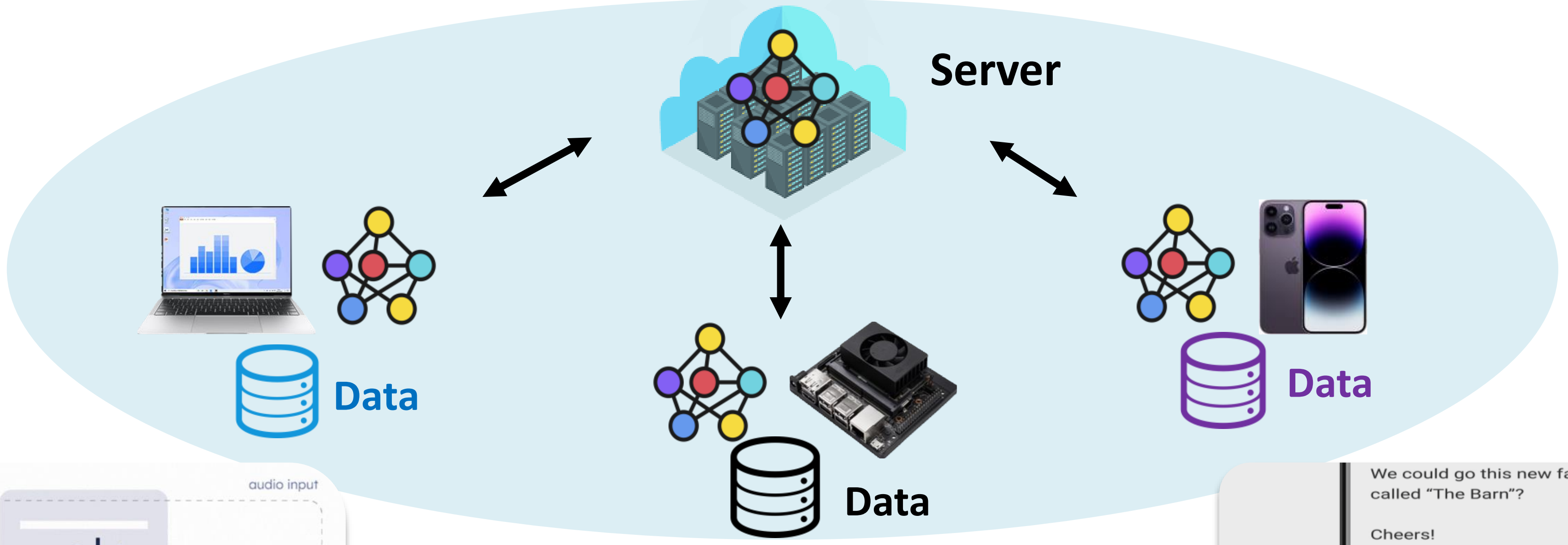
LATTE: Layer Algorithm-aware Training Time Estimation for Heterogeneous Federated Learning

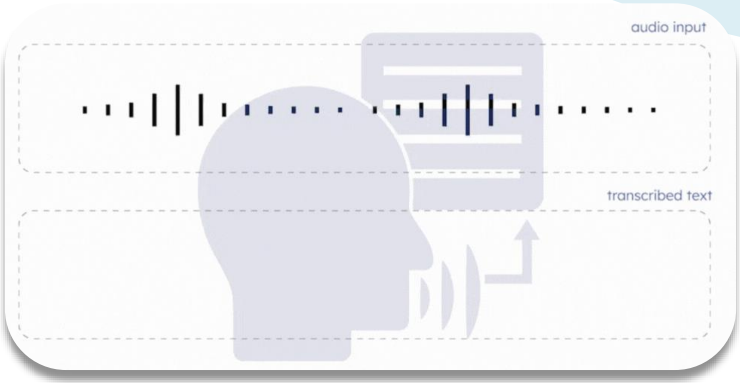
Kun Wang[1], Zimu Zhou[2], Zhenjiang Li[1]

[1] Department of Computer Science, City University of Hong Kong
[2] School of Data Science, City University of Hong Kong

