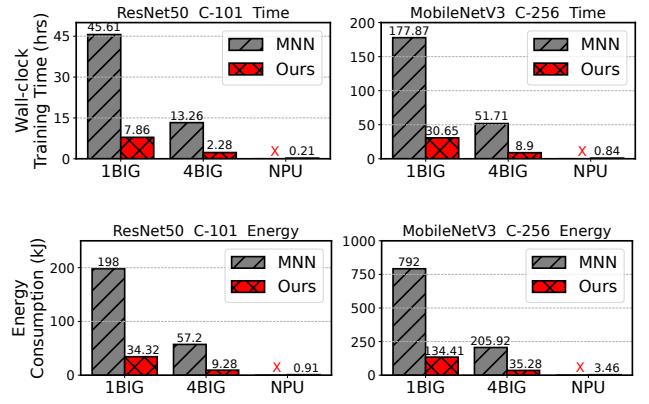

Figure 17: Performance on COTS devices with different processors compared to MNN. "1BIG", "4BIG" and "NPU" are run on Huawei Mate30 with 1 CPU big core, 4 CPU big cores and NPU, respectively. "X" means not supported.

computation and buffer-reuse. (2) PieBridge consumes more memory than vanilla PET. This is because PieBridge uses additional memory for swapping.

5.4 NPU Acceleration

Commercial off-the-shelf mobile devices typically integrate various processors into their SoCs for different purposes, such as CPU, GPU, DSP, ISP, NPU, etc.[4] While NPUs offer high-precision (FP32/FP16) AI computing ability, their operators and software stacks are closed-source and only inference-friendly (e.g., only high-level non-configurable ABIs available for Kirin NPUs [6]).

Fortunately, PieBridge's design is NPU-friendly, as its frozen subnets can reuse NPU operators and software stacks for acceleration, while trainable blocks that require weight updates can still be computed on the CPU. This is a "syntactic sugar" at the implementation level, completely orthogonal to the core design of PieBridge.

We run PieBridge on a HuaiWei Mate30 smartphone with Kirin 990 half-precision (FP16) NPU, and compare its performance to a popular on-device training framework MNN [11]. The default fine-tuning strategy of MNN is full model fine-tuning with 4 CPU big cores. As is shown in Figure 17, with the assistance of key techniques in PieBridge and NPU offloading, on-device fine-tuning becomes practical and even efficient: a medium-sized fine-tuning workload (ResNet50, Caltech-101) requires only 0.21 hours and 0.91 kJ of energy, whereas MNN takes 13.26 hours and 57.2 kJ of energy. On much larger datasets (Caltech-256), PieBridge also requires only 0.84 hours and 3.46 kJ of energy.

6 RELATED WORK

Time-efficient on-device training. A considerable amount of literature/frameworks have investigated how to enable and accelerate training on resource-constrained edge devices [11, 16, 17, 55, 73, 92, 102, 103, 106]. For instance, ElasticTrainer [55] reduces

---

[4]Mobile GPUs have been proven unsuitable for DNN training [32], and DSPs, due to their low-precision (INT8/INT4) computing characteristics, can only be used for training from scratch instead of fine-tuning pre-trained weights [92].

training time by non-deterministically training a portion of the network. Mandheling [92] directly downsamples the entire network and maps it to the hardware accelerator. There are also many open-source frameworks tailored for on-device training, such as TFLite [16] from Google and MNN [11] from Alibaba. `PieBridge` is compatible with the above frameworks and hardware accelerator oriented works. The difference between `PieBridge` and other works, such as ElasticTrainer and PruneTrain, which also employ dynamic neural network architectures, lies in that `PieBridge` does not compromise on parameter efficiency.

Neural network compression/approximation. Neural network compression/approximation refers to approximating the original network with simpler and lighter networks, which can come from network structure approximation (e.g., structural pruning [18, 60, 69, 70] or sparse weights [84]) or data precision approximation (e.g., INT8/INT4 quantization [37, 101]). Different from the aforementioned works, `PieBridge`'s NN approximation is dynamic for each data point, and is accurate enough for lossless training.

