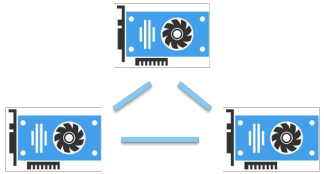
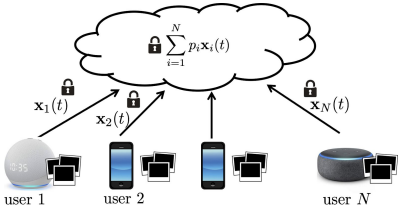
USC University of Southern California

Distributed ML System for Large-scale Models: *Dynamic* Distributed Training and *Scalable* Federated Learning



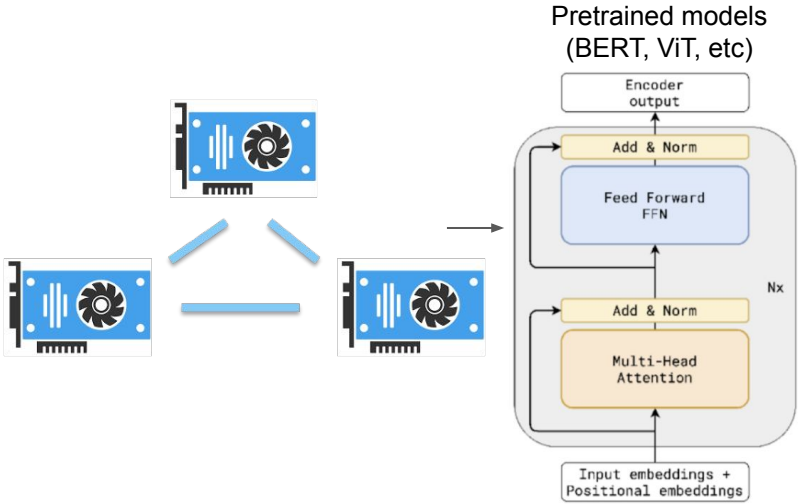


Chaoyang He
PhD student (2018-present), CS, USC
Former R&D Manager and Staff Software Engineer at Tencent

Mahdi Soltanolkotabi
Associate Professor
CS & ECE, USC

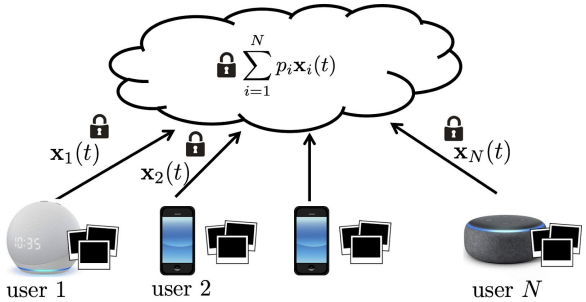
Salman Avestimehr
Professor, ECE&CS, USC
Director, USC-Amazon Trusted ML Center

Fundamental Research to Alexa AI Applications



Pretrained models
(BERT, ViT, etc)

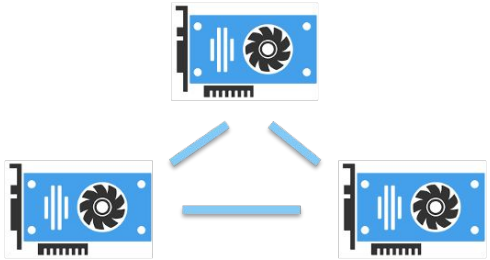
Model Compression for edge devices

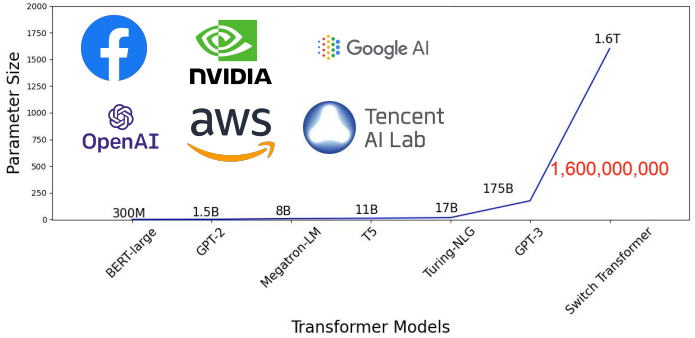


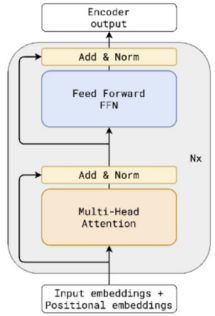
1. Distributed Training in GPU cluster

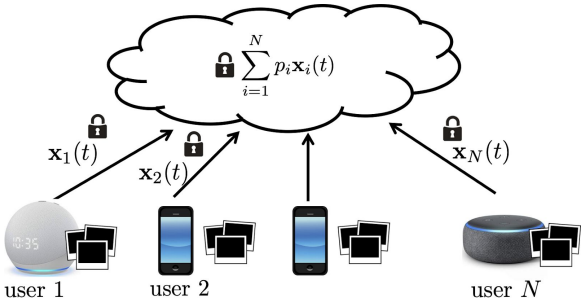
2. Federated Learning at the edge

Distributed Training v.s. Federated Learning





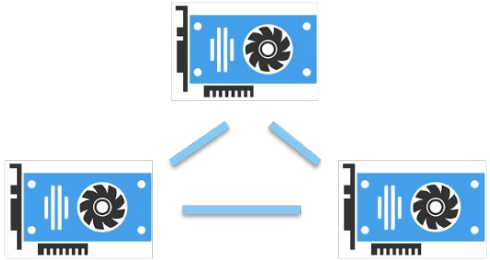




In the Federated Learning setting, the data is distributed across millions of devices in a highly uneven fashion. These devices have low computational compatibility, significantly higher-latency, lower-throughput connections

--- Federated Learning and Open Problems, Foundations and Trends in Machine Learning (FnTML) 2021; …, Chaoyang He, and many Googlers

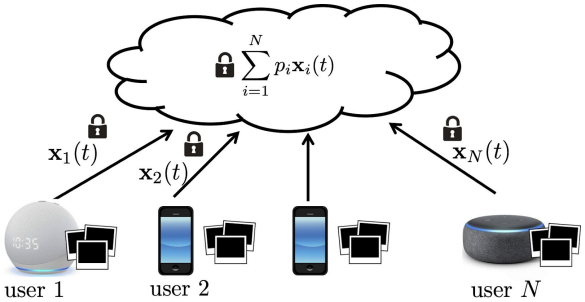
Distributed Training v.s. Federated Learning



Goals: training large model in reasonable time, energy cost, and hardware resources to obtain a higher accuracy

Challenge:

1. communication cost
2. memory constraints
3. computational efficiency
4. straggler/failure

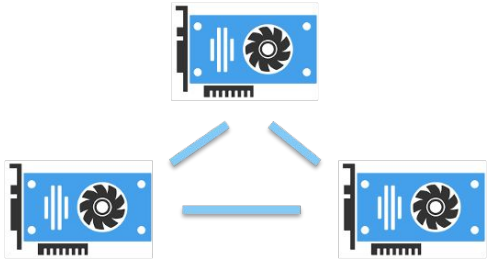


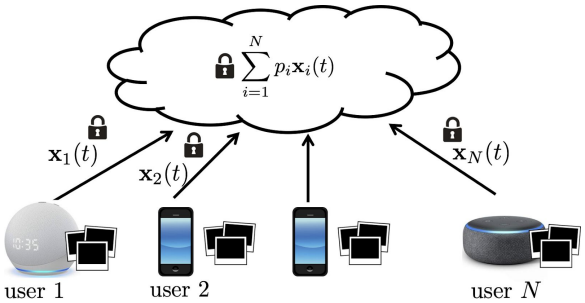
Goals: data and model privacy; mitigate the difficulty of data centralization; ubiquitous ML in 5G/IoT future.

Challenges:

1. data heterogeneity and label deficiency
2. system heterogeneity; resource-constraint; scalability
3. trustworthiness: security, privacy, fairness, personalization, interpretability, etc

Distributed Training v.s. Federated Learning





In essence, three methodologies are useful to both:

1. Dynamic system and stochastic ML lead to complicated interaction between these objectives.

PipeTransformer, ICML 2021

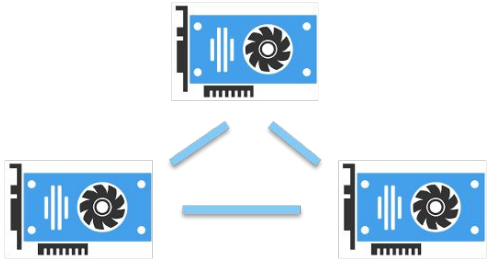
2. On demand: focus on the key demand for different use cases (ML applications), optimizing one of the objectives but not deteriorating the others.

FedML Lib and Ecosystem, NeurIPS 2020 FL workshop, Best Paper Award

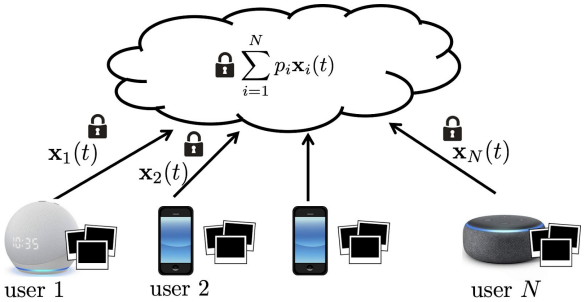
3. Trade-off among multiple objectives including accuracy, efficiency, security, and privacy.

Group Knowledge Transfer, NeurIPS 2020

Outline: Distributed ML for Large-scale Models



- **Part 1: Dynamic Distributed Training**
  PipeTransformer: Automated Elastic Pipelining for Distributed Training of Large-scale Models. ICML 2021
  https://DistML.ai



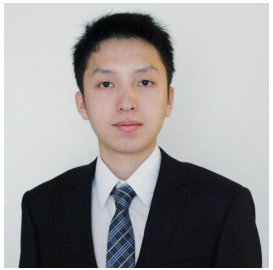
- **Part 2: Scalable Federated Learning**
  [Algorithm] Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge. NeurIPS 2020

  [System] FedML: A Research Library and Benchmark for Federated Machine Learning
  (Best Paper Award, NeurIPS 2020 FL Workshop)

  [Diverse Applications]
  FedML Ecosystem: Ubiquitous Distributed Training for Diverse AI Applications at the Edge.
  https://FedML.ai

PipeTransformer: Automated Elastic Pipelining for Distributed Training of Large-scale Models

Chaoyang He
PhD student (2018-present), CS, USC
Former R&D Manager at Tencent
Researcher, Tencent AI Lab



Shen Li
Research Scientist, Facebook AI
Team Lead, PyTorch Distributed
CS PhD, UIUC

Mahdi Soltanolkotabi
Associate Professor
CS, ECE, USC

Salman Avestimehr
Professor, ECE&CS, USC
Director, USC-Amazon ML Center

https://DistML.ai
https://chaoyanghe.com/pipetransformer/

August 18, 2021

PipeTransformer: Automated Elastic Pipelining for Distributed Training of Large-scale Models

by Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr

In this blog post, we describe the first peer-reviewed research paper that explores accelerating the hybrid of PyTorch DDP (`torch.nn.parallel.DistributedDataParallel`) [1] and Pipeline (`torch.distributed.pipeline`) - PipeTransformer: Automated Elastic Pipelining for Distributed Training of Large-scale Models (Transformers such as BERT [2] and ViT [3]), published at ICML 2021.

PipeTransformer leverages automated elastic pipelining for efficient distributed training of Transformer models. In PipeTransformer, we designed an adaptive on-the-fly freeze algorithm that can identify and freeze some layers gradually during training and an elastic pipelining system that can dynamically allocate resources to train the remaining active layers. More specifically, PipeTransformer automatically excludes frozen layers from the pipeline, packs active layers into fewer GPUs, and forks more replicas to increase data-parallel width. We evaluate PipeTransformer using Vision Transformer (ViT) on ImageNet and BERT on SQuAD and GLUE datasets. Our results show that compared to the state-of-the-art baseline, PipeTransformer attains up to 2.83-fold speedup without losing accuracy. We also provide various performance analyses for a more comprehensive understanding of our algorithmic and system-wise design.

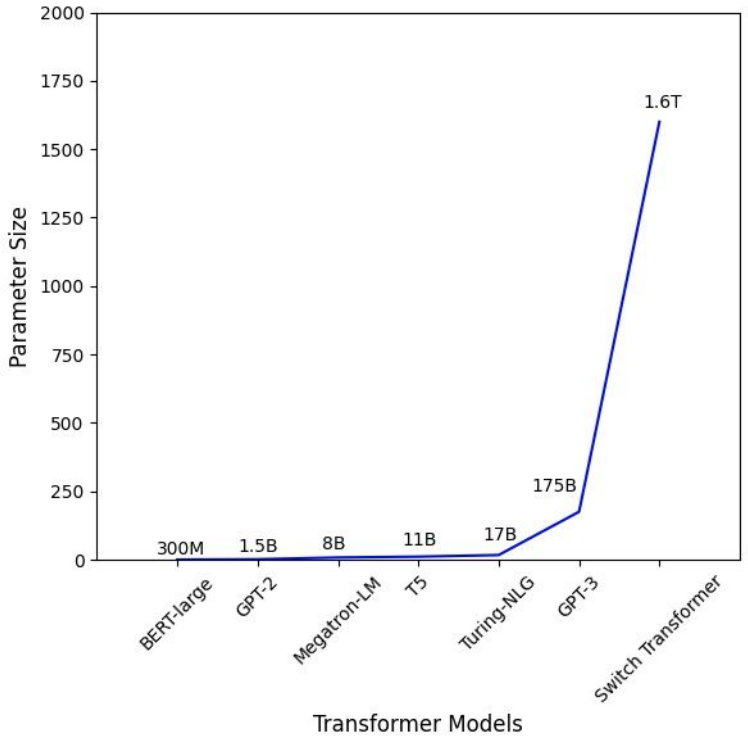
Next, we will introduce the background, motivation, our idea, design, and how we implement the algorithm and system with PyTorch Distributed APIs.

https://pytorch.org/blog/pipetransformer-automated-elastic-pipelining/

Outline

- Background and Related Works
- Motivation and Ideas
- Overall Design (Animation)
- AutoFreeze: Freeze Algorithm
- AutoPipe: Elastic Pipelining
- AutoDP: Spawning More Pipeline Replicas
- AutoCache: Cross-pipeline Caching
- Experimental Results
- Future Works

Background



After 2021-06: 100 Trillion?

The parameter number of deep neural networks (Transformers) is dramatically increasing!

Background

ICML International Conference On Machine Learning

USC University of Southern California

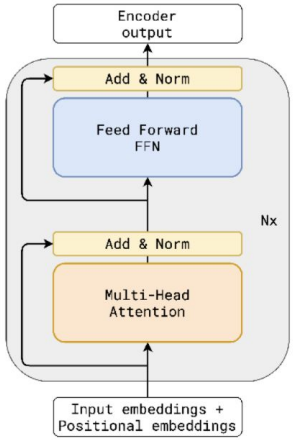
On the Opportunities and Risks of Foundation Models

Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles Hamed Nilforoshan
Julian Nyarko Giray Ogut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*[1]

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

Background

The cost of Search/Translation models (1/3)

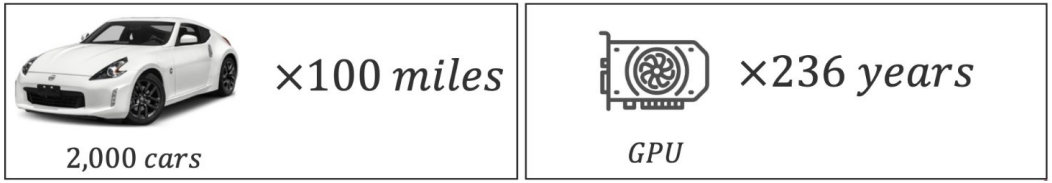


$2.3B$ *parameters*

"Our best quality dense single Transformer model (2.3B parameters) … was trained for 235.5 TPU v3 core-years."
– 2021 paper from openreview.net
- "TPU v3 core": equivalent to a GPU, uses 100W

2.3B transformer model costs 206,000 kWh…
… takes 236 years to train on one GPU

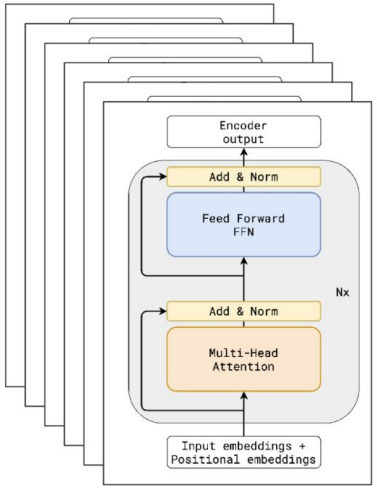
206,000 kWh equals …
- Driving 2,000 electric cars to another state (300 miles)
- Round-trip by 2,000 gas cars to another city (100 miles)



*Note: This page is from AAAI 2021 Tutorial (https://sites.google.com/view/aaai-2021-tutorial-ah9/home)

Background

The cost of Search/Translation models (2/3)



retrained 100 times

Models re-trained 100s of times before service

- R&D – test designs
- Hyperparameter optimization
- Upgrade with newer data

100x retraining of 2.3B model uses 20 gWh…
…takes 23.6 millennia to train on one GPU

20 gWh equals …

- Round-trip by 200,000 gas cars to another city (100 miles)
- Yearly electricity of 2,000 USA homes

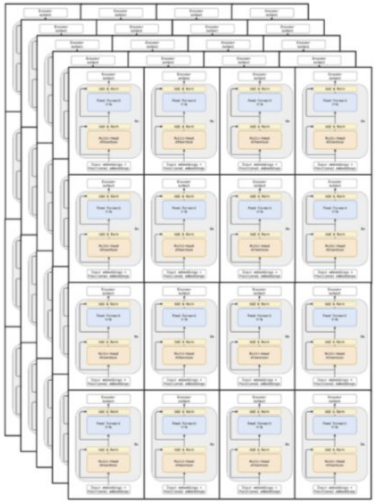
2,000 *homes* ×365 *days*

GPU ×23.6 *millennia*

*Note: This page is from AAAI 2021 Tutorial (https://sites.google.com/view/aaai-2021-tutorial-ah9/home)

Background

The cost of Search/Translation models (3/3)



scaled to 175B parameters
retrained 100 times

2.3B is tiny vs. recent models @ Google, OpenAI, …
- OpenAI GPT3 – 175B parameters
- Google model under open review – 600B parameters

Extrapolating – 100x retraining of 175B model costs 1500 gWh…
…and would have taken 1,800 millennia to train on one GPU
…except you can't fit 175B parameters on one GPU's memory!

We can't avoid distributed computing
Yet we need to reduce costs to be realistic & responsible

1500 gWh equals …
- Yearly home electricity of small city (150,000 homes)

an entire city ×365 *days*

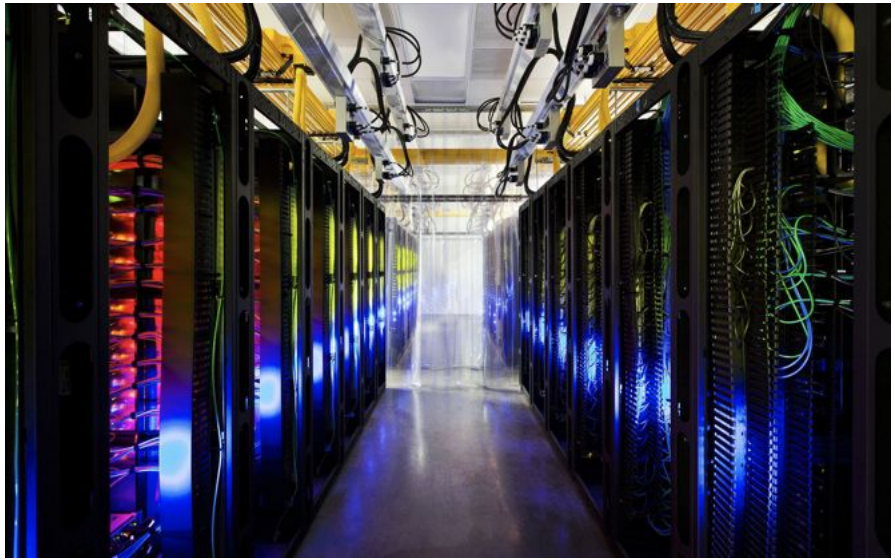
GPU ×1800 *millennia*

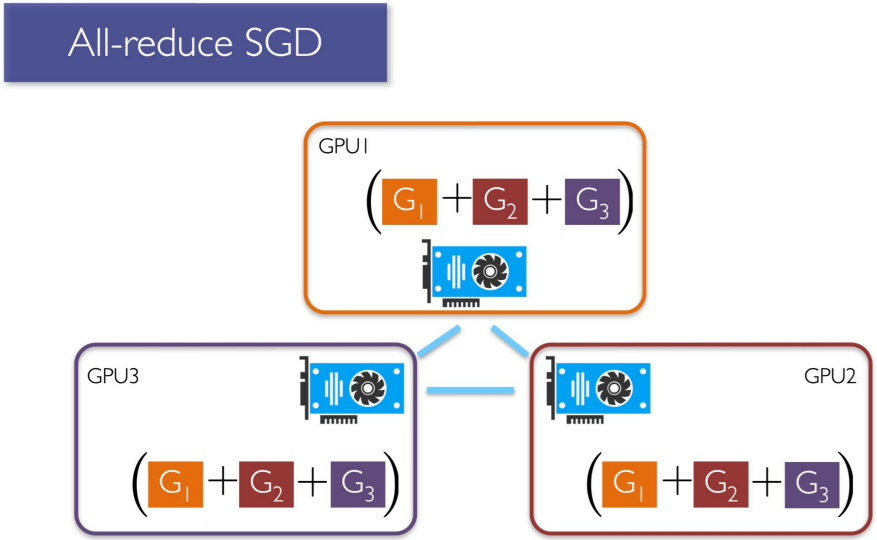
Out of Memory!

