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

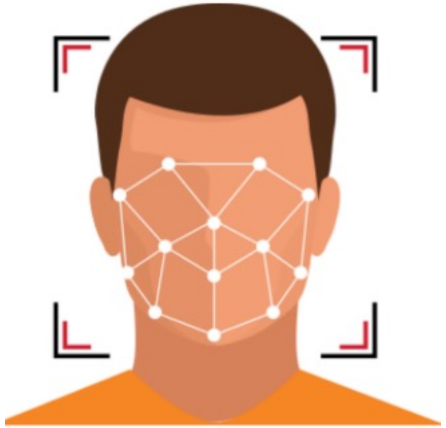
Wangsong Yin[1], Daliang Xu[1], Gang Huang[1], Ying Zhang[1], Shiyun Wei[2], Mengwei Xu[3], Xuanzhe Liu[1]

[1]Peking University, [2]Zhongguancun Laboratory,

[3]Beijing University of Posts and Telecommunications

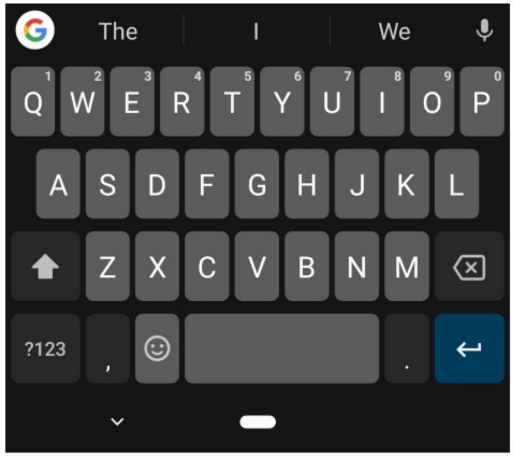
ACM SenSys 2024

On-Device Training NNs



Face detection

Voiceprint verification



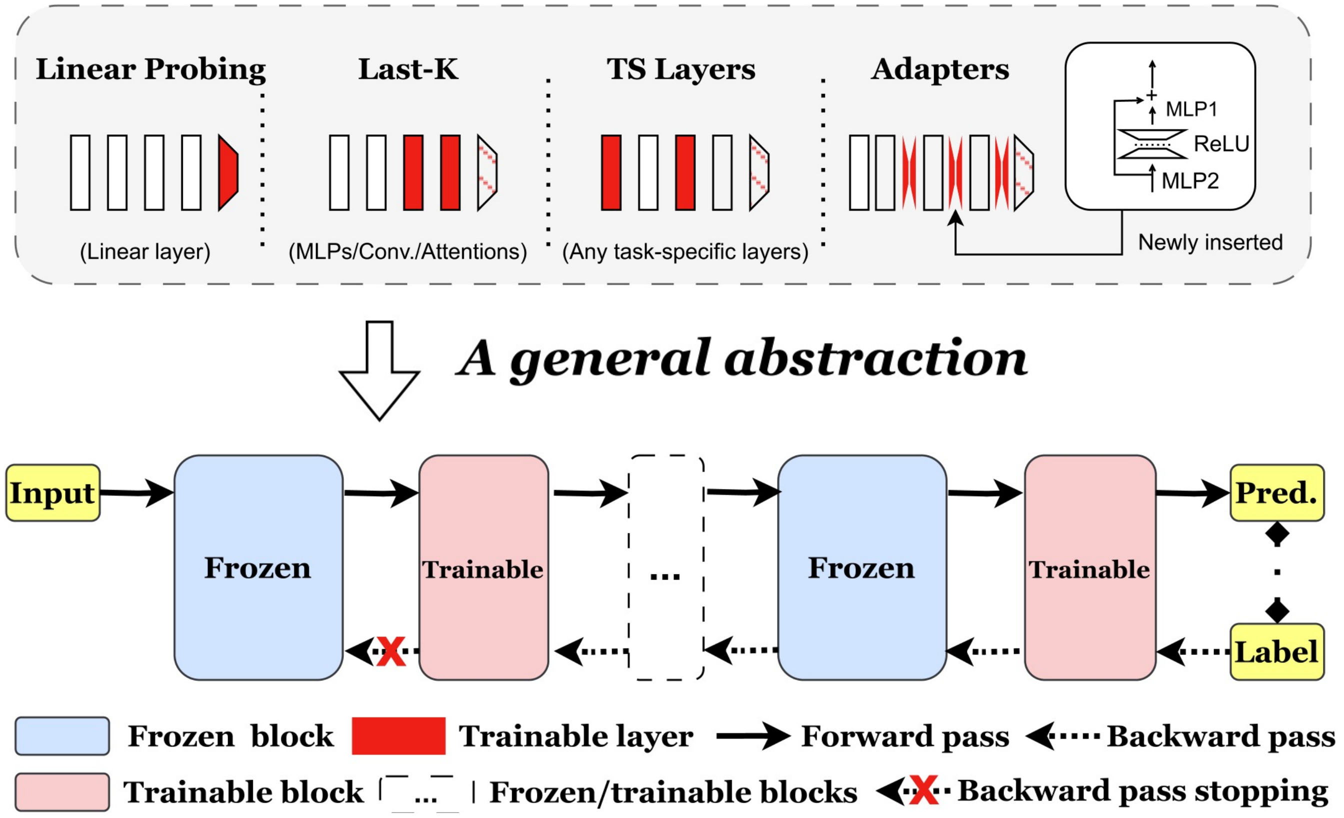
Input habit prediction

Training (fine-tuning) NNs on devices maximizes the protection of privacy data!