Foundation models as a mobile system service. Compared to traditional on-device AI paradigm that individually deploys a compact model for each task [28, 28, 34, 72, 95, 97, 98], recent advances in LLMs [24, 26, 39, 77, 105] and multimodal [63–65, 79, 82, 85] suggest that a highly parameterized OS-level foundational model [96, 99] with stronger generalization and emergent capabilities is more suitable for the evolving workloads in on-device AI. `PieBridge` sheds light on building such a one-for-all foundation ML system on devices to continuously and personally serve ubiquitous AI tasks.

Routing diverse data to multiple networks. Several prior arts [57, 66, 88] also leverage the idea of routing diverse data to multiple networks that has diverse compute cost. Appealnet [66] proposes a two-head little network for routing data to two cascaded big-little networks at inference time. The difficulty in Appealnet is defined as a loose measurement on softmax probability. Selective Query [57] trains a routing model for assigning the to-be-inferenced sample to either the cloud or the device. These work mainly focus on *inference* optimizations, which are much more simple and deterministic compared to training. `PieBridge`'s data routing identifies several unique aspects in training, including feature-level difficulty and training sample importance to convergence.

7 DISCUSSION

Data diversity. Although there are some early efforts [23, 86] that explore the diversity, their definition cannot be directly applied to PET mainly due to the following reasons. Firstly, they mainly focus on reducing the inference-time overhead by layer-level early exit for diverse samples. NN inference has a higher tolerance for extracted features. For instance, when a feature is tortured, one classification network can still output the correct answer if the highest logit is in the label class, regardless the other classes. While in training, this may lead to a severely affected convergence. Instead, `PieBridge` sets stringent restriction for training by a feature-level matching in its difficulty definition. Secondly, early exiting is essentially a pruning method with sequential layer-level heuristic, which is much more coarse-grained than `PieBridge`'s IDC pruning that prunes neurons in independent dimensions.

Handling diversity distribution drift. One practical issue that `PieBridge` faces in real-world deployment is the data diversity distribution drift. As `PieBridge` focuses on on-device fine-tuning, the distribution of on-device data may differ from on-cloud pre-training data. To this end, `PieBridge` has the following conclusions and alternatives. Firstly, as elaborated in §5.2 and §3.3.1, the pattern of data difficulty is much more simple and deterministic than the distribution on concrete tasks, thus showing strong transferablity. For instance, an NN trained on ImageNet still needs fine-tuning on caltech-101 dataset, while the difficulty pattern that captured by `PieBridge`'s policy model on ImageNet has been nearly equivalent to caltech-101. Besides, `PieBridge` has several alternatives when the fine-tuning data is completely different from pre-training. One can crowd-source a portion of user-permitted trainable data to the cloud, federated learn the policy model, or directly on-device continuously train the policy model.

Other approximation techniques. Sparsity and quantization is not a good fit for `PieBridge`, as they can only provide a limited wall-clock speedup on mobile SoC processors. For instance, to the best of our knowledge, the Raspberry Pi has no dedicated compute units for quantized data formats like int4. Thereby, a highly quantized network only achieves an insignificant speedup on these devices since most of its operators are compute-bound. Sparsity is also not well-supported on mainstream mobile SoCs. Early exiting is essentially a coarse-grained pruning, thus also sub-optimal compared to pruning. `PieBridge` employs structural pruning, which is the most suitable technique direction for PET acceleration on mobile devices.

Applicability to language models. Currently, `PieBridge` is mainly designed for on-device tuning CV models. The LLMs are typically much heavier than CV models (e.g., llama-7B), and their fine-tuning on extremely resource-constrained mobile devices is still not a practical solution for demands like personalization. Nevertheless, the main techniques of `PieBridge` are generally applicable to LLMs, with minor adjustments like supporting LoRA structures and profiling token-level data diversity. We leave this as our future work.

8 CONCLUSION

This work has proposed `PieBridge`, a framework for time-efficient on-device parameter efficient training. It innovatively leverages data diversity and neural network approximations for reducing the computation cost of frozen layers. `PieBridge` achieves up to 2.5× on-device fine-tuning speedup compared to state-of-the-art PET methods, and up to 6.6× speedup compared to traditional full model fine-tuning. `PieBridge` makes on-device fine-tuning more practical for COTS devices equipped with NPUs.