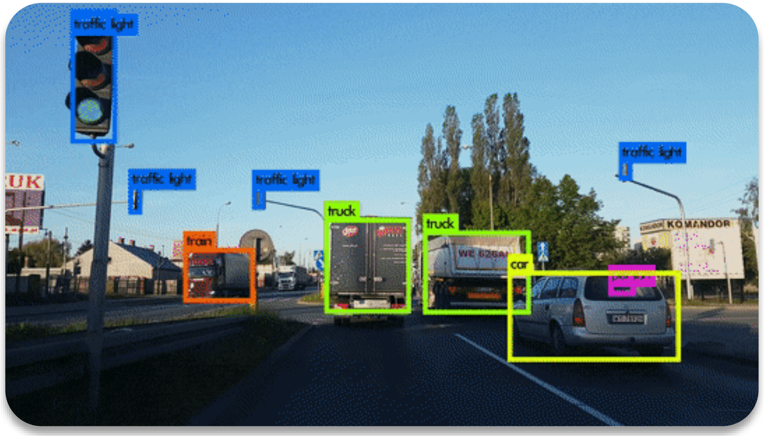
香港城市大學
City University of Hong Kong

Background - On-Device Federated Training

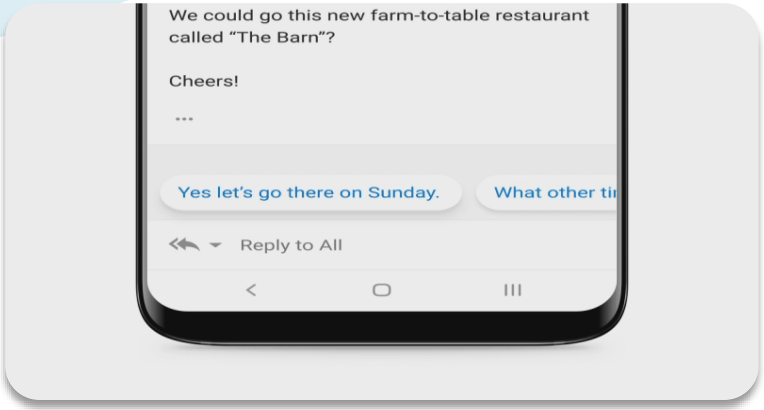




Speech Recognition

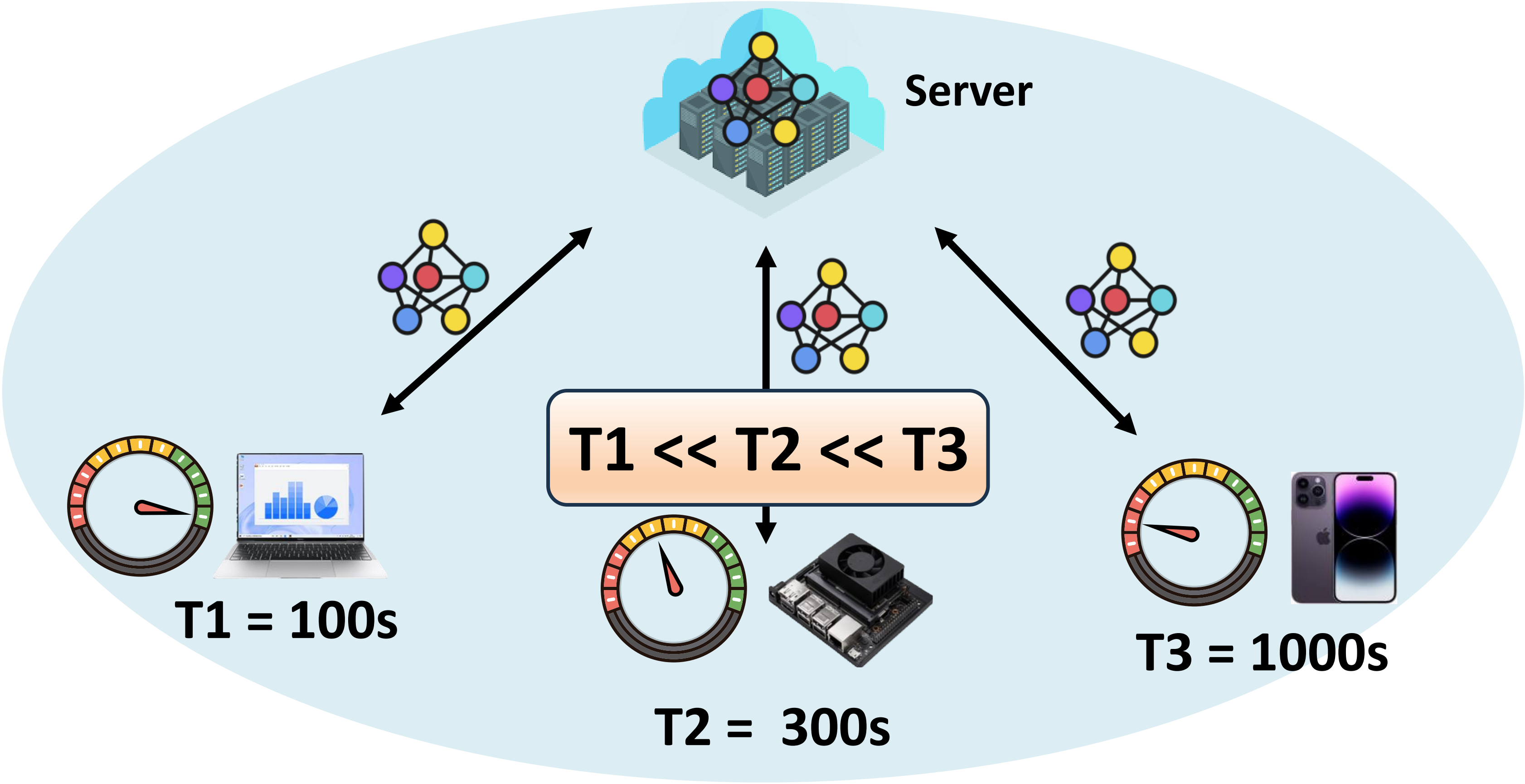


Autonomous Driving



Smart Reply helper

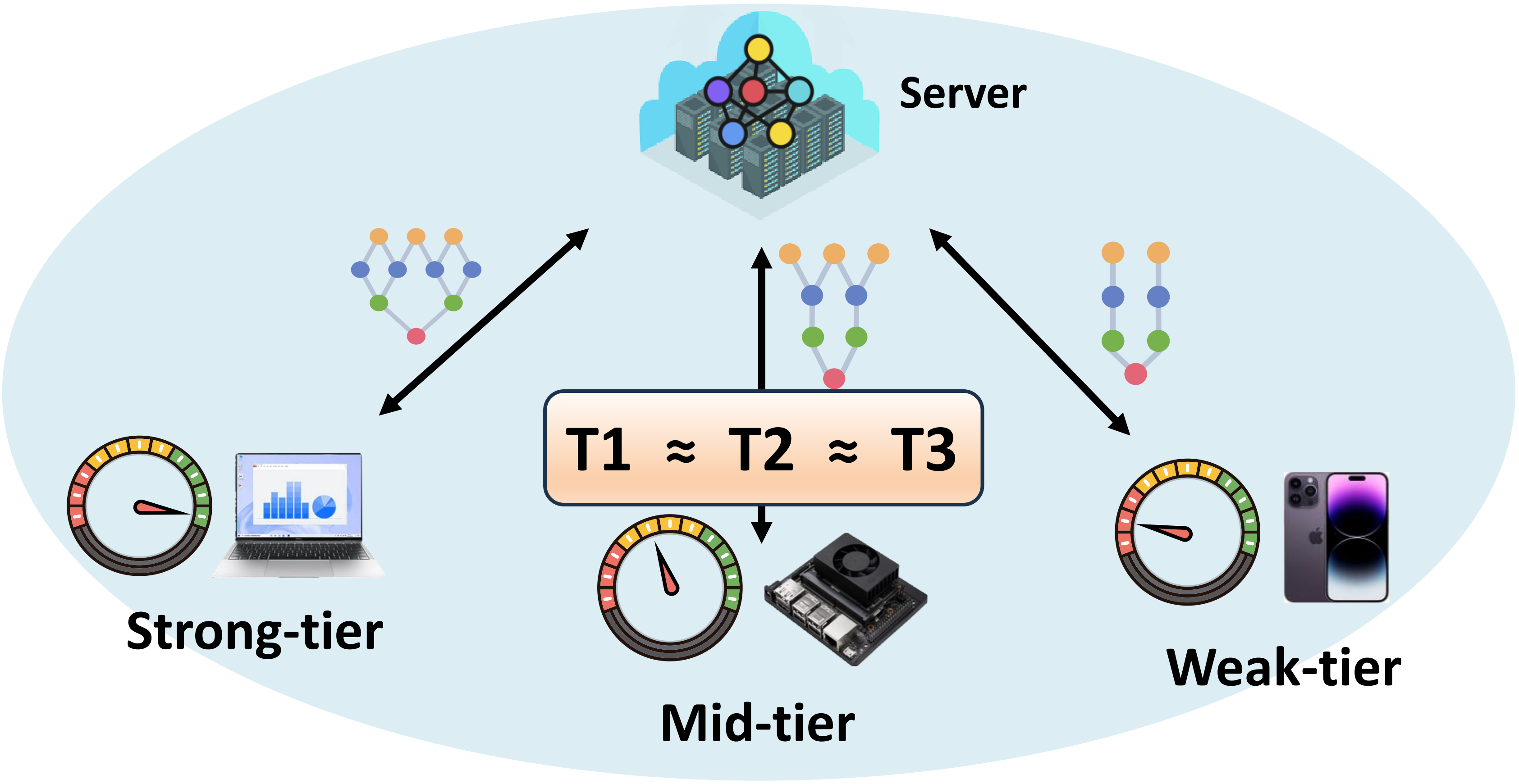
Huge training time gap between devices



Has low
FL efficiency

Stronger Devices need to wait for Weaker Devices

Idea - Heterogeneous Federated Learning

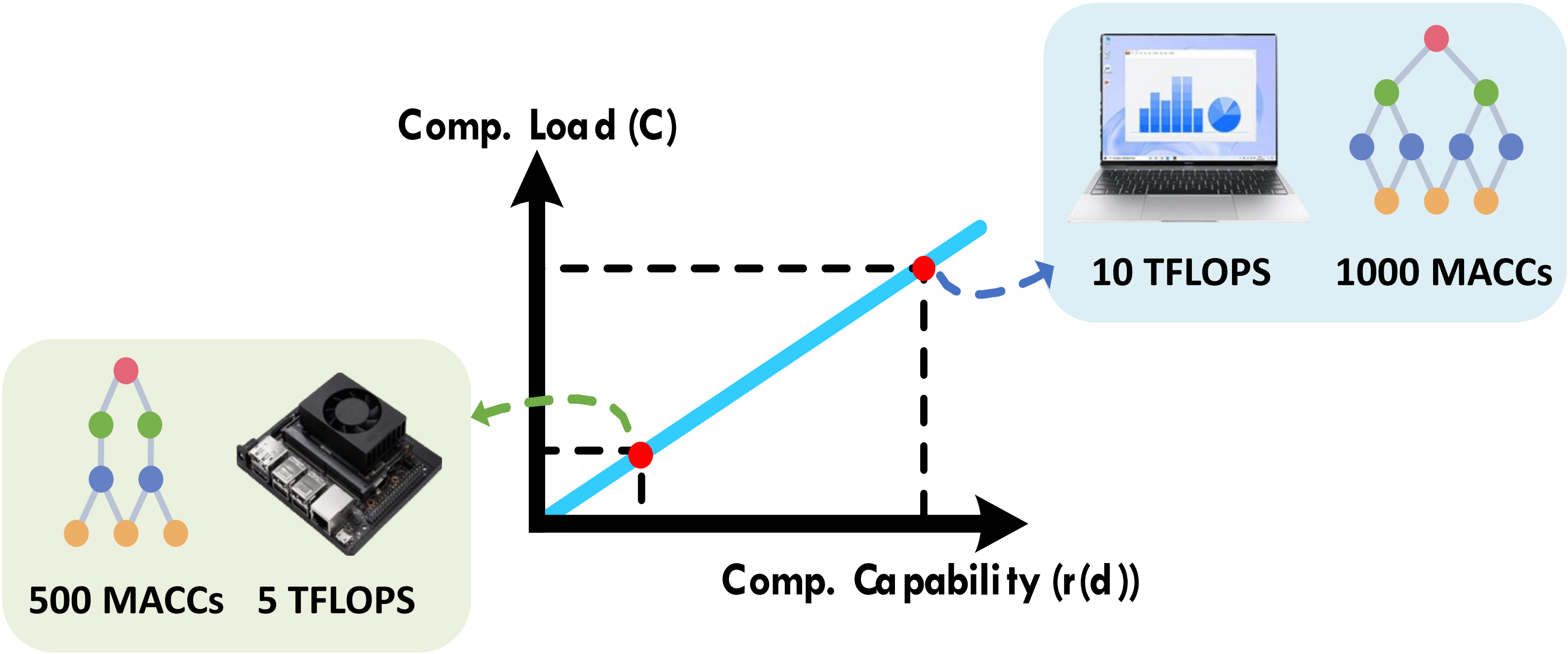


It was proved that can really improve FL efficiency !