Popular Paradigm: Parameter-Efficient Training

Parameter-Efficient Training (PET)

Achieving on-par accuracy with training all parameters by only tuning a tiny portion of parameters



(Typically static) trainable/frozen blocks

Popular Paradigm: Parameter-Efficient Training

Parameter-Efficient Training (PET)

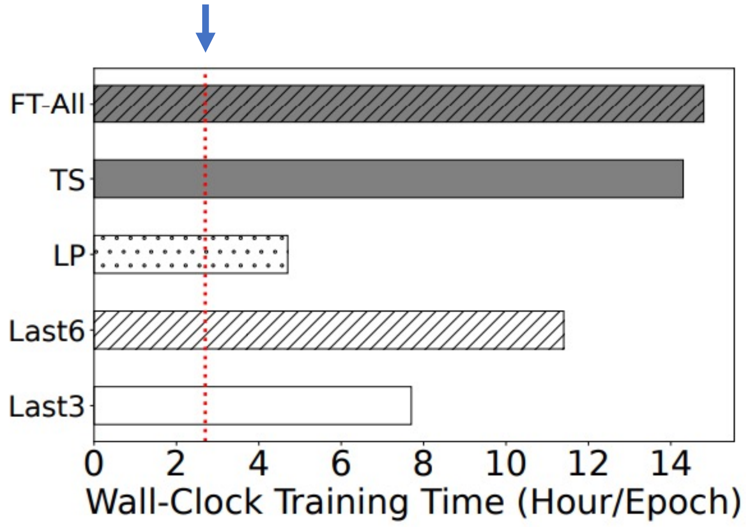
Achieving on-par accuracy with training all parameters by only tuning a tiny portion of parameters

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%

Trainable parameters are very few! (e.g. usually <1%)

Key Problem: Param. Efficiency ≠ Time Efficiency

Daily idle time on mobile devices[1]



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

Time breakdown in a PET iteration

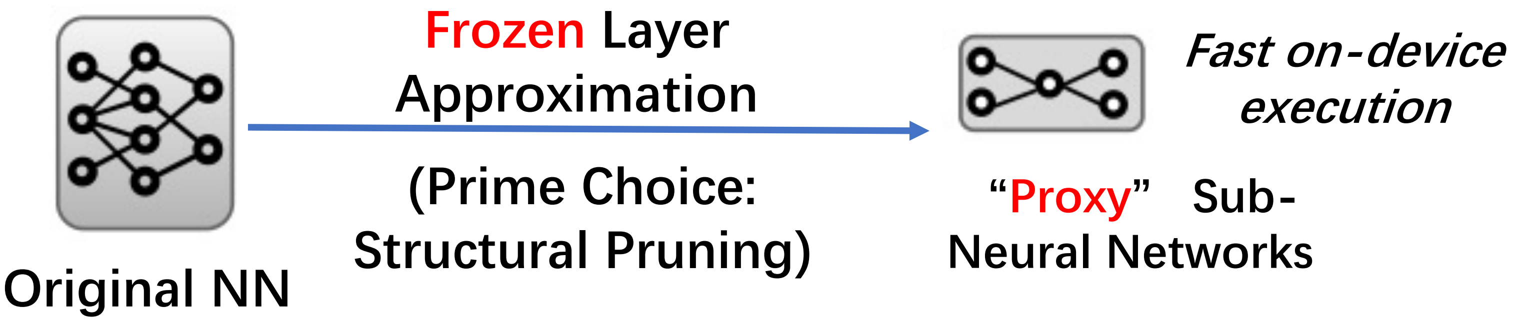
Training time is still too long!

Time consumption:
Frozen >> trainable blocks!

[1] Mengwei Xu, etc. 2018. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. Proceedings of the ACM on IMWUT (2018), 1–26

Frozen Layer Approximation During Training

Can we *approximate* the frozen blocks for a faster on-device PET?

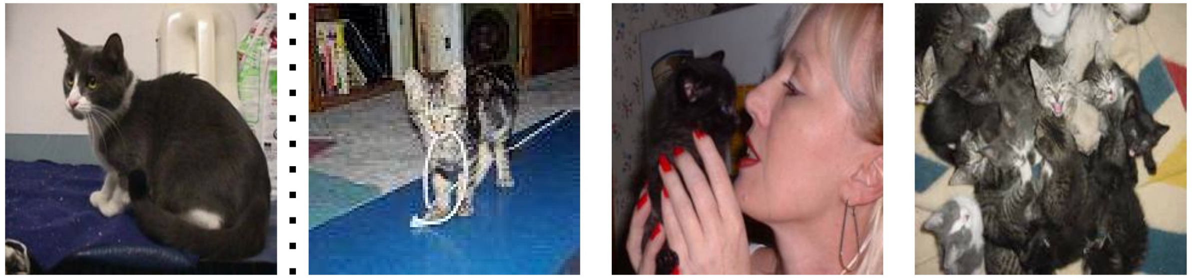


Convergence accuracy drops significantly！

Training all samples by a proxy sub-NN with 50% retained compute leads to an accuracy drop of up to 29.11%!