9 ACKNOWLEDGEMENT

REFERENCES

[1] 2023. Caltech-101. https://www.tensorflow.org/datasets/catalog/caltech101.
[2] 2023. Caltech-256. https://paperswithcode.com/dataset/caltech-256.
[3] 2023. Describable Textures Dataset. https://www.robots.ox.ac.uk/~vgg/data/dtd/.
[4] 2023. Dogs vs. Cats. https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset.
[5] 2023. Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant. https://machinelearning.apple.com/research/hey-siri.
[6] 2023. Hi-AI Foundation. https://developer.huawei.com/consumer/en/doc/development/hiai-Guides/introduction-0000001051486804.
[7] 2023. HuaWei Mate30. https://consumer.huawei.com/ph/phones/mate30/specs/.
[8] 2023. ImageNet-21K. https://arxiv.org/abs/2104.10972.
[9] 2023. ImageNet2012. https://www.image-net.org/challenges/LSVRC/2012/.
[10] 2023. MI 10. https://www.mi.com/in/mi-10/.
[11] 2023. MNN. https://github.com/alibaba/MNN.
[12] 2023. Nvidia jetson tx2. https://developer.nvidia.com/embedded/jetson-tx2.
[13] 2023. Once-For-All. https://github.com/mit-han-lab/once-for-all.
[14] 2023. Raspberry Pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
[15] 2023. TensorFlow. https://www.tensorflow.org.
[16] 2023. TensorFlow Lite. https://www.tensorflow.org/lite?hl=zh-cn.
[17] 2023. TensorFlow.js. https://github.com/tensorflow/tfjs.
[18] 2023. Torch-Pruning. https://github.com/VainF/Torch-Pruning.
[19] 2024. jtop. https://github.com/rbonghi/jetson_stats.
[20] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. 2020. Knowledge Distillation from Internal Representations. arXiv:1910.03723 [cs.CL] https://arxiv.org/abs/1910.03723
[21] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. 2020. Knowledge Distillation from Internal Representations. arXiv:1910.03723 [cs.CL]
[22] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In Advances in Neural Information Processing Systems, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6c340f25839e6acdc73414517203f5f0-Paper.pdf
[23] Robert J. N. Baldock, Hartmut Maennel, and Behnam Neyshabur. 2021. Deep Learning Through the Lens of Example Difficulty. arXiv:2106.09647 [cs.LG] https://arxiv.org/abs/2106.09647
[24] BELLEGroup. 2023. BELLE: Be Everyone's Large Language model Engine. https://github.com/LianjiaTech/BELLE.
[25] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous learning of video analytics models on edge compute servers. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). 119–135.
[26] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]
[27] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 11285–11297. https://proceedings.neurips.cc/paper_files/paper/2020/file/81f7acabd411274fcf65ce2070ed568a-Paper.pdf
[28] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In International Conference on Learning Representations. https://arxiv.org/pdf/1812.00332.pdf
[29] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. arXiv:1905.13536 [cs.CV]
[30] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Return of the devil in the details: Delving deep into convolutional nets. arXiv preprint arXiv:1405.3531 (2014).
[31] Tiffany Yu-Han Chen, Hari Balakrishnan, Lenin Ravindranath, and Paramvir Bahl. 2016. GLIMPSE: Continuous, Real-Time Object Recognition on Mobile Devices. GetMobile: Mobile Comp. and Comm. 20, 1 (jul 2016), 26–29. https://doi.org/10.1145/2972413.2972423
[32] Anish Das, Young D. Kwon, Jagmohan Chauhan, and Cecilia Mascolo. 2022. Enabling On-Device Smartphone GPU based Training: Lessons Learned. arXiv:2202.10100 [cs.LG]