*Note: This page is from AAAI 2021 Tutorial (https://sites.google.com/view/aaai-2021-tutorial-ah9/home)

Background

USC University of Southern California



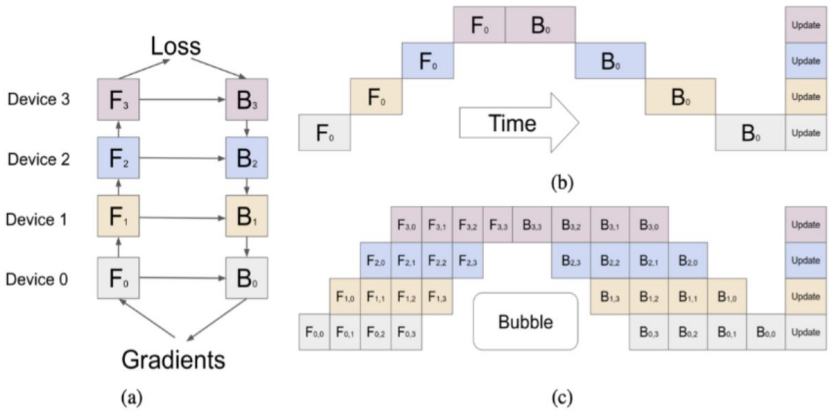


Related Works

1. System-wise: Distributed Training System Design and Optimization

Data and Model Parallel	**Data Inter-Batch:** ByteScheduler (SOSP'19) Crossbow (VLDB'19)	**Data Intra-Batch:** PT Pipe PT DDP (VLDB'20) GPipe (NeurIPS'19)
Model Inter-Operator PT RPC TF + gRPC (EuroSys'19)	PipeDream (SOSP'19) HetPipe (ATC'20)	PT RPC + DDP PT RPC + Pipe Parallax (EuroSys'19) BytePS (OSDI'20)
Model Intra-Operator Mesh-TF (NeurIPS'18) TouFu (EuroSys'19)		FlexFlow (MLSys'19) GPT-3 (NeurIPS'20) ZeRO/DeepSpeed (SC'20)

Related Works



Pipeline Parallelism

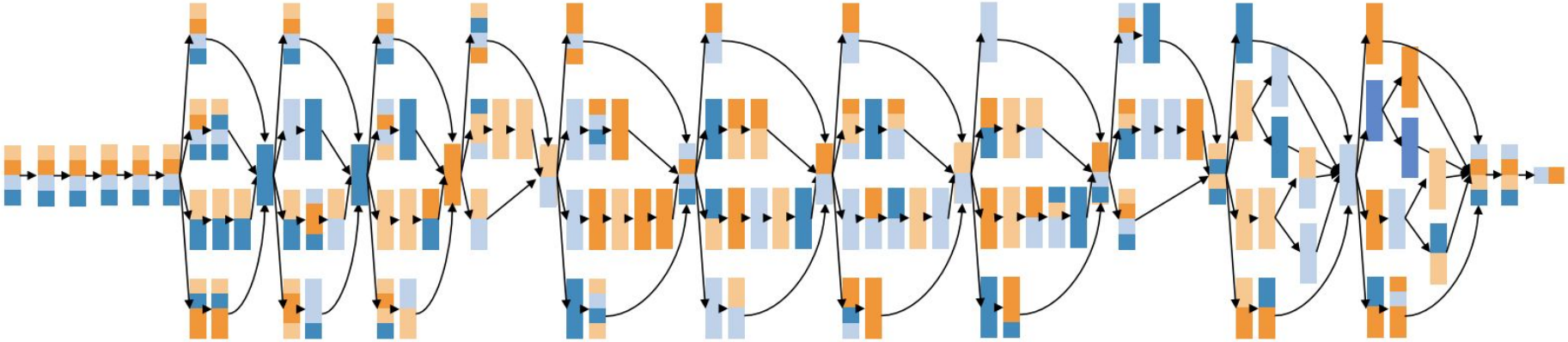
Hybrid of Data Parallelism and Pipeline Parallelism

Related Works

ICML International Conference On Machine Learning

USC University of Southern California

Model Parallelism

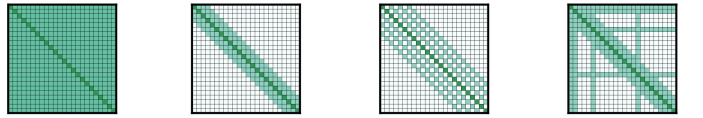


Intra-layer and inter-layer model parallelism.
https://FlexFlow.ai

Related Works

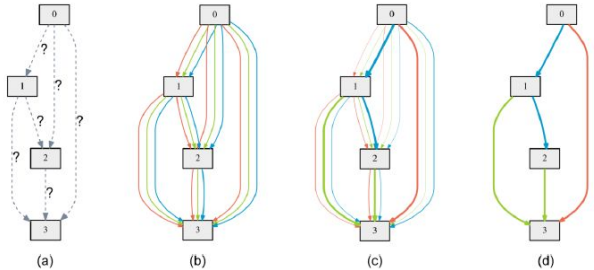
2. ML-wise: Model Architecture and Training Algorithm

1) **Architecture Optimization Manually:**
LinFormer (AAAI'2021, Best Paper Award)


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

2) **Automated Architecture Design:**
Neural Architecture Search: FBNet (CVPR' 2019, 400+ citations)

3) **Spare Training: pruning, quantization, etc**
Lottery Ticket Hypothesis (ICLR 2019, Best Paper Award)


(a) (b) (c) (d)

4) **Progressive Training**

5) **SGD-based Distributed Optimization:**
LARS (ICLR 2020): Large Batch Optimization for Deep Learning: Training BERT in 76 minutes

...

Our Motivation and Idea

1. (Sys) Distributed Training System

Pro:
Efficiency in Computation/Communication/Memory

Con:
View the model/SGD optimization as black box

2. (ML) Model Architecture and Training Algorithm

Pro:
Improve the efficiency mathematically, fundamentally

Con:
1. Lack of system design to amplify the algorithmic advantages
2. the model-wise optimization is not friendly to distributed training

What if we co-design?

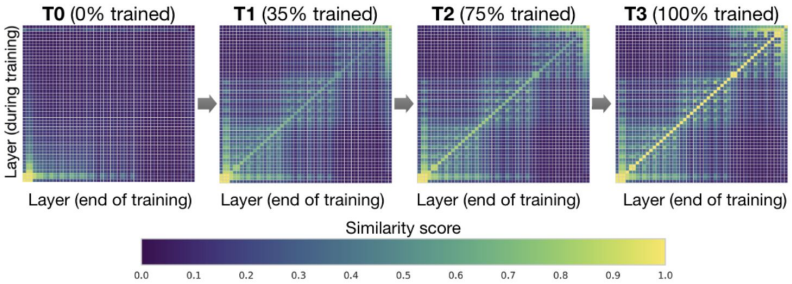
Hybrid of Pipeline and Data Parallelism

1. **Progressive Training**
2. **Dynamic Neural Networks**

(https://arxiv.org/pdf/2102.04906.pdf)

Elastic Distributed Training System!

Progressive Training



Freeze Training [1]

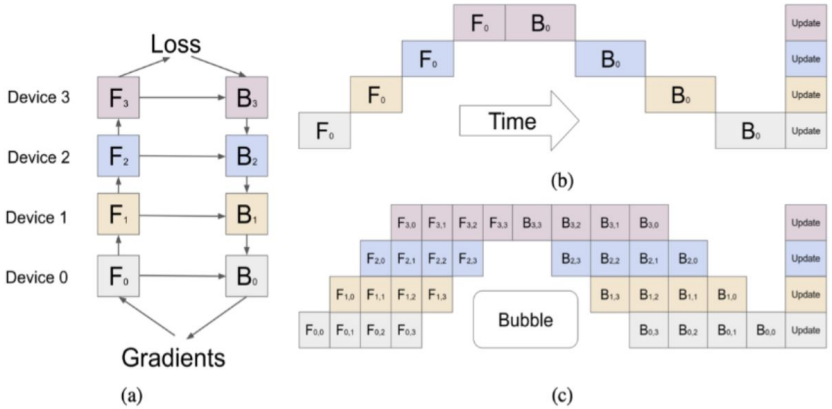
Algorithm 1 Progressive stacking

$M_0' \leftarrow \text{InitBERT}(L/2^k)$
$M_0 \leftarrow \text{Train}(M_0')$ {Train from scratch.}
for $i \leftarrow 1$ to k **do**
  $M_i' \leftarrow \text{Stack}(M_i)$ {Doubles the number of layers.}
  $M_i \leftarrow \text{Train}(M_i')$ {M_i has $L/2^{k-i}$ layers.}
end for
return M_k

Progressive Stacking [2]

[1] Freeze Training: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. NeurIPS 2017
[2] Efficient Training of BERT by Progressively Stacking. ICML 2019
[3] Accelerating Training of Transformer-Based Language Models with Progressive Layer Dropping. NeurIPS 2020. Minjia Zhang
[4] On the Transformer Growth for Progressive BERT Training. NACCL 2021

Distributed Training System



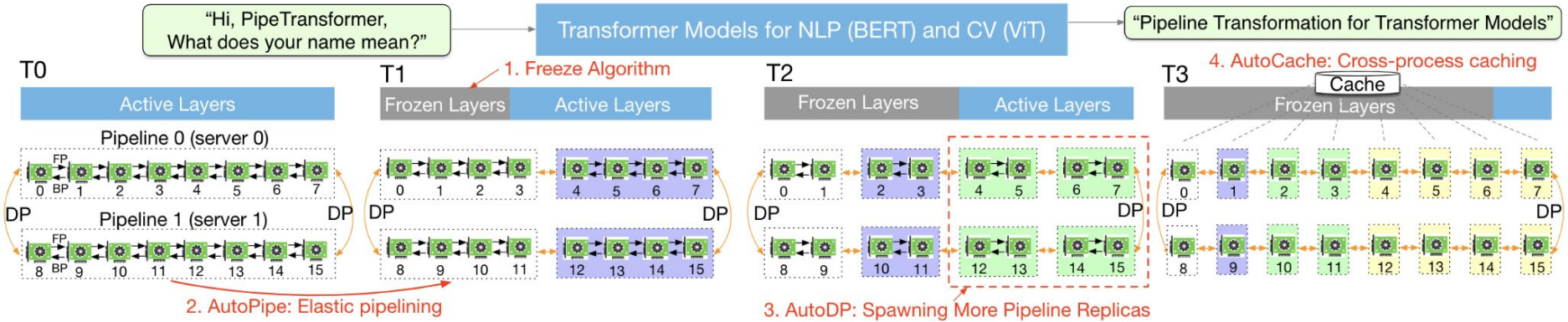
Pipeline Parallelism

Hybrid of Data Parallelism and Pipeline Parallelism

Key observations when applying progressive training (e.g., freeze training) to the above training systems:

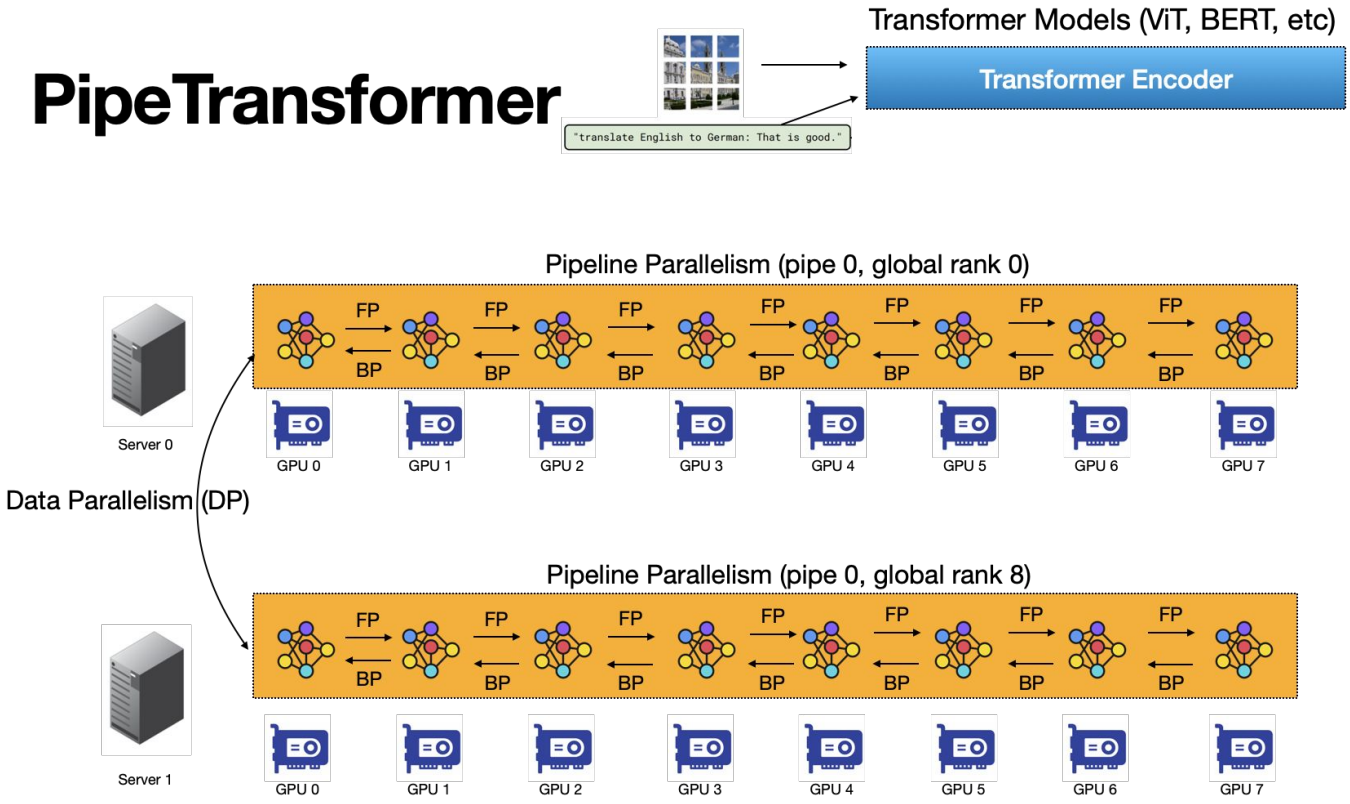
1. The computation cost becomes unbalanced in pipeline-parallelism
2. The memory cost is reduced gradually
3. The communication cost among DP workers should be reduced gradually