Opportunity: Training-Data Diversity

Difficulty



Clean *Blurry* *Obstructed* *Complicated*

Importance

British Shorthair *Tabby Cat 1* *Tabby Cat 2* *Tabby Cat 3*

- Data Difficulty
  - Samples differ in the difficulty level of extracting high quality features.
  - Measure data difficulty by *the minimal subnet that captures feature that on-par with the non-approximated network.*

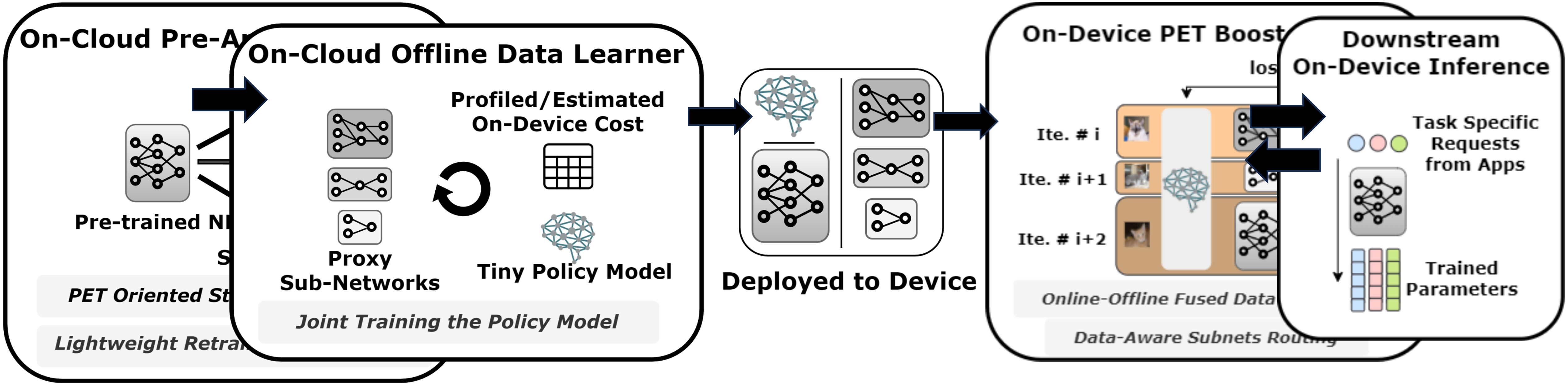
$$\phi(x) = max(n \in \{0, 1, \cdots, N\}), \quad s.t. \quad \mathcal{F}_n(x) \simeq \mathcal{F}_0(x). \tag{1}$$

- Data Importance
  - Samples differ in the contribution to model convergence.
  - Measure data importance by *loss value.*

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

Dynamically assign the most suitable approximation for diverse training data!

Our System: PieBridge



Cloud Offline Stage

Generating multi-level subnets

Learning a tiny policy model

Device Online Stage

Perceiving diff./imp. of on-device data

Routing training data to proper subnet

Stage#1: Cloud Offline

How to generate FAST and ACCURATE subnets with fine-grained tradeoffs?

INT8 Quantization

❌ Limited benefit on the same processor

Sparsification

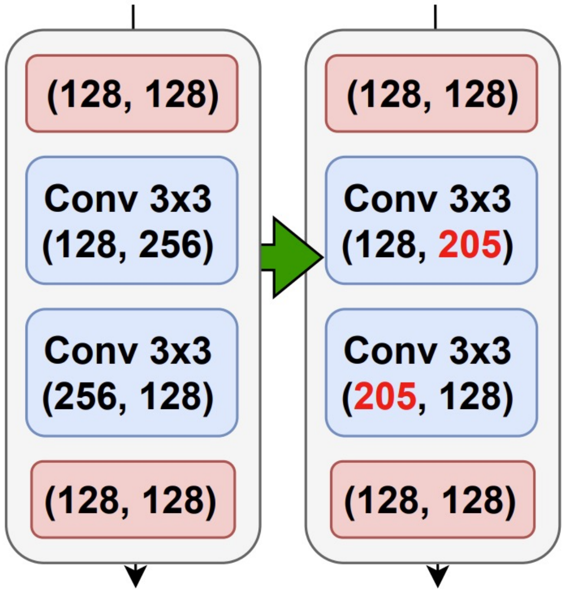
❌ Limited support from on-device NN libs

EXIT Early Exiting

❌ Too coarse granularity (a whole NN layer)

