大语言模型时代下的边缘智能系统
Edge Intelligence System in LLM Era

个人介绍

- 徐梦炜，北邮特聘副研究员/博士生导师
  - 入选中国科协青托，北京市科技新星，微软“铸星计划”学者等
  - 主要研究领域：**边缘智能系统、卫星计算系统**
  - 主页： https://xumengwei.github.io/
  - 代码： https://github.com/UbiquitousLearning

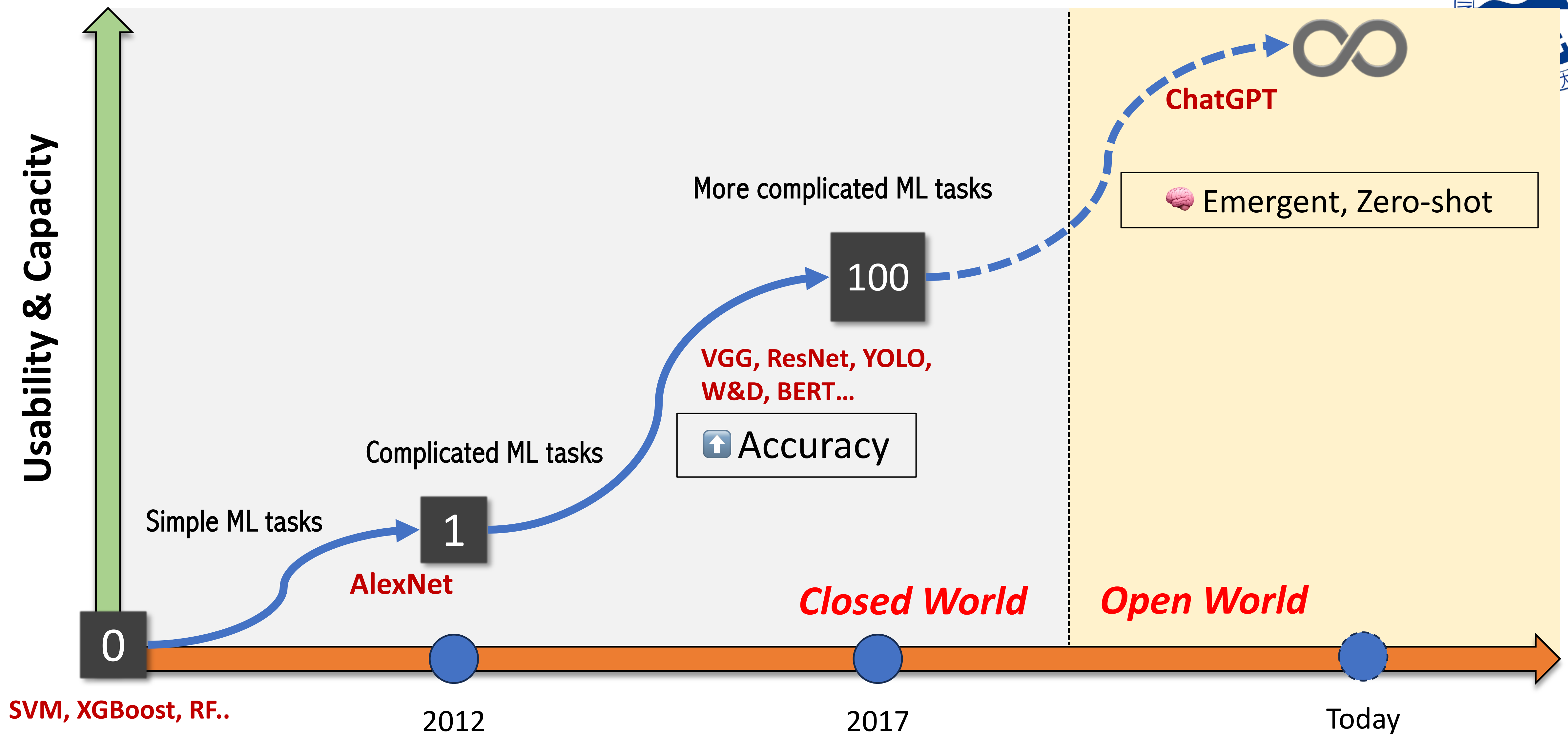
Bachelor/PhD
2011-2022

Visiting Scholar
2018-2019

Asst. Professor
2020-present

What ChatGPT means to AI..

- “ChatGPT is just a smarter chatbot”
  - As a product, yes
    - But think about it: Moss is also a chatbot; robots/humans are chat bots with physical ability
  - As a research, hell no
    - It is a generative model that theoretically knows everything on Internet and can accomplish any NLP tasks
  - It’s also
    - a series of papers cited by 10,000 times
    - a startup company worthy of 30,000,000,000 dollars.
  - It’s also the one who opens the Pandora’s box

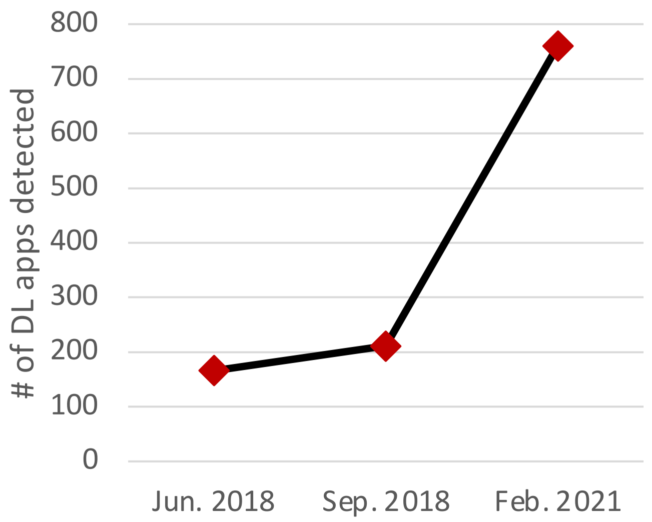


Unseen complicated ML tasks
ChatGPT
Emergent, Zero-shot
More complicated ML tasks
100
VGG, ResNet, YOLO, W&D, BERT…
Accuracy
Usability & Capacity
Complicated ML tasks
1
Simple ML tasks
AlexNet
Closed World
Open World
0
SVM, XGBoost, RF..
2012
2017
Today

Some trends

- **DNN-embedded apps are increasing rapidly**

Jun. 2018	Sep. 2018	Mar. 2021
	↑27%	↑260%
166/16,500	211/16,500	760/16,500
1.0%	1.3%	4.6%



- **DNN-embedded apps are popular apps**
  - **Contributing to billions of downloads**

[1] Mengwei Xu, et al. "A First Look at Deep Learning Apps on Smartphones" In the Web Conference (WWW) 2019

面向泛在学习的机器学习系统

中國計算機學會通訊 第 17 卷 第 10 期 2021 年 10 月

徐梦炜[1] 黄 康[2] 刘譞哲[3]
[1] 北京邮电大学
[2] 领规科技
[3] 北京大学

关键词：人工智能 终端设备 系统软件

Ubiquitous Learning

The devices can learn from the environments
at anywhere and anytime

- **Autonomous**: on-device transfer learning / personalization / …
- **Cooperative**: federated learning / split learning / …

Outline – (Federated) Training on Devices

Training on edge

- [WWW’21] FLBench — ☑ *Simulation integrity*
- [MobiCom’22] Mandheling — ☑ *Time & energy*
- [MobiSys’22] Melo — ☑ *memory*
- [INFOCOM’23] FLScheduler — ☑ *privacy*

Making on-device training (FL) practical!

Fine-tuning LLM on edge

- [MobiCom’23] FedAdapter — ☑ *Network & speed*
- [MobiCom’23] FedFSL — ☑ *Label scarcity*
- [Going-on] ForwardFL — ☑ *Time & energy & scalability & etc..*

Outline – (Federated) Training on Devices

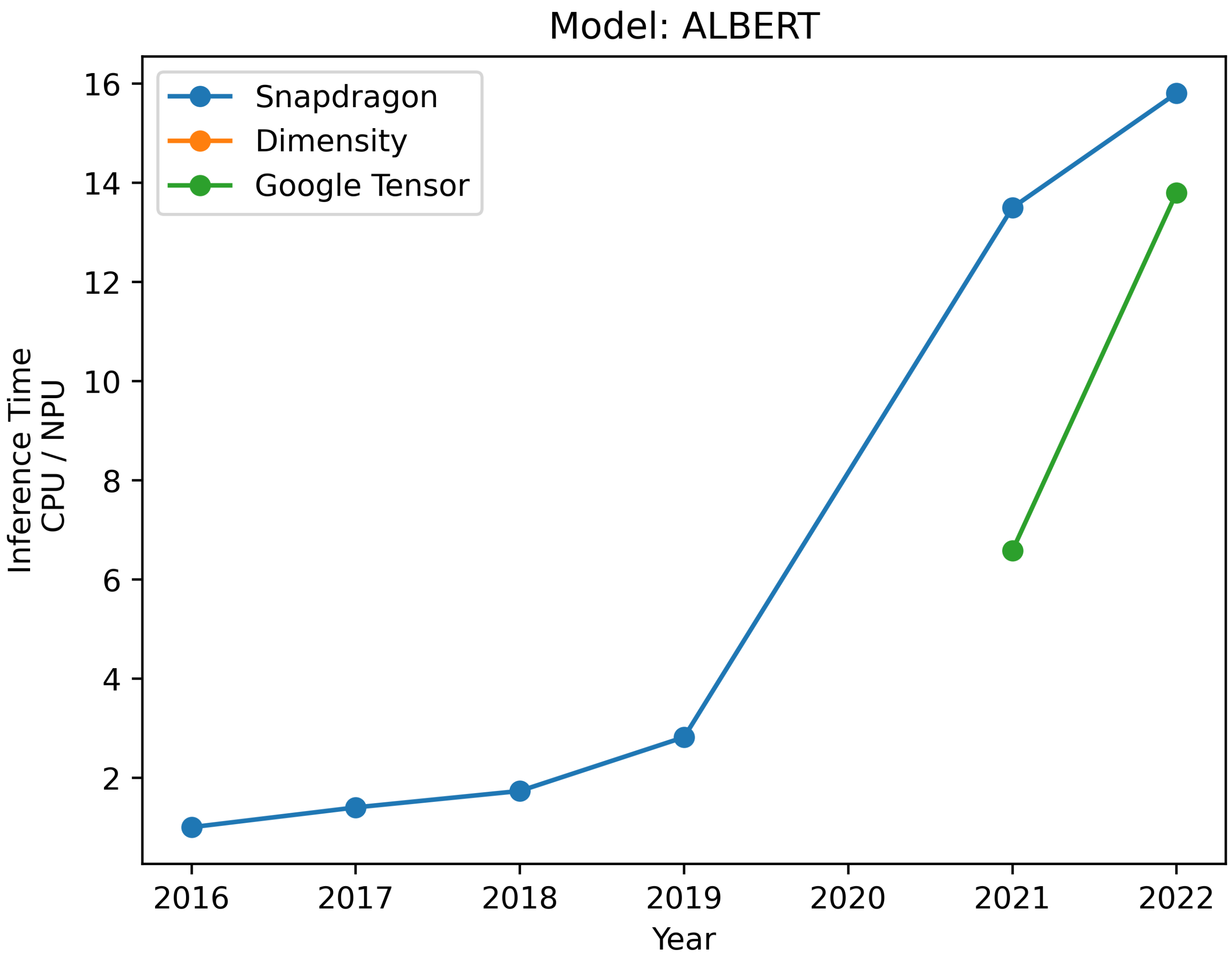
Training on edge

- [WWW'21] FLBench — ☑ *Simulation integrity*
- [MobiCom'22] Mandheling — ☑ *Time & energy*
- [MobiSys'22] Melo — ☑ *memory*
- [INFOCOM'23] FLScheduler — ☑ *privacy*