[33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In International Conference on Learning Representations. https://openreview.net/forum?id=YicbFdNTTy
[34] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. ACM. https://doi.org/10.1145/3241539.3241559
[35] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. DepGraph: Towards Any Structural Pruning. arXiv:2301.12900 [cs.AI] https://arxiv.org/abs/2301.12900
[36] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 610, 7930 (2022), 47–53.
[37] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG]
[38] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. 2022. An image is worth one word: Personalizing text-to-image generation using textual inversion. arXiv preprint arXiv:2208.01618 (2022).
[39] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A Dialogue Model for Academic Research. Blog post. https://bair.berkeley.edu/blog/2023/04/03/koala/
[40] Negar Goli and Tor M. Aamodt. 2020. ReSprop: Reuse Sparsified Backpropagation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).
[41] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. 2018. FoggyCache: Cross-Device Approximate Computation Reuse. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (New Delhi, India) (MobiCom '18). Association for Computing Machinery, New York, NY, USA, 19–34. https://doi.org/10.1145/3241539.3241557
[42] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. 2019. Spottune: transfer learning through adaptive fine-tuning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 4805–4814.
[43] Jialiang Han, Yun Ma, Qiaozhu Mei, and Xuanzhe Liu. 2021. Deeprec: On-device deep learning for privacy-preserving sequential recommendation in mobile commerce. In Proceedings of the Web Conference 2021. 900–911.
[44] Yikun Han, Chunjiang Liu, and Pengfei Wang. 2023. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. arXiv:2310.11703 [cs.DB] https://arxiv.org/abs/2310.11703
[45] Jingrui He. 2017. Learning from Data Heterogeneity: Algorithms and Applications. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. 5126–5130. https://doi.org/10.24963/ijcai.2017/735
[46] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. arXiv:1911.05722 [cs.CV]
[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]
[48] Zhuoxun He, Lingxi Xie, Xin Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. 2019. Data Augmentation Revisited: Rethinking the Distribution Gap between Clean and Augmented Data. arXiv:1909.09148 [cs.LG]
[49] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML] https://arxiv.org/abs/1503.02531
[50] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In International Conference on Machine Learning. PMLR, 2790–2799.
[51] 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. 2019. Searching for MobileNetV3. arXiv:1905.02244 [cs.CV]
[52] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]
[53] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In International Conference on Learning Representations. https://openreview.net/forum?id=nZeVKeeFYf9
[54] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Multi-Scale Dense Networks for Resource Efficient Image Classification. arXiv:1703.09844 [cs.LG]
[55] Kai Huang, Boyuan Yang, and Wei Gao. 2023. ElasticTrainer: Speeding Up On-Device Training with Runtime Elastic Tensor Selection. In Proceedings of

the 21st Annual International Conference on Mobile Systems, Applications and Services. 56–69.

[56] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In International Conference on Learning Representations. https://openreview.net/forum?id=rkE3y85ee

[57] Anil Kag, Igor Fedorov, Aditya Gangrade, Paul Whatmough, and Venkatesh Saligrama. 2023. Efficient Edge Inference by Selective Query. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=jpR98ZdIm2q

[58] Angelos Katharopoulos and François Fleuret. 2017. Biased importance sampling for deep neural network training. arXiv preprint arXiv:1706.00043 (2017).

[59] Simon Kornblith, Jonathon Shlens, and Quoc V Le. 2019. Do better imagenet models transfer better?. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2661–2671.

[60] Gokul Krishnan, Xiaocong Du, and Yu Cao. 2019. Structural Pruning in Deep Neural Networks: A Small-World Approach. arXiv:1911.04453 [cs.LG]

[61] Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. 2022. Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution. In International Conference on Learning Representations. https://openreview.net/forum?id=UYneFzXSJWh

[62] Dong-Hyun Lee et al. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In Workshop on challenges in representation learning, ICML, Vol. 3. Atlanta, 896.

[63] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. arXiv:2301.12597 [cs.CV]

[64] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. arXiv:2201.12086 [cs.CV]

[65] Junnan Li, Ramprasaath R. Selvaraju, Akhilesh Deepak Gotmare, Shafiq Joty, Caiming Xiong, and Steven Hoi. 2021. Align before Fuse: Vision and Language Representation Learning with Momentum Distillation. arXiv:2107.07651 [cs.CV]

[66] Min Li, Yu Li, Ye Tian, Li Jiang, and Qiang Xu. 2021. AppealNet: An Efficient and Highly-Accurate Edge/Cloud Collaborative Architecture for DNN Inference. arXiv:2105.04104 [cs.LG] https://arxiv.org/abs/2105.04104

[67] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190 (2021).