Ours: structural pruning with heuristic

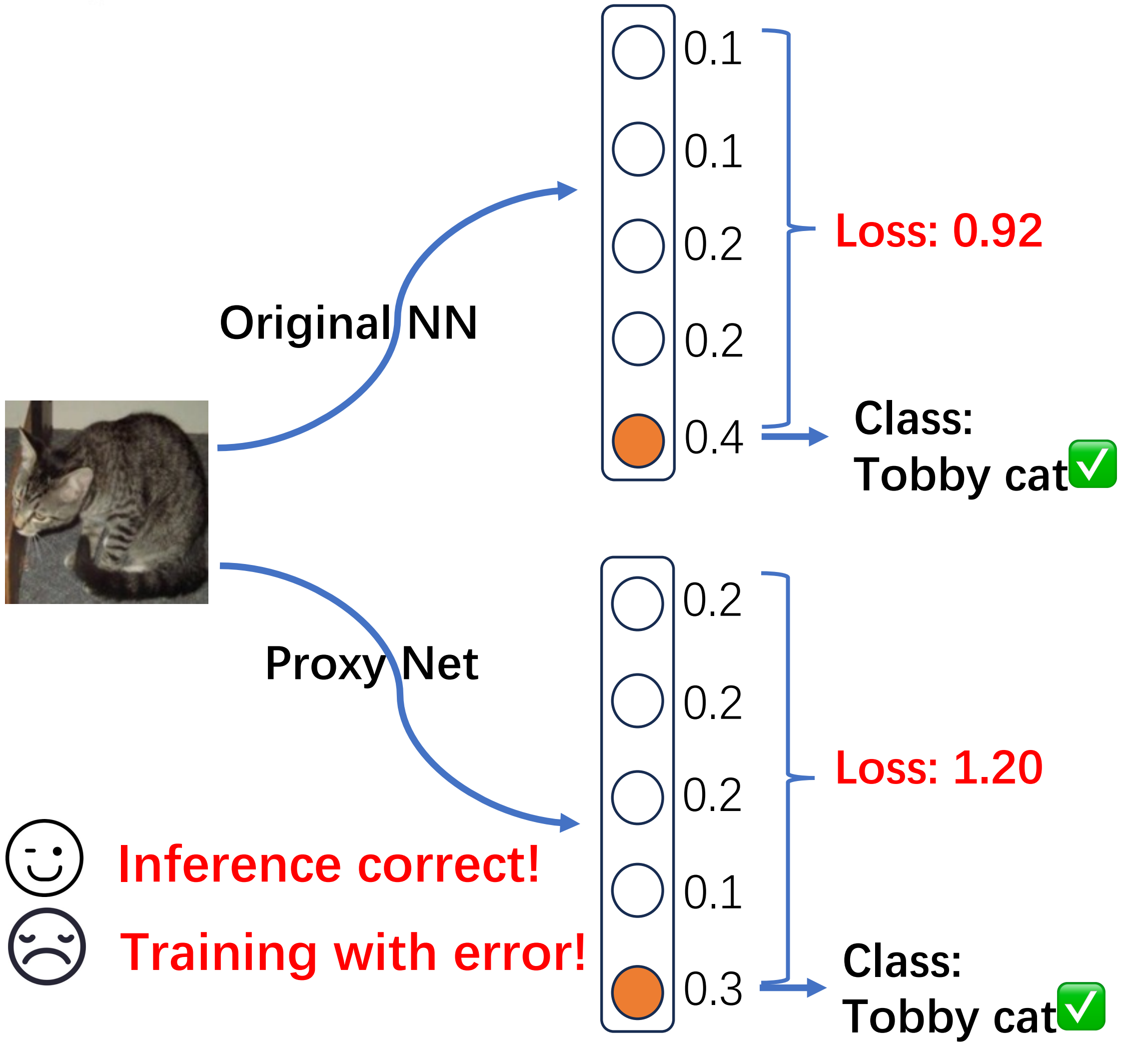
✅ E2e latency reduction ✅ Fine-grained tradeoff



- Identifying and pruning dimensions that are independent to trainable blocks
- Multiple level proxy nets for a fine-grained approximation
- Generating subnets with an even pruning step

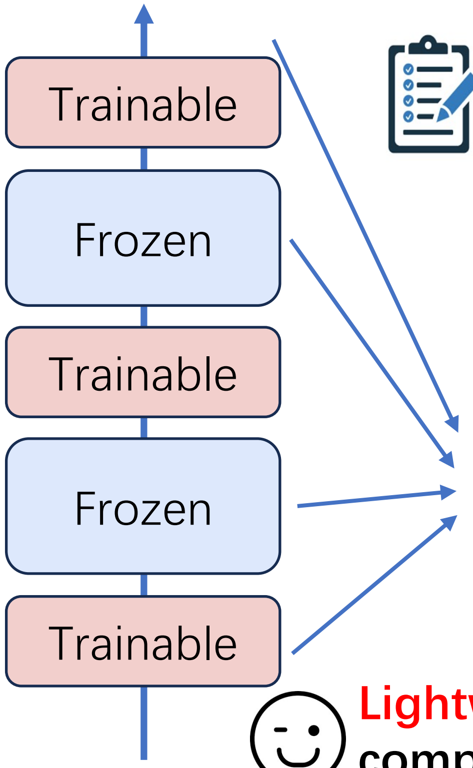
Stage#1: Cloud Offline

! Is directly generated (pruned) subnet a good approximation? Not yet!



Inference correct!

Training with error!



A retraining mechanism: adding extra constraints for generated proxy subnets

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

Lightweight: low resource consumption compared to pretraining stage

Once for all: retrain once and reuse many times/devices for a subnet

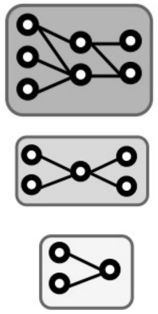
Stage#1: Cloud Offline

! Perceiving data diversity online introduces extra system level overhead for on-device training···

Needs traversing all proxy subnets

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

❌Up to 10x slower training!



What is the corresponding proxy subnet of current sample difficulty?