Overall Design

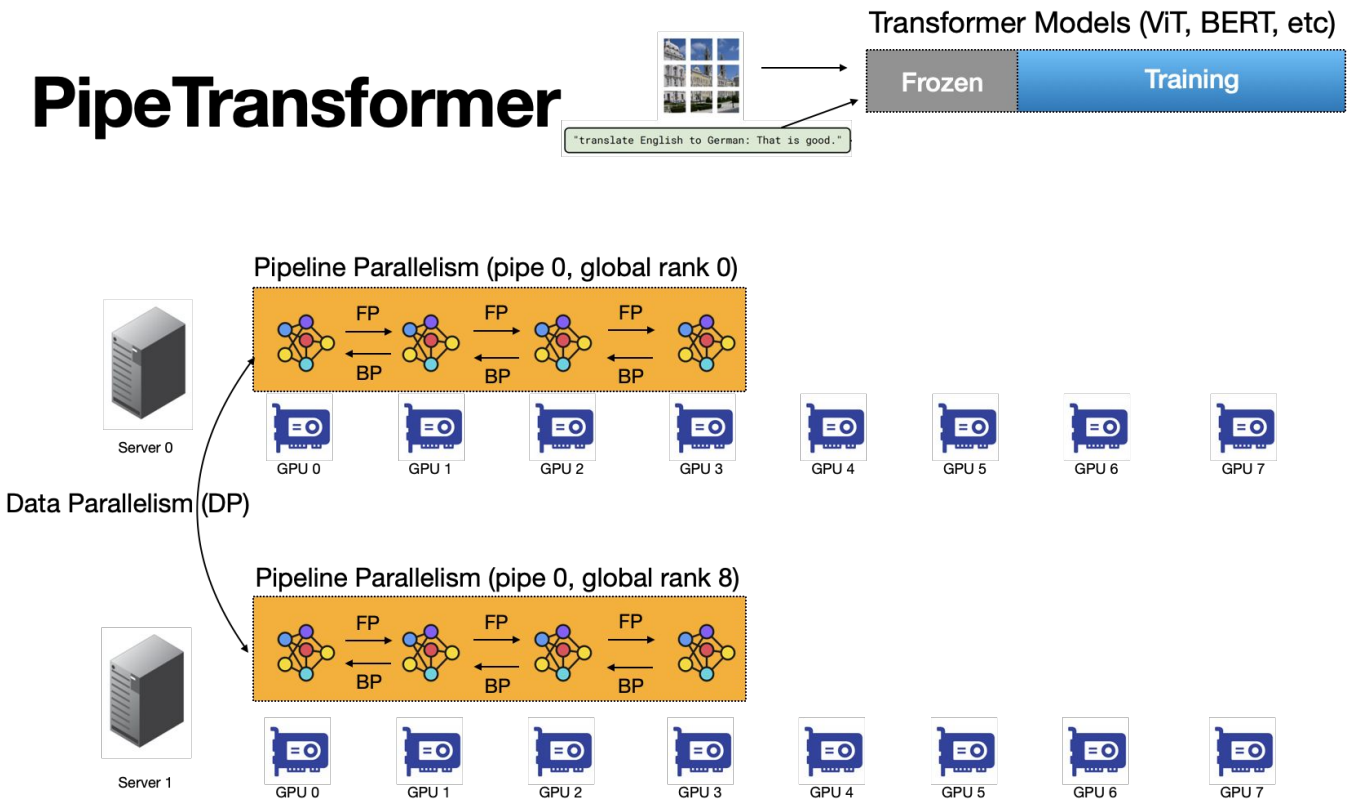


The process of PipeTransformer's automated and elastic pipelining

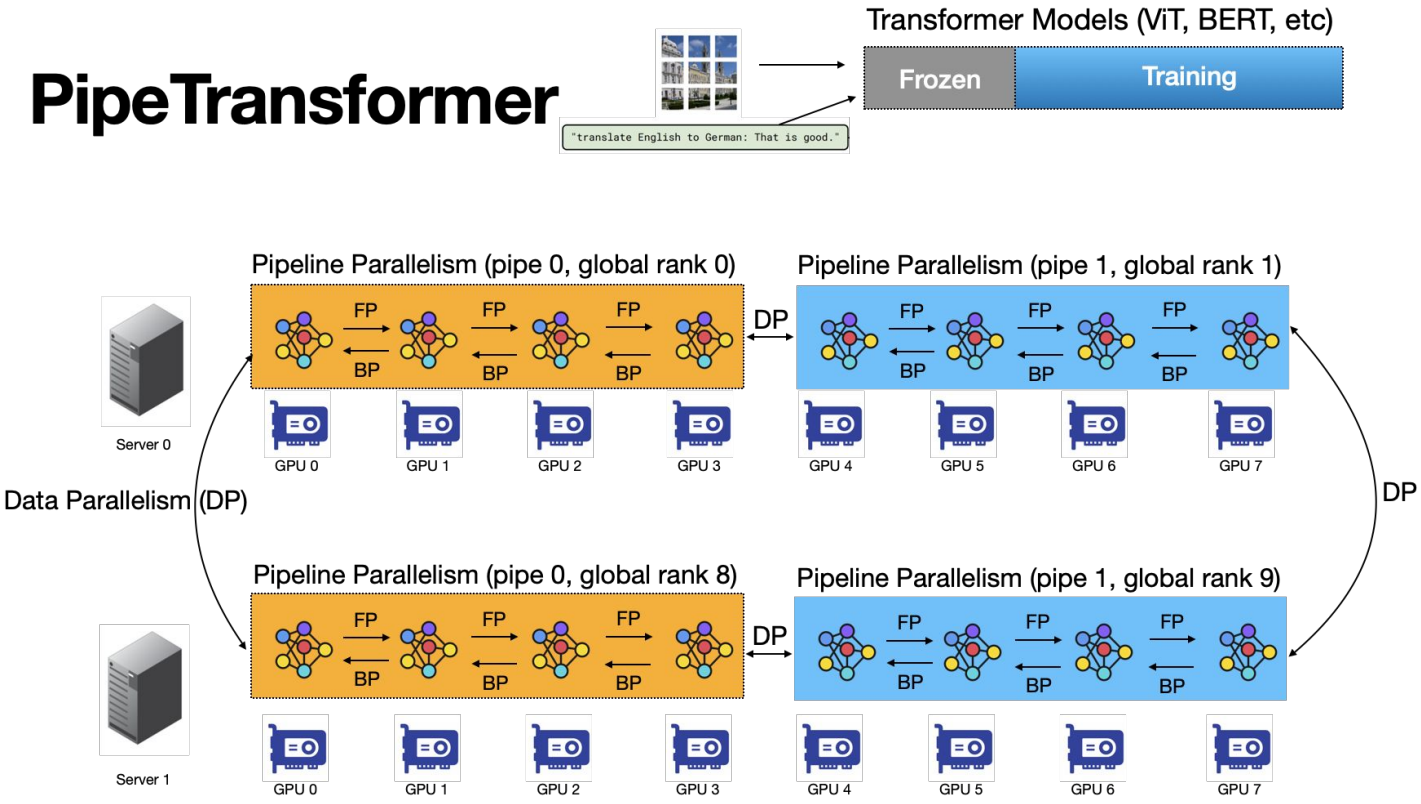
PipeTransformer Animation



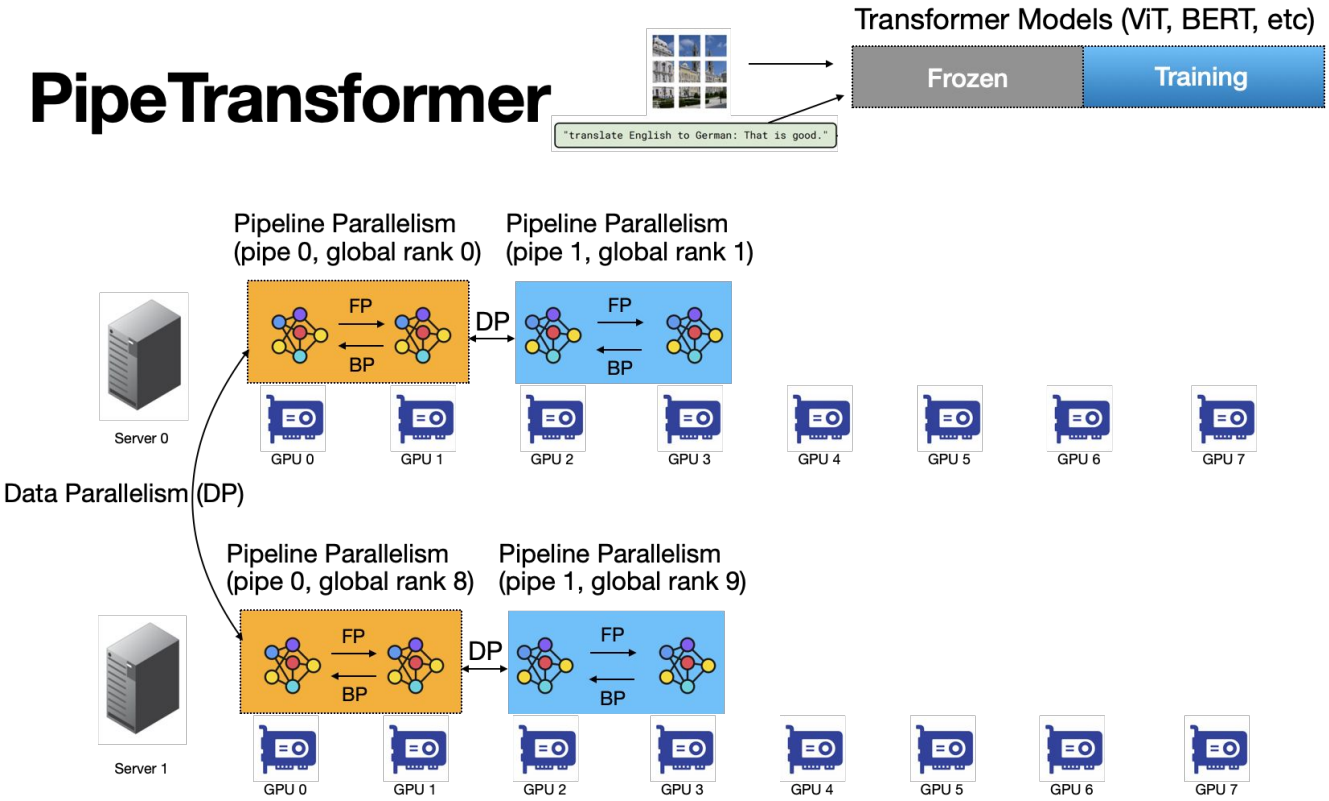
PipeTransformer Animation



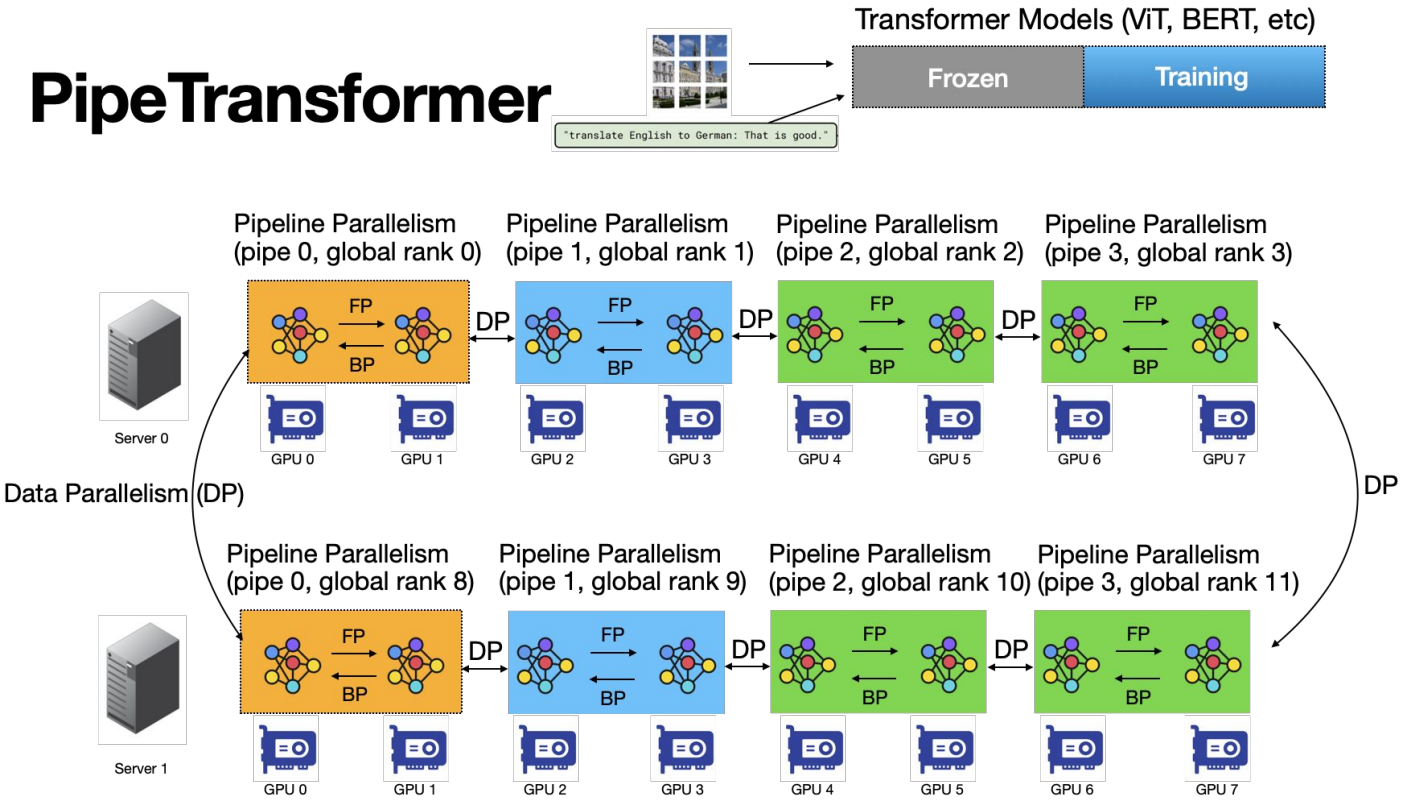
PipeTransformer Animation



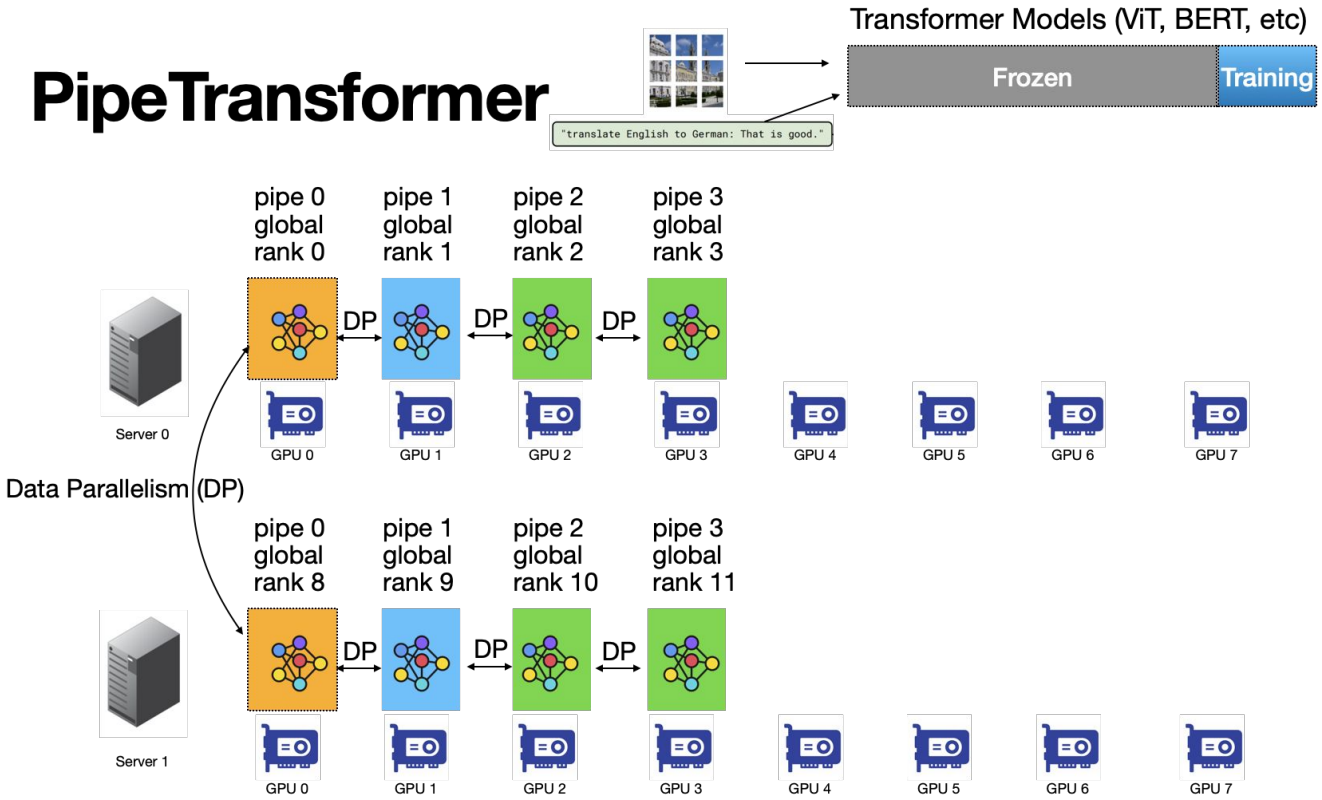
PipeTransformer Animation



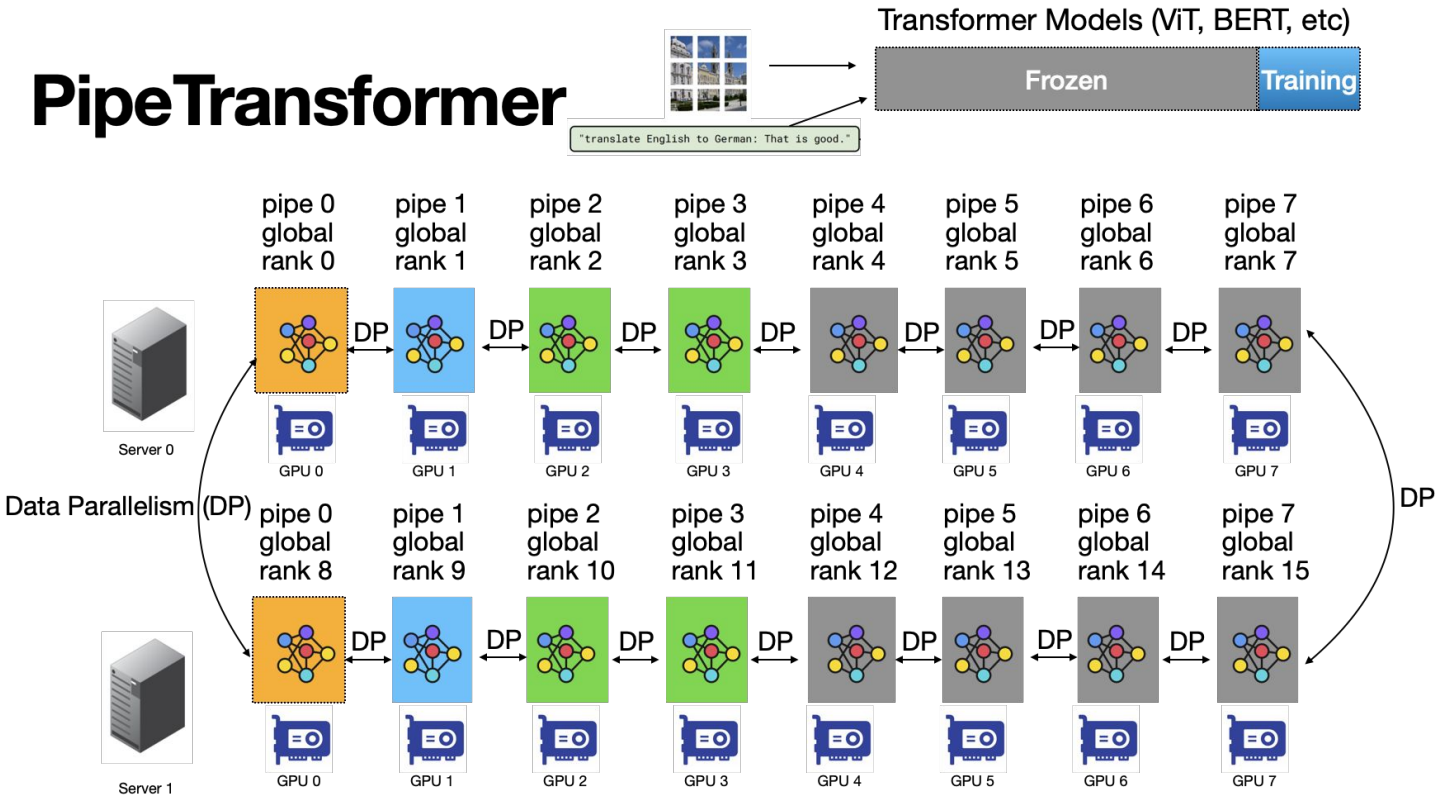
PipeTransformer Animation



PipeTransformer Animation



PipeTransformer Animation



Overall Design

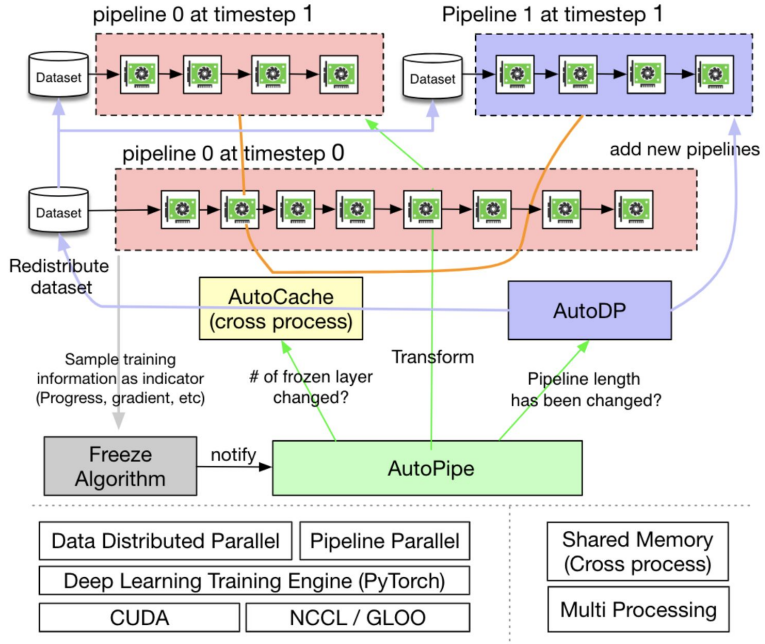
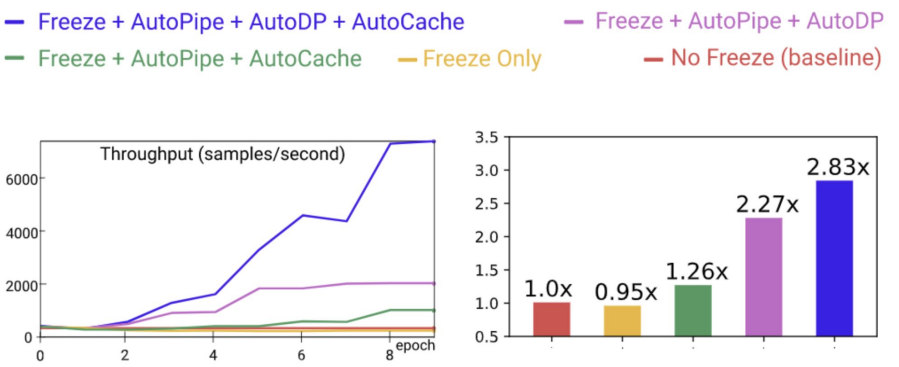


Figure 3. Overview of `PipeTransformer` Training System



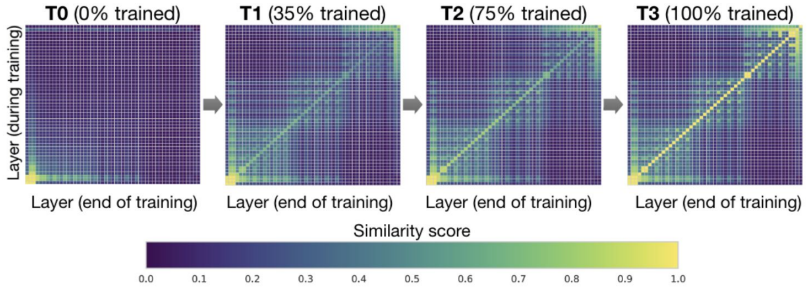
(a) Sample Throughput (b) Speedup Ratio Comparison

Figure 9. Speedup Breakdown (ViT on ImageNet)

https://DistML.ai

https://chaoyanghe.com/pipetransformer/

Freeze Algorithm



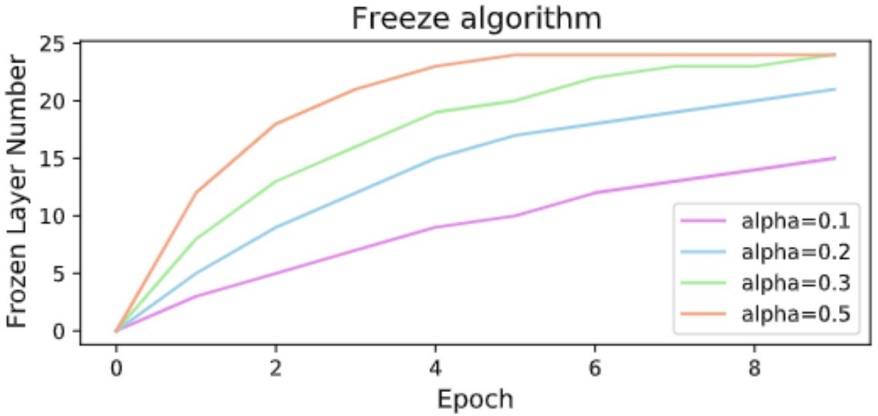


Figure 4. Freeze Algorithm Using Different α

$$\min\left(L_{\text{frozen}}^{(T-1)} + \alpha(L - L_{\text{frozen}}^{(T-1)}),\ \underset{\ell \in \{L_{\text{frozen}}^{(T-1)},\ldots,L\}}{\operatorname{argmin}} \left\|\boldsymbol{g}_\ell^{(T)}\right\|\right)$$

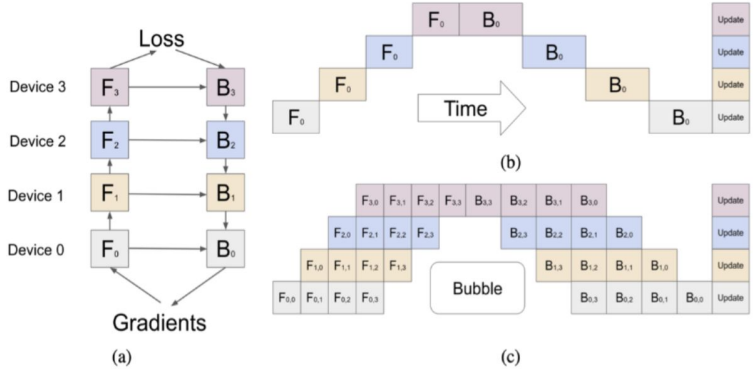
$$\text{where } T \geq 1,\ L_{\text{frozen}}^{(0)} = 0, \text{ and } \alpha \in (0,1) \tag{1}$$

AutoPipe: Elastic Pipelining

ICML International Conference On Machine Learning

USC University of Southern California

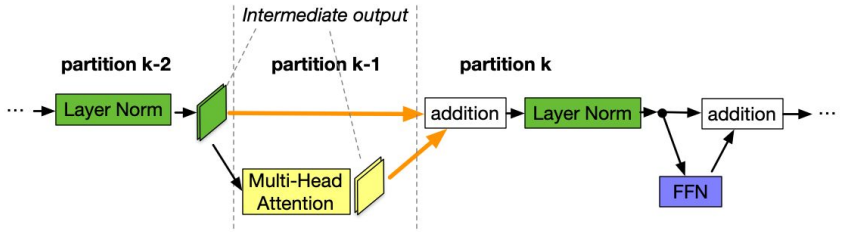
(1) Basic Usage of PyTorch Pipeline



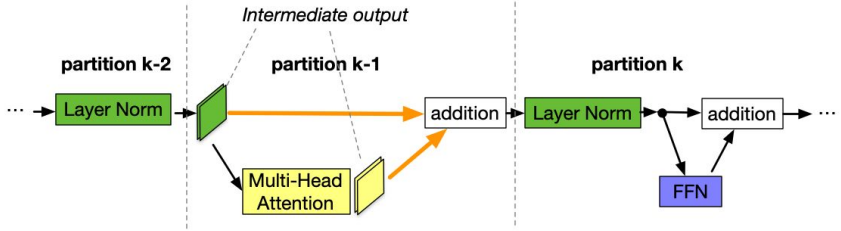

```
# Step 1: build a model including two linear layers
fc1 = nn.Linear(16, 8).cuda(0)
fc2 = nn.Linear(8, 4).cuda(1)

# Step 2: wrap the two layers with nn.Sequential
model = nn.Sequential(fc1, fc2)

# Step 3: build Pipe (torch.distributed.pipeline.sync.Pipe)
model = Pipe(model, chunks=8)

# do training/inference
input = torch.rand(16, 16).cuda(0)
output_rref = model(input)
```

AutoPipe: Elastic Pipelining

(1) Pipeline Partitioning Strategy



Partition in the middle of skip connection:
the parameter number is balanced, but it requires more communications



NO Partition in the middle of skip connection:
the parameter number is slightly unbalanced, but it requires less communications

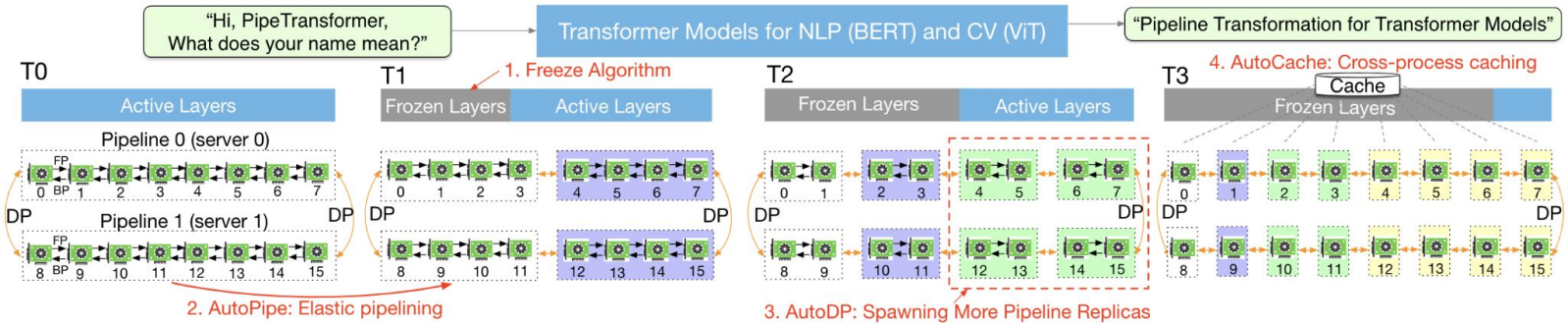
trade-off of computational load, communication cost, and memory consumption among partitions (each partition is loaded to one GPU)

AutoPipe: Elastic Pipelining

ICML International Conference On Machine Learning

USC University of Southern California

(2) Pipeline Compression



$$\text{compress the pipeline if } M_{GPU}^{(T)} \leq M_{GPU}^{(0)}$$

$$\text{where } M_{GPU}^{(T)} \Leftrightarrow \max_{k \in \{0,\cdots,K-1\}} S_{p_k}$$

To avoid extensive memory profiling, the compression algorithm **uses the parameter size as a proxy** for the training memory footprint.

AutoPipe: Elastic Pipelining

ICML International Conference On Machine Learning

USC University of Southern California

(3) Dynamic Number of Micro Batches

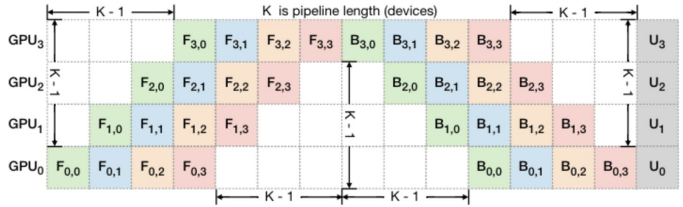
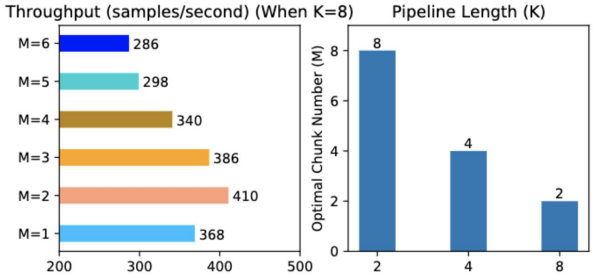


Figure 6. Pipeline Bubble: $F_{d,b}$, $B_{d,b}$, and U_d denote forward, backward, and the optimizer update of micro-batch b on device d, respectively. The total bubble size in each iteration is $(K-1)$ times per micro-batch forward and backward cost.



(b) Profiling Optimal Chunk Number

When the pipeline compressed (K becomes smaller), the bubble is shrunk (left figure)

However, we find that the micro-batches size (M) also needs to be adjusted accordingly (right figure shows the optimal M in different K).