[68] Yutong Lin, Ze Liu, Zheng Zhang, Han Hu, Nanning Zheng, Stephen Lin, and Yue Cao. 2022. Could Giant Pre-trained Image Models Extract Universal Representations? Advances in Neural Information Processing Systems 35 (2022), 8332–8346.

[69] Sicong Liu, Bin Guo, Ke Ma, Zhiwen Yu, and Junzhao Du. 2021. AdaSpring: Context-adaptive and Runtime-evolutionary Deep Model Compression for Mobile Applications. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 5, 1, Article 24 (March 2021), 22 pages. https://doi.org/10.1145/3448125

[70] Sicong Liu, Xiaochen Li, Zimu Zhou, Bin Guo, Meng Zhang, Haocheng Shen, and Zhiwen Yu. 2023. AdaEnlight: Energy-aware Low-light Video Stream Enhancement on Mobile Devices. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 6, 4, Article 172 (Jan. 2023), 26 pages. https://doi.org/10.1145/3569464

[71] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Association for Computational Linguistics, Dublin, Ireland, 61–68. https://doi.org/10.18653/v1/2022.acl-short.8

[72] Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D. Lane, and Mengwei Xu. 2024. Small Language Models: Survey, Measurements, and Insights. arXiv:2409.15790 [cs.CL] https://arxiv.org/abs/2409.15790

[73] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. 2019. PruneTrain: Fast Neural Network Training by Dynamic Sparse Model Reconfiguration. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (Denver, Colorado) (SC '19). Association for Computing Machinery, New York, NY, USA, Article 36, 13 pages. https://doi.org/10.1145/3295500.3356156

[74] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. arXiv:2305.11627 [cs.CL] https://arxiv.org/abs/2305.11627

[75] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making learned query optimization practical. In Proceedings of the 2021 International Conference on Management of Data. 1275–1288.

[76] Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. 2022. Prioritized Training on Points that are Learnable, Worth Learning, and not yet Learnt. In Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162), Kamalika Chaudhuri,

[77] Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 15630–15649. https://proceedings.mlr.press/v162/mindermann22a.html

[78] Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. 2023. Crosslingual Generalization through Multitask Finetuning. arXiv:2211.01786 [cs.CL]

[78] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc Friedman, and Alekh Jindal. 2021. Steering query optimizers: A practical take on big data workloads. In Proceedings of the 2021 International Conference on Management of Data. 2557–2569.

[79] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV]

[80] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. Advances in neural information processing systems 30 (2017).

[81] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. 2020. Do adversarially robust imagenet models transfer better? Advances in Neural Information Processing Systems 33 (2020), 3533–3545.

[82] Yixuan Su, Tian Lan, Huayang Li, Jialu Xu, Yan Wang, and Deng Cai. 2023. PandaGPT: One Model To Instruction-Follow Them All. arXiv:2305.16355 [cs.CL]

[83] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. 2017. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. In Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70), Doina Precup and Yee Whye Teh (Eds.). PMLR, 3299–3308. https://proceedings.mlr.press/v70/sun17c.html

[84] Haotian Tang*, Shang Yang*, Zhijian Liu, Ke Hong, Zhongming Yu, Xiuyu Li, Guohao Dai, Yu Wang, and Song Han. 2023. TorchSparse++: Efficient Training and Inference Framework for Sparse Convolution on GPUs. In IEEE/ACM International Symposium on Microarchitecture (MICRO).

[85] Zineng Tang, Ziyi Yang, Chenguang Zhu, Michael Zeng, and Mohit Bansal. 2023. Any-to-Any Generation via Composable Diffusion. arXiv:2305.11846 [cs.CV]

[86] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. arXiv:1709.01686 [cs.NE] https://arxiv.org/abs/1709.01686

[87] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. 2022. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services. 450–463.