You have to calculate it during training to find out···

Can we offline learn data difficulty for a low overhead online prediction？

Covering usual difficulty patterns

IMAGENET

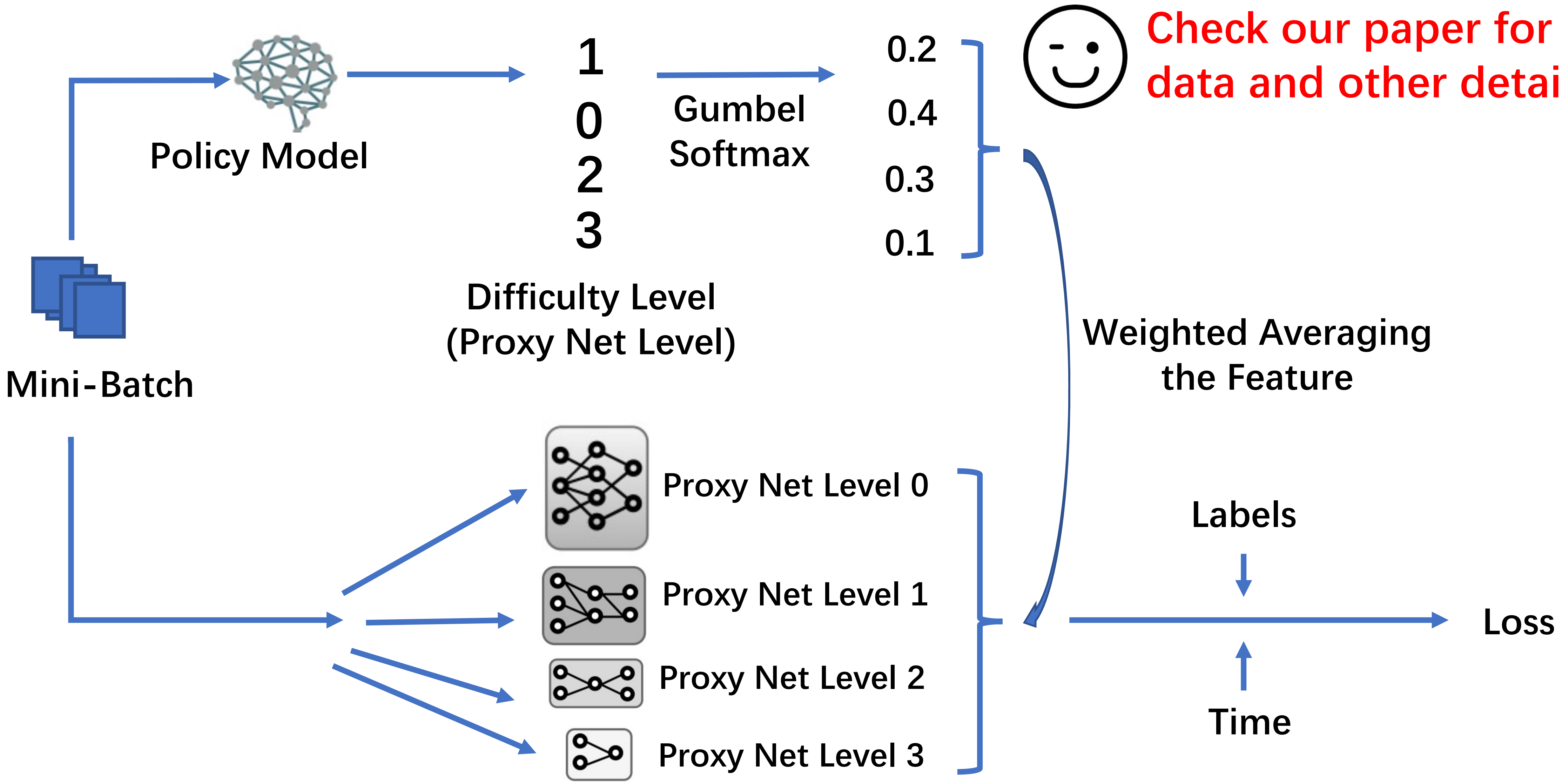
General pretraining dataset

Cloud learning

Isomorphic to proxy nets yet with much fewer parameters

Tiny policy model

Stage#1: Cloud Offline



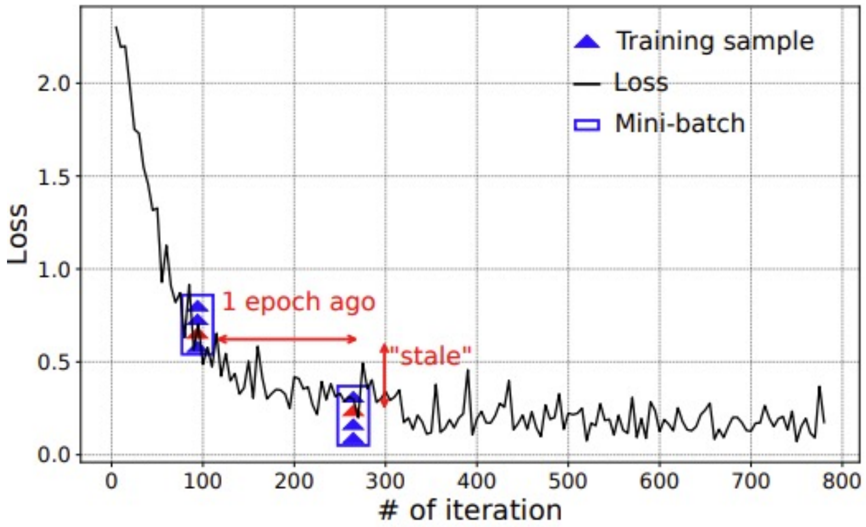
Check our paper for training data and other details!

Stage#2: Device Online

Intra mini-batch scheduling proxy nets: inefficient!