Different devices but Similar Training Time

Recent work - Fine-grained method

$$T = \frac{C_1}{r(d_1)} = \frac{C_2}{r(d_2)}$$

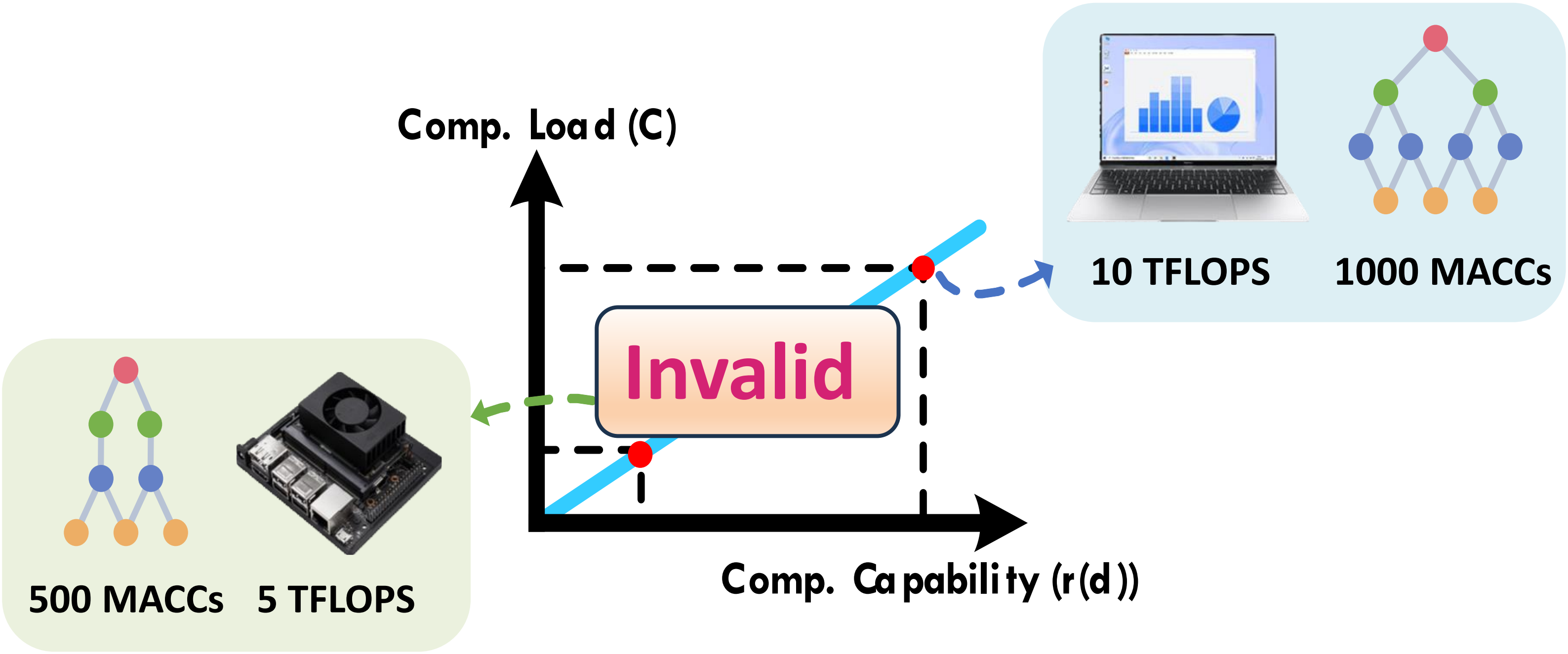


Allocating sub-models according

device's computing power (i.e., FLOPS)

Recent work - Fine-grained method

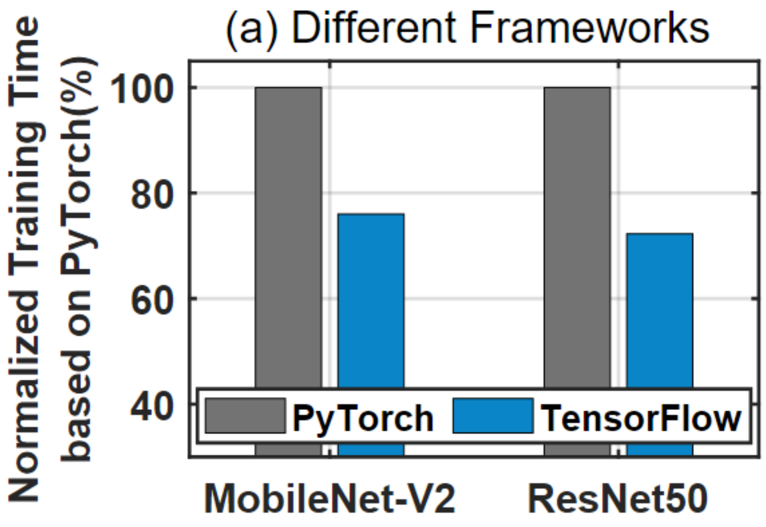
$$T = \frac{C_1}{r(d_1)} = \frac{C_2}{r(d_2)}$$

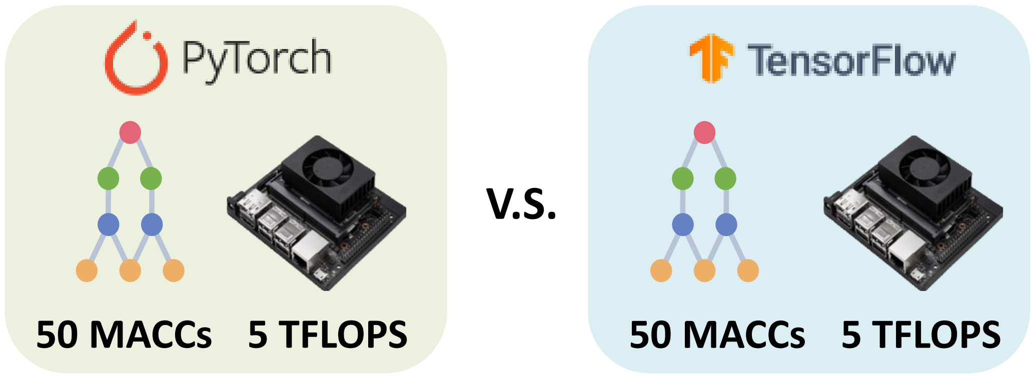


This training time modeling is still over-simple!

Problem 1 - Training Time Inconsistency

- **Even training same models in the same devices, training time has huge gap by using different DL frameworks.**





Training Time Inconsistency Problem

Key Observation - Layer Algorithm Diversity

- There are several **candidate layer algorithms implementation** in DL frameworks.

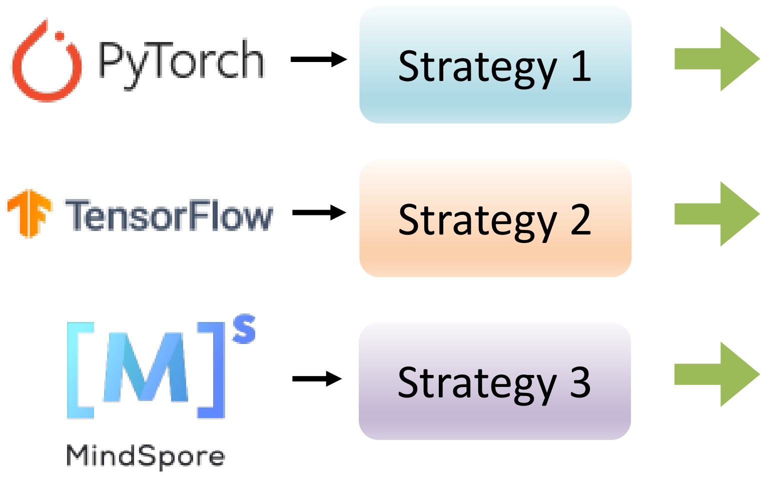
```
static const algo_t Fwd_algos[] = {
    CUDNN_CONVOLUTION_FWD_ALGO_GEMM,
```

```
static const algo_t Bwd_algos[] = {
    CUDNN_CONVOLUTION_BWD_FILTER_GEMM,
    CUDNN_CONVOLUTION_BWD_FILTER_FFT,
    CUDNN_CONVOLUTION_BWD_FILTER_FFT_TILING,
    CUDNN_CONVOLUTION_BWD_FILTER_IMPLICIT_GEMM,
    CUDNN_CONVOLUTION_BWD_FILTER_IMPLICIT_PRECOMP_GEMM,
    CUDNN_CONVOLUTION_BWD_FILTER_DIRECT,
    CUDNN_CONVOLUTION_BWD_FILTER_WINOGRAD,
    CUDNN_CONVOLUTION_BWD_FILTER_WINOGRAD_NONFUSED,
};
```

Analyzing source codes of DL framework’s training mechanism

Key Observation - Different Algorithm Selecting Strategy

- **Different DL frameworks may select different layer algorithms as implementation.**