[88] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An Efficient Multi-Level Inference System for Large Language Models. In Proceedings of the Eighteenth European Conference on Computer Systems (Rome, Italy) (EuroSys '23). Association for Computing Machinery, New York, NY, USA, 233–248. https://doi.org/10.1145/3552326.3587438

[89] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. 2023. AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments. In Proceedings of the 29th Annual International Conference on Mobile Computing and Networking. 1–17.

[90] Paul Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Venkataramanaiah, Jae-sun Seo, and Matthew Mattina. 2019. FixyNN: Energy-Efficient Real-Time Mobile Computer Vision Hardware Acceleration via Transfer Learning. In Proceedings of Machine Learning and Systems, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 107–119. https://proceedings.mlsys.org/paper_files/paper/2019/file/e09d45e14e9ece7142217550ddd3c4d0-Paper.pdf

[91] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, 2246–2251. https://doi.org/10.18653/v1/2020.acl-main.204

[92] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Guang Huang, Xin Jin, and Xuanzhe Liu. 2022. Mandheling: Mixed-Precision On-Device DNN Training with DSP Offloading. arXiv:2206.07509 [cs.NI]

[93] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2022. Mandheling: Mixed-Precision On-Device DNN Training with DSP Offloading. In Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (Sydney, NSW, Australia) (MobiCom '22). Association for Computing Machinery, New York, NY, USA, 214–227. https://doi.org/10.1145/3495243.3560545

[94] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. 2018. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2, 4 (2018), 1–26.

[95] Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, Qiyang Zhang, Zhenyan Lu, Li Zhang, Shangguang Wang, Yuanchun Li, Yunxin Liu, Xin Jin, and Xuanzhe Liu. 2024. A Survey of Resource-efficient LLM and Multimodal Foundation Models. arXiv:2401.08092 [cs.LG] https://arxiv.org/abs/2401.08092

[96] Bufang Yang, Lixing He, Neiwen Ling, Zhenyu Yan, Guoliang Xing, Xian Shuai, Xiaozhe Ren, and Xin Jiang. 2023. EdgeFM: Leveraging Foundation Model for Open-set Learning on the Edge. (2023).

[97] Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. 2024. LLM as a System Service on Mobile Devices. arXiv:2403.11805 [cs.OS] https://arxiv.org/abs/2403.11805

[98] Wangsong Yin, Rongjie Yi, Daliang Xu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. 2024. ELMS: Elasticized Large Language Models On Mobile Devices. arXiv:2409.09071 [cs.DC] https://arxiv.org/abs/2409.09071

[99] Jinliang Yuan, Chen Yang, Dongqi Cai, Shihe Wang, Xin Yuan, Zeling Zhang, Xiang Li, Dingge Zhang, Hanzi Mei, Xianqing Jia, Shangguang Wang, and Mengwei Xu. 2023. Rethinking Mobile AI Ecosystem in the LLM Era. arXiv:2308.14363 [cs.AI]

[100] Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. 2022. InFi: End-to-End Learnable Input Filter for Resource-Efficient Mobile-Centric Inference. In Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (Sydney, NSW, Australia) (MobiCom '22). Association for Computing Machinery, New York, NY, USA, 228–241. https://doi.org/10.1145/3495243.3517016

[101] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8Bit BERT. In 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS). IEEE. https://doi.org/10.1109/emc2-nips53020.2019.00016

[102] Xiao Zeng, Ming Yan, and Mi Zhang. 2021. Mercury: Efficient On-Device Distributed DNN Training via Stochastic Importance Sampling. In ACM Conference on Embedded Networked Sensor Systems (SenSys).

[103] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroidLite: A Release-and-Inhibit Control Approach to Resource-Efficient Deep Neural Networks on Mobile Devices. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems (Virtual Event, Japan) (SenSys '20). Association for Computing Machinery, New York, NY, USA, 463–475. https://doi.org/10.1145/3384419.3430716

[104] Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. 2023. No Privacy Left Outside: On the (In-) Security of TEE-Shielded DNN Partition for On-Device ML. In 2024 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, 52–52.

[105] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910 [cs.LG]

[106] Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Chen Wang, Wei-Ming Chen, and Song Han. 2023. PockEngine: Sparse and Efficient Fine-tuning in a Pocket. In IEEE/ACM International Symposium on Microarchitecture (MICRO).