AutoPipe: Elastic Pipelining

ICML International Conference On Machine Learning

USC University of Southern California

(4) AutoPipe algorithm: put all together

Algorithm 1 `AutoPipe` Algorithm

Input: model $\mathcal{F}$, layer number L and L_{frozen}, pipeline length K, frozen layer cost factor λ_{frozen}
Return: model $\mathcal{F}_{\text{frozen}}$, model $\mathcal{F}_{\text{pipe}}$, updated K;
`def m_partition(`$\mathcal{F}, L, L_{\text{frozen}}$`):` *//see 3.2.1*
$\mathcal{F}_{\text{frozen}}$ = `Sequential()`; model size $S_{\text{frozen}} = 0$
$\mathcal{F}_{\text{pipe}}$ = `Sequential()`; per-layer size S_{pipe} = `[]`
for layer index = L_{frozen} **to** L **do**
  $f_{\text{ATT}_i}, f_{\text{MLP}_i} \leftarrow f_i$
  $\mathcal{F}_{\text{pipe}}$.append($f_{\text{ATT}_i}$); S_{pipe}.append(m_size(f_{ATT_i}))
  $\mathcal{F}_{\text{pipe}}$.append($f_{\text{MLP}_i}$); S_{pipe}.append(m_size(f_{MLP_i}))
return $\mathcal{F}_{\text{frozen}}$,$S_{\text{frozen}}$,$\mathcal{F}_{\text{pipe}}$,$S_{\text{pipe}}$
`def load_balance(`$\mathcal{F}_{\text{pipe}}$, S_{pipe}, K`):` *//Section 3.2.1*
B_L=dict(), B_S=dict() *// balanced L and S*
$L_{\text{assigned}} = 0$; S_{total} = sum(S_{pipe})
for partition index = k **to** K **do**
  mean=S_{total}/(K - k);
  var=np.var($S_{\text{pipe}}[L_{\text{assigned}}$:])/($K$ - k)
  for sublayer index i = L_{assigned} **to** len(S_{pipe}) **do**
    S_k = S_{pipe}[i]
    criterion=B_S[i]-S_{frozen}(1.0- λ_{frozen})+S_k
    if criterion < mean + var **then**
      B_S+=S_k; B_L+=1; L_{assigned}+=1; S_{total}-=S_k
    else
      break
return B_L, B_S
$\mathcal{F}_{\text{frozen}}$,$S_{\text{frozen}}$,$\mathcal{F}_{\text{pipe}}$,$S_{\text{pipe}}$ = m_partition($\mathcal{F}$,L, L_{frozen})
while $K \geq 2$ **do**
  B_L, B_S = load_balance($\mathcal{F}_{\text{pipe}}$, S_{pipe}, $K/2$)
  B_S[0] -= S_{frozen}(1.0 - λ_{frozen});
  $M_{GPU}^{(T)}$ = max(B_S) *//Equation 2*
  if $M_{GPU}^{(T)} < M_{GPU}^{(0)}$ **then**
    K=$K/2$
  else
    break
load $\mathcal{F}_{\text{frozen}}$ and $\mathcal{F}_{\text{pipe}}$ to K GPUs using B_S and B_L
Pipe($\mathcal{F}_{\text{pipe}}$, chunks= get_optimal_chunks (K))

1. A trade-off in Pipeline partition
2. Pipeline compression
3. optimal micro-batches chunk number (M)

AutoDP:
Spawning More Pipeline Replicas

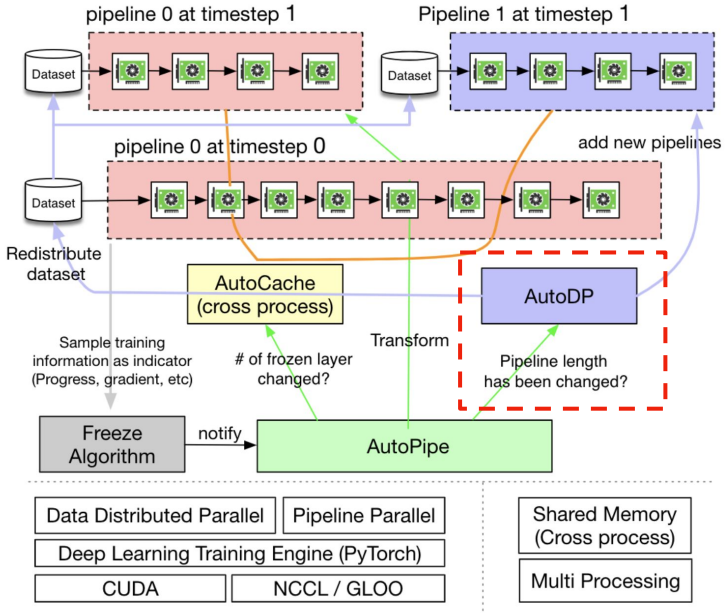
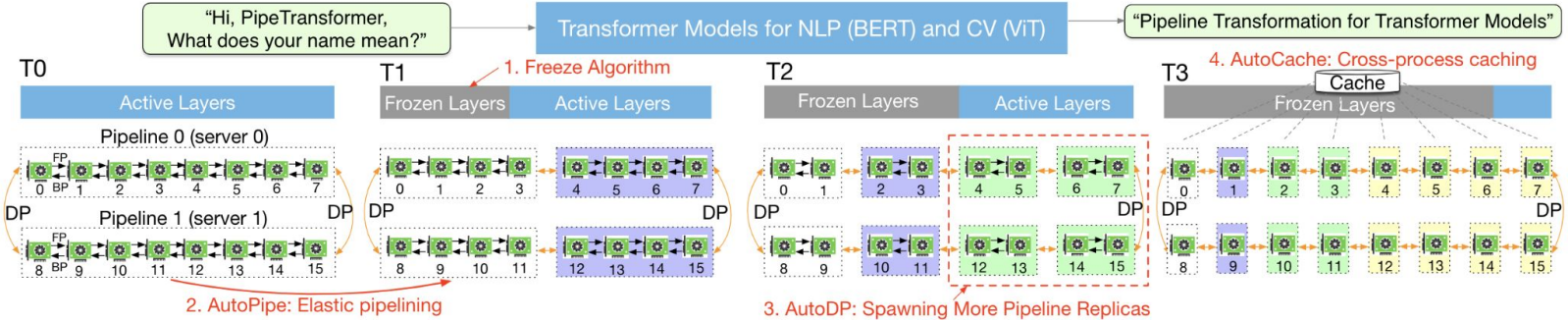


Figure 3. Overview of `PipeTransformer` Training System

AutoDP:
Spawning More Pipeline Replicas



Key challenges when adding more pipeline on the fly of training:

1) **DDP Communication:** Collective communications in PyTorch DDP requires **static** membership, which prevents new pipelines from connecting with existing ones;

2) **State Synchronization:** **newly activated processes** must be **consistent** with existing pipelines in the training progress (e.g., epoch number and learning rate), weights and optimizer states, the boundary of frozen layers, and pipeline GPU range;

3) **Dataset Redistribution:** the **dataset** should be **re-balanced** to match a dynamic number of pipelines. This not only avoids stragglers but also ensures that gradients from all DDP processes are equally weighted.

AutoDP:
Spawning More Pipeline Replicas

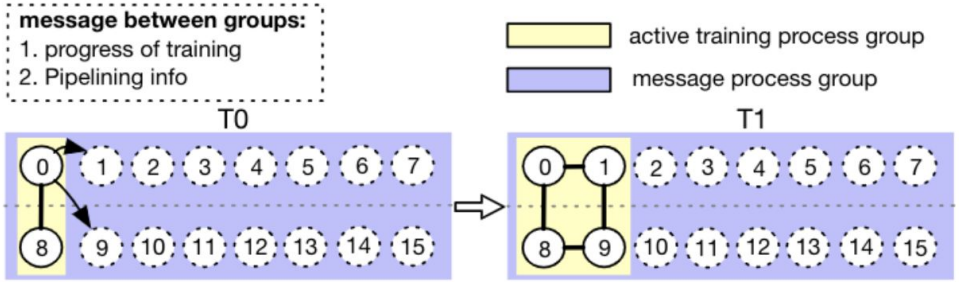


Figure 7. AutoDP: handling dynamical data parallel with messaging between double process groups (Process 0-7 belong to machine 0, while process 8-15 belong to machine 1)

Our idea:

1. **creating two process groups.** Each process handles one pipeline
2. the active training process group (yellow) handles the training.
3. the message process group (purple) handles **State Synchronization and Dataset Redistribution** by **messaging passing between two groups** with MPI communication.

AutoDP:
Spawning More Pipeline Replicas

ICML International Conference On Machine Learning

USC University of Southern California

```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

# initialize the process group (this must be called in the initialization of PyTorch DDP)
dist.init_process_group(init_method='tcp://' + str(self.config.master_addr) + ':' +
str(self.config.master_port), backend=Backend.GLOO, rank=self.global_rank, world_size=self.world_size)
...

# create active process group (yellow color)
self.active_process_group = dist.new_group(ranks=self.active_ranks, backend=Backend.NCCL, timeout=timedelta(days=365))
...

# create message process group (yellow color)
self.comm_broadcast_group = dist.new_group(ranks=[i for i in range(self.world_size)], backend=Backend.GLOO, timeout=timede
...

# create DDP-enabled model when the number of data-parallel workers is changed. Note:
# 1. The process group to be used for distributed data all-reduction.
If None, the default process group, which is created by torch.distributed.init_process_group, will be used.
In our case, we set it as self.active_process_group
# 2. device_ids should be set when the pipeline length = 1 (the model resides on a single CUDA device).

self.pipe_len = gpu_num_per_process
if gpu_num_per_process > 1:
    model = DDP(model, process_group=self.active_process_group, find_unused_parameters=True)
else:
    model = DDP(model, device_ids=[self.local_rank], process_group=self.active_process_group, find_unused_parameters=True)

# to broadcast message among processes, we use dist.broadcast_object_list
def dist_broadcast(object_list, src, group):
    """Broadcasts a given object to all parties."""
    dist.broadcast_object_list(object_list, src, group=group)
    return object_list
```

AutoCache:
Cross-pipeline Caching

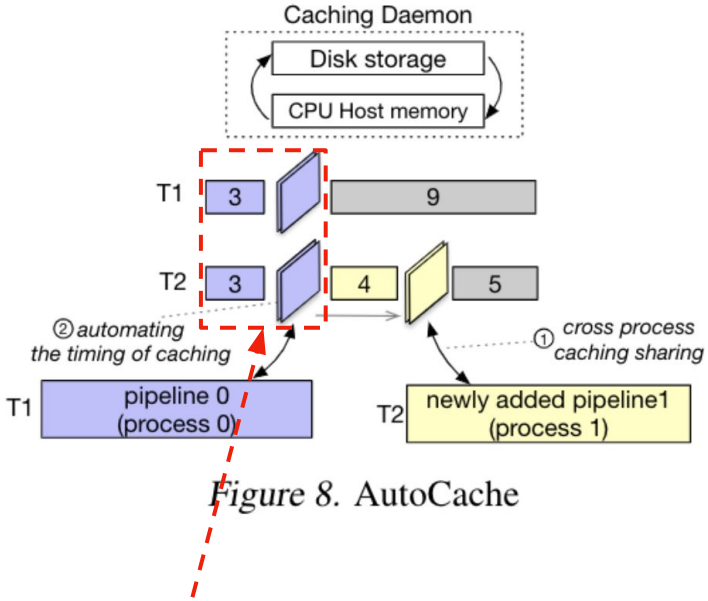


Figure 8. AutoCache

In this example, **the first 3 layers (purple)** at two time steps T1 and T2(epochs) are the same computation, so T2 can reuse the caching from T1.

Experimental Results

ICML International Conference On Machine Learning

USC University of Southern California

1. Overall Speedup

Table 1. Speedup for ViT and BERT Training

Dataset	Baseline		PipeTransformer		
	Accuracy	Training time	Accuracy	Training time	Training Speedup
ImageNet	80.83 ± 0.05	26h 30m	82.18 ± 0.32	9h 21m	**2.83** ×
CIFAR-100	91.21 ± 0.07	35m 6s	91.33 ± 0.05	12m 23s	2.44 ×
SQuAD 1.1	90.71 ± 0.18	5h 7m	90.69 ± 0.23	2h 26m	2.10 ×

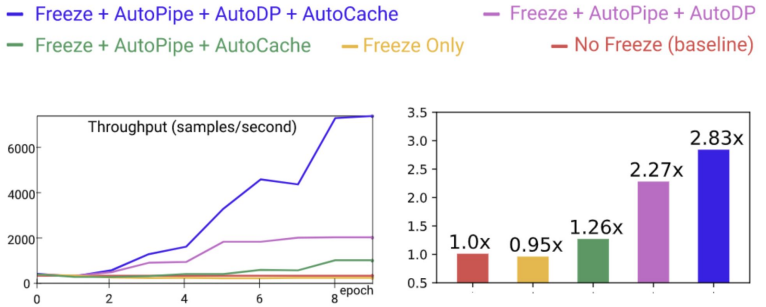
Evaluation on Various datasets and models, including tasks in both CV and NLP.

Experimental Results

ICML International Conference On Machine Learning

USC University of Southern California

2. Breakdown for speedup



(a) Sample Throughput (b) Speedup Ratio Comparison

Figure 9. Speedup Breakdown (ViT on ImageNet)

Key takeaway:

1. the main speedup is the result of elastic pipelining which is achieved through the joint use of AutoPipe and AutoDP (purple)

2. AutoCache's contribution is amplified by AutoDP (green v.s. blue: more parallel DP workers can use caching)

3. freeze training alone without system-wise adjustment even downgrades the training speed (yellow)
(the underlying mechanism of PyTorch is not tailored for freeze training, forcing CUDACachingAllocator to split blocks or launch new memory allocations)

Experimental Results

3. Breakdown for communication v.s. computation

Table 2. Communication Cost v.s. Computational Cost

	Dataset	Overall Cost	Communication Cost	Computation Cost	Communication Cost Ratio
ViT-Base (87M)	ImageNet	9h 21m	34m	8h 47m	5.9 %
BERT-large (340M)	SQuAD	2h 26m	16m 33s	2h 9m	8.8%

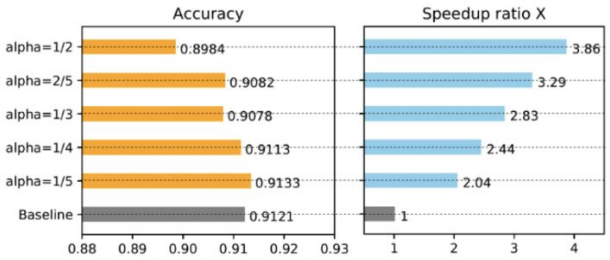
Communication Infrastructure: InfiniBand CX353A where **cross-machine bandwidth is 5GB/s**, and GPU-to-GPU bandwidth within a machine (PCI 3.0, 16 lanes) is **15.754GB/s**.

Key takeaway:
1. Communication cost is not the main bottleneck when we use InfiniBand for medium-scale models (< 500M such as ViT-base and BERT-large), but it is still non-trivial even under freeze training.
2. Recent progress in NLP and CV has scaled up the model size to billion/trillion-level (GPT-3 - 175B [1], Switch Transformer - 1.7T [2]), which will make the ratio of communication much higher than our experimental results.

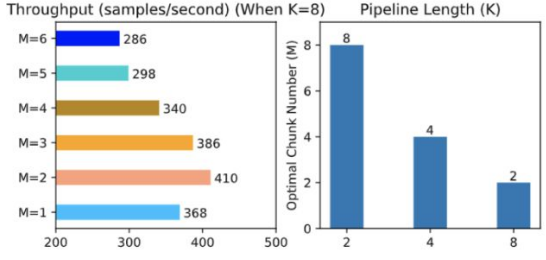
[1] Language Models are Few-Shot Learners. 2020
[2] Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. 2021

Experimental Results

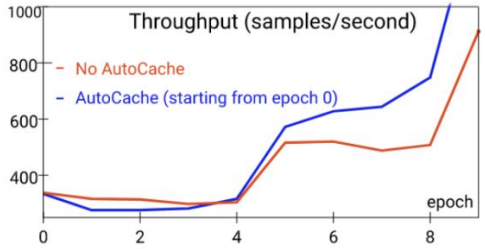
4. Performance Analysis



(a) Tuning α in Freeze Algorithm



(b) Profiling Optimal Chunk Number



(c) Timing of Caching

Figure 10. Some Results of Performance Analysis

The trade-off between accuracy and efficiency.

1. Optimal chunk number in dynamic pipeline is different.
2. Timing of caching is important.

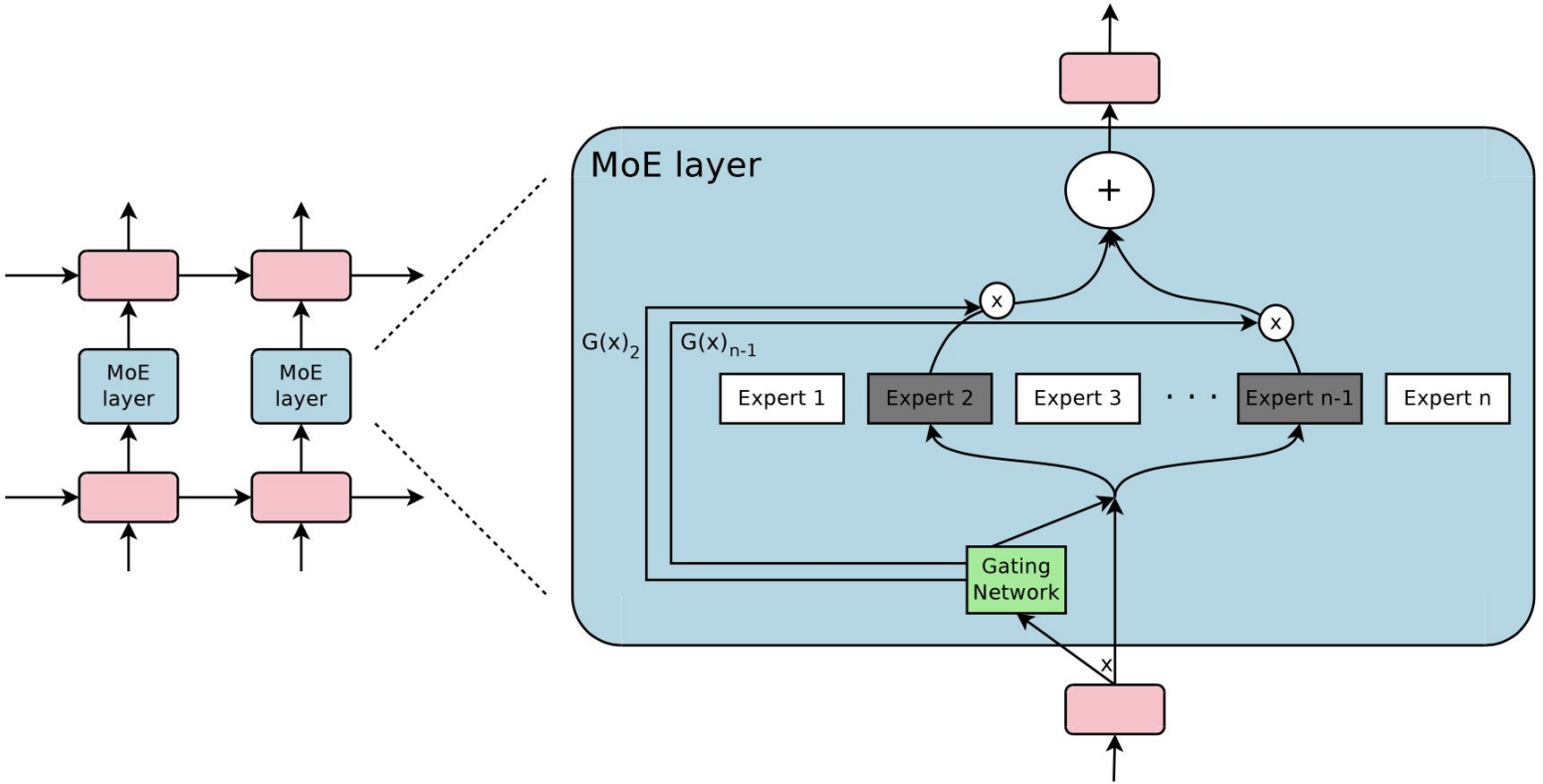
We automate these two optimization strategies.

Future works

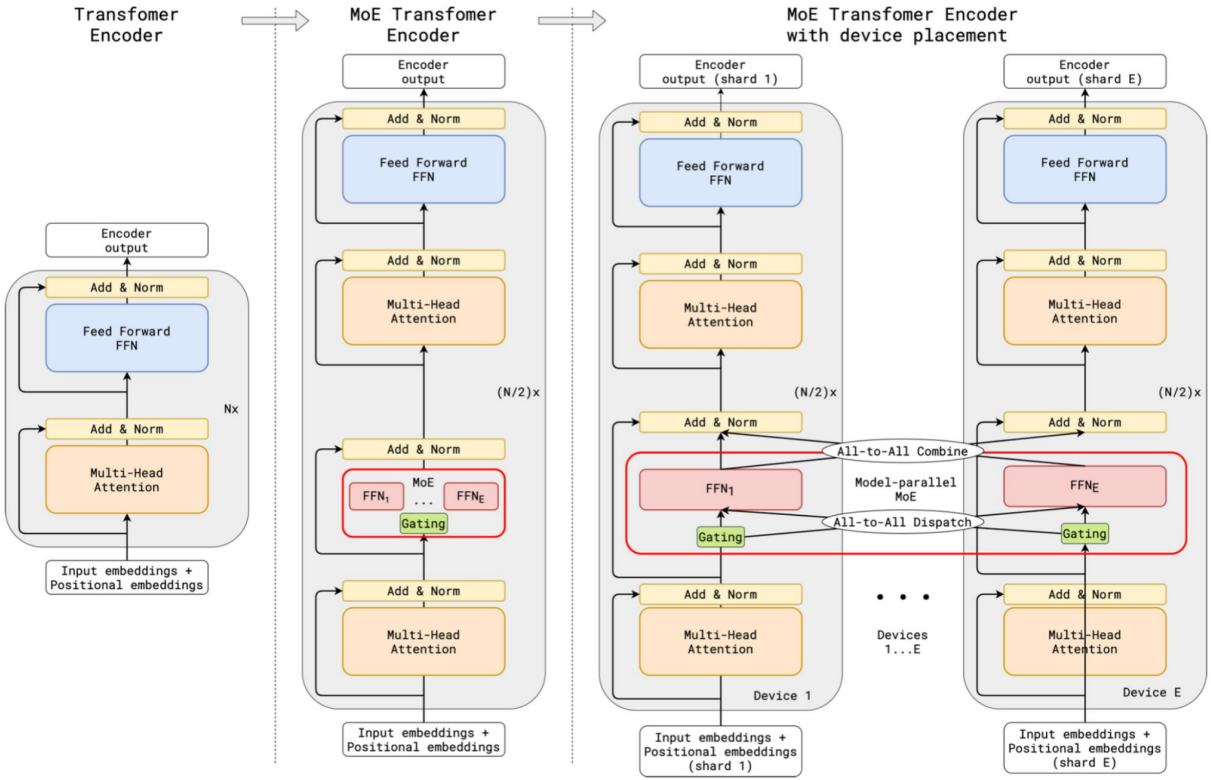
Distributed training tasks that can be dynamic:

1. Elastic cloud-based distributed training system
2. Accelerating NAS in extremely large search space
3. Federated AutoML (NAS, HPO)
4. Cross-silo Federated Learning
5. Pruning-based distributed training
6. IoT device-based elastic edge training

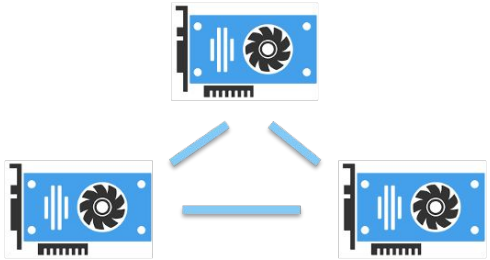
Ongoing Research - Mixture of Experts



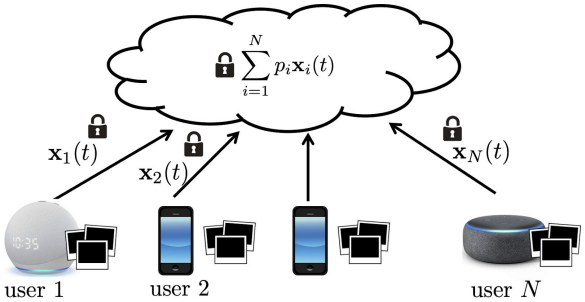
Ongoing Research - Mixture of Experts



Outline: Distributed ML for Large-scale Models



- **Part 1: Dynamic Distributed Training**
  PipeTransformer: Automated Elastic Pipelining for Distributed Training of Large-scale Models. ICML 2021
  https://DistML.ai

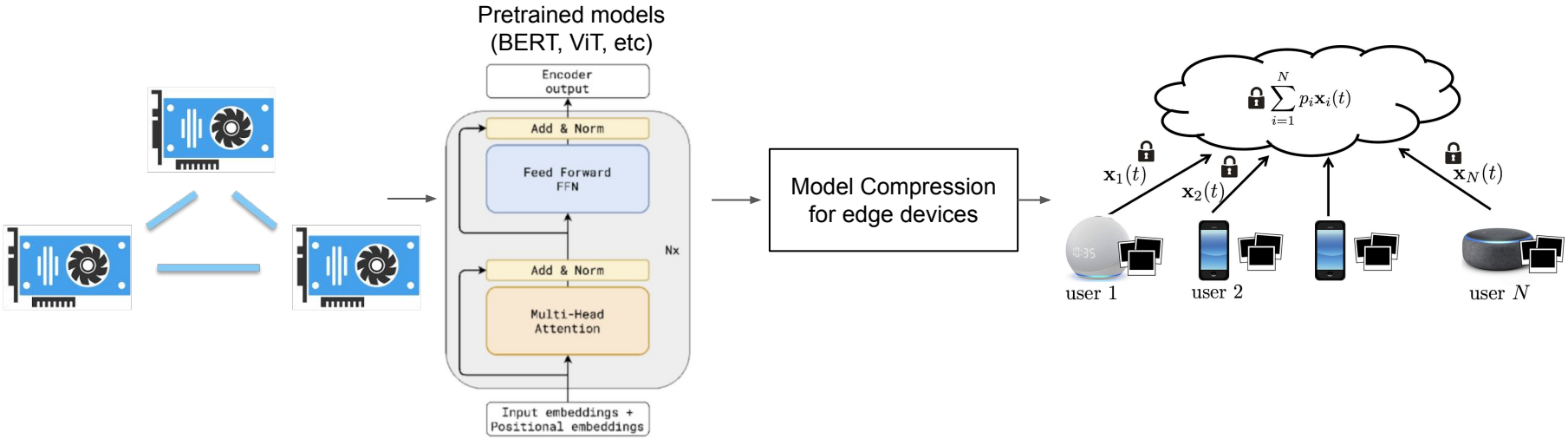


- **Part 2: Scalable Federated Learning**
  [Algorithm] Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge. NeurIPS 2020