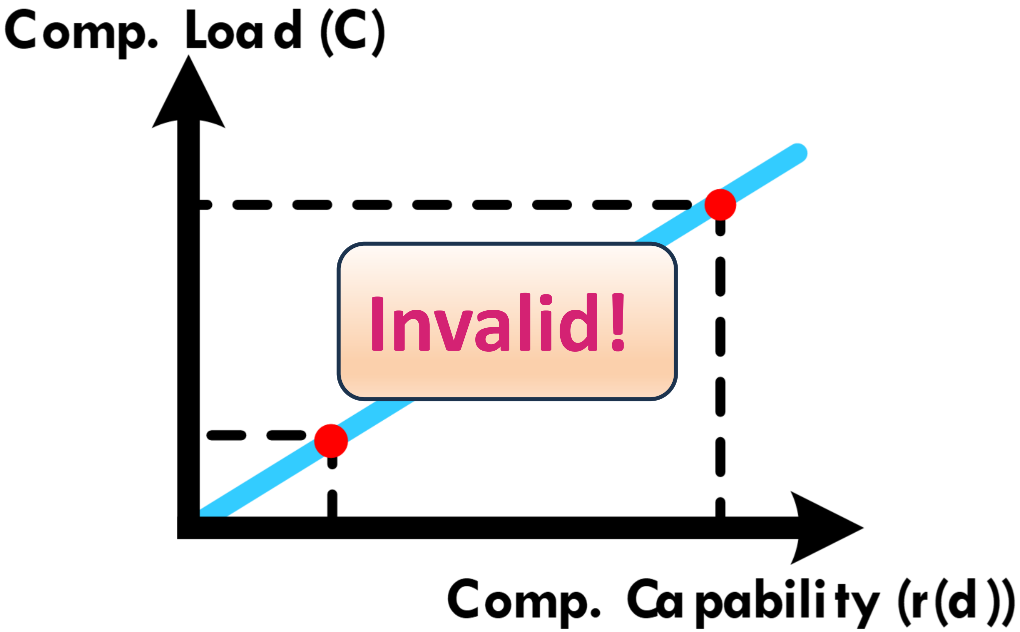
Different Convolution Layer Algorithms

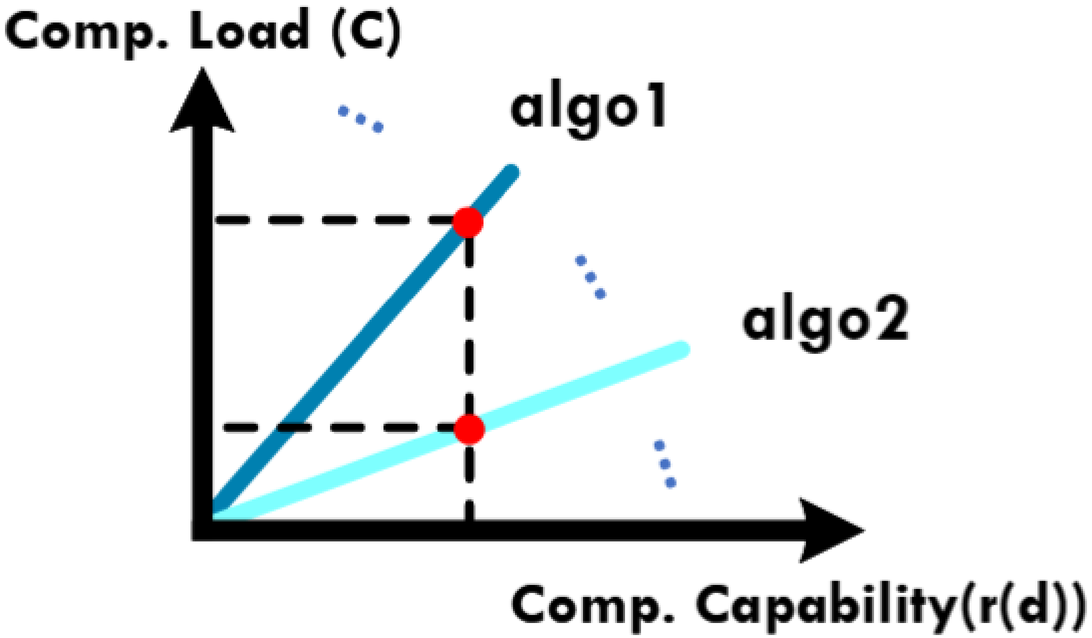
Layer Algorithms	Generality	Memory Efficiency
GEMM	+++	++
FFT	+	+++
FFT_TILING	++	++
IMPLICIT_GEMM	+++	+++
IMPLICIT_PRECOMP_GEMM	+++	++
DIRECT	++	+
WINOGRAD	+	+++
WINOGRAD_NONFUSED	++	++

Different layer algorithms have different computation workloads thus different training time.

Training Time Modeling Reformulation

$$T = \frac{C_1}{r(d_1)} = \frac{C_2}{r(d_2)} \quad \Rightarrow \quad T = \frac{C(algo_1)}{r(d_1)} = \frac{C(algo_2)}{r(d_2)}$$

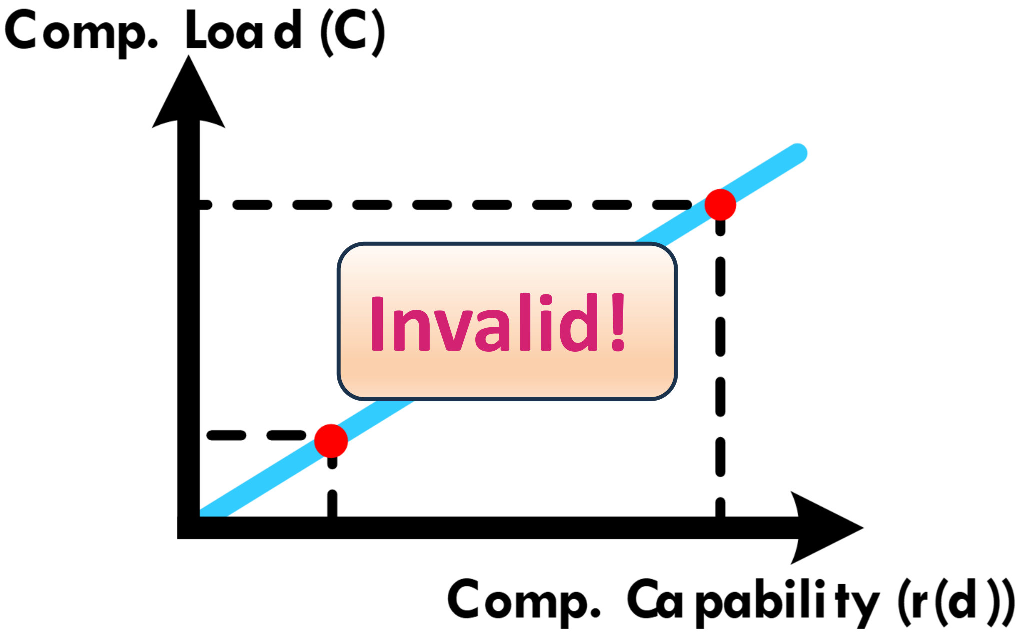


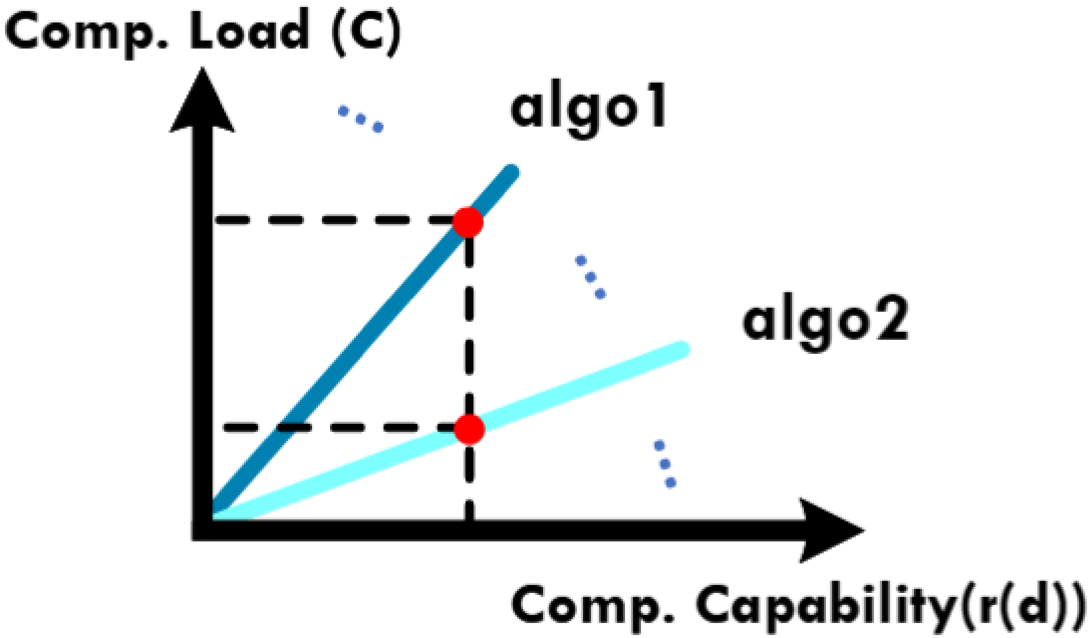


Accurate training time modeling need to consider the layer algorithms.