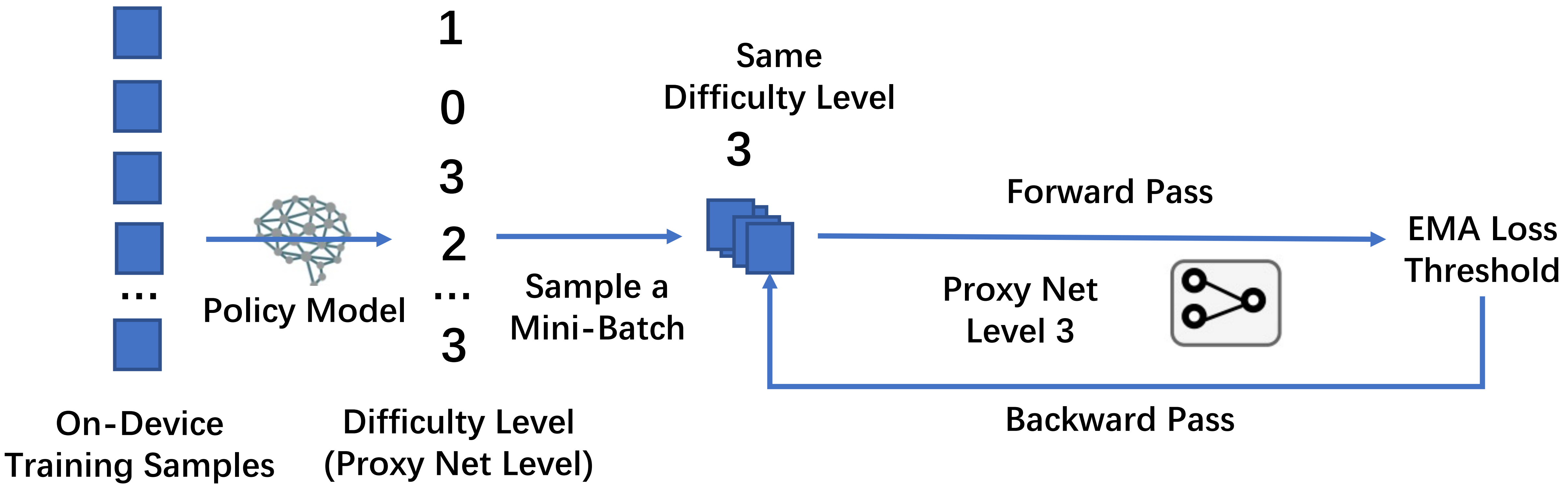
Since the mini-batch size is tiny on wimpy mobile devices (e.g. 4/8), lets schedule the proxy network assignment at mini-batch granularity!

Data importance: determined by current training state!

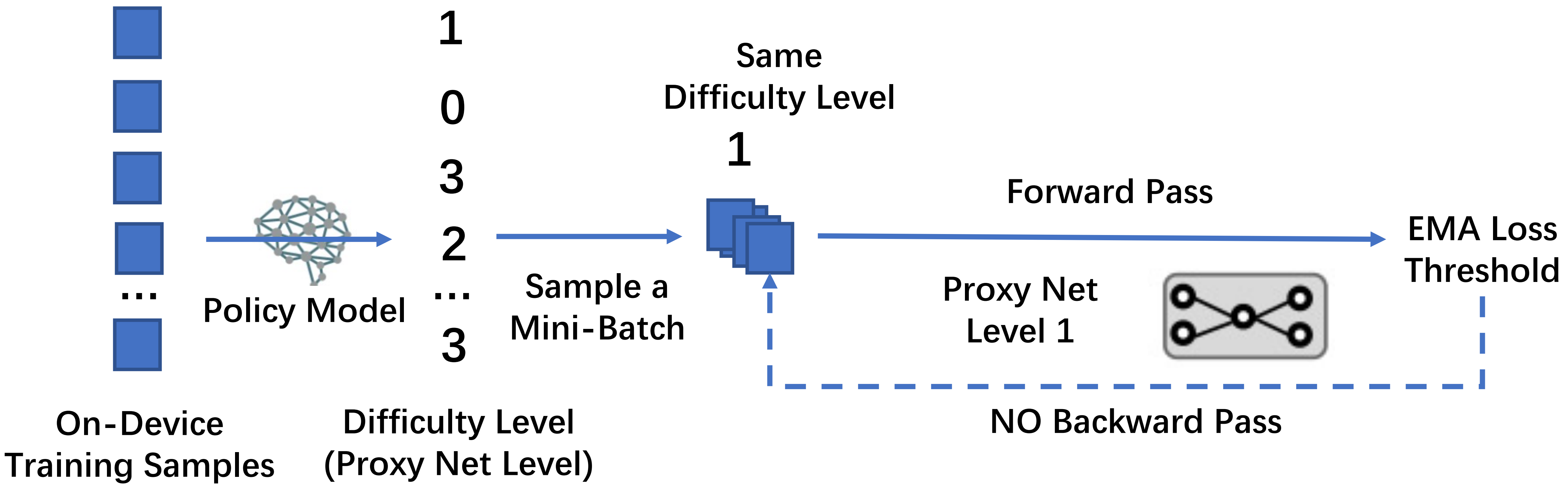


Lets harness it only for backward stage!

Stage#2: Device Online



Stage#2: Device Online



Please refer to the Algo. 1 in our paper for more details…

Implementation and Evaluation

- **Datasets**
  - **Cloud (pre)training**
    - **ImageNet2012**
  - **On-device training**
    - **Caltech101**
    - **Caltech256**
    - **DogsvsCats**
    - **DTD**
- **Models**
  - **MobileNetV3-L**
  - **ResNet50**
  - **ViT_base**

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.	

Device hardware/software

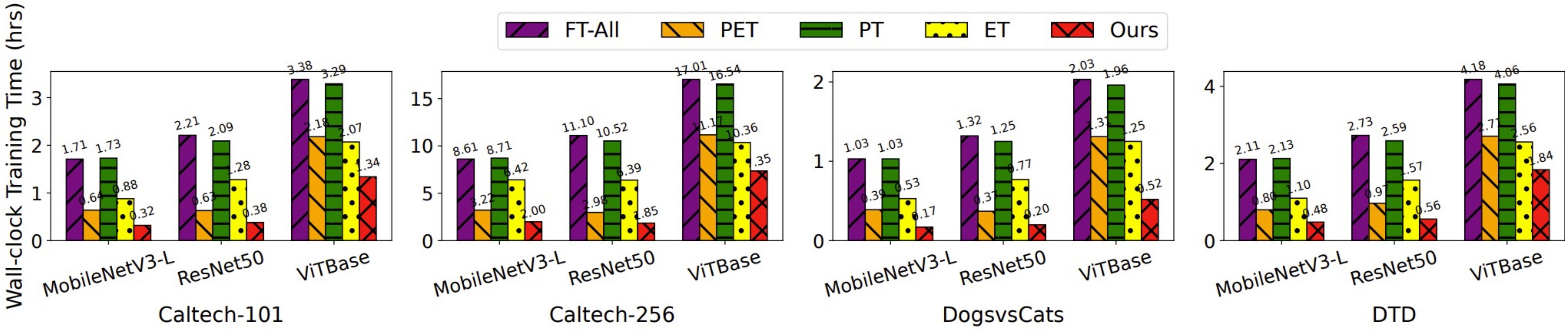
- **Baselines**
  - **Full parameter training (`FT-All`)**
  - **Parameter efficient training (PET)**
    - **ResNet50：Linear Probing**
    - **MobileNetV3：Last-3**
    - **ViT_base：Adapters(12)**
  - **PruneTrain（PT）** [1]
  - **ElasticTrainer（ET）** [2]