  [System] FedML: A Research Library and Benchmark for Federated Machine Learning
  (Best Paper Award, NeurIPS 2020 FL Workshop)

  [Diverse Applications]
  FedML Ecosystem: Ubiquitous Distributed Training for Diverse AI Applications at the Edge.
  https://FedML.ai

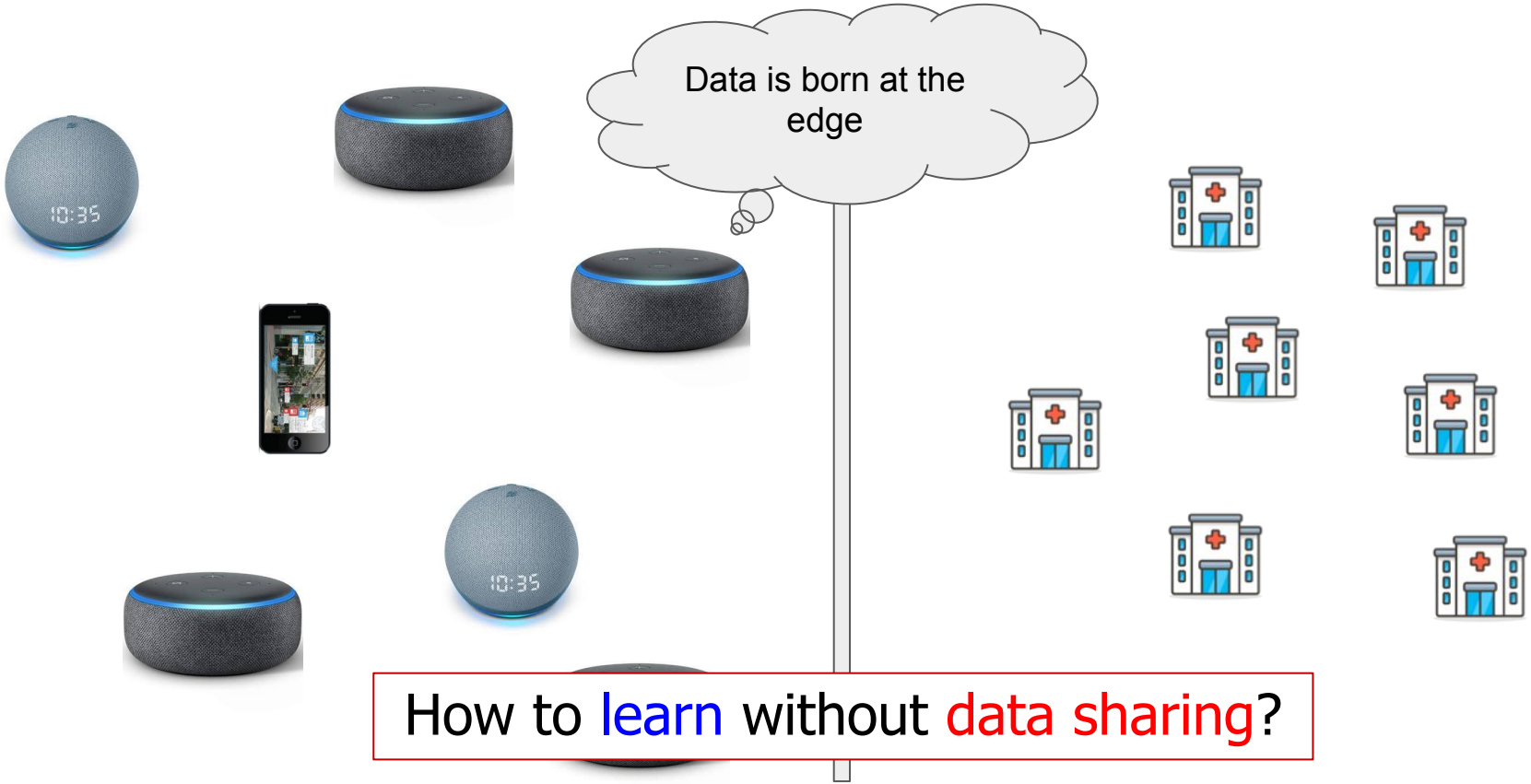
Fundamental Research to Alexa AI Applications



1. Distributed Training in GPU cluster

2. Federated Learning at the edge

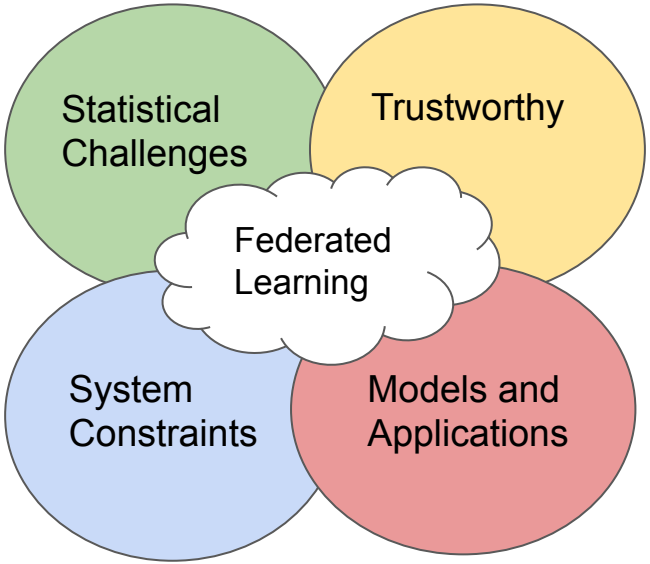
Goal: enabling secure collaborative learning at the edge



Introduction: Widely Interdisciplinary

distributed optimization
personalization and non-IID
transfer learning
un/semi-supervised learning
neural architecture search
continual learning/meta learning
...

communication efficiency
computation efficiency
wireless communication
cloud computing
embedded training system
...



preserving privacy
robust to adversarial attack
fairness
incentive mechanism
...

computer vision
natural language processing
data ming
IoT/5G
...

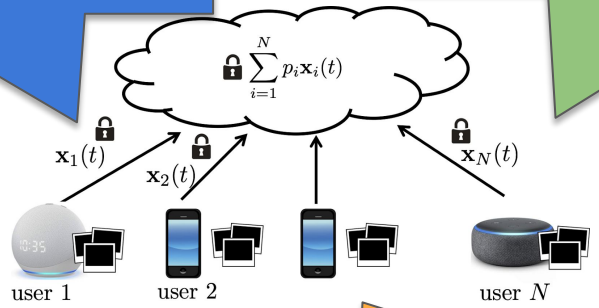
Conference Venues: NeurIPS, ICML, ICLR, MLSys, CVPR/ICCV/ECCV, ACL/EMNLP, AISTATS, AAAI

Foundations of Algorithm Design

- ❑ User selection, optimal scaling
- ❑ Heterogeneity, personalization, and fairness
- ❑ Unsupervised federated learning

System Bottlenecks/Opportunities

- ❑ **Resource-constrained FL (small edge models, large server models)**
- ❑ Scalability: 1K users → 1M users
- ❑ Federated neural architecture search

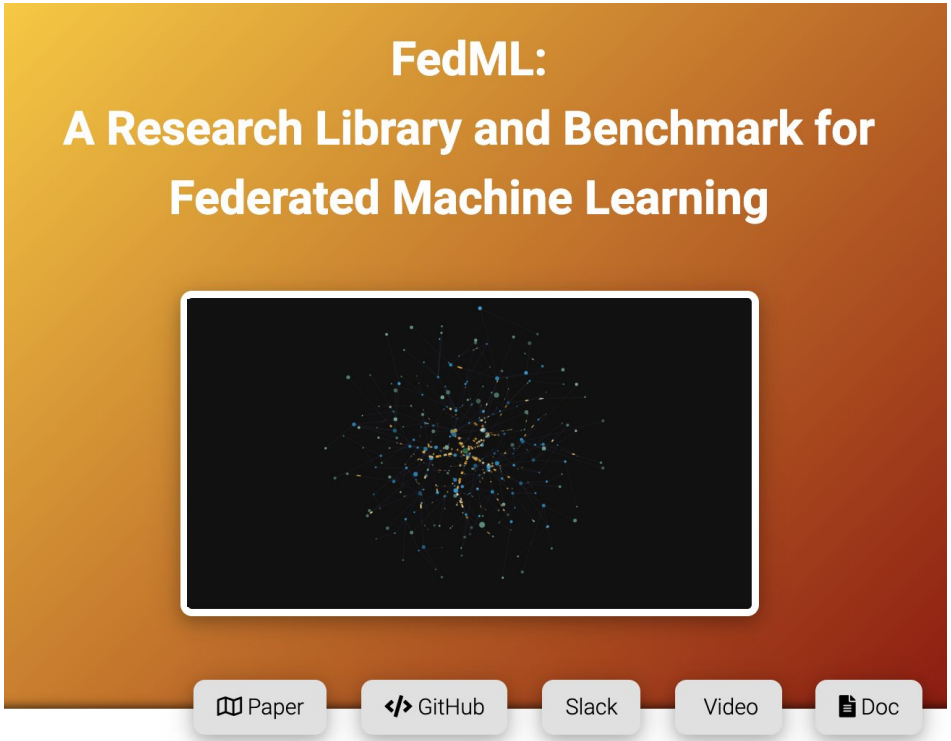


Trustworthy

- ❑ Secure and resilient model aggregation
- ❑ Adversarial users (data/model poisoning)
- ❑ Leveraging trusted computing environments

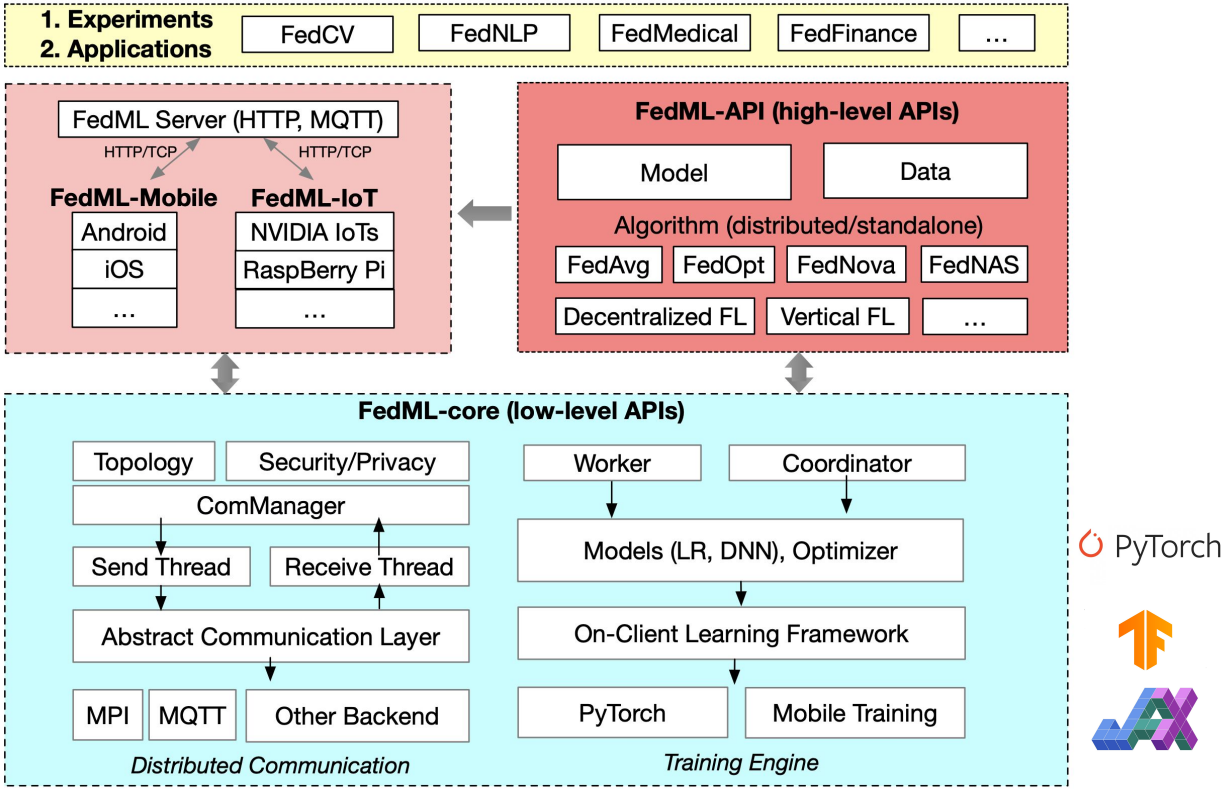
USC University of Southern California

FEDML
MAKING COLLABORATION INTELLIGENT

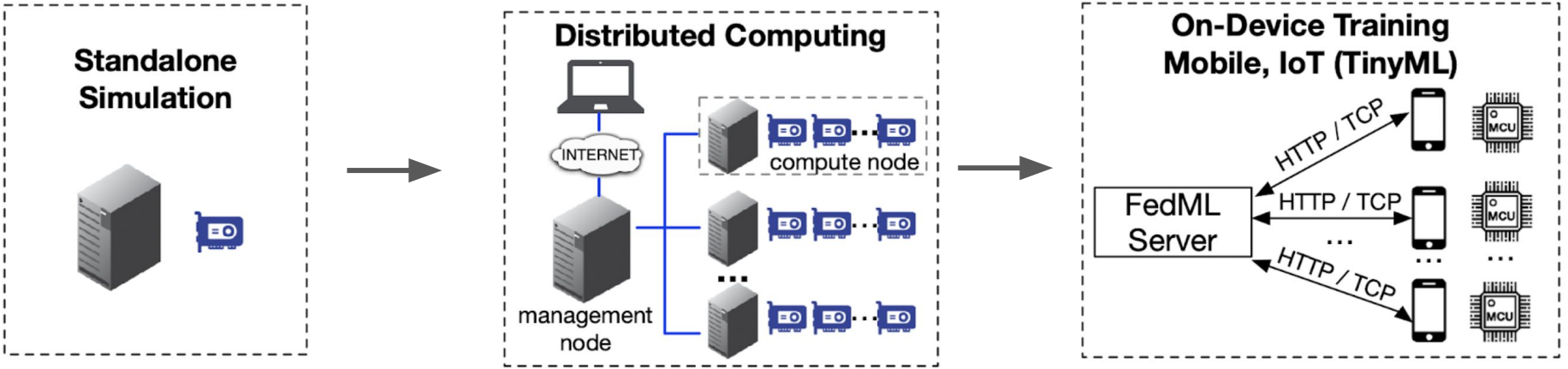


https://www.avestimehr.com/fedml
https://FedML.ai

Overview


1. Experiments
2. Applications
FedCV
FedNLP
FedMedical
FedFinance
...
FedML Server (HTTP, MQTT)
HTTP/TCP
HTTP/TCP
FedML-Mobile
FedML-IoT
Android
iOS
...
NVIDIA IoTs
RaspBerry Pi
...
FedML-API (high-level APIs)
Model
Data
Algorithm (distributed/standalone)
FedAvg
FedOpt
FedNova
FedNAS
Decentralized FL
Vertical FL
...
FedML-core (low-level APIs)
Topology
Security/Privacy
ComManager
Send Thread
Receive Thread
Abstract Communication Layer
MPI
MQTT
Other Backend
Distributed Communication
Worker
Coordinator
Models (LR, DNN), Optimizer
On-Client Learning Framework
PyTorch
Mobile Training
Training Engine
PyTorch

FedML System Capability



- Single process
- Runs on any server
- Good for small models (LR, 2CNN, Bi-LSTM)
- And small datasets (FMNIST, Shakespeare)

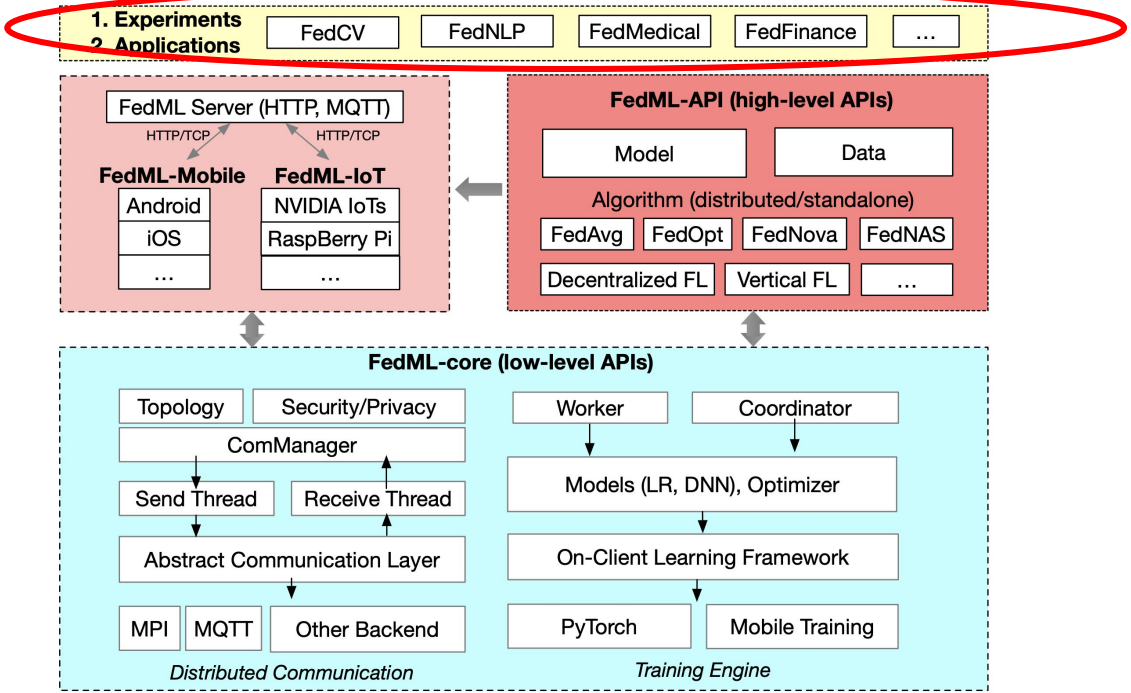
- Multiple-processes in parallel
- Can be seamlessly distributed across nodes
- Train big models (ResNet, MobileNet, Efficient Net, Transformer-based)
- And large datasets (CIFAR10/100, Google Landmark, COCO, ImageNet)

- Multiple processes with explicit message passing
- Great for resource constrained edge devices (Smartphone and IoT) - low memory, low computational power, limited communication bandwidth
- Python-centric simplicity

FedML Ecosystem

USC University of Southern California

FEDML
MAKING COLLABORATION INTELLIGENT



PyTorch

FedML Ecosystem

FedML
MAKING COLLABORATION INTELLIGENT

FedNLP

FedCV

FedGraphNN

FedIoT

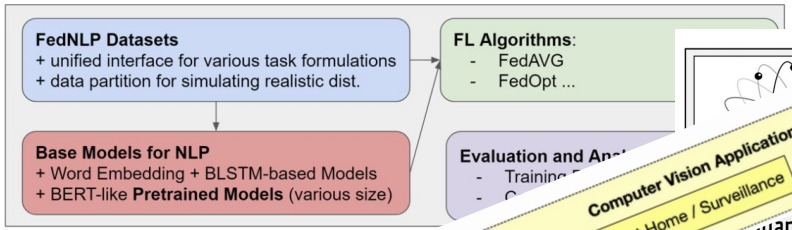
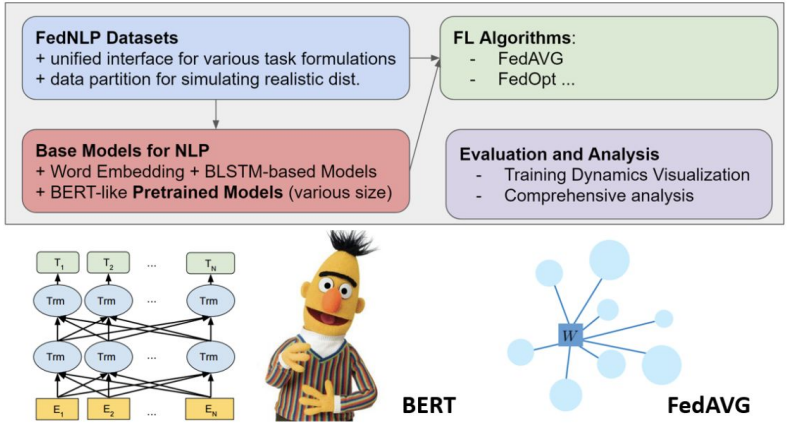
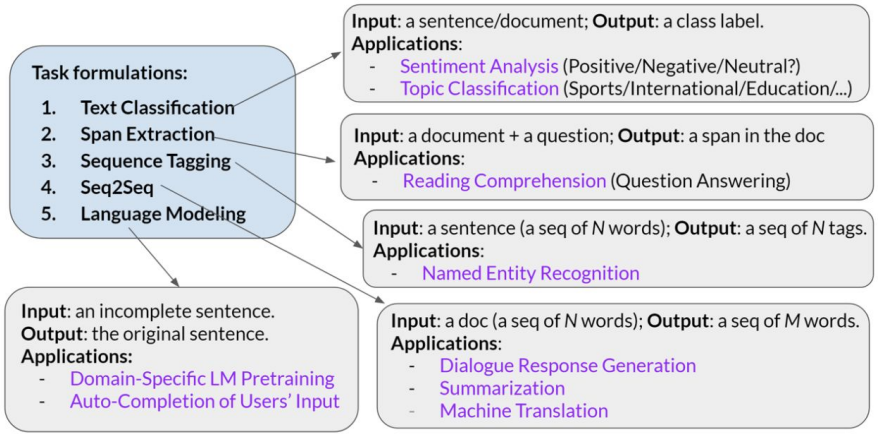


Figure 2: Tasks in different datasets focus on different levels of properties of molecules.

FedNLP


FedNLP Datasets
+ unified interface for various task formulations
+ data partition for simulating realistic dist.
FL Algorithms:
- FedAVG
- FedOpt ...
Base Models for NLP
+ Word Embedding + BLSTM-based Models
+ BERT-like Pretrained Models (various size)
Evaluation and Analysis
- Training Dynamics Visualization
- Comprehensive analysis
BERT
FedAVG


Task formulations:
1. Text Classification
2. Span Extraction
3. Sequence Tagging
4. Seq2Seq
5. Language Modeling
Input: a sentence/document; Output: a class label.
Applications:
- Sentiment Analysis (Positive/Negative/Neutral?)
- Topic Classification (Sports/International/Education/...)
Input: a document + a question; Output: a span in the doc
Applications:
- Reading Comprehension (Question Answering)
Input: a sentence (a seq of N words); Output: a seq of N tags.
Applications:
- Named Entity Recognition
Input: a doc (a seq of N words); Output: a seq of M words.
Applications:
- Dialogue Response Generation
- Summarization
- Machine Translation
Input: an incomplete sentence.
Output: the original sentence.
Applications:
- Domain-Specific LM Pretraining
- Auto-Completion of Users' Input

FedCV

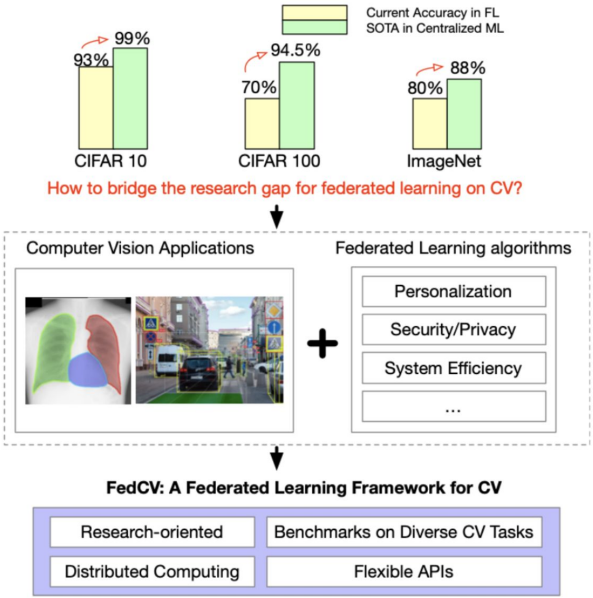


Figure 1. Our philosophy of federated learning on computer vision: connecting the algorithmic FL research and CV application-drive research with an unified research framework.