Training Time Modeling Reformulation

$$T = \frac{C_1}{r(d_1)} = \frac{C_2}{r(d_2)} \Rightarrow T = \frac{C(algo_1)}{r(d_1)} = \frac{C(algo_2)}{r(d_2)}$$





Now we indeed can allocate sub-models efficiently, however…

Problem 2 - Most Strategies are not optimal

PyTorch → Strategy 1

TensorFlow → Strategy 2

MindSpore → Strategy 3

Different Convolution Layer Algorithms

Layer Algorithms	Generality	Memory Efficiency
GEMM	+++	++
FFT	+	+++
FFT_TILING	++	++
IMPLICIT_GEMM	+++	+++
IMPLICIT_PRECOMP_GEMM	+++	++
DIRECT	++	+
WINOGRAD	+	+++
WINOGRAD_NONFUSED	++	++

Slower

Fastest

Slower

Most strategies selecting the sub-optimal algorithms as their layer implementation

Problem 2 - Most Strategies are not optimal

PyTorch → Strategy 1 → GEMM — Slower

TensorFlow → Strategy 2 → IMPLICIT_GEMM — Fastest

MindSpore → Strategy 3 → WINOGRAD — Slower

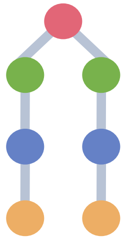
Different Convolution Layer Algorithms

Layer Algorithms	Generality	Memory Efficiency
GEMM	+++	++
FFT	+	+++
FFT_TILING	++	++
IMPLICIT_GEMM	+++	+++
IMPLICIT_PRECOMP_GEMM	+++	++
DIRECT	++	+
WINOGRAD	+	+++
WINOGRAD_NONFUSED	++	++

PyTorch

Algorithm_1

Slower



40 MACCs

Smaller sub-model

How to design a better selecting strategy?

PyTorch
TensorFlow
MindSpore

New Strategy

Different Convolution Layer Algorithms

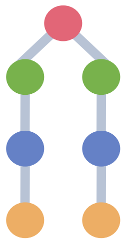
Layer Algorithms	Generality	Memory Efficiency
GEMM	+++	++
FFT	+	+++
FFT_TILING	++	++
IMPLICIT_GEMM	+++	+++
IMPLICIT_PRECOMP_GEMM	+++	++
DIRECT	++	+
WINOGRAD	+	+++
WINOGRAD_NONFUSED	++	++

✓ Fastest

PyTorch

Algorithm_1

✗ Slower



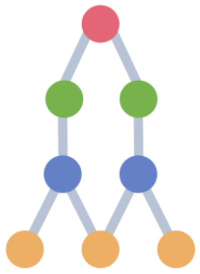
Smaller sub-model

40 MACCs

PyTorch

Algorithm_4

✓ Fastest



larger sub-model

60 MACCs

Observation - TensorFlow's Exhaustive Testing Strategy

TensorFlow

Exhaustive Testing

PyTorch

Heuristic & Blackbox

As input

13 dims Layer Configs

I/O_size

Filter

Stride

...

Padding

Layer Algorithms
GEMM
FFT
FFT_TILING
IMPLICIT_GEMM
IMPLICIT_PRECOMP_GEMM
DIRECT
WINOGRAD
WINOGRAD_NONFUSED

Always optimal

but slowly

Ranking →

Optimal Algorithm

output →

Sub-optimal Algorithm

Observation - PyTorch's Heuristic Blackbox Strategy

TensorFlow
Exhaustive Testing

Ranking →

Optimal Algorithm

Layer Algorithms
GEMM
FFT
FFT_TILING
IMPLICIT_GEMM
IMPLICIT_PRECOMP_GEMM
DIRECT
WINOGRAD
WINOGRAD_NONFUSED

PyTorch
Heuristic &
End-to-end

output →

Sub-optimal Algorithm

As input

13 dims Layer Configs
I/O_size
Filter
Stride
...
Padding

Only 74% optimal but fast

Thinking

- **How to design an accurate and fast algorithm selecting strategy?**

How to design a new strategy combine with all the advantages?

Slow but Accurate

TensorFlow

Exhaustive Testing

Fast but Inaccurate

PyTorch

Heuristic & End-to-end

Design 1 - Layer-Algorithm Selector

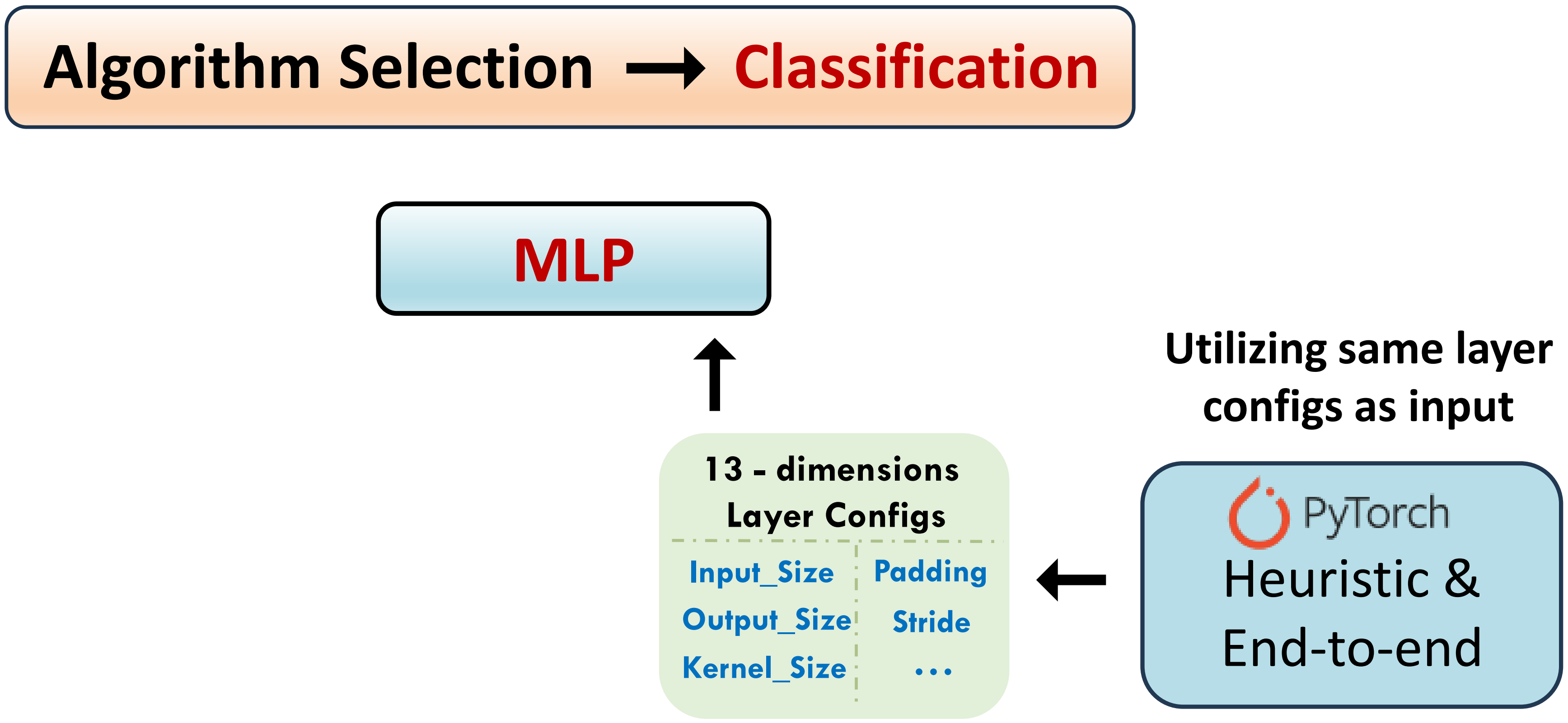
- **Data-driven based Layer-Algorithm Selector**