[1] Sangkug Lym, etc. 2019. PruneTrain: Fast Neural Network Training by Dynamic Sparse Model Reconfiguration

[2] Kai Huang, Boyuan Yang, and Wei Gao. 2023. ElasticTrainer: Speeding Up On-Device Training with Runtime Elastic Tensor Selection.

Wall-Clock Training Time

- **Wall-Clock training time on standalone datasets**



- **Compared to baselines, PieBridge accelerates on-device training by up to 6.6x**
- **Compared to vanilla PET, PieBridge accelerates on-device training by 1.4--2.5x**

Training Accuracy

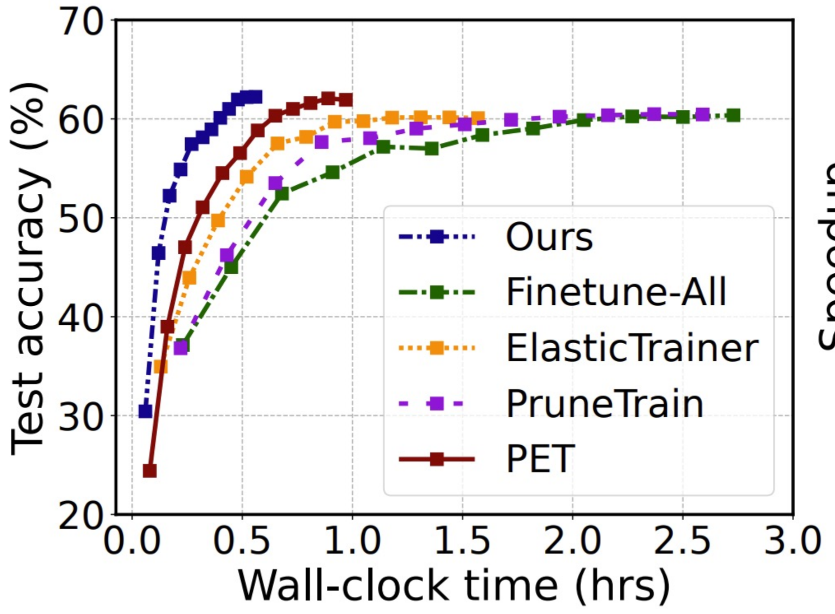
- **Best convergence accuracy on standalone datasets**

	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*

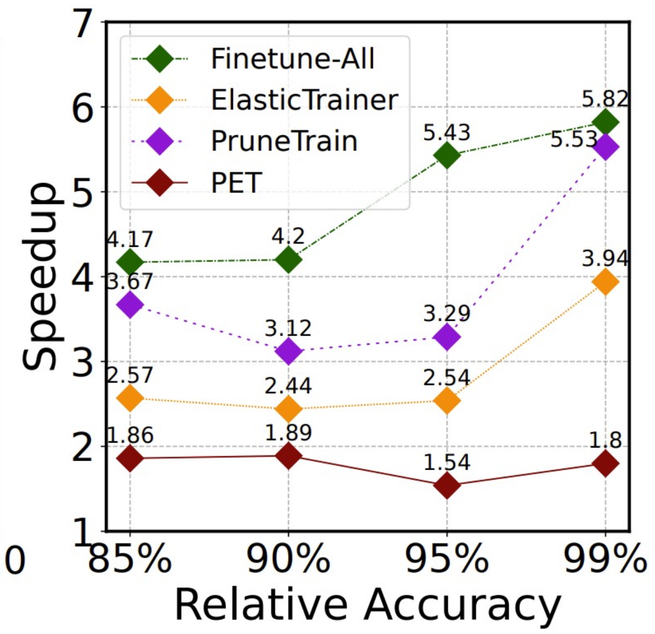
- **Compared to baselines, PieBridge has no noticeable influence on accuracy (<2%)**

Training Accuracy

- **Performance under various accuracy budgets**
  - **There exists a marginal effect of training**
  - **Under an 85%/90%/95%/99% accuracy budget**
    - **PieBridge speeds up PET by 1.86/1.89/1.54/1.80x**
    - **PieBridge speeds up FT-All by 4.17/4.20/5.43/5.82x**