Fine-tuning LLM on edge

- [MobiCom'23] FedAdapter — ☑ *Network & speed*
- [MobiCom'23] FedFSL — ☑ *Label scarcity*
- [Going-on] ForwardFL — ☑ *Time & energy & scalability & etc..*

Motivation



- Mobile NPUs are increasingly powerful
  - More than 10x speedup over mobile CPU

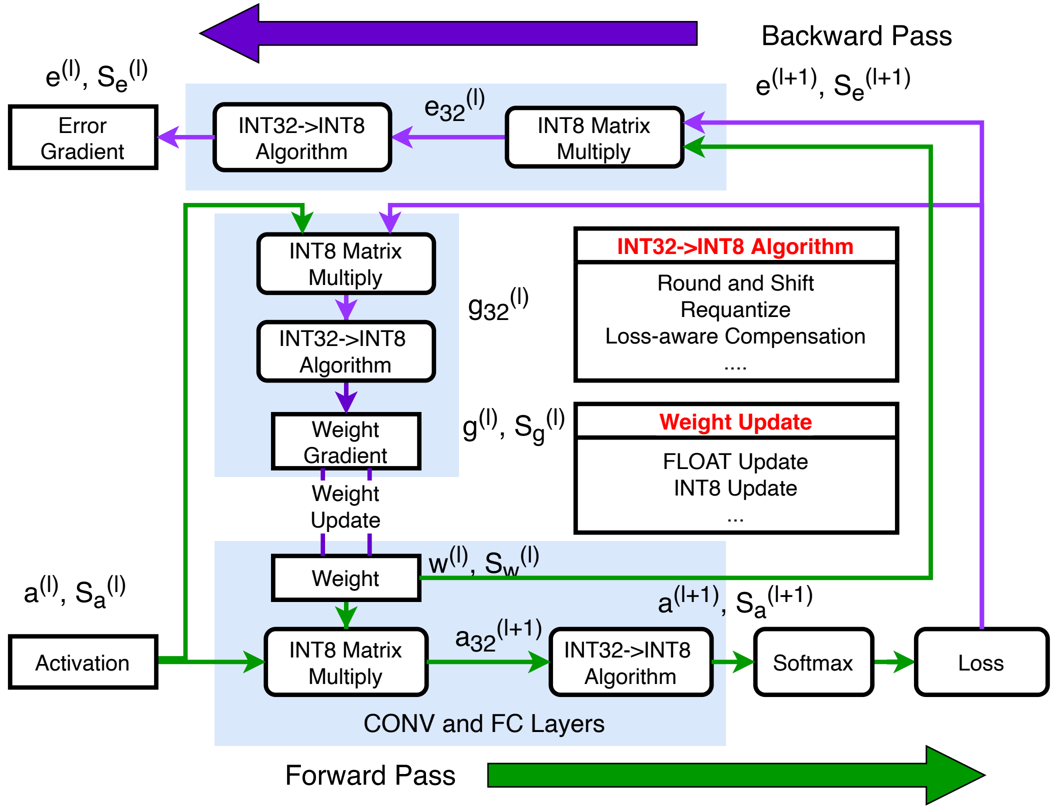
Motivation

- NPUs are becoming ubiquitous on mobile SoCs, can we use them to accelerate training?
  - The key issue: mobile NPUs often operate on **low-precision formats**

Vendor	**Supported data formats**	**SDK**
Qualcomm/AIP (HTA/HTP)	INT8 (Since Snapdragon 855， HTA、HTP) FP16 (Since Snapdragon 8Gen1， HTP) INT4 (Since 8Gen2， HTP)	SNPE (Snapdragon Neural Processing Engine)
Huawei/Kirin NPU	FP16	HiAI Foundation，
MediaTek APU	INT8	NeuroPilot SDK
Google Edge TPU	INT8 (both 1.0 and 2.0) FP16 (both 1.0 and 2.0, highly optimized in 2.0)	TFLite delegate
Rockchip NPU	INT8/INT16 (mostly) FP16 (Only RK3588)	RockChip SDK

An abstraction

- Making Mandheling a unified framework for various mixed-precision training algorithms – through a few configurations



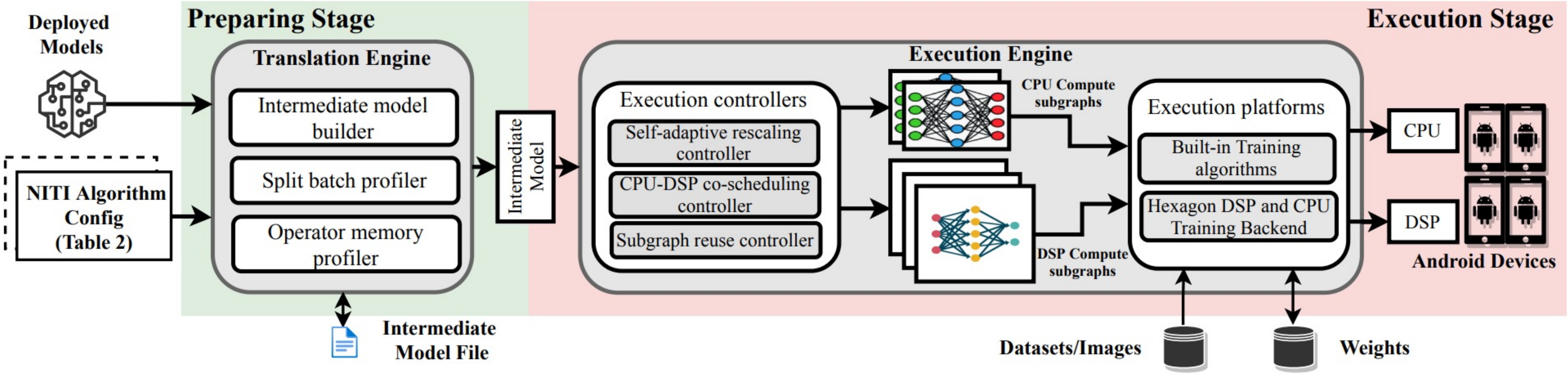
Mixed-precision algo.	W	A	G	WU	support
NITI [67]	INT8	INT8	INT8	INT8	✓
Octo [82]	INT8	INT8	INT8	INT8	✓
Adaptive Fixed-Point [79]	INT8/INT16	INT8	INT8	FP32	✓
WAGEUBN [74]	INT8	INT8	INT8	FP24	✓
MLS Format [81]	INT8	INT8	INT8	FP32	✓
Chunk-based [68]	FP8	FP8	FP8	FP16	×
Unifiec					×

"W", "A ... pdate.

Attribute	Contents	
	key	value
Translation	FP32 Conv	INT8 Conv+ReduceMax+Shift
	FP32 MaxPool	INT8 MaxPool
Backprop.	FP32 Conv Error Grad.	INT8 Deconv
	FP32 Conv Weight Grad.	INT8 ConvBackpropFilter
Weight	Initializer	Xavier_normal
	Type	INT8
	Update	INT8
Optimizer	Loss	Cross Entropy
	Optimizer	SGD

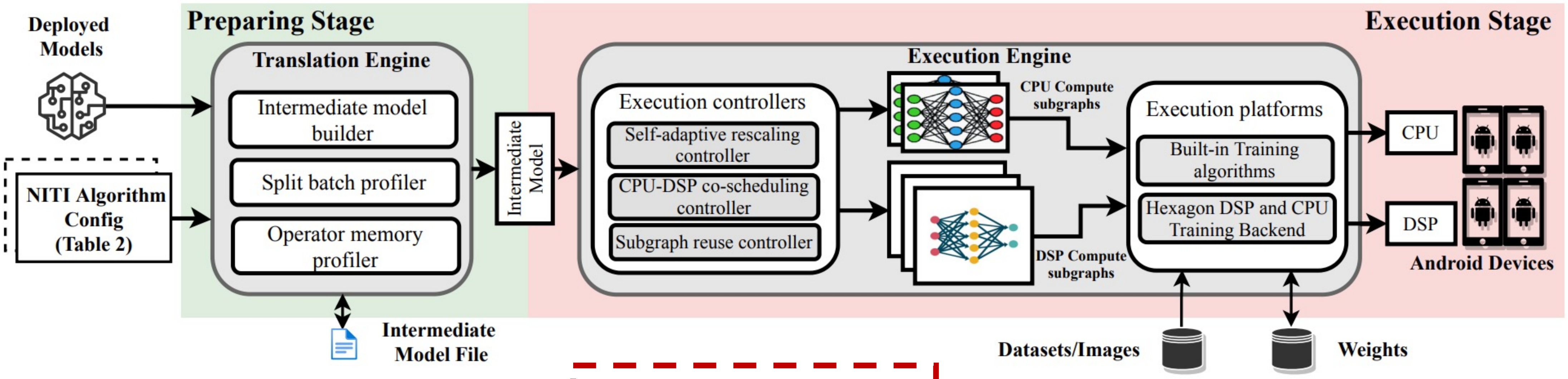
Table 2: A typical NITI algorithm training config.