Algorithm Selection → Classification

MLP

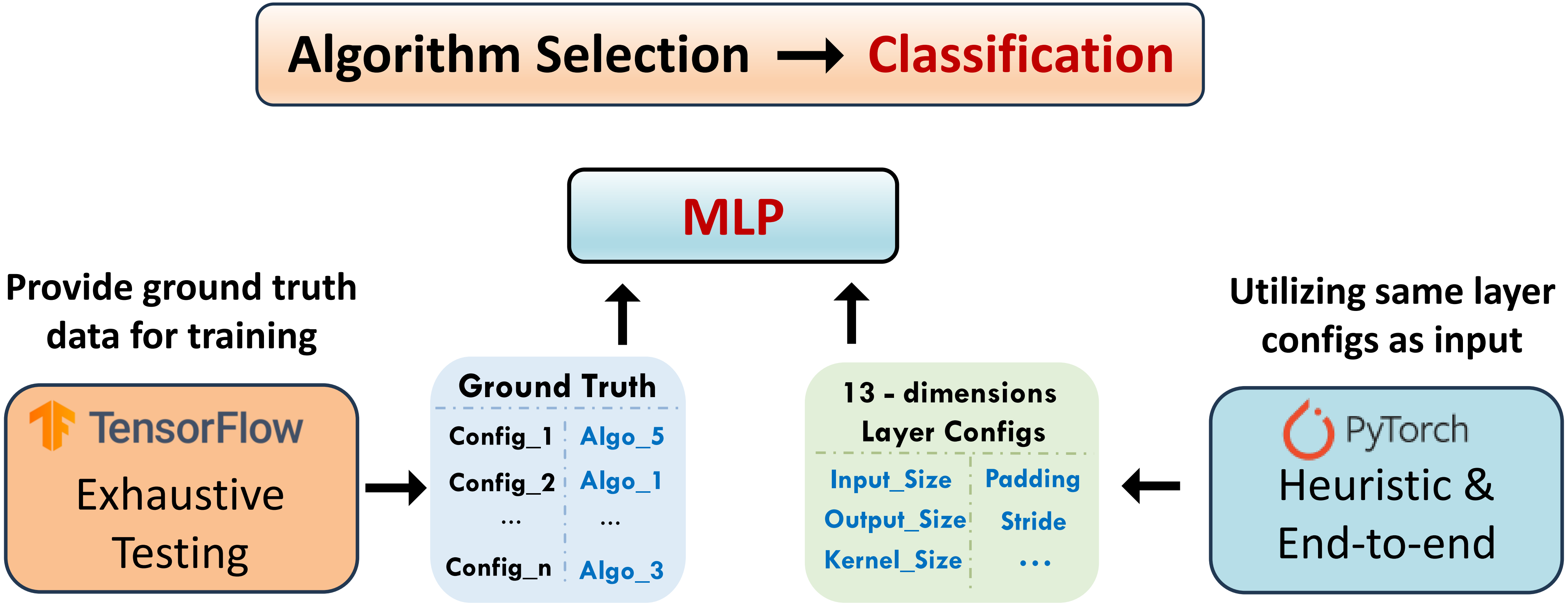
Design 1 - Layer-Algorithm Selector

- **Data-driven based Layer-Algorithm Selector**



Design 1 - Layer-Algorithm Selector

- **Data-driven based Layer-Algorithm Selector**

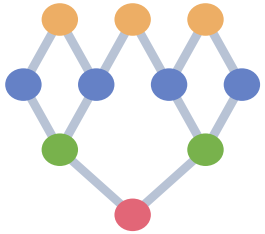


Design 2 - Training Time Estimator

- **Training Time Estimation by profiling**

Design 2 - Training Time Estimator

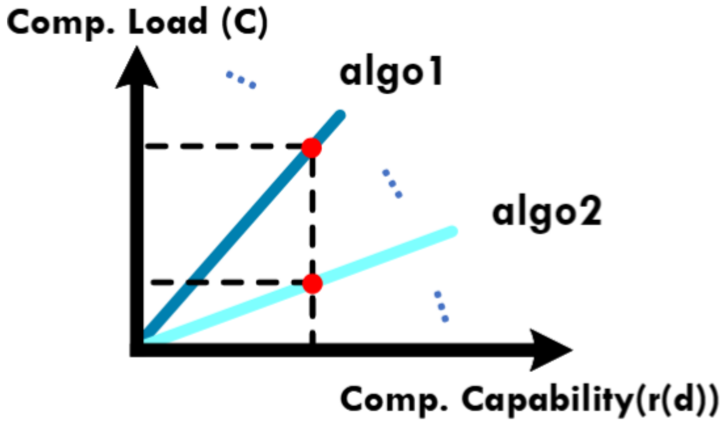
- **Training Time Estimation by profiling**



→ **MLP** → **Algotithm_4** →

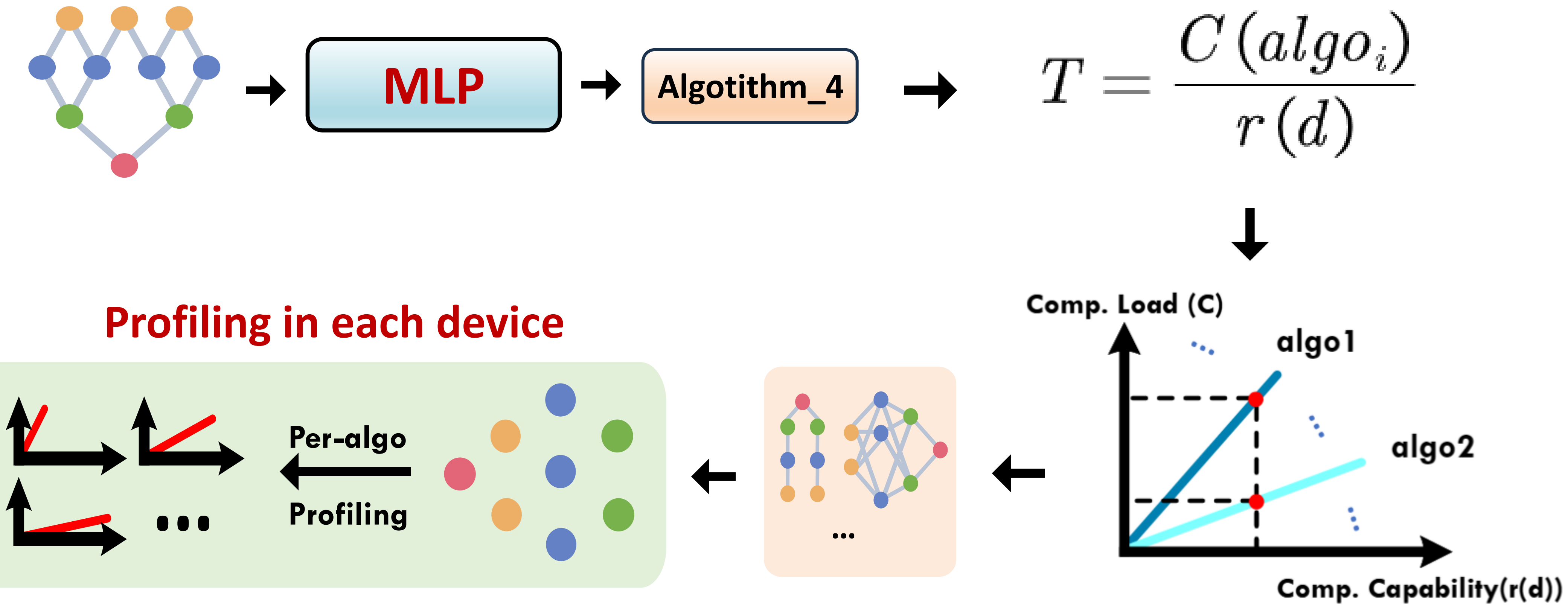
$$T = \frac{C\left(algo_i\right)}{r\left(d\right)}$$