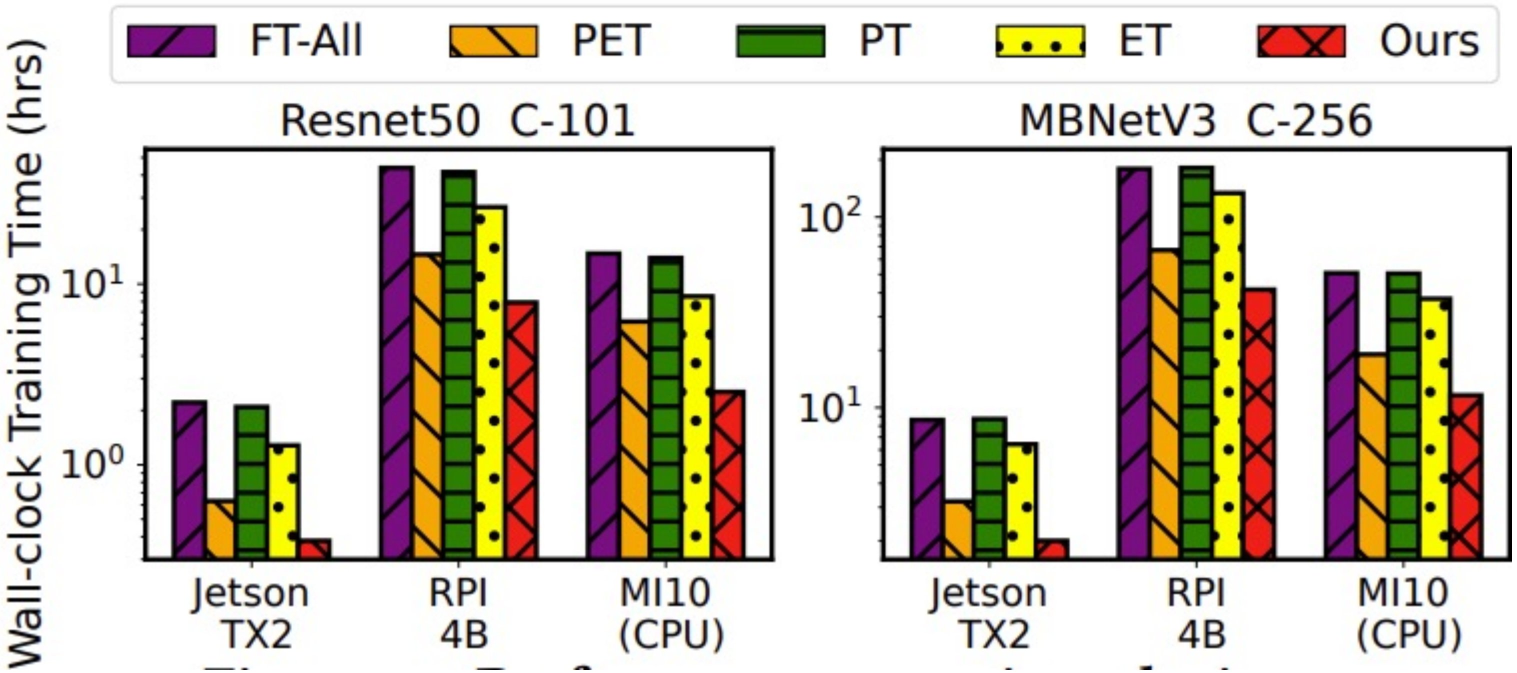
(a) Time to accuracy



(b) Speedup

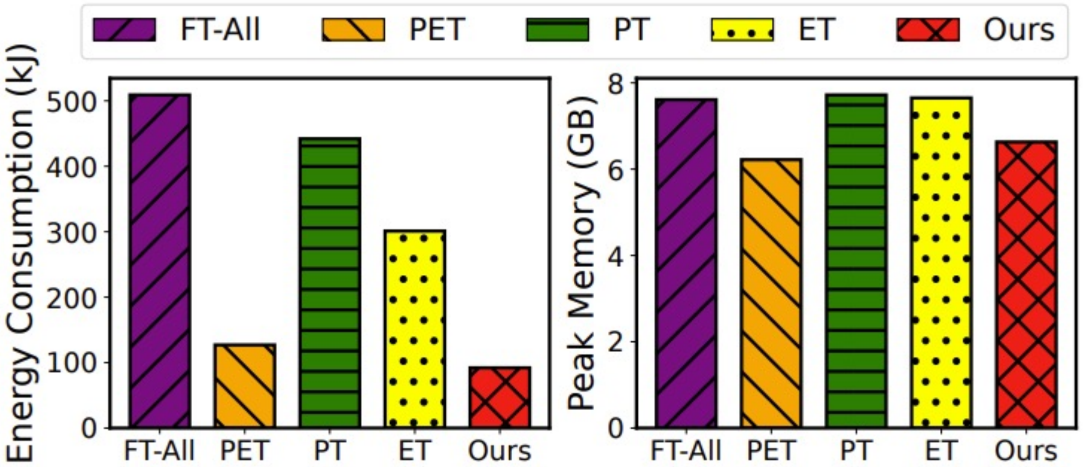
Performance on Various Devices

- **Performance on various devices**
  - **Jetson TX2，Raspberry Pi 4B，MI 10 smartphone**
  - **Speeds up PET by 1.61/1.63/1.72x**
  - **A clear performance gain on both CPU and GPU devices**



Others

- **Energy consumption**
  - 5.53x lower than FT-All
  - 1.28x lower than PET
- **Memory consumption**
  - No significant increase of memory consumption compared to baselines



- **Other sensitivity analysis**
  - # of proxy nets (pruning step)
  - Policy model training data distribution
  - Few-shot learning performance
  - …

Please refer to our paper!

Take Aways

- We present PieBridge, an on-device training framework with both time- and parameter- efficiency.
- PieBridge innovatively leverages training data diversity and proxy subnetwork approximation to reduce the overhead of frozen layers during training.
- PieBridge achieves up to 6.6x training speedup compared to strong baselines, paving the way for personalized on-device AI.

yws@stu.pku.edu.cn

Code: https://github.com/yinwangsong/PieBridge