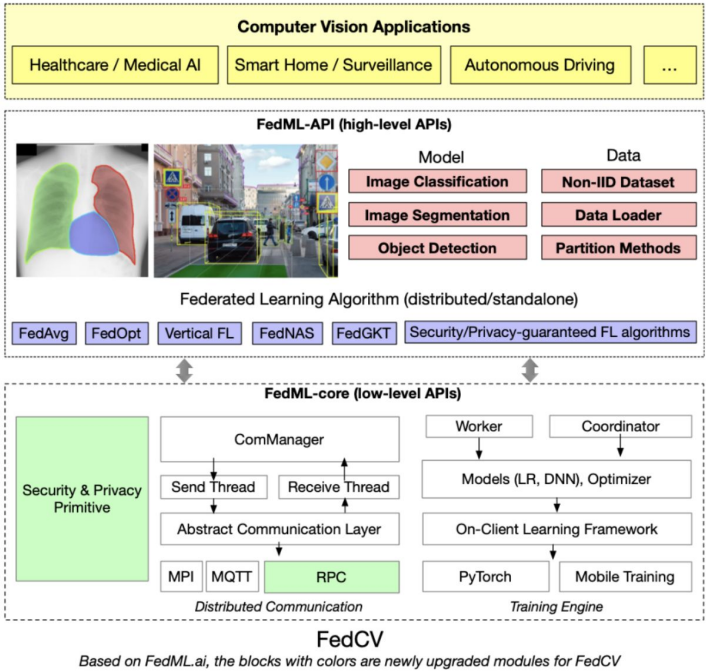


Figure 2. Overview of `FedCV` System Architecture Design

FedGraphNN

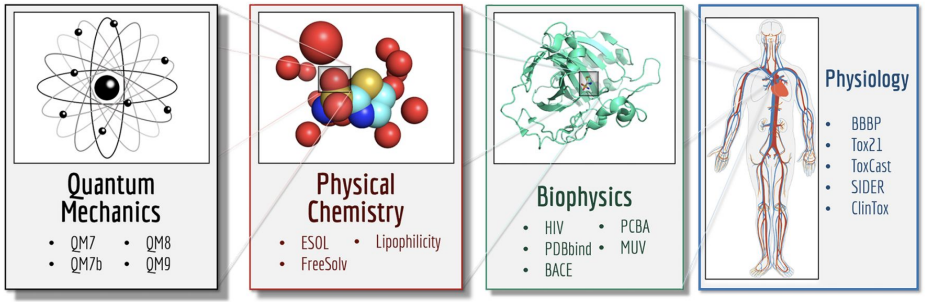


Figure 2: Tasks in different datasets focus on different levels of properties of molecules.

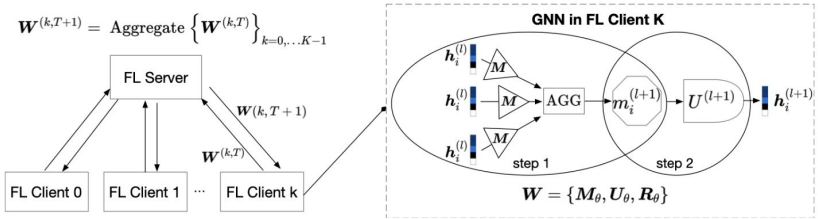


Figure 1: Formulation of `FedGraphNN` (Federated Graph Neural Network)

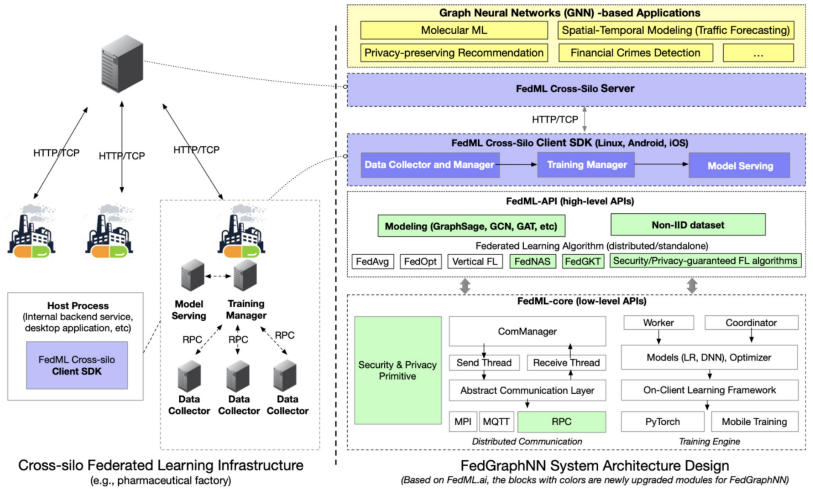
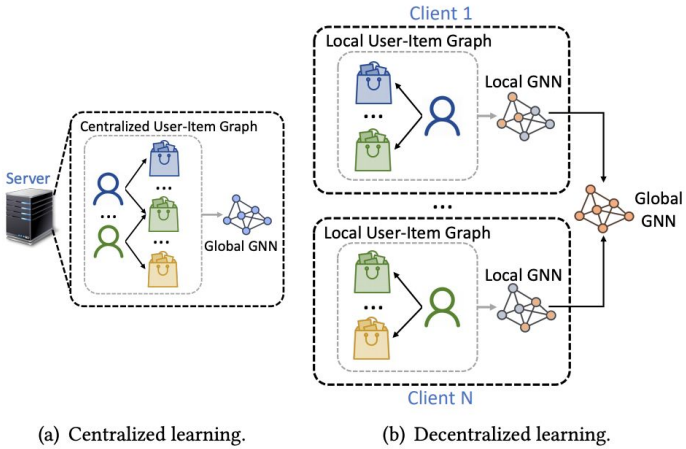


Figure 2. Overview of `FedGraphNN` System Architecture Design

FedGraphNN



(a) Centralized learning. (b) Decentralized learning.

Figure 1: Comparisons between centralized and decentralized training of GNN based recommendation models.

1. *Graph level*. We believe molecular machine learning is a paramount application in this setting, where many small graphs are distributed between multiple edge devices;

2. *Sub-graph level*. This scenario typically pertains to social networks or knowledge graphs that need to be partitioned into many small sub-graphs due to data barriers between different departments in a giant company, as demonstrated in (Wu et al., 2021).

3. *Node level*. When the privacy of a specific node in a graph is important, node-level GNN-based FL is useful in practice. The IoT setting is a good example (Zheng et al., 2020);

4. *Link level* is also a promising direction that is relevant when the privacy of edges (eg: connections in a social network) is of importance.

FedIoT: “Internet of Things & 5G + Federated Learning”

Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders [1]

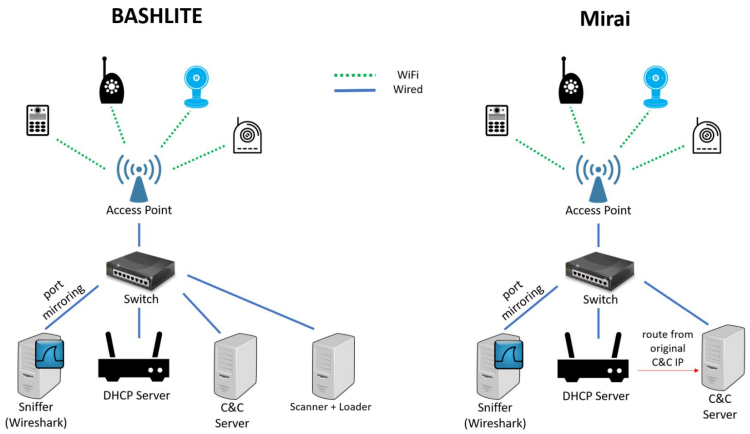


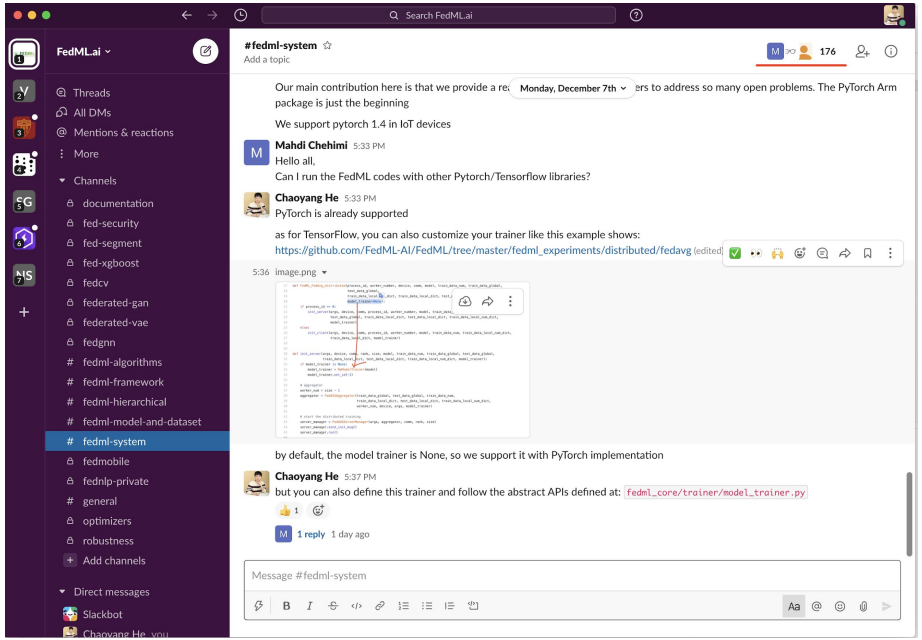
Fig. 1: Lab setup for detecting IoT botnet attacks

[1] N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders. IEEE PERVASIVE COMPUTING 2018

FedML Community

FEDML
MAKING COLLABORATION INTELLIGENT

slack
fedml.slack.com



Growth Statistics:

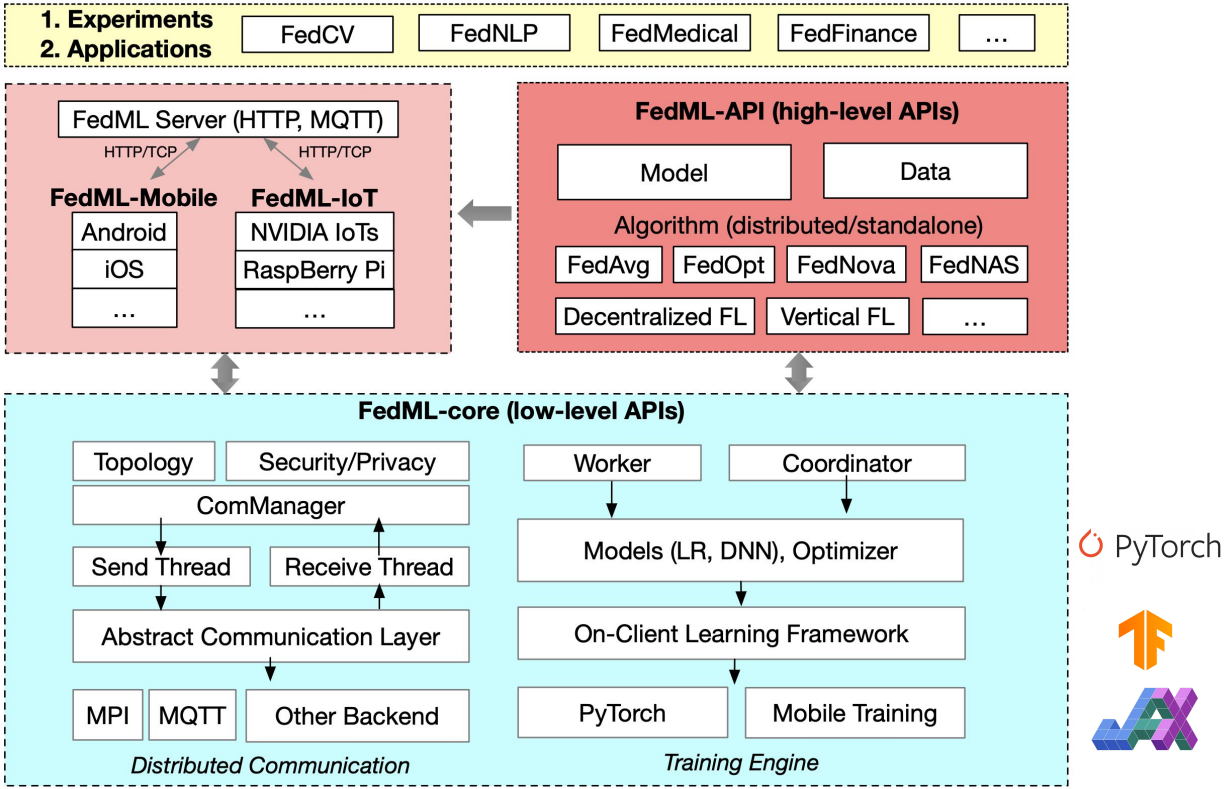
Slack Community Users: 640

GitHub Stars: 780
GitHub Forks: 216

GitHub Ranking in “Federated Learning”: 3
(the first two are industry leaders: Tencent/WeBank, Google)

Short Paper won Best Paper Awards in NeurIPS 2020 SpicyFL Workshop

FedML System Overview



FedML-API - Simplicity is Our Key

```
# topology configuration
device = init_training_device(process_id, worker_number - 1, args.gpu_num_per_server)

# load data
dataset = load_data(args, args.dataset)
[train_data_num, test_data_num, train_data_global, test_data_global
train_data_local_num_dict, train_data_local_dict, test_data_local
                                                                    set

# create model
model = create_model(args, model_name=args.model, output_dim=dataset[

# start "federated averaging (FedAvg)"
FedML_FedAvg_distributed(process_id, worker_number, device, comm,
                         model, train_data_num, train_data_global, test_data_global,
                         train_data_local_num_dict, train_data_local_dict, test_data_local_dict, args)
```

- FedAvg, FedAvg_Robustness, FedOpt (server optimizer), FedNova (client optimizer) FedGKT, FedNAS, Decentralized FL, Vertical FL, Split Learning, etc.

FedML Feature List

Dataset	MNIST, Synthetic Federated EMNIST; Shakespeare; Fed-CIFAR100; stackoverflow (NWP); CIFAR10, CIFAR100, CINIC10 (ImageNet+CIFAR10) Google Landmark, COCO, ImageNet *Support non-IID partition tool for heterogeneous distribution*
Model	LR, CNN (2 layers), RNN (Bi-LSTM), ResNet, MobileNet V3, EfficientNet Transformer/BERT, etc.
Federated Optimizers and Algorithms	FedAvg, FedOpt (server optimizer, ICLR 2021), FedNova (client optimizer, NeurIPS 2020), FedAvg_Robustness (NeurIPS 2020), FedGKT (NeurIPS 2020), FedNAS (CVPRw 2020), Turbo-Aggregate, Decentralized FL (NeurIPS 2020 FL), Vertical FL, Split Learning, etc *Support both cross-device and cross-silo settings.*
Platform Supports	Distributed Computing, IoT/Mobile, Standalone Simulation

Three computing paradigms, SOTA optimizers, various datasets and models!

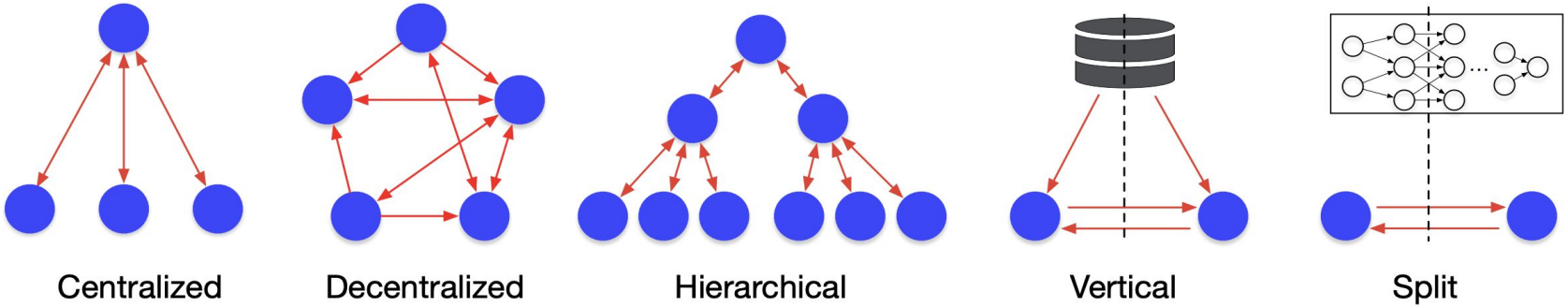
FedML Feature List - Privacy/Security

Dataset	MNIST, Synthetic Federated EMNIST; Shakespeare; Fed-CIFAR100; stackoverflow (NWP); CIFAR10, CIFAR100, CINIC10 (ImageNet+CIFAR10) Google Landmark, COCO, ImageNet *Support non-IID partition tool for heterogeneous distribution*
Model	LR, CNN (2 layers), RNN (Bi-LSTM), ResNet, MobileNet V3, EfficientNet Transformer/BERT, etc.
Federated Optimizers and Algorithms	FedAvg, FedOpt (server optimizer, ICLR 2021), FedNova (client optimizer, NeurIPS 2020), FedAvg_Robustness (NeurIPS 2020), FedGKT (NeurIPS 2020), FedNAS (CVPRw 2020), Turbo-Aggregate, Decentralized FL (NeurIPS 2020 FL), Vertical FL, Split Learning, etc *Support both cross-device and cross-silo settings.*
Platform Supports	Distributed Computing, IoT/Mobile, Standalone Simulation

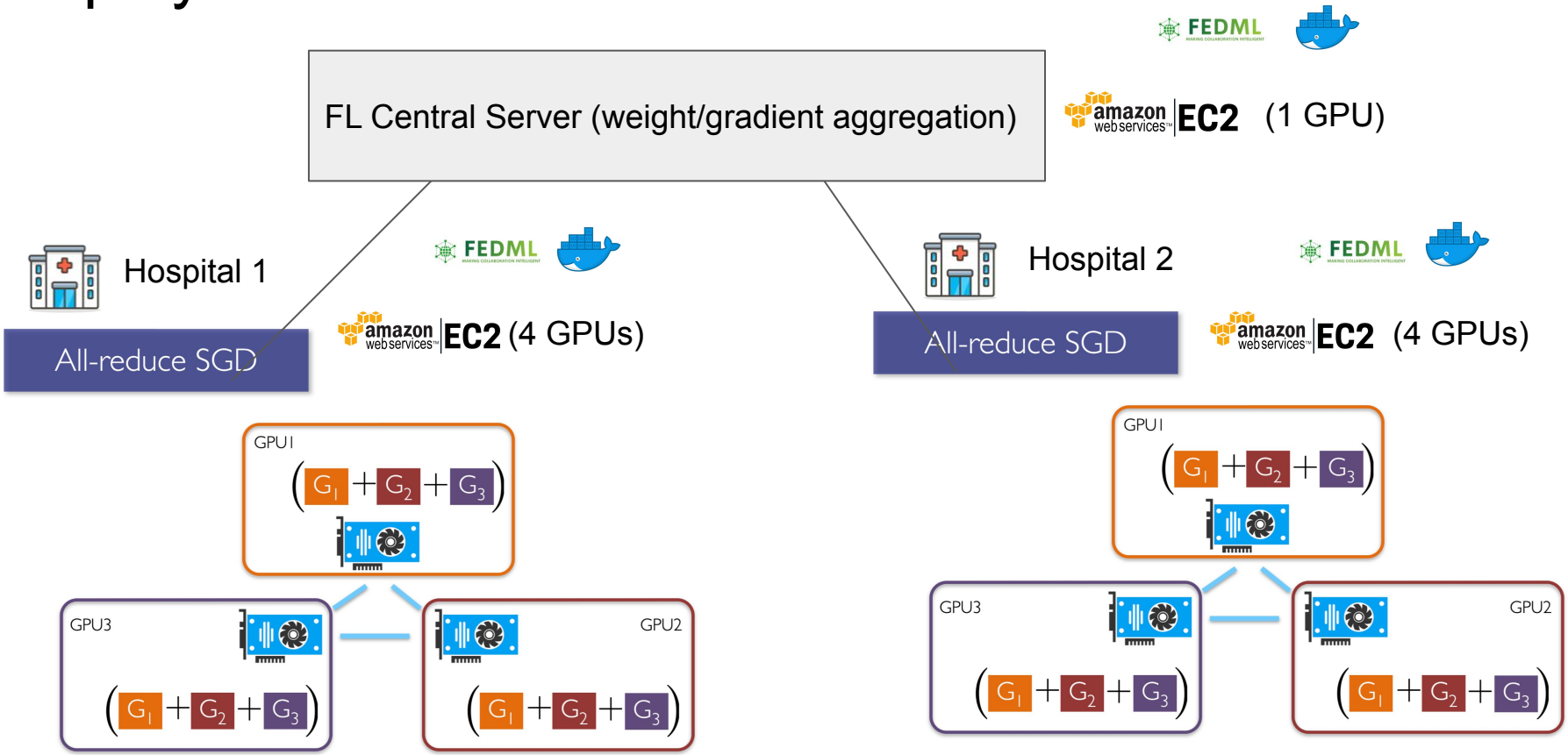
Three computing paradigms, SOTA optimizers, various datasets and models!