↓

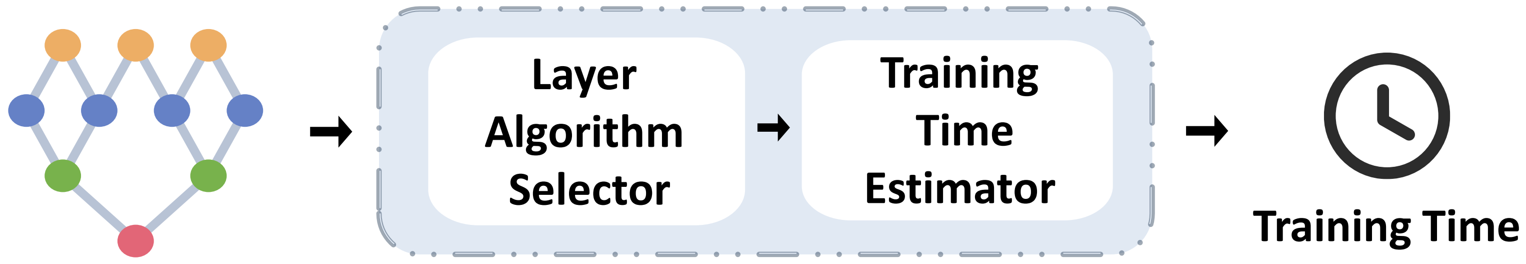


Design 2 - Training Time Estimator

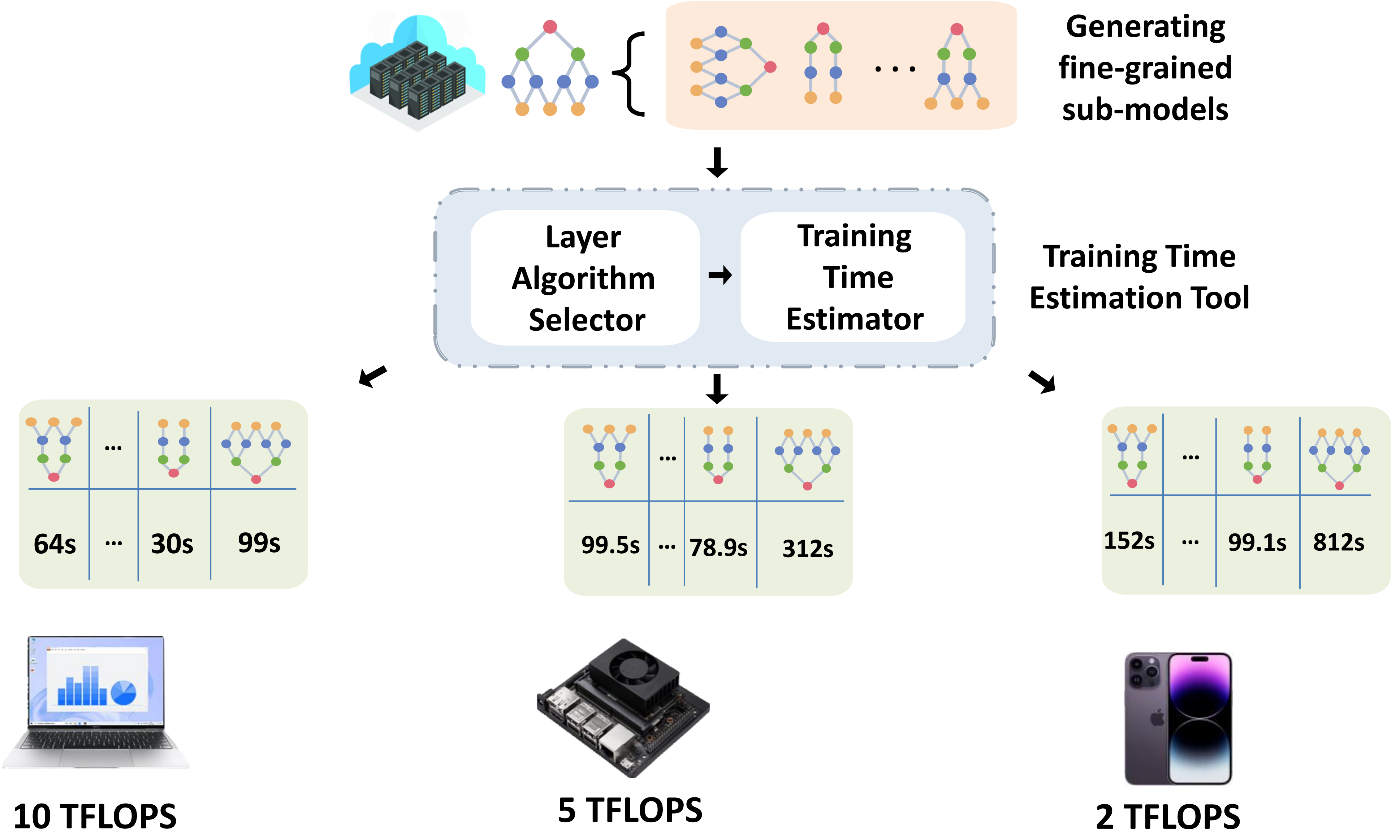
- **Training Time Estimation by profiling**