System overview



Challenges	DSP-unfriendly operators	Slow dynamic rescaling (quantization ops)	Exhausted data cache	Costly compute graph preparation
Techniques	***CPU-DSP co-scheduling***	***Self-adaptive rescaling***	***Batch splitting***	***DSP-compute subgraph reuse***

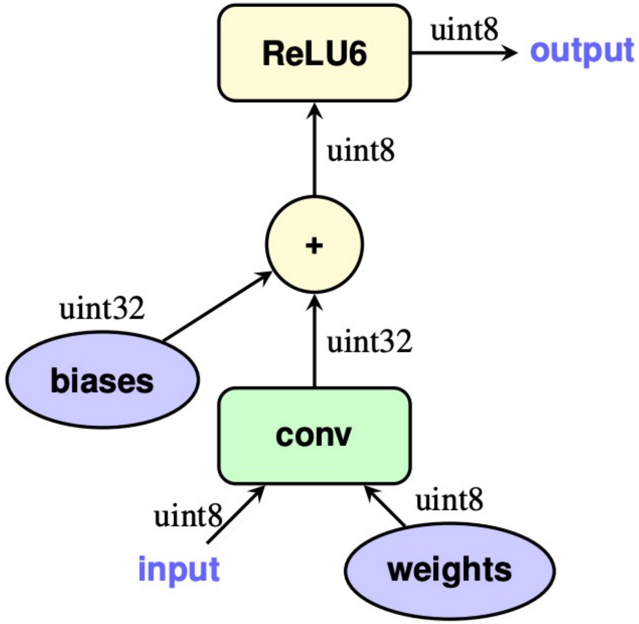
System overview



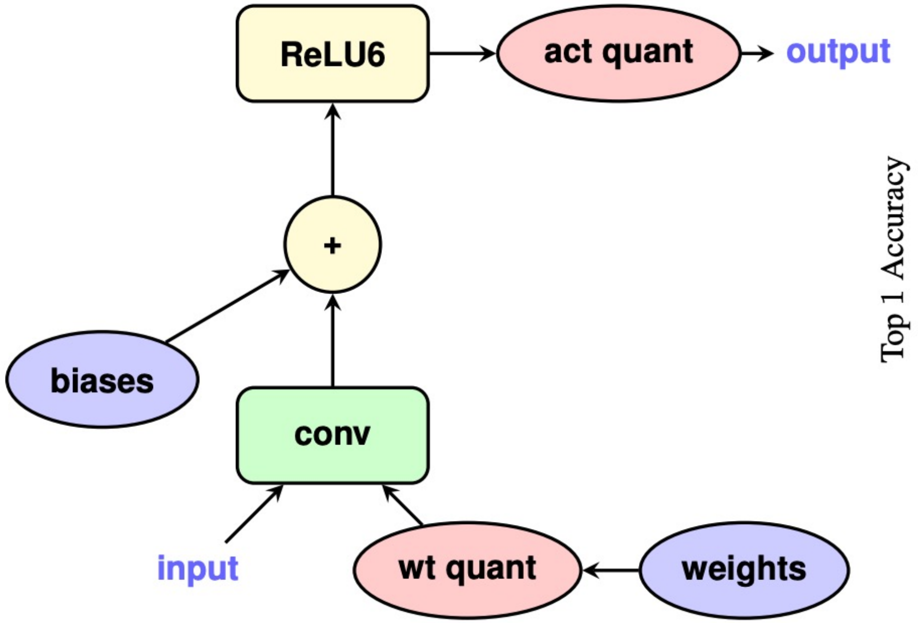
Challenges	DSP-unfriendly operators	Slow dynamic rescaling (quantization ops)	Exhausted data cache	Costly compute graph preparation
Techniques	***CPU-DSP co-scheduling***	***Self-adaptive rescaling***	***Batch splitting***	***DSP-compute subgraph reuse***

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.



(a) Integer-arithmetic-only inference



(b) Training with simulated quantization

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
  - Memory-intensive

```
int scale = 0;
/* Calculate INT32 temporal results */
for(int i = 0; i < length; i++) {
  Tensor x = input[i];
  Tensor w = weight[i];
  // CONV or matrix multiply
  Tensor temp_result = x * w;
  // count leading zero
  Tensor clz = clz(temp_result);
  int tscale = 32 - max(clz) - 7;
  scale = scale > tscale ? scale :
        tscale;
  temp_output[i] = temp_result;
}
/* Cast the INT32 to INT8 values */
for(int i = 0; i < length; i++) {
  Tensor temp = temp_output[i];
  // Downscale
  Tensor int8_result = temp / scale;
  result[i] = int8_result;
}
```

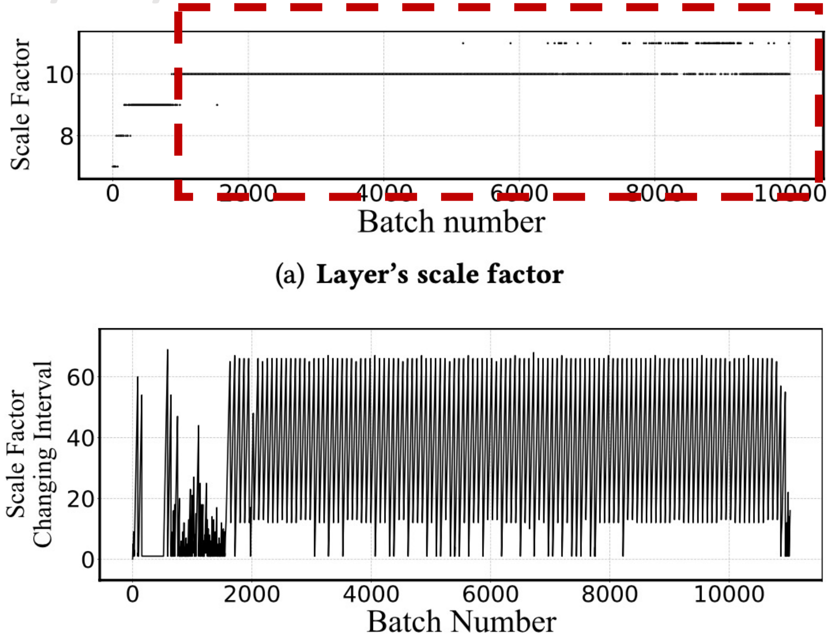
Listing 1: Key C code snippet of dynamic rescaling

```
  scale = 0

loop0:
  v0 = vmem ptr_i
  v1 = vmem ptr_w

  v2 = vrmpy v0, v1

  v3 = vclz v2
  tscale = vmax v3
  scale = mux scale >
        tscale, scale,
        tscale
  vmem ptr_t, v2
end loop0
loop1:
  v0 = vmem ptr_t

  v3 = vmpye v0, scale
  vmem ptr_v, v3
end loop1
```

Listing 2: Asm code version

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
- Opportunity
  - Very few candidates of n
  - Changing frequency is low



(a) **Layer's scale factor**

(b) **Layer's scale factor changing interval**

Figure 4: The scale factor and its changing interval of the first CONV layer in training VGG11 model (batch size = 64) on CIFAR-10 dataset.

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
- Opportunity
  - Very few candidates of n
  - Changing frequency is low
- **Solution: self-adaptive instead of every batch**
  - Determining the adapting frequency based on historical traces

Highlighted results

- Implementation
  - 15k LoC in C/C++ and 800 LoC in assembly
  - Reuse ops on CPU from MNN
- Setups
  - 3 devices
  - 6 models
  - 2 datasets (CIFAR-10 & ImageNet)
- Baselines
  1. TFLite-FP32
  2. MNN-FP32
  3. MNN-INT8
  4. MNN-INT8-GPU
- Algorithm: NITI[1]

Devices	CPU	GPU	DSP
XiaoMI 11 Pro Snapdragon 888	2.84GHz Cortex-X1 3× 2.4GHz Cortex A78 4× 1.8GHz Cortex A55	Adreno 660 GPU 700MHz	Hexagon 780 DSP 500MHz
XiaoMI 10 Snapdragon 865	2.84GHz A77 3× 2.4GHz Cortex A77 4× 1.8GHz Cortex A55	Adreno 650 GPU 587MHz	Hexagon 698 DSP 500MHz
Redmi Note9 Pro Snapdragon 750G	2× 2.2GHz Cortex A77 6× 1.8GHz Cortex A55	Adreno 619 GPU 950MHz	Hexagon 694 DSP 500MHz

Table 5: Devices used in the experiments.

Model	Input Data	FLOPs	# of CONVs
VGG-11 [60]	CIFAR-10	914 M	8
VGG-16 [60]	CIFAR-10	1.35 G	13
VGG-19 [60]	ImageNet	26.92 G	16
ResNet-34 [29]	CIFAR-10	7.26 G	36
ResNet-18 [29]	ImageNet	11.66 G	20
InceptionV3 [62]	CIFAR-10	2.43 G	16

Table 6: DNN models used in the experiments.

[1] Wang, Maolin, et al. "Niti: Training integer neural networks using integer-only arithmetic." IEEE Transactions on Parallel and Distributed Systems (2022).

Highlighted results

- Per-batch training time reduced by up to 8.3x.

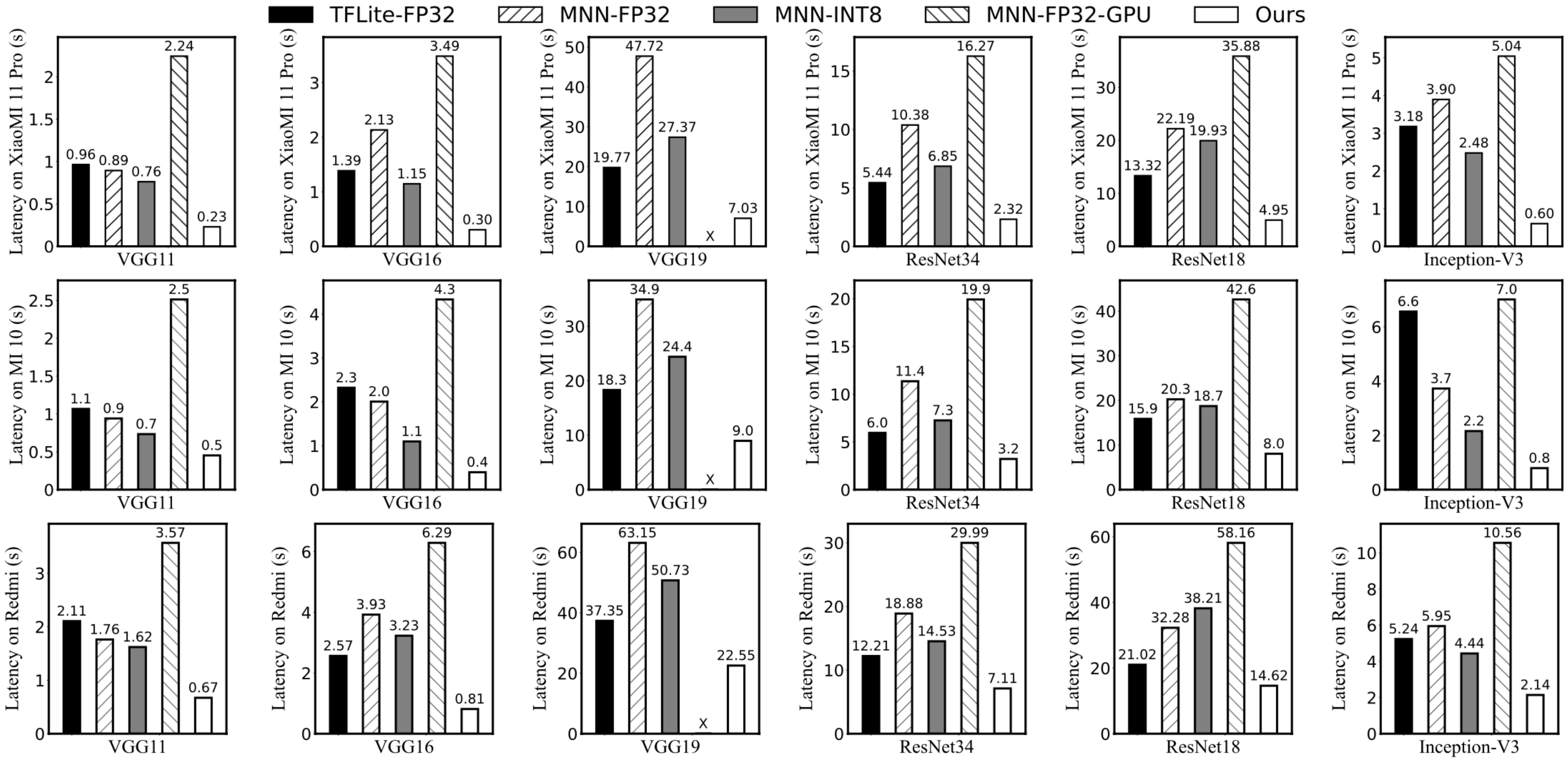


Figure 5: Per-batch training time on different models (batch size = 64) on different devices.

Highlighted results

- Per-batch energy consumption reduced by up to 12.5x.

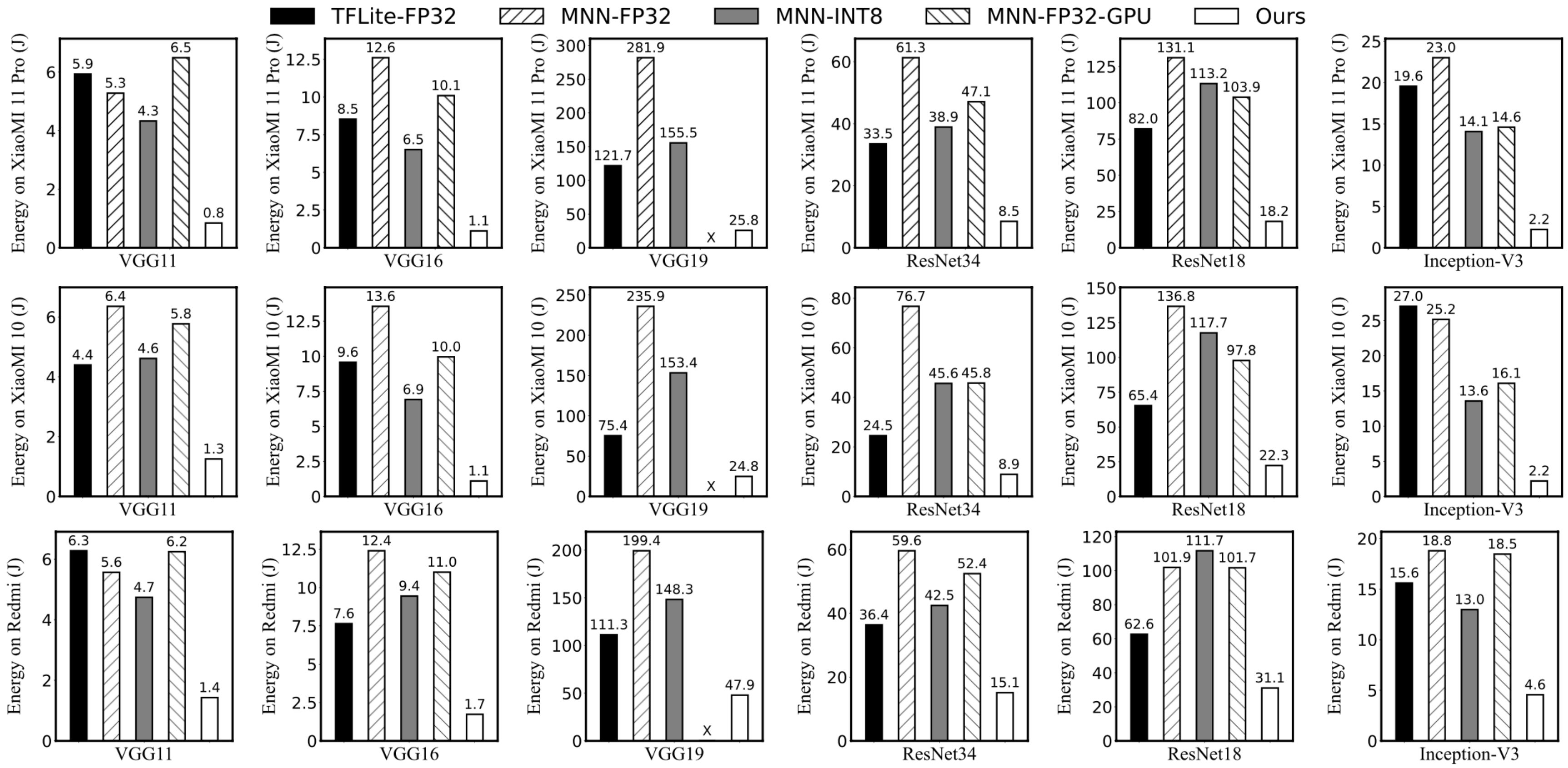


Figure 6: Per-batch energy consumption on different models (batch size = 64) on different devices.

Highlighted results

- In end-to-end convergence tasks
  - Time reduced by 5.7x on average
  - Energy consumption reduced by 7.8x on average
  - 19.%--2.7% accuracy loss

Dataset	Model	Methods	Acc.	Training Cost to Convergence		
				Round number	Clock Hours	Energy (WH)
Centralized CIFAR-10	VGG11	MNN-FP32	89.87%	150	29.13	187.01
		MNN-INT8	87.17%	150	24.77	153.33
		Ours	87.17%	150	7.50	31.39
Centralized CIFAR-10	ResNet18	MNN-FP32	92.49%	150	223.55	1,435.19
		MNN-INT8	90.62%	150	135.71	840.04
		Ours	90.62%	150	35.68	149.32
Federated FEMNIST	LeNet	MNN-FP32	84.18%	990	0.97	0.00057
		MNN-INT8	82.04%	4,960	0.39	0.00029
		Ours	82.04%	4,960	0.19	0.00007
Federated CIFAR-100	VGG16	MNN-FP32	71.15%	1,960	8.35	2.74
		MNN-INT8	68.42%	2,200	1.56	1.26
		Ours	68.42%	2,200	0.78	0.21

Table 8: A summary of end-to-end training cost till convergence under different training scenarios.

Outline – (Federated) Training on Devices

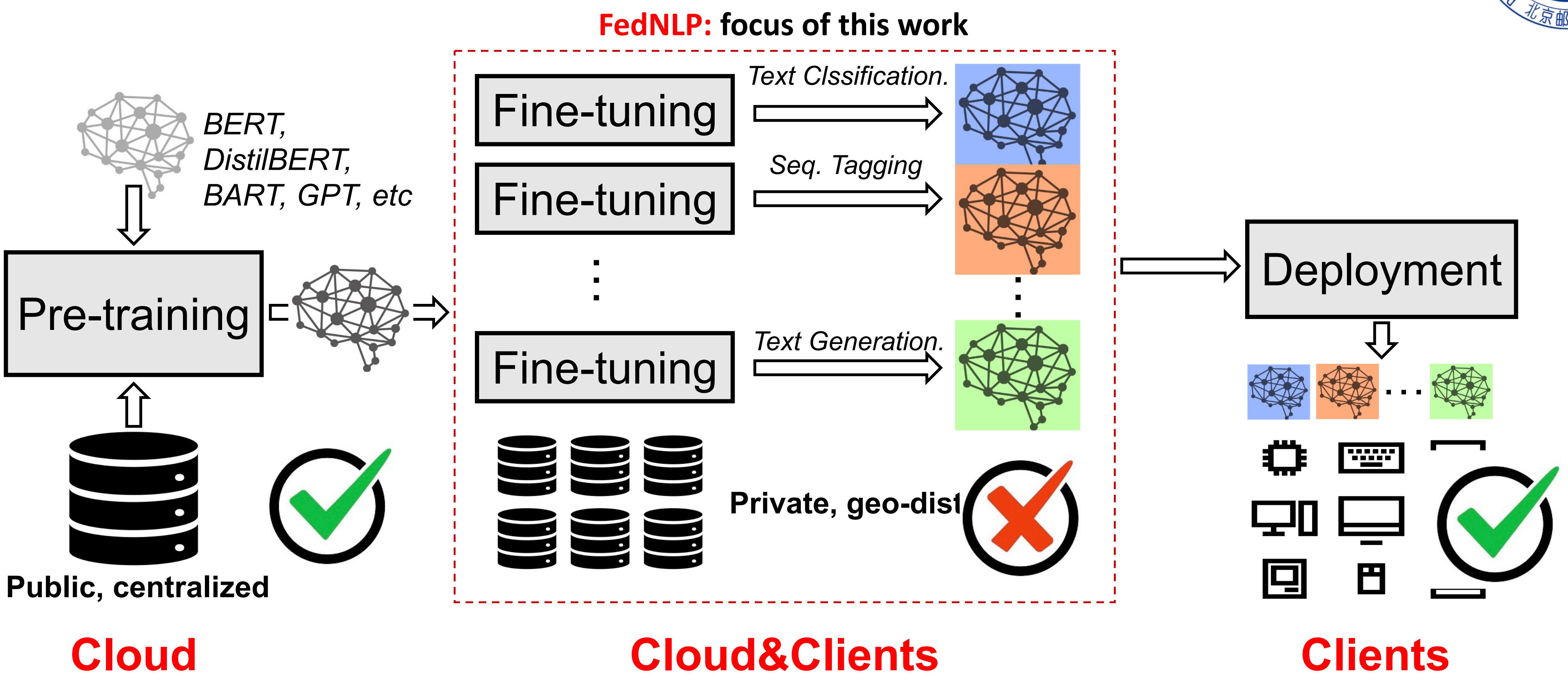
Training on edge

- [WWW'21] FLBench — ☑ *Simulation integrity*
- [MobiCom'22] Mandheling — ☑ Time & energy
- [MobiSys'22] Melo — ☑ *memory*
- [INFOCOM'23] FLScheduler — ☑ *privacy*

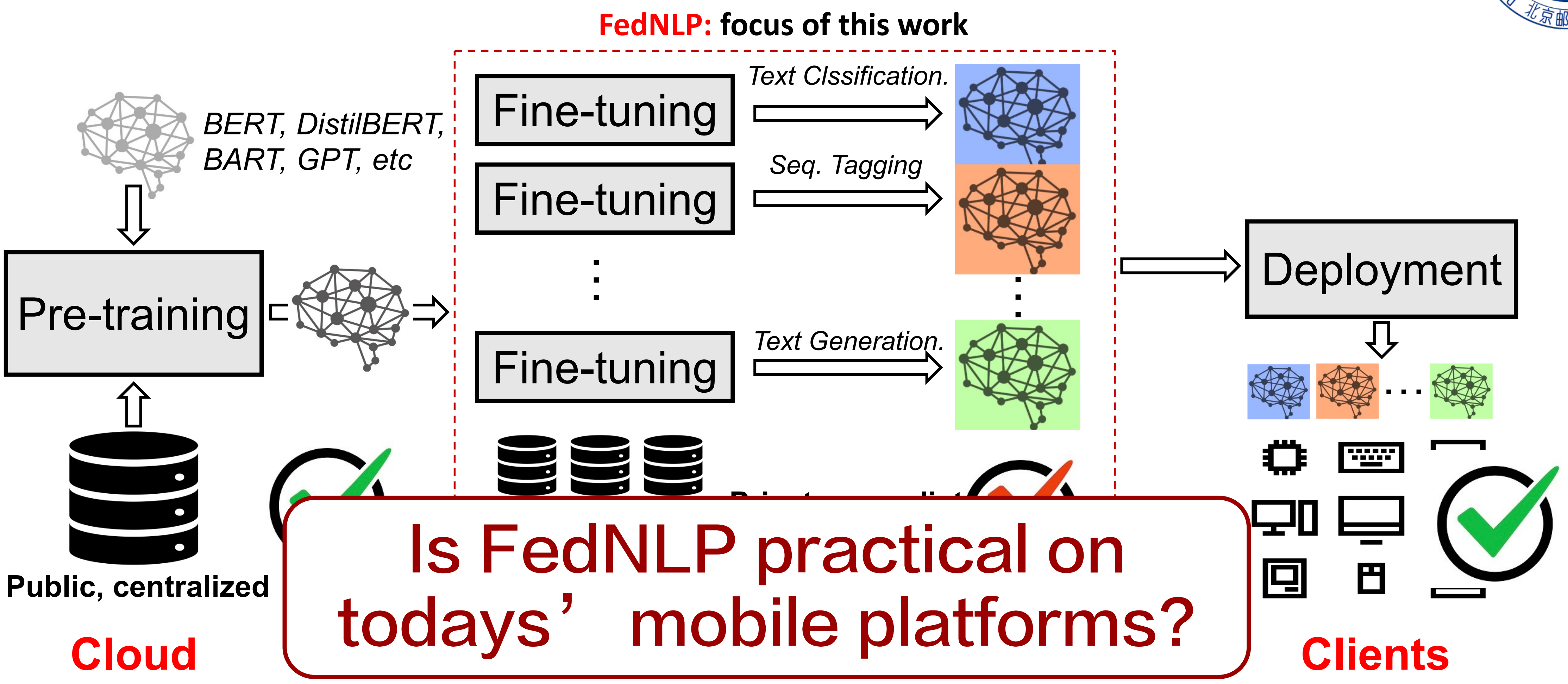
Fine-tuning LLM on edge

- **[MobiCom'23] FedAdapter** — ☑ Network & speed
- [MobiCom'23] FedFSL — ☑ *Label scarcity*
- [Going-on] ForwardFL — ☑ *Time & energy & scalability & etc..*

Background


FedNLP: focus of this work
BERT,
DistilBERT,
BART, GPT, etc
Pre-training
Public, centralized
Fine-tuning
Fine-tuning
Fine-tuning
Text Clssification.
Seq. Tagging
Text Generation.
Private, geo-dist
Deployment
Cloud
Cloud&Clients
Clients

Background

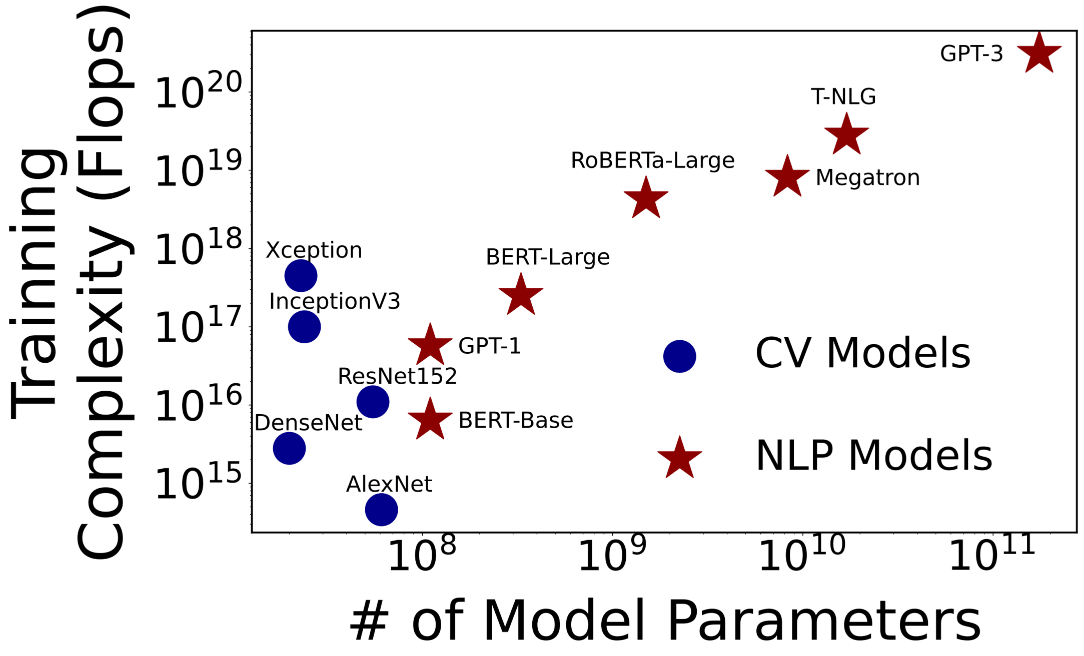


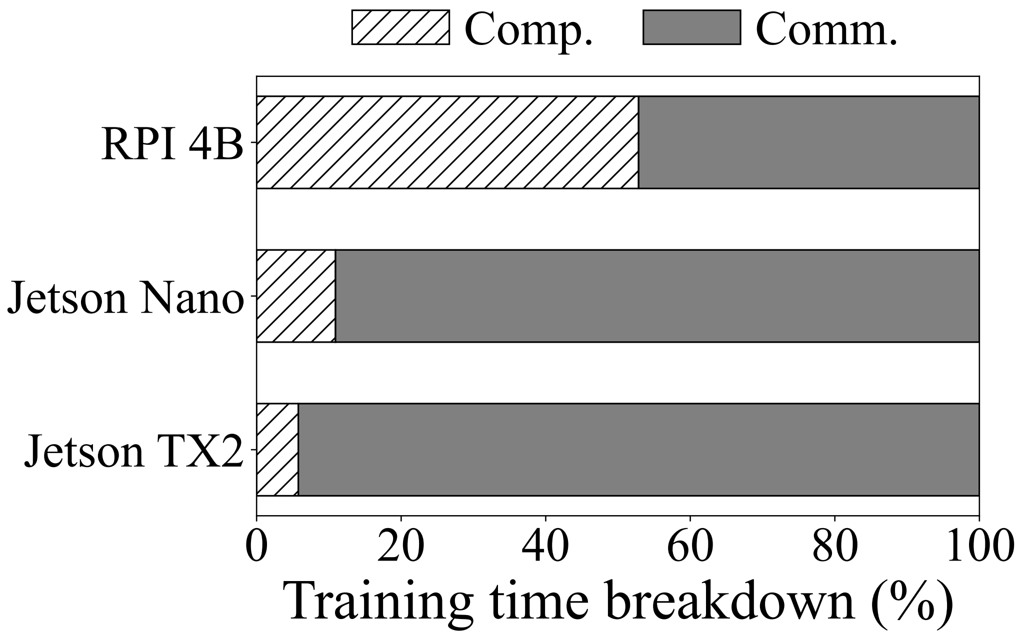
Is FedNLP practical on todays’ mobile platforms?

Challenges

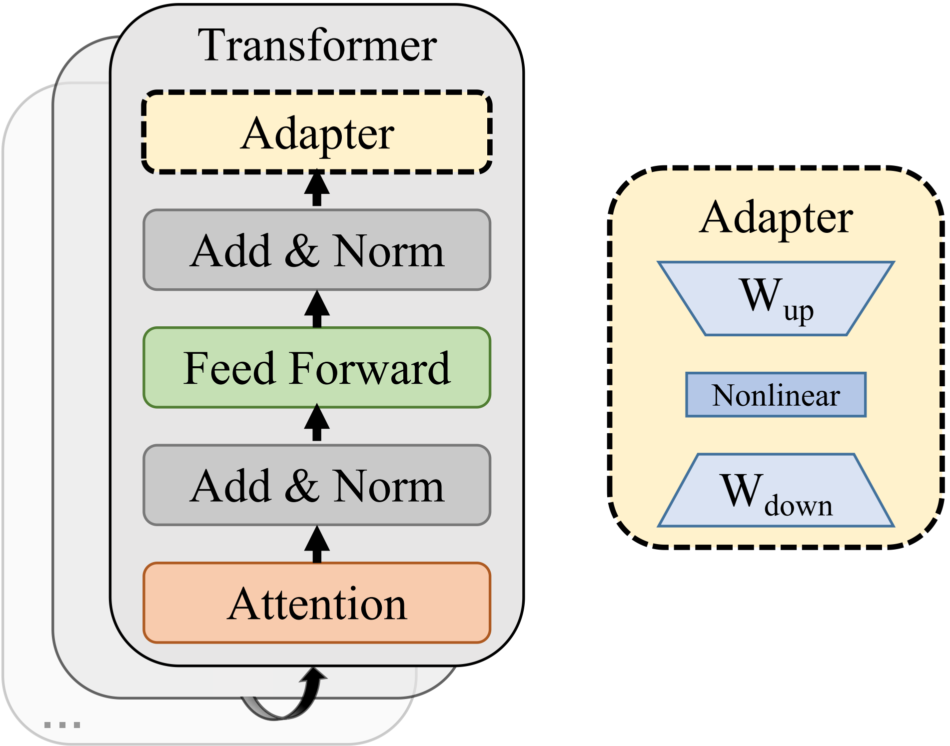
LLM training on mobile platforms:

- Transformer-based NLP models are highly costly.
- Network transmission dominates the training delay.





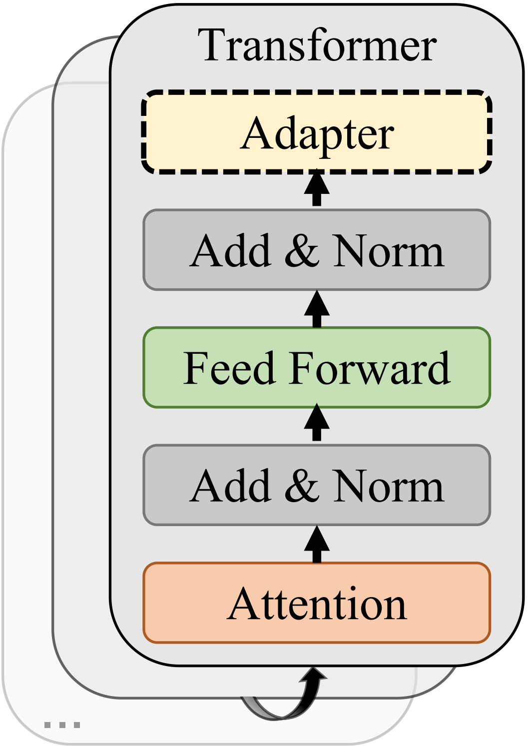
Key building block: pluggable adapters

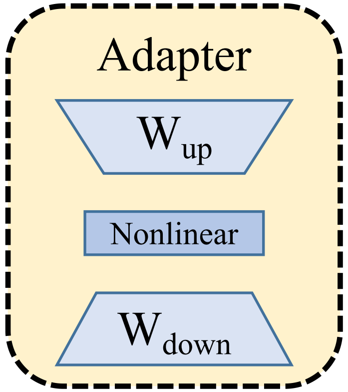


Model	Method	Training Time	Updated Paras.
BERT	Full Fine-tuning	1.86 sec	110.01×10^6
	Adapter	1.14 sec	0.61×10^6
DistilBERT	Full Fine-tuning	0.91 sec	67×10^6
	Adapter	0.56 sec	0.32×10^6

Table 1: Computation and communication cost of inserting adapters into each transformer block (width=32) and full-model tuning on Jetson TX2.

Key building block: pluggable adapters





Model	Method	Training Time	Updated Paras.
BERT	Full Fine-tuning	1.86 sec	110.01×10^6
	Adapter	1.14 sec	0.61×10^6
DistilBERT	Full Fine-tuning	0.91 sec	67×10^6
	Adapter	0.56 sec	0.32×10^6

Table 1: Computation and communication cost of inserting adapters into each transformer block (width=32) and full-model tuning on Jetson TX2.

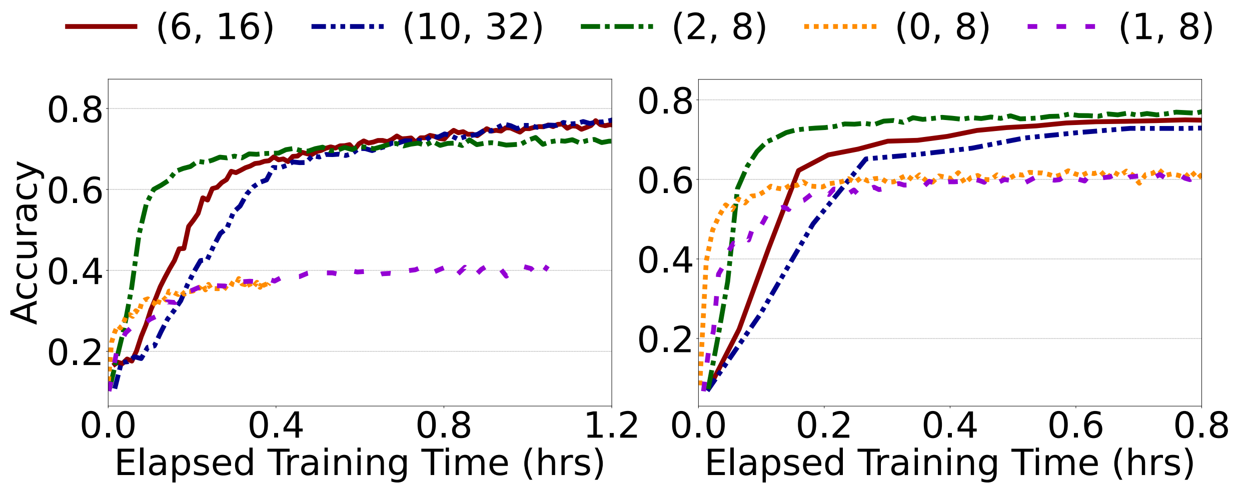
How to find an "optimal" adapter towards fast convergence? (It's not like AutoML/NAS!)

Adapter configuration challenges

- Large adapter configuration space
- Design must be online
- No silver bullet configuration

Model	Datasets	Optimal adapter configuration **(depth, width)** towards different target accuracy				
		99%	**95%**	**90%**	**80%**	**70%**
BERT	**20news**	(2,64)	(2,32)	(2,8)	(2,8)	(2,8)
	agnews	(3,16)	(2,16)	(2,8)	(0,8)	(0,8)
	semeval	(10,8)	(6,8)	(6,8)	(2,8)	(2,8)
	ontonotes	(12, 32)	(12, 32)	(10, 32)	(0, 16)	(0, 16)

Table 2: The optimal adapter configuration (i.e., best time-to-accuracy) for different target accuracy (ratio to the full convergence) and different datasets.

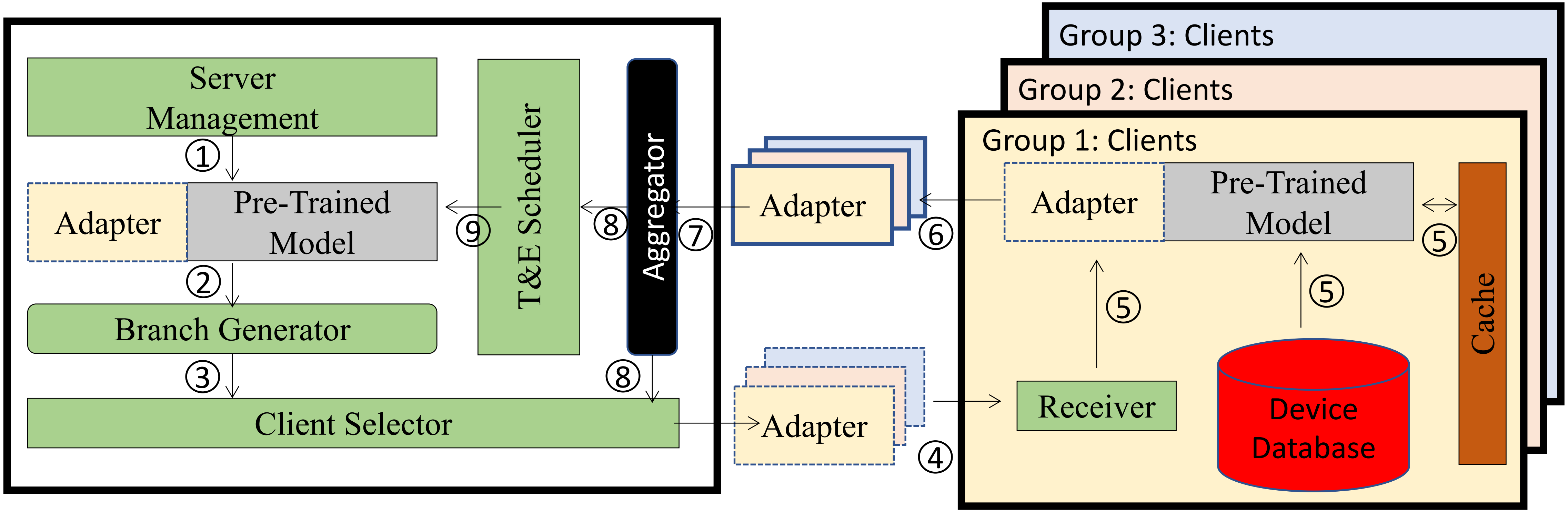


(a) SEMEVAL (b) 20NEWS

Figure 4: Across different target accuracy and FedNL tasks, the optimal adapter configuration (dept width) varies. Tested with BERT and Jetson TX2.

Our design: trial-and-error

A. Progressive training; **B.** Identifying timing and direction to upgrade configuration through sideline trails.



Highlighted results

- Implementation
  - FedNLP[1]
  - AdapterHub[2]
- Setups
  - 3 devices
  - 2 models (BERT & DistilBERT)
  - 4 datasets
- Baselines
  1. Vanilla Fine-Tuning (FT)
  2. FineTuning-Quantized (FTQ)
  3. LayerFreeze-Oracle (LF_{oracle})
  4. LayerFreeze-Quantized-Oracle (LFQ_{oracle})

Device	Processor	Per-batch Latency (s)
Jetson TX2 [1]	256-core NVIDIA Pascal™ GPU.	0.88
Jetson Nano [2]	128-core NVIDIA CUDA® GPU.	1.89
RPI 4B [3]	Broadcom BCM2711B0 quad-core A72 64-bit @ 1.5GHz CPU.	18.27

Table 3: Development boards used in experiments.

Task	Dataset	# of Clients	Labels	Non-IID	Samples
TC	20NEWS [44]	100	20	/	18.8k
TC	AGNEWS [92]	1,000	4	a=10	127.6k
TC	SEMEVAL [31]	100	19	a=100	10.7k
ST	ONTONOTES [60]	600	37	a=10	5.5k

Table 4: Datasets and settings used in experiments for Text Classification and Sequence Tagging. "a" is a parameter that controls the datasets' non-IID level [50].

[1] Yuchen Lin B, He C, Zeng Z, et al. FedNLP: Benchmarking Federated Learning Methods for Natural Language Processing Tasks[J]. Findings of NAACL, 2022.
[2] Pfeiffer J, Rücklé A, Poth C, et al. AdapterHub: A Framework for Adapting Transformers. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020: 46-54

Highlighted results

- AdaFL reduces model convergence delays significantly.

Datasets	20NEWS			AGNEWS			SEMEVAL			ONTONOTES		
Relative Accuracy	99%	95%	90%	99%	95%	90%	99%	95%	90%	99%	95%	90%
FT	44.0	23.4	13.1	31.1	10.1	5.2	124.3	89.9	61.7	76.1	55.9	35.6
FTQ	12.7	6.8	3.8	9.1	2.6	1.7	32.0	23.1	15.9	21.2	15.5	9.9
LF_{oracle}	18.5	8.1	4.3	9.6	1.4	1.1	74.0	46.8	33.2	82.5	43.8	24.5
LFQ_{oracle}	5.2	2.5	1.1	1.6	0.3	0.2	16.8	11.0	7.7	23.9	12.9	7.2
AdaFL	**1.3**	**0.4**	**0.1**	**0.2**	**0.03**	**0.02**	**2.3**	**1.1**	**0.6**	**4.5**	**2.4**	**1.3**

Table 5: Elapsed training time taken to reach different relative target accuracy. NLP model: BERT. Unit: Hour.

from 44hrs to 1.3hrs

Highlighted results

- AdaFL outperforms baselines in various network environments and on various client hardware.

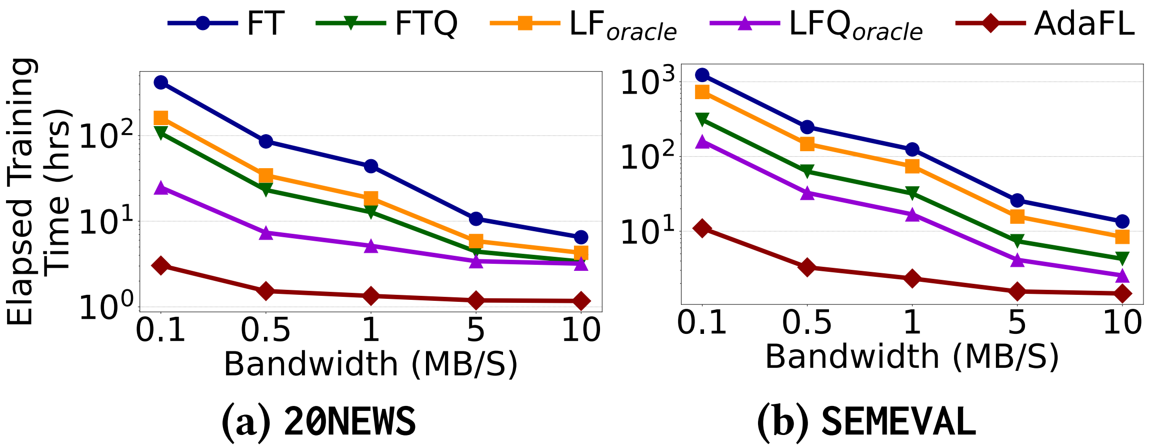


Figure 6: AdaFL outperforms baselines under all network bandwidths with 99% target accuracy.

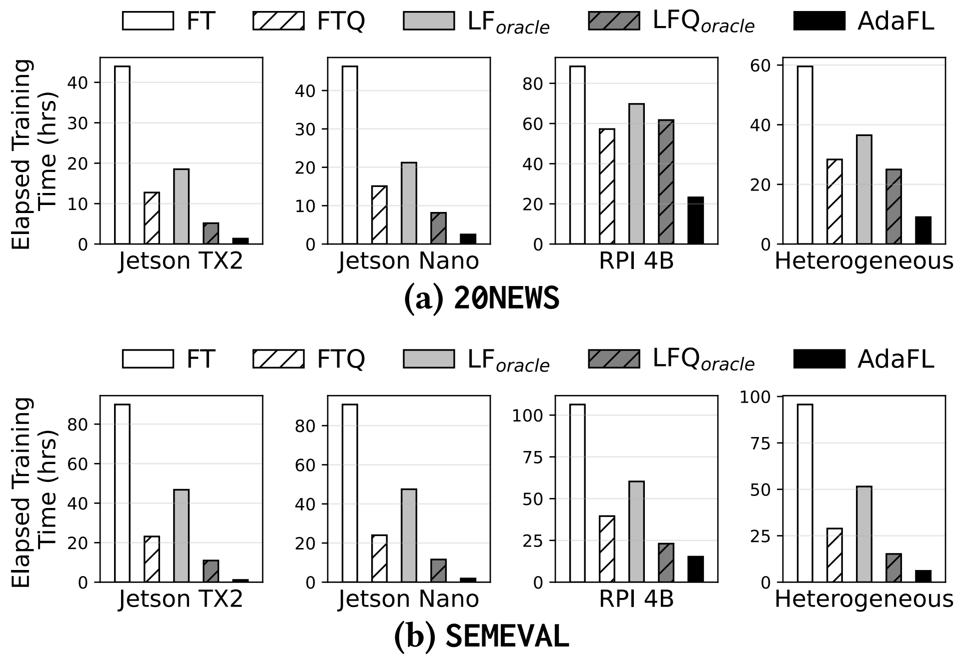


Figure 7: Convergence delays with a variety of client hardware. Training targets 99% relative target accuracy. "Heterogeneous" means the device capacity is uniformly distributed between three boards.

Outline – (Federated) Training on Devices

Training on edge

- [WWW'21] FLBench — ☑ *Simulation integrity*
- [MobiCom'22] Mandheling — ☑ Time & energy
- [MobiSys'22] Melo — ☑ *memory*
- [INFOCOM'23] FLScheduler — ☑ *privacy*

Fine-tuning LLM on edge

- [MobiCom'23] FedAdapter — ☑ Network & speed
- [MobiCom'23] FedFSL — ☑ *Label scarcity*
- **[Going-on] ForwardFL** — ☑ Time & energy & scalability & etc..

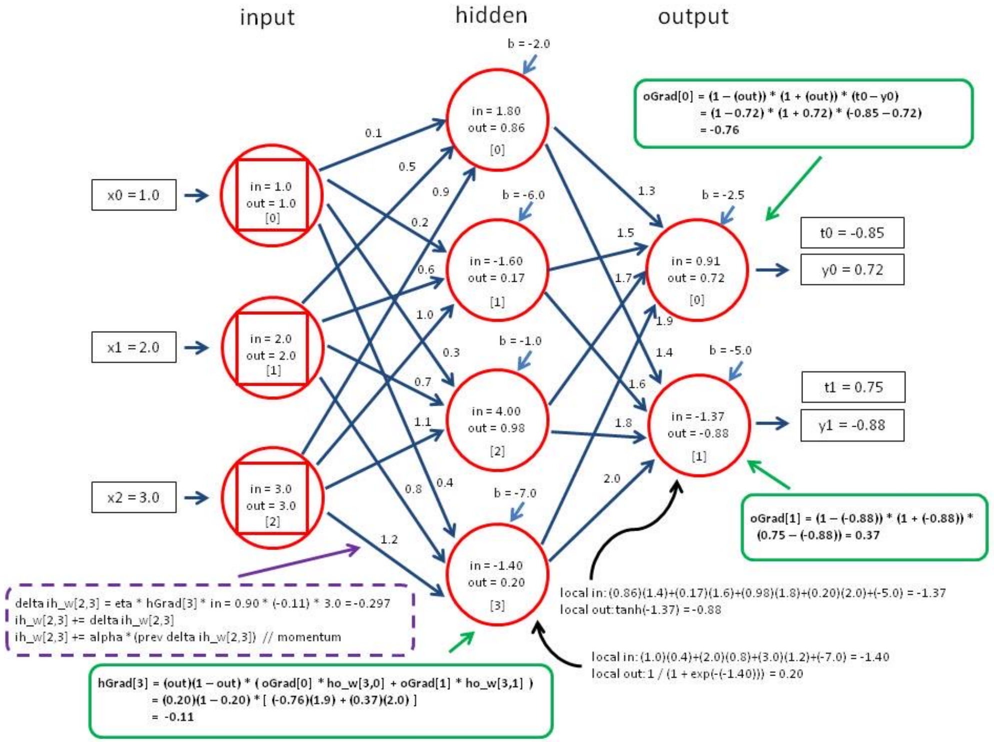
Time to retire *backprop*. in FedLLM

- Those optimizations are good, but NOT good enough to bring FedLLM to real world

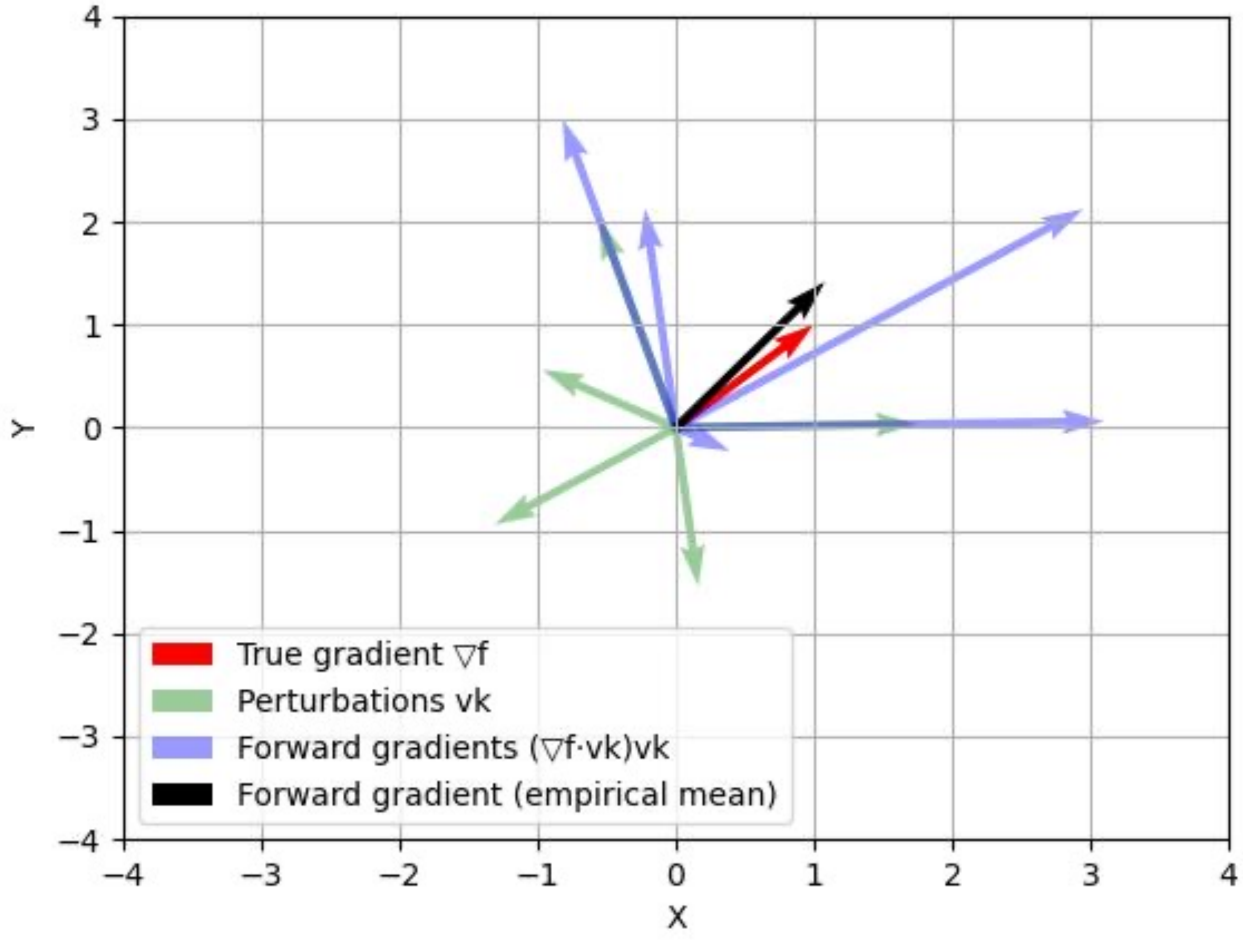
Huge Memory Footprint

NPU-unfriendly Operators

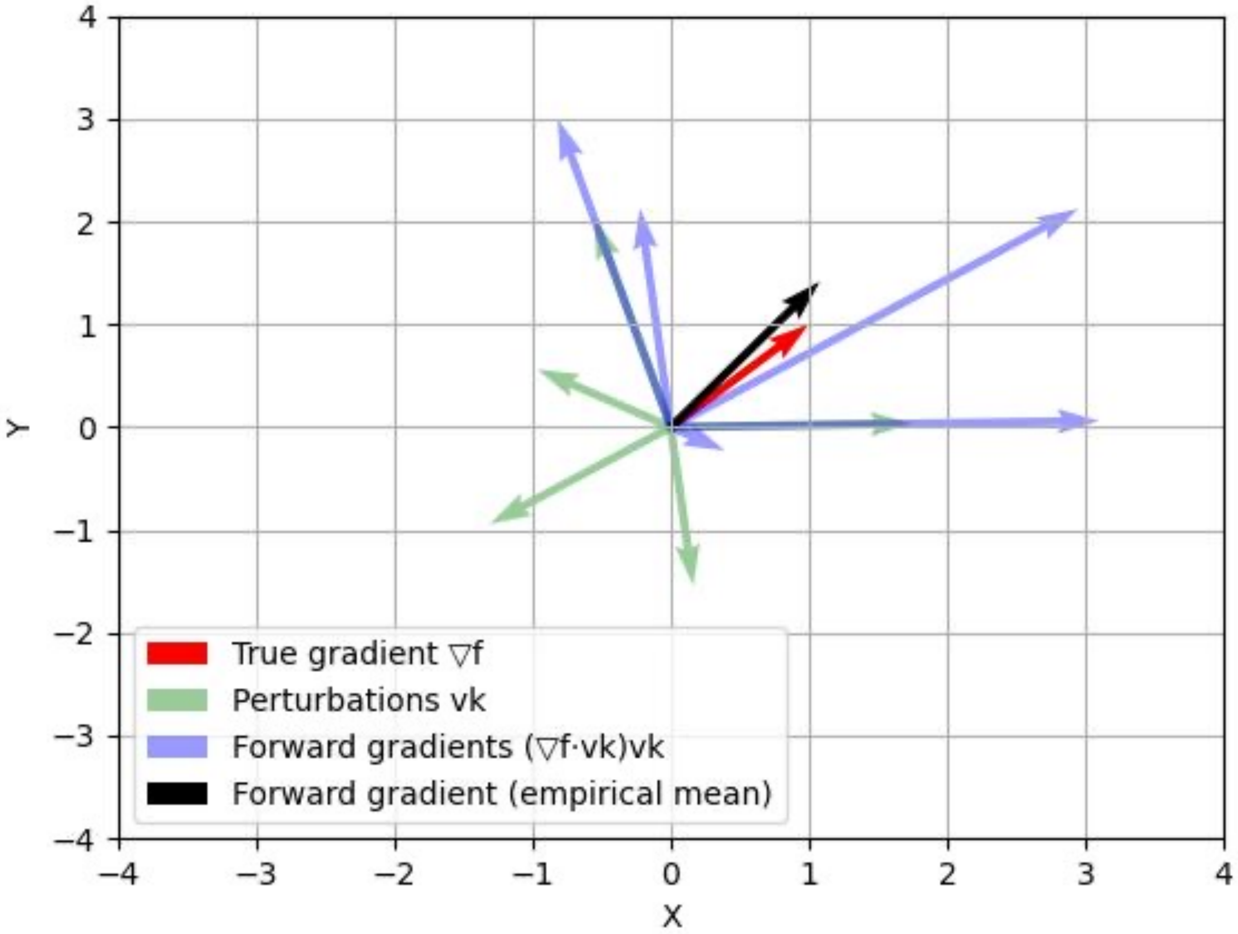
Low Client Scalability



Forward gradient: guess-then-verify



Forward gradient: guess-then-verify

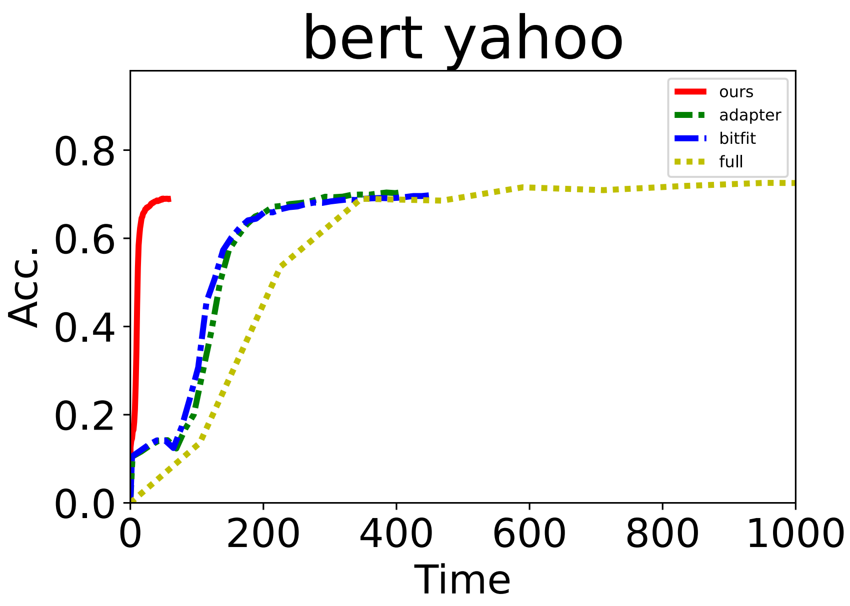


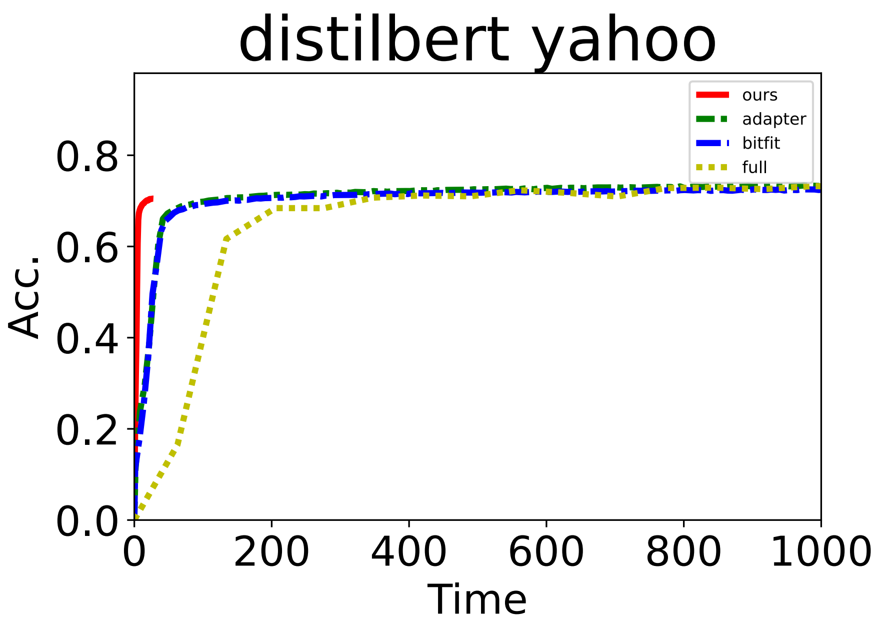
Okay, forward gradient can be traced back to 1980s (also called weight perturbation methods), but it never goes real?

- Because of its increased demand of computing/data with the **trainable** parameter size

Some early results

- Delivers about 2 orders of magnitudes speedup
  - By leveraging NPU and more clients
- Enables federated learning of LLaMA-7B over real smartphones
  - For the very first time





A few thoughts on mobile/edge LLM..

The Golden Era for Mobile/Edge Research

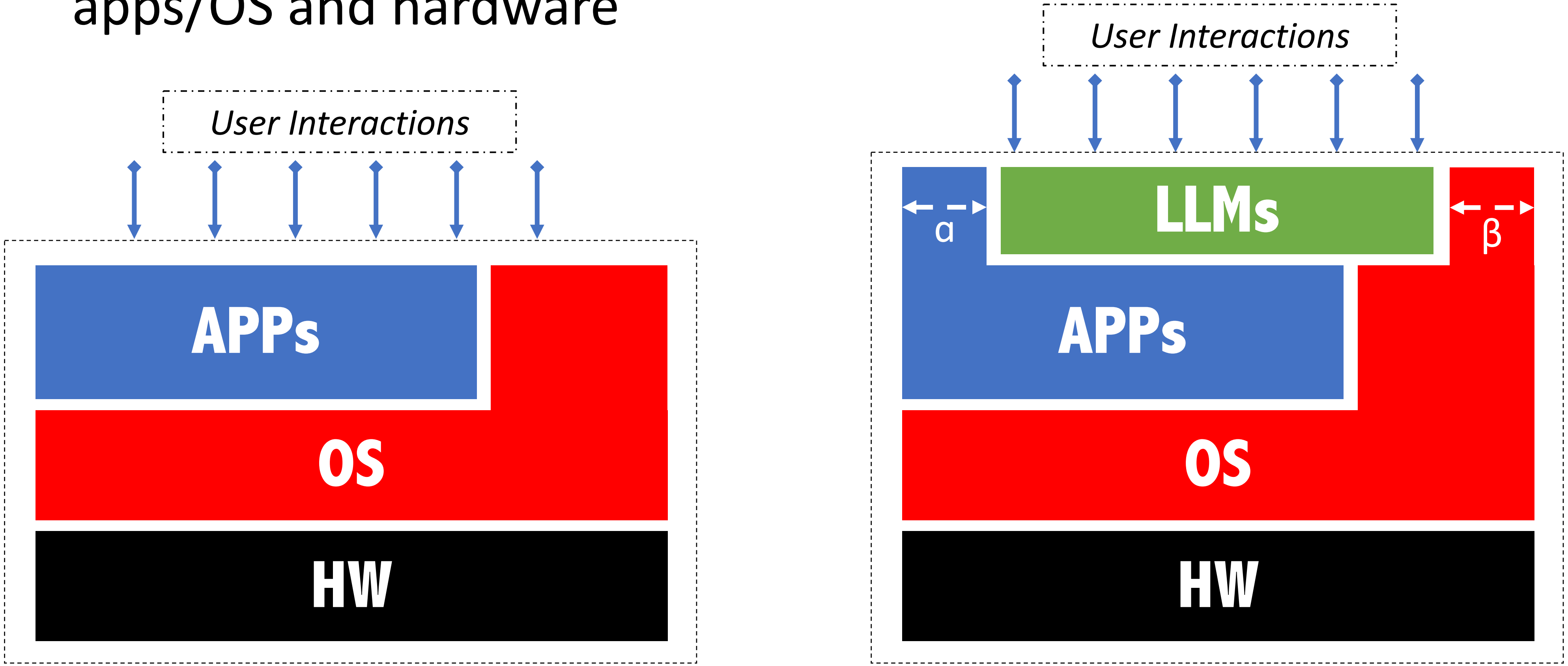
- Since iPhone 2007..
- The next long-term goal of mobile research: ChatGPT on smartphone
  - Takes ~5 years
    - maybe LLaMA-2-65B in 1 year first?
  - Takes collective efforts from hardware/architecture, mobile system, ML algorithm communities
- **Old stories:** data privacy, low delay, low power consumption, etc..
- **New techniques:** memory-bounded LLMs, foundation model + adapters, generative and augoregressive, MoE, etc..

The Golden Era for Mobile/Edge Research

- For LLMs deployed on cloud –
  - **How to protect data privacy?**
- For LLMs deployed on devices –
  - **How to efficiently scale the model size?**
- A hybrid mode, e.g., a cascade –
  - **How to split the workloads?**

LLM is the new Operating System

- Users interact with LLM, while LLM manages/utilizes old-time apps/OS and hardware



α+β -> 0

LLM is the new Operating System

- Users interact with LLM, while LLM manages/utilizes old-time apps/OS and hardware

- LLaMA wants to be (or already is) the new Android?
    - Think about its ecosystem: LLaMA.cpp, various LoRa adapters..

Exploration atop or below LLM?

- Another way to go: build systems for LLM, or build systems with LLM
- When a software layer is finalized, most research/industry opportunities go above
  - Very very few system researchers rebuild OS now
  - Very very few network researchers rebuild network stacks now

Auto-GPT, Agent-GPT, babyAGI, HuggingGPT, Web LLM, CAMEL, GPTRGB, PandaGPT..

→ *Easier to handle, potentially high impacts, but more crowded and competitive*

LLMs

GPTQ, Mixture-of-Experts, [EuroSys'23] Tabi, [MLSys'23] Flex, [OSDI'22] Orca, [ATC'22] PetS..

→ *More fundamental, potentially extremely-high impacts but technically/financially challenging*

Takeaways

- Machine (deep) learning is happening everywhere at anytime
- The system support for edge intelligence is still at very preliminary stage – so many open problems!
- LLMs bring new challenges and opportunities
- Open to discussion and collaboration!