System Config diversity

Diverse topologies:



Deployment for Cross-silo FL

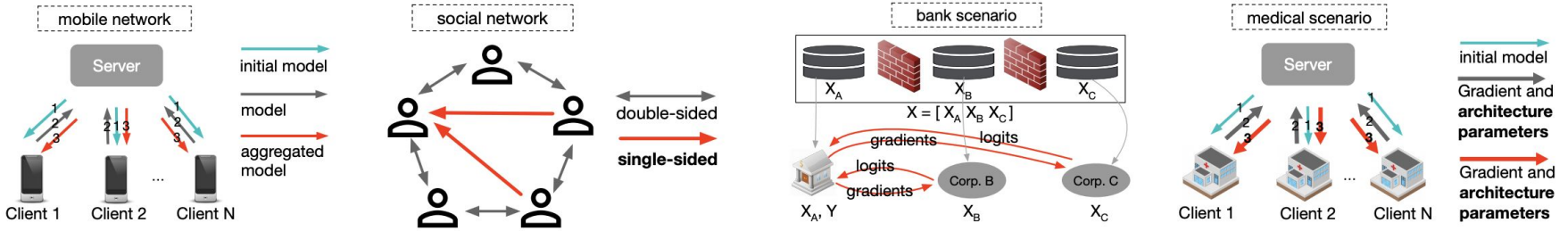


FedML Live Demo

FedML-core - Worker-oriented Programming Interface

```
import torch.distributed as dist
...
    ...
                Training procedure-oriented programming
    def train():
        ...
        model = Net()
        optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

```
class FedAvgClientManager(WorkerMananger):        worker-oriented
    def __init__(self, ..., topology_manager, trainer):    programming
        ...
        self.topology_manager = topology_manager
        self.trainer = trainer

    def register_message_receive_handlers(self):
```



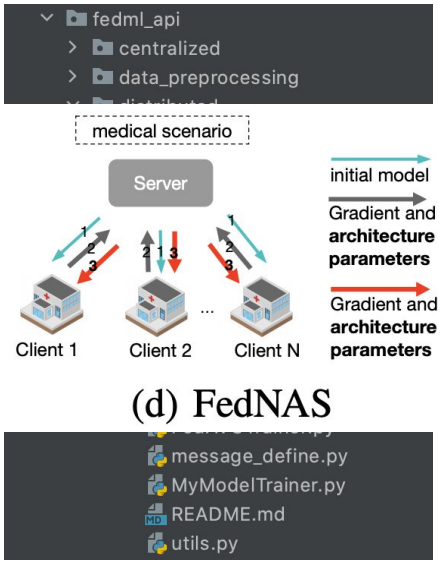

```
...
```

```
self.send_message_to_client(receive_id, params_B)
```

(a) Training procedure-oriented programming

(b) Worker-oriented programming

Figure 2: A worker-oriented programming design pattern of FedML.



FedML-core - Trainer Customization

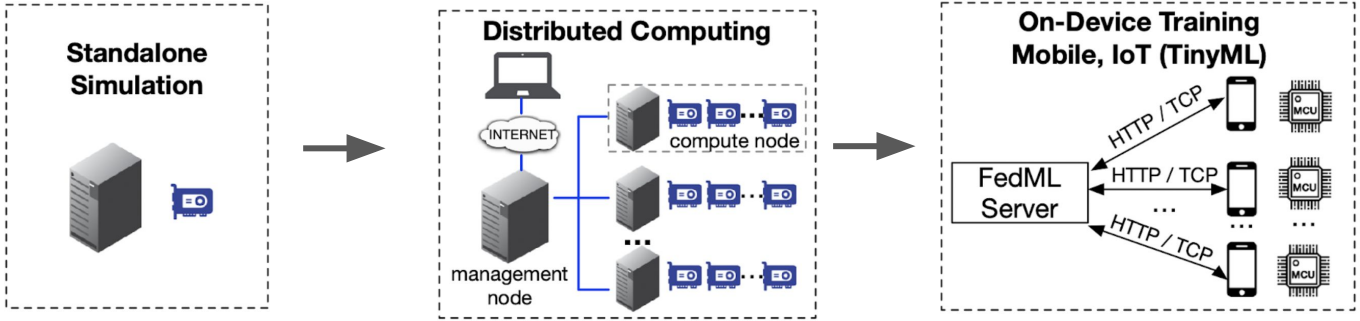
```
# topology configuration
device = init_training_device(process_id, worker_number - 1,
                              args.gpu_num_per_server)


# load data
dataset = load_data(args, args.dataset)
[train data num, test_data_num, train_data_global,
test data global,
train data local num dict, train data local_dict,
test_data_local_dict, class_num] = dataset


# create model
model = create_model(args, model name=args.model,
                     output_dim=dataset[7])

# start "federated averaging (FedAvg)"
FedML_FedAvg_distributed(process_id, worker_number, device,
                         Comm, model, train_data_num,
                         train_data_global, test_data_global,
                         train_data_local_num_dict,
                         train_data_local_dict,
                         test_data_local_dict, args,
                    model_trainer)
```

```
class ModelTrainer(ABC):
    """
    def __init__(self, model):
        self.model = model
        self.id = 0

    def set_id(self,
trainer_id):
        self.id = trainer_id

    @abstractmethod
    def set_model_params(self,

model_parameters):
        Pass

    @abstractmethod
    def train(self,
train_data,
              device, args):
        Pass

    @abstractmethod
    def test(self, test_data,
device,
             args):
        pass
```

PyTorch

FedML supports diverse platforms

```
class ModelTrainer(ABC):
    """
    def __init__(self, model):
        self.model = model
        self.id = 0

    def set_id(self,
trainer id):
        self.id = trainer_id

    @abstractmethod
    def set_model_params(self,

model_parameters):
        Pass

    @abstractmethod
    def train(self,
train data,
              device, args):
        Pass

    @abstractmethod
    def test(self, test_data,
device,
             args):
        pass
```



PyTorch

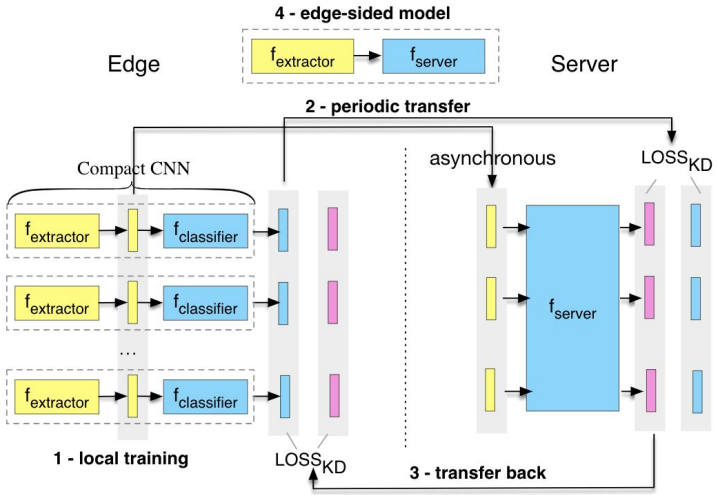
Write once, run everywhere:

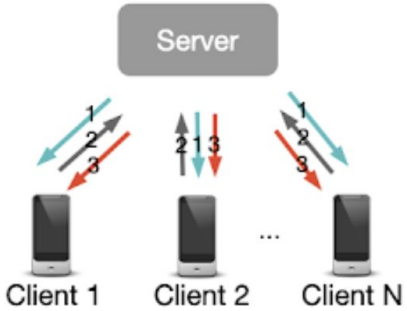
Reusing the same trainer class definition in three platforms

Let the library do the algorithm and platform-specific implementation

FedGKT: An Algorithmic Example of FedML:

FedGKT is a new distributed training framework, and requires specific communication protocol (*soft labels, hidden features*, etc) and training paradigms (*training models in client and server alternatively*).





Step 1: Model weight

Step 2&3: Hidden Representation, Logits (soft labels)

NEURAL INFORMATION PROCESSING SYSTEMS

USC University of Southern California

Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge

Chaoyang He
PhD Student, USC

Murali Annavaram
Professor, USC

Salman Avestimehr
Professor, USC

FEDML
MAKING COLLABORATION INTELLIGENT

https://fedml.ai

Introduction

- **Extremely Large DNN:** the SOTA model in NLP (Natural Language Processing) domain has billion or even trillion of parameters [1], and the SOTA model in CV (Computer Vision) domain has around 4 million parameters [2].

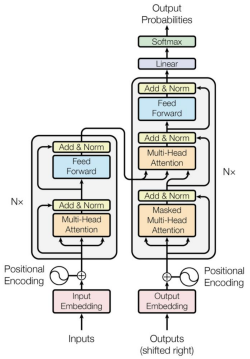
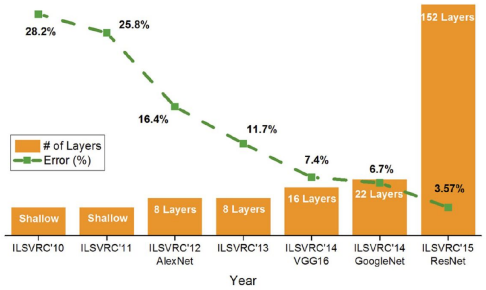
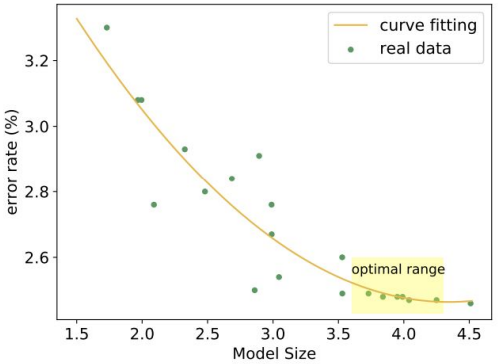


Figure 1: The Transformer - model architecture.



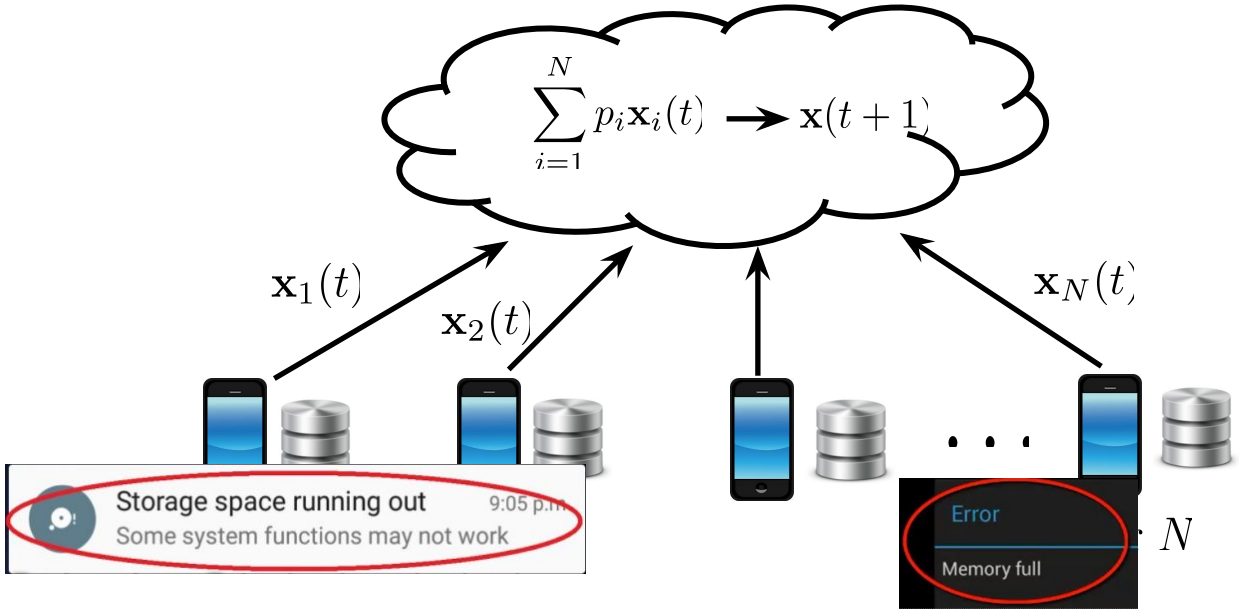


[1] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. 2019-10. https://arxiv.org/abs/1910.10683

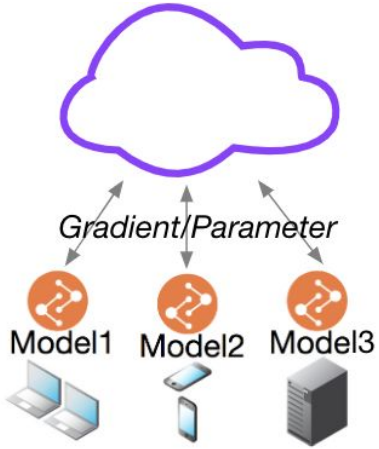
[2] MiLeNAS: Efficient Neural Architecture Search via Mixed-Level Reformulation. CVPR 2020
Chaoyang He, Haishan Ye, Li Shen, Tong Zhang.

Motivation

Edge devices are personal devices with limited memory and storage and computation capabilities

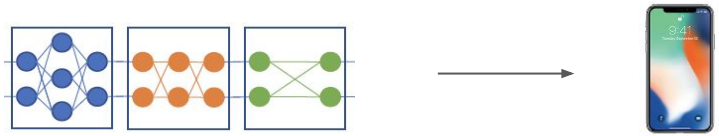


Motivation



Challenges of **Federated Deep Learning** - Training large modern DNN (ResNet, Transformers, etc) on-device is difficult due to data and system resource constraints:

1. Limited memory and computation (no GPU accelerator)
2. Limited bandwidth and unstable wireless communication
3. Data heterogeneity



How to efficiently and effectively **train** large DNN over resource-constrained edge devices?

Related Works: a Basic Formulation of Federated Learning

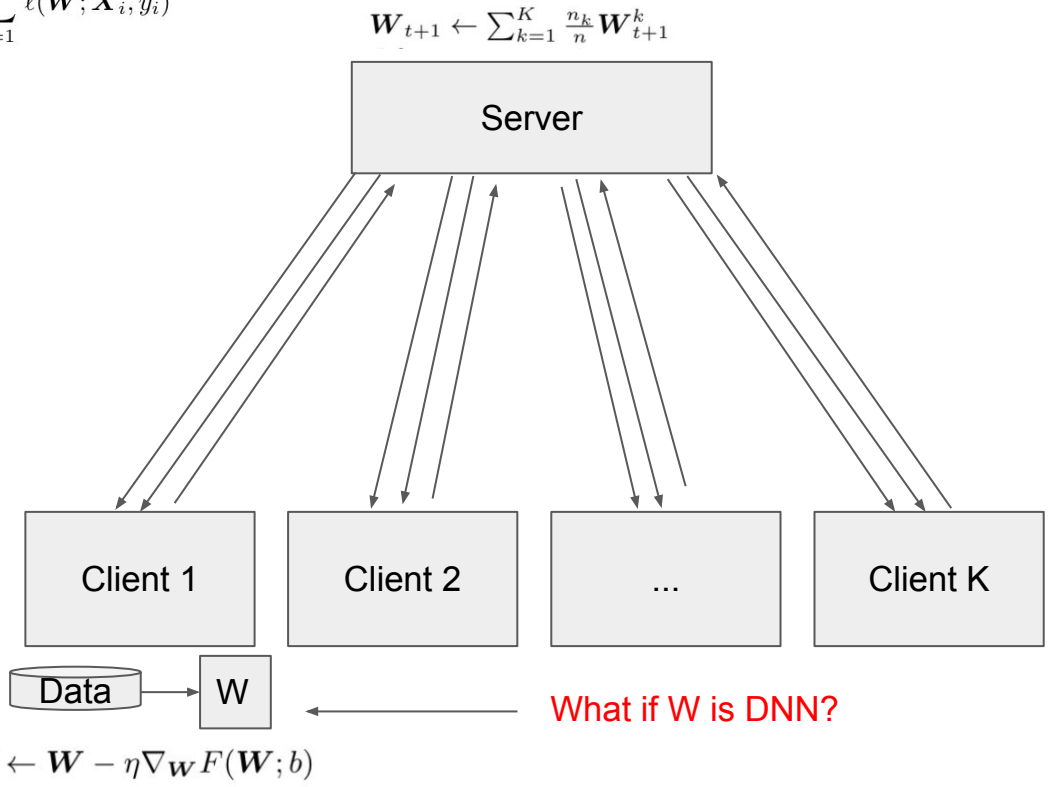
$$\min_{\boldsymbol{W}} F(\boldsymbol{W}) \stackrel{\text{def}}{=} \min_{\boldsymbol{W}} \sum_{k=1}^{K} \frac{N^{(k)}}{N} \cdot f^{(k)}(\boldsymbol{W}), \text{where } f^{(k)}(\boldsymbol{W}) = \frac{1}{N^{(k)}} \sum_{i=1}^{N^{(k)}} \ell(\boldsymbol{W}; \boldsymbol{X}_i, y_i)$$

Algorithm 1 `FedAvg` Algorithm: A Challenge Perspective

Initialization: there is a number of clients in a network; the client k has local dataset $\mathcal{D}^k$; each client's local model is initialized as $\boldsymbol{W}_0$;

Server_Executes:
for each round $t = 0, 1, 2, \ldots$ **do**
  $S_t \leftarrow$ (sample a random set of clients)
  for each client $k \in S_t$ **in parallel do**
    $\boldsymbol{W}_{t+1}^k \leftarrow \text{ClientUpdate}(k, \boldsymbol{W}_t)$
  end for
  $\boldsymbol{W}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \boldsymbol{W}_{t+1}^k$
end for

Client_Training$(k, \boldsymbol{W})$: *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{D}^k$ into batches)
for each local epoch i with $i = 1, 2, \cdots$ **do**
  for batch $b \in \mathcal{B}$ **do**
    $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} F(\boldsymbol{W}; b)$
  end for
end for
return $\boldsymbol{W}$ to server



[1] McMahan et al., Communication-Efficient Learning of Deep Networks from Decentralized Data, AISTATS 2017

Related Works: a Basic Formulation of Federated Learning

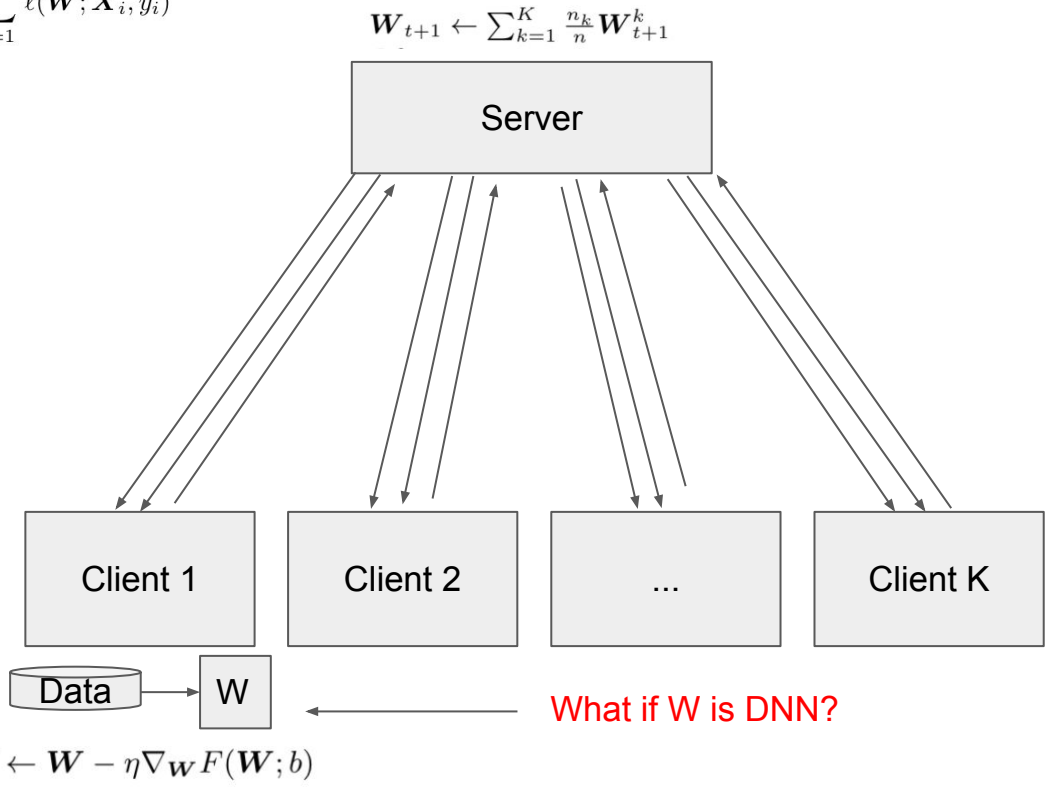
$$\min_{\boldsymbol{W}} F(\boldsymbol{W}) \stackrel{\text{def}}{=} \min_{\boldsymbol{W}} \sum_{k=1}^{K} \frac{N^{(k)}}{N} \cdot f^{(k)}(\boldsymbol{W}), \text{where } f^{(k)}(\boldsymbol{W}) = \frac{1}{N^{(k)}} \sum_{i=1}^{N^{(k)}} \ell(\boldsymbol{W}; \boldsymbol{X}_i, y_i)$$

Algorithm 1 `FedAvg` Algorithm: A Challenge Perspective

Initialization: there is a number of clients in a network; the client k has local dataset $\mathcal{D}^k$; each client's local model is initialized as $\boldsymbol{W}_0$;

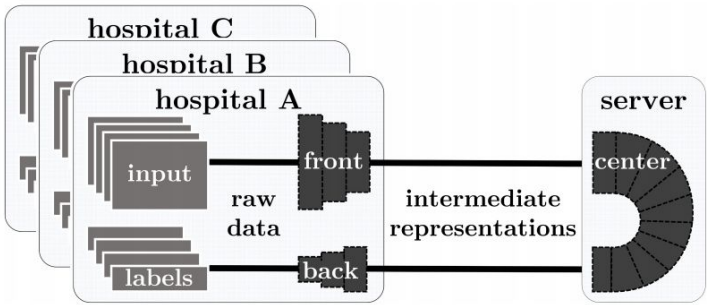
Server_Executes:
for each round $t = 0, 1, 2, \ldots$ **do**
  $S_t \leftarrow$ (sample a random set of clients)
  for each client $k \in S_t$ **in parallel do**
    $\boldsymbol{W}_{t+1}^k \leftarrow \text{ClientUpdate}(k, \boldsymbol{W}_t)$
  end for
  $\boldsymbol{W}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \boldsymbol{W}_{t+1}^k$
end for

Client_Training$(k, \boldsymbol{W})$: *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{D}^k$ into batches)
for each local epoch i with $i = 1, 2, \cdots$ **do**
  for batch $b \in \mathcal{B}$ **do**
    $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} F(\boldsymbol{W}; b)$
  end for
end for
return $\boldsymbol{W}$ to server



[1] McMahan et al., Communication-Efficient Learning of Deep Networks from Decentralized Data, AISTATS 2017

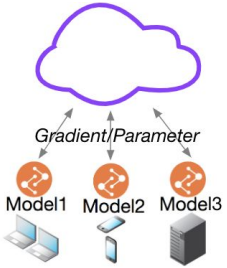
Related Works: Split Learning



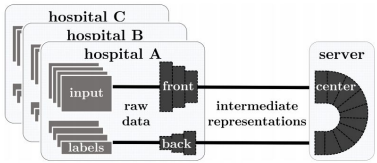
[2] Poirot et al., Split Learning for collaborative deep learning in healthcare, NeurIPS 2019

Related Works

Federated Learning:
Pros:
1) exchanging gradients/models periodically;
2) reusing *distributed optimization* methods in conventional distributed training in the data center environment.
Cons:
1) Communication cost for each gradient/model is much higher than a hidden vector
2) training large DNN models on device is prohibited due to resource constraints.



Split Learning:
Pros:
1) lower communication cost than FL: only exchanging the hidden vector of each training sample/mini-batch;
2) a portion of the entire DNN architecture is trainable on resource constrained devices. 3) hidden vector is more secure to resist adversarial attacks.
Cons:
1) Do not support periodical synchronization, which prohibits off-the-shell optimization methods; 2) the straggler problem becomes more severe because one SGD iteration has been split into four rounds of communication.



Insights

Can we design a learning framework that leverages advantages of FL and SL?

To be more specific, is there a framework that supports:

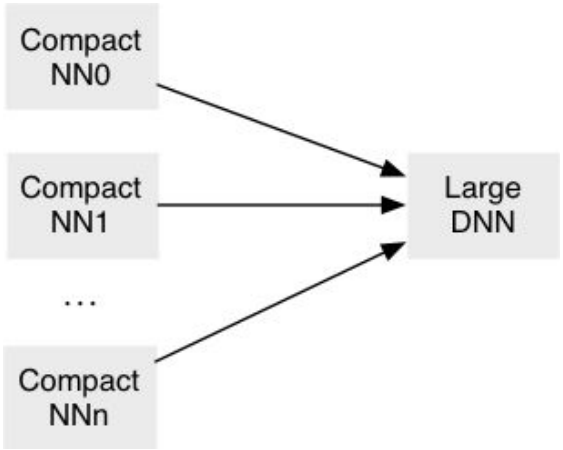
1. Computation Efficient: Computation-efficient on-device training (trainable) like SL
2. Communication Efficient: Exchanging hidden vectors during training like SL
3. Low Communication Frequency: Supporting periodical training like FL (local SGD)
4. No Accuracy Compromise: Preserving or outperforming the model accuracy of FL and SL

Our Idea

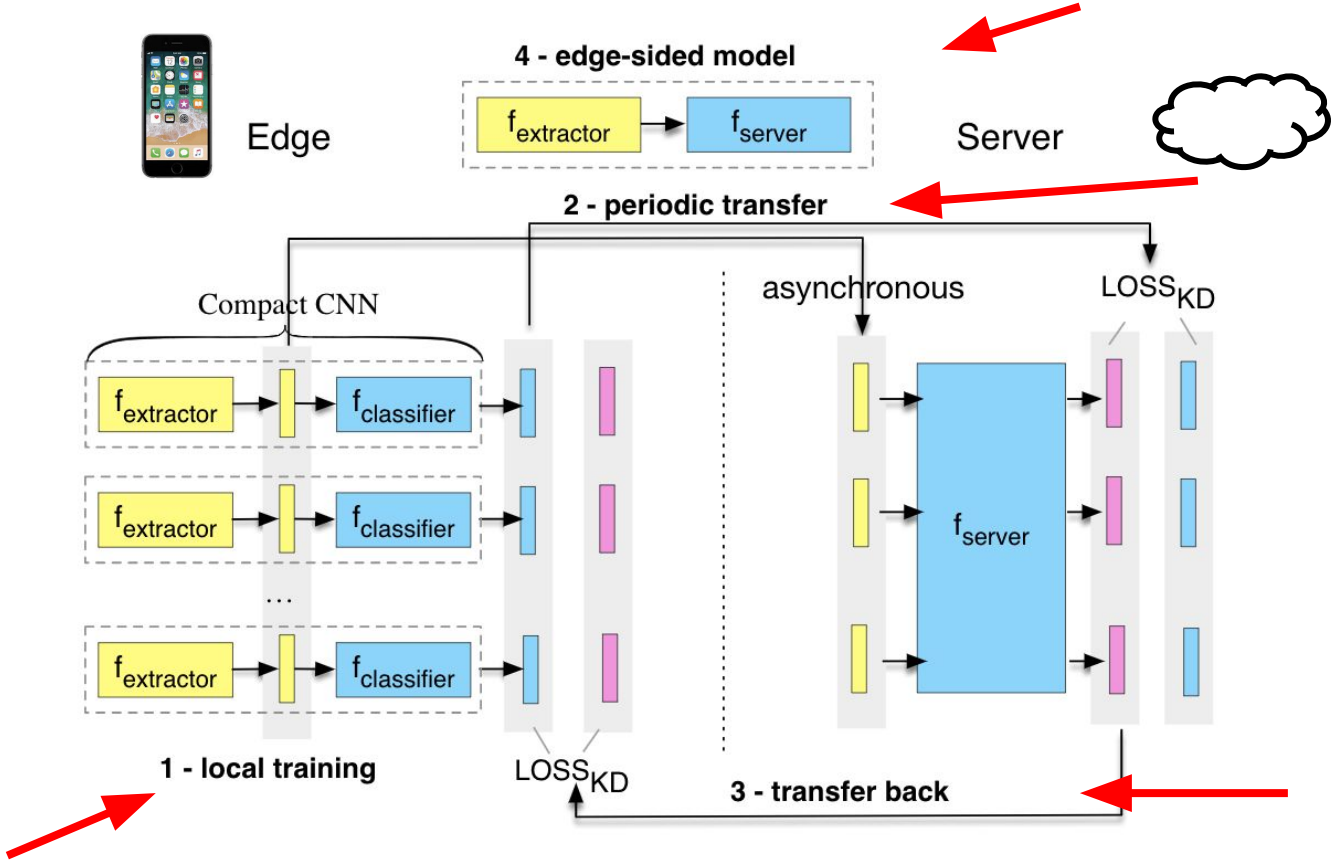
1. Transfer Data (Centralized Training)
2. Transfer Model (Federated Learning)

→ 3. Can we only transfer “knowledge”?

Transfer knowledge from many small networks to a larger one which has more capacity to obtain high accuracy

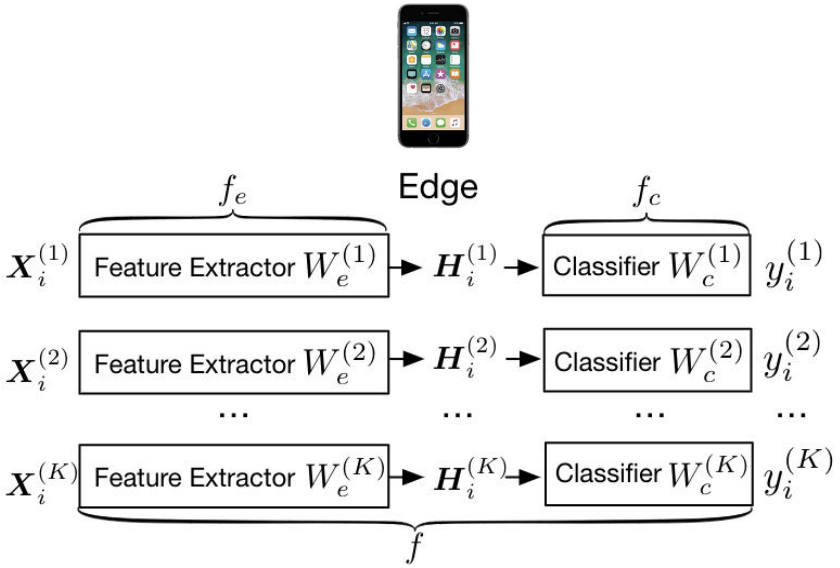


FedGKT Overview



Problem Formulation

First Formulation

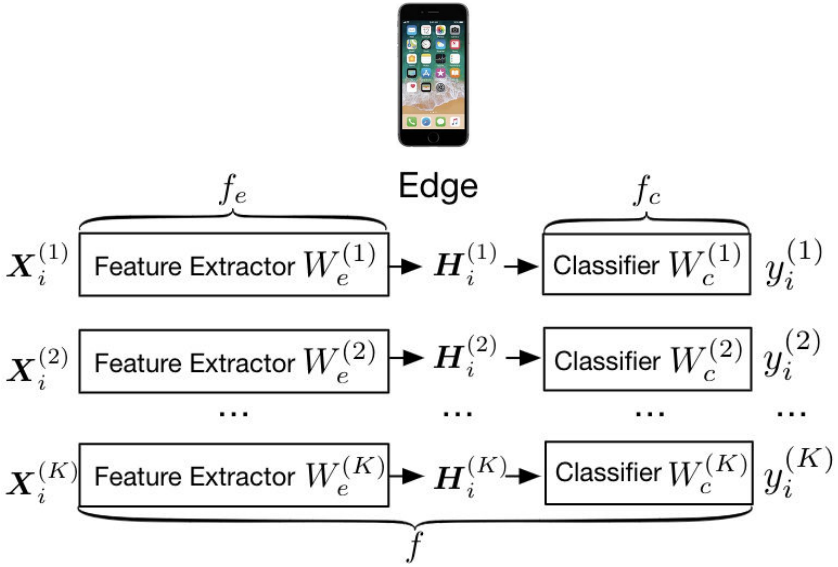


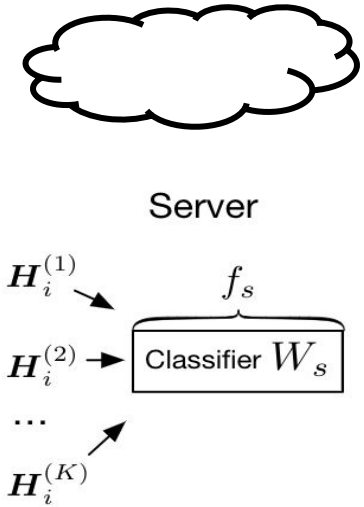
$$\underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} F_c(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)}) = \underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} \sum_{i=1}^{N^{(k)}} \ell_c\big(f_c^{(k)}(\boldsymbol{W}_c^{(k)}; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), y_i^{(k)}\big)$$

find a good and small feature extractor

Problem Formulation

First Formulation: ideal





$$\underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} F_c(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)}) = \underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} \sum_{i=1}^{N^{(k)}} \ell_c\big(f_c^{(k)}(\boldsymbol{W}_c^{(k)}; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)};\boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), y_i^{(k)}\big)$$

find a good and small feature extractor

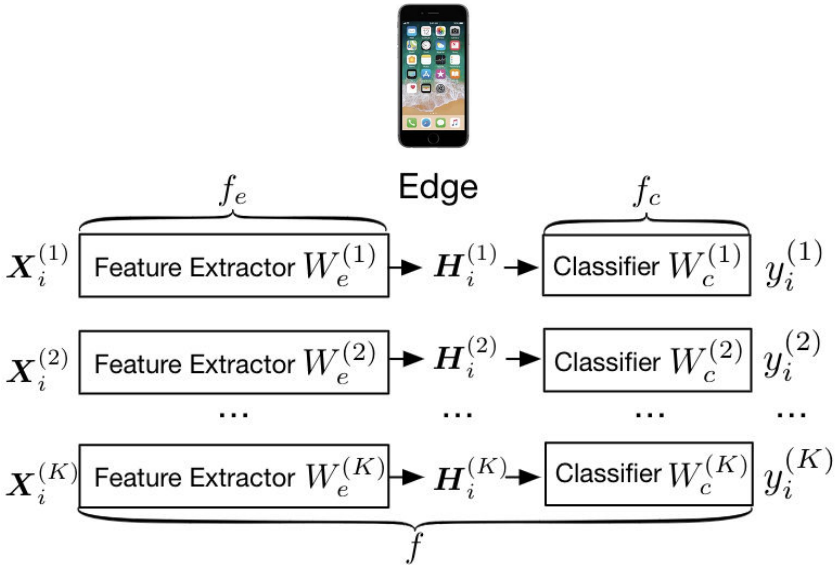
$$\underset{\boldsymbol{W}_s}{\operatorname{argmin}} F_s(\boldsymbol{W}_s,\boldsymbol{W}_e^*) = \underset{\boldsymbol{W}_s}{\operatorname{argmin}} \sum_{k=1}^{K}\sum_{i=1}^{N^{(k)}} \ell_s\left(f_s(\boldsymbol{W}_s;\boldsymbol{H}_i^{(k)}), y_i^{(k)}\right)$$

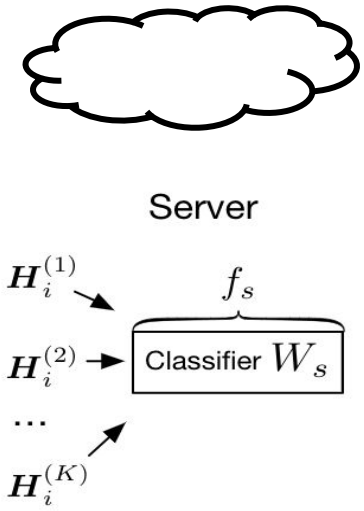
$$\textit{subject to: } \boldsymbol{H}_i^{(k)} = f_e^{(k)}(\boldsymbol{W}_e^{(k)};\boldsymbol{X}_i^{(k)})$$

find a great and large classifier by leveraging users' feature extractors

Problem Formulation

First Formulation: ideal





$$\underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} F_c(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)}) = \underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} \sum_{i=}^{N^{(k)}}$$

$$= \underset{\boldsymbol{W}_s}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i=1}^{N^{(k)}} \ell_s \left(f_s(\boldsymbol{W}_s; \boldsymbol{H}_i^{(k)}), y_i^{(k)} \right)$$

$$= f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{X}_i^{(k)})$$

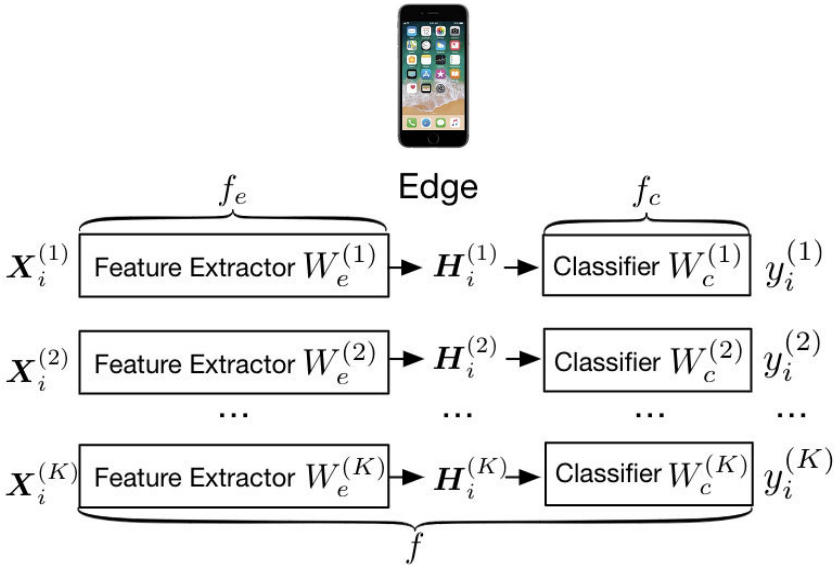
- light-weight model at the clients
- large/effective model at the server
- small communication requirement: feature size is much smaller than the model size

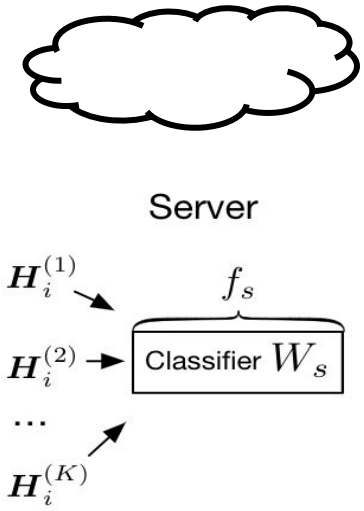
find a good and small f

large classifier by leveraging
rs' feature extractors

Problem Formulation

First Formulation: practical





$$\underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} F_c(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)}) = \underset{(\boldsymbol{W}_e^{(k)},\boldsymbol{W}_c^{(k)})}{\operatorname{argmin}} \sum_{i=}^{N}$$

$$= \underset{\boldsymbol{W}_s}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i=1}^{N^{(k)}} \ell_s \left(f_s(\boldsymbol{W}_s; \boldsymbol{H}_i^{(k)}), y_i^{(k)} \right)$$

$$= f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{X}_i^{(k)})$$

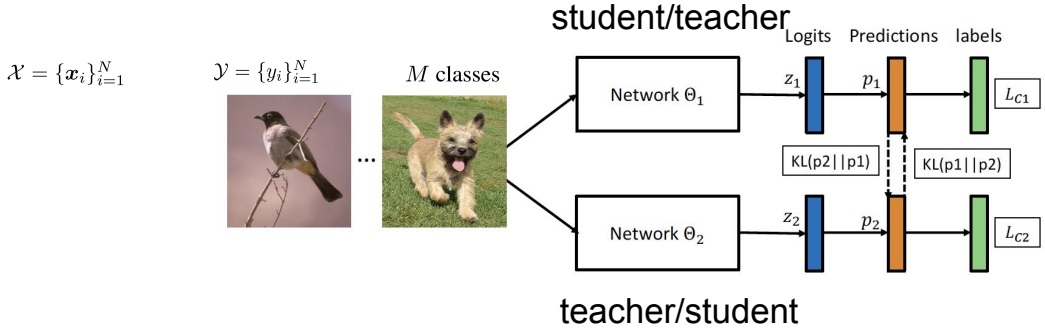
- users can't train good feature extractors alone with its small data
- users' poor feature extractors also mess up the training at the server

find a good and small feature extractor

arge classifier by leveraging users' feature extractors

Knowledge Distillation

- **Bidirectional Knowledge Distillation (CVPR 2018, ICLR 2018)**



$$p_1^m(\boldsymbol{x}_i) = \frac{exp(z_1^m)}{\sum_{m=1}^{M} exp(z_1^m)} \qquad L_{C_1} = -\sum_{i=1}^{N}\sum_{m=1}^{M} I(y_i, m)\log(p_1^m(\boldsymbol{x}_i)) \qquad D_{KL}(\boldsymbol{p}_2\|\boldsymbol{p}_1) = \sum_{i=1}^{N}\sum_{m=1}^{M} p_2^m(\boldsymbol{x}_i)\log\frac{p_2^m(\boldsymbol{x}_i)}{p_1^m(\boldsymbol{x}_i)}.$$

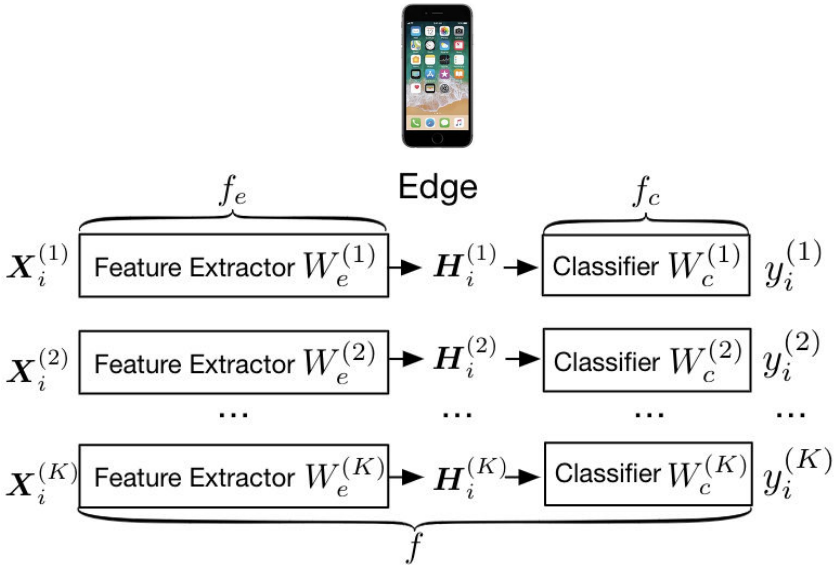
Transfer from 2 to 1:

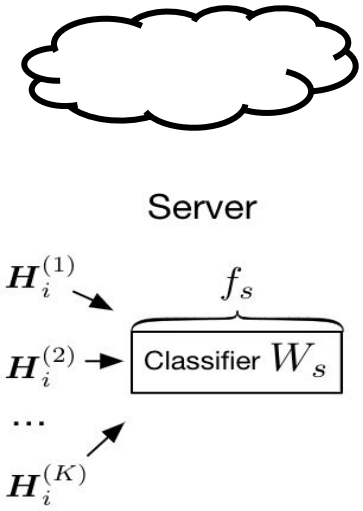
The overall loss function L_{Θ_1} for network Θ_1 is defined as

$$L_{\Theta_1} = L_{C_1} + D_{KL}(\boldsymbol{p}_2\|\boldsymbol{p}_1)$$

Problem Formulation

Reformulation as Bidirectional Knowledge Transfer





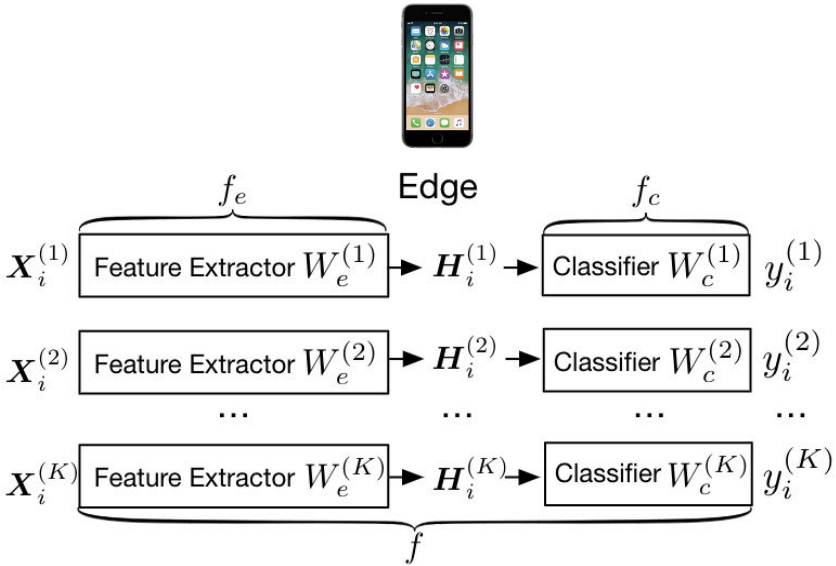
$$\underset{\boldsymbol{W}^{(k)}}{\operatorname{argmin}} F_c(\boldsymbol{W}_s^*, \boldsymbol{W}^{(k)}) = \text{CE loss of the classifier} + \ell_{KD}\left(\boldsymbol{z}_s^*, \boldsymbol{z}_c^{(k)}\right)$$

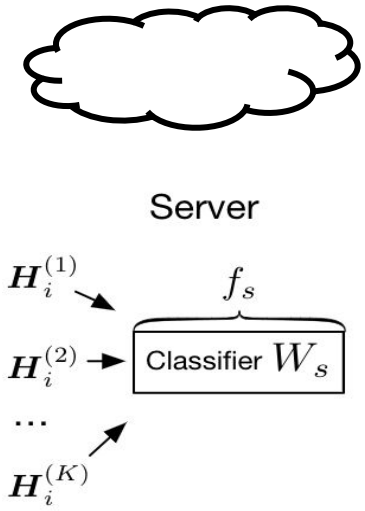
$$\underset{\boldsymbol{W}_s}{\operatorname{argmin}} F_s(\boldsymbol{W}_s, \boldsymbol{W}_e^{(k)*}) = \text{CE loss of the classifier} + \sum_{k=1}^{K} \ell_{KD}\left(\boldsymbol{z}_c^{(k)*}, \boldsymbol{z}_s\right)$$

The logits z_s and z_c are the output of the last fully connected layer in the server model and the client model

Problem Formulation

Reformulation as Bidirectional Knowledge Transfer



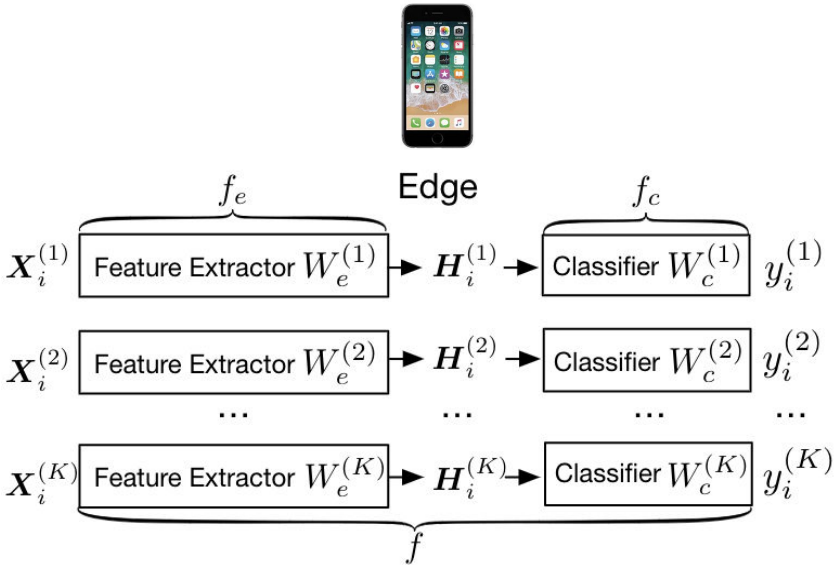


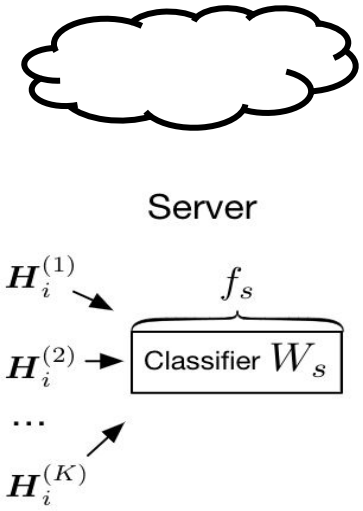
$$\underset{\boldsymbol{W}^{(k)}}{\operatorname{argmin}} F_c(\boldsymbol{W}_s^*, \boldsymbol{W}^{(k)}) = \text{CE loss of the classifier} + \ell_{KD}\left(\boldsymbol{z}_s^*, \boldsymbol{z}_c^{(k)}\right) \qquad \underset{\boldsymbol{W}_s}{\operatorname{argmin}} F_s(\boldsymbol{W}_s, \boldsymbol{W}_e^{(k)*}) = \text{CE loss of the classifier} + \sum_{k=1}^{K} \ell_{KD}\left(\boldsymbol{z}_c^{(k)*}, \boldsymbol{z}_s\right)$$

The logits z_s and z_c are the output of the last fully connected layer in the server model and the client model

Problem Formulation

Reformulation as Bidirectional Knowledge Transfer





$$\underset{\boldsymbol{W}^{(k)}}{\operatorname{argmin}} F_c(\boldsymbol{W}_s^*, \boldsymbol{W}^{(k)}) = \text{CE loss of the classifier} + \ell_{KD}\left(\boldsymbol{z}_s^*, \boldsymbol{z}_c^{(k)}\right)$$

$$\underset{\boldsymbol{W}_s}{\operatorname{argmin}} F_s(\boldsymbol{W}_s, \boldsymbol{W}_e^{(k)*}) = \text{CE loss of the classifier} + \sum_{k=1}^{K} \ell_{KD}\left(\boldsymbol{z}_c^{(k)*}, \boldsymbol{z}_s\right)$$

users bring their predictions closer to server model's prediction

- users absorb the server model knowledge to improve feature extraction
- server absorbs users' knowledge to train a better classifier

FedGKT: Alternating Minimization

Server

$$\underset{\boldsymbol{W}_s}{\operatorname{argmin}} F_s(\boldsymbol{W}_s, \boldsymbol{W}_e^{(k)*}) = \underset{\boldsymbol{W}_s}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i=1}^{N^{(k)}} \ell_{CE}\big(f_s(\boldsymbol{W}_s; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)*}; \boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), y_i^{(k)}\big) + \sum_{k=1}^{K} \ell_{KD}\big(\boldsymbol{z}_c^{(k)*}, \boldsymbol{z}_s\big) \tag{8}$$

$$\textit{where } \boldsymbol{z}_c^{(k)*} = f_c^{(k)}(\boldsymbol{W}_c^{(k)}; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)*}; \boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), \textit{and } \boldsymbol{z}_s = f_s(\boldsymbol{W}_s; \boldsymbol{H}_i^{(k)}) \tag{9}$$

$$\underset{\boldsymbol{W}^{(k)}}{\operatorname{argmin}} F_c(\boldsymbol{W}_s^*, \boldsymbol{W}^{(k)}) = \underset{\boldsymbol{W}^{(k)}}{\operatorname{argmin}} \sum_{i=1}^{N^{(k)}} \ell_