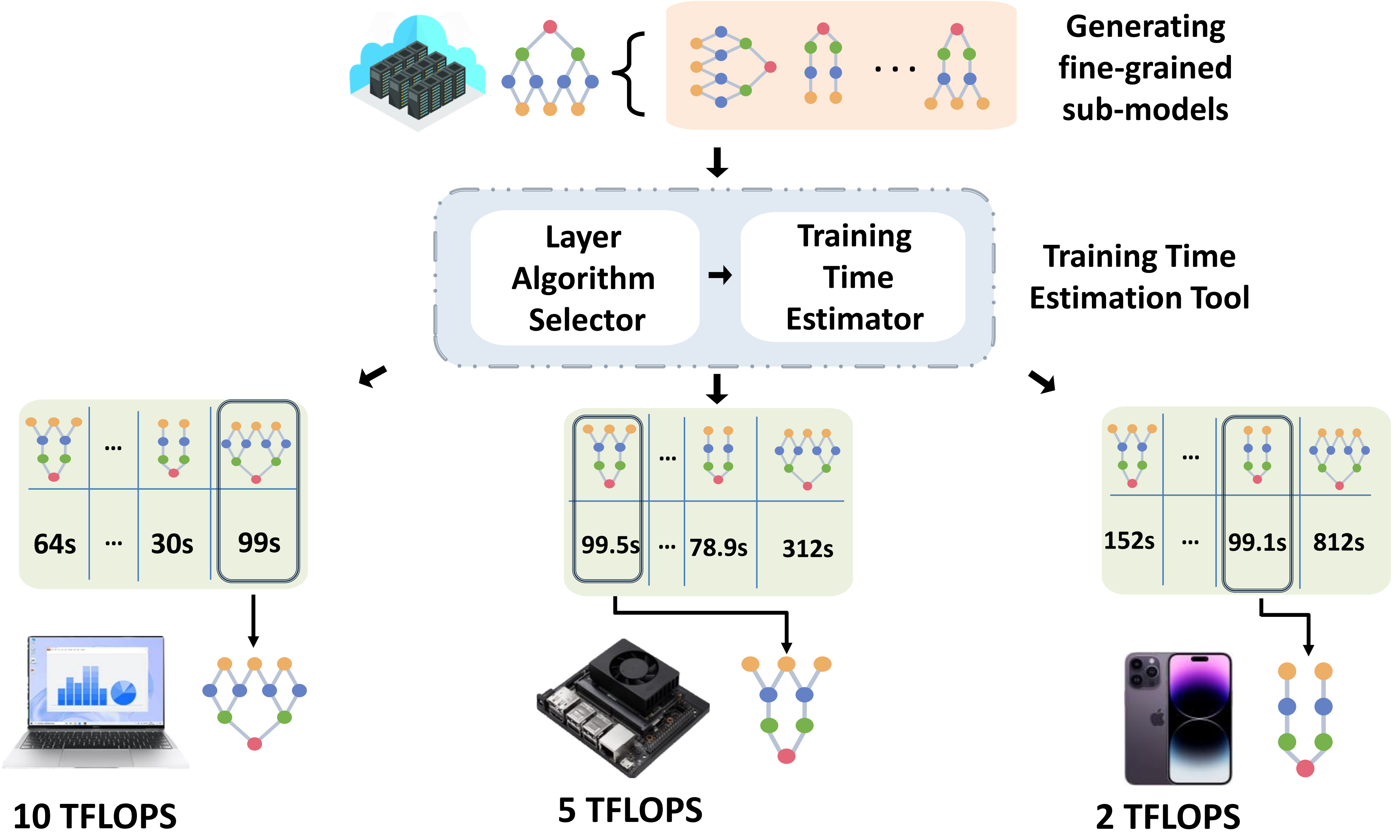
Put all pieces together



Put all pieces together

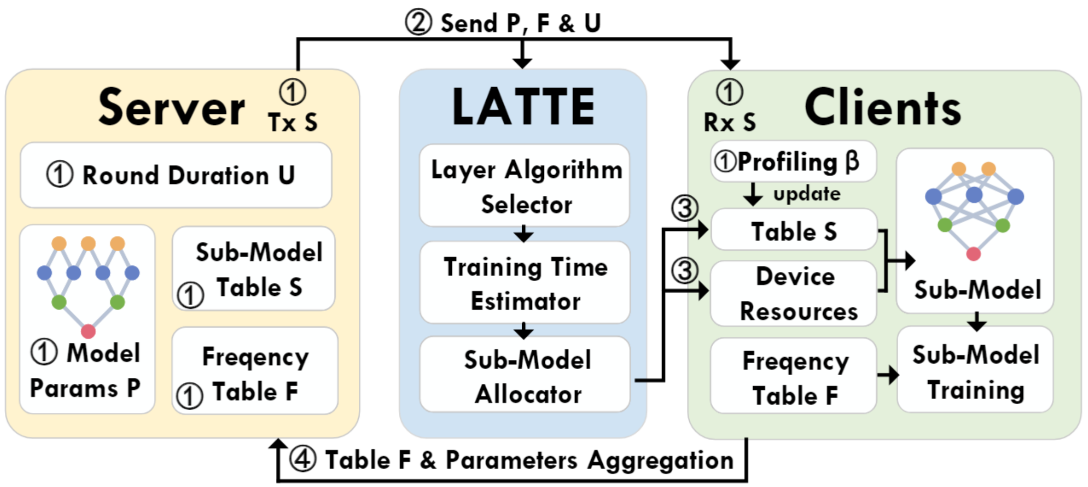


Put all pieces together

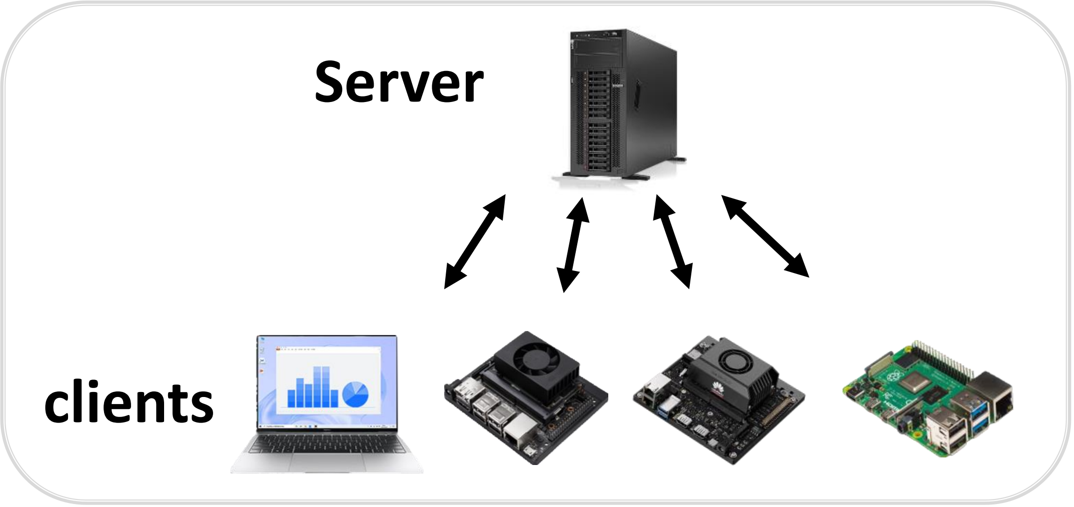


Implementation

Proposed LATTE Framework



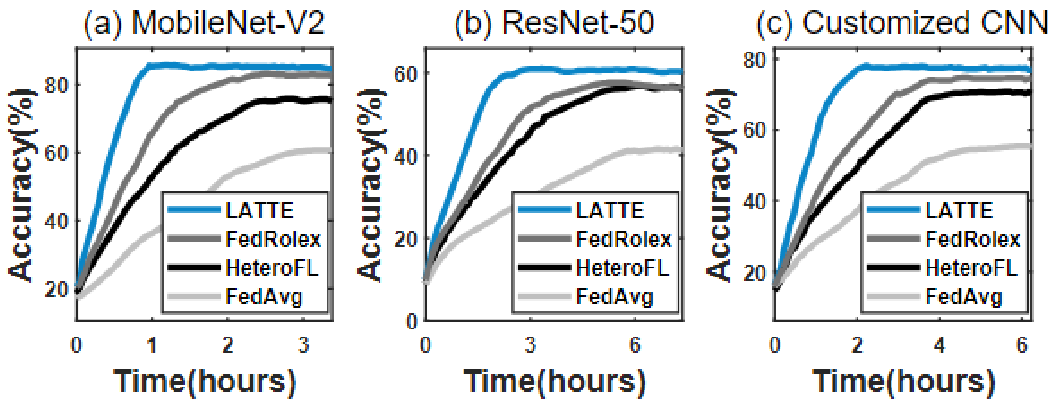
Test-bed



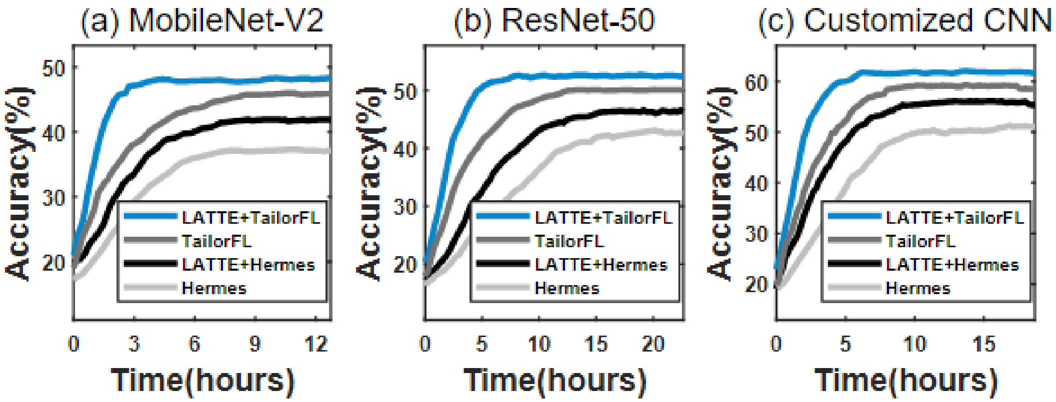
1. **Time-to-Convergence:** Represent whether our system can accelerate converge speed.
2. **Layer Algorithm Selector's accuracy:** Represent the Layer Algorithm Selector's performance.
3. **Training Time Estimator's Precision:** Represent the Training Time Estimator's performance.

Metrics

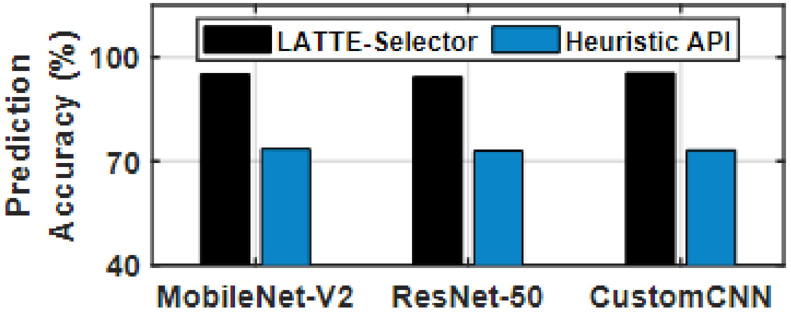
Evaluation



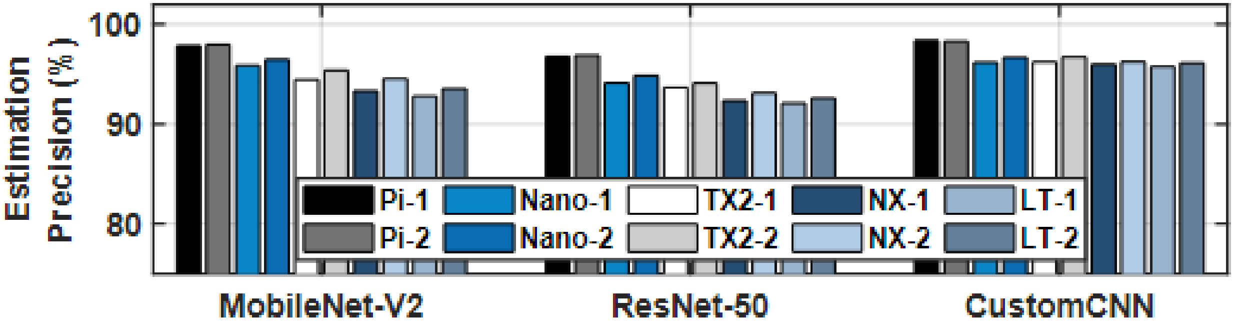
Compared with SOTA methods



Evaluating each components



Evaluating Selector's Performance



Evaluating Estimator's Performance

Conclusion

- We reveal the problem of development-chain diversity in federated learning systems and identify diverse layer algorithms as the key to explain the variability in training time. Based on this, accurate estimation of model training time can be achieved without complex operator or kernel-level modeling.

- We devise LATTE, with a novel layer algorithm selector and training time estimator, to accurately estimate the single-pass (forward/backward) propagation latency of a model given its architecture, expected hardware and runtime memory. We further showcase its usability in a client-side sub-model selection for HFL

- We conduct extensive experiments to evaluate LATTE in five typical HFL scenarios. The results show significant improvements in performance compared to seven classical or state-of-the-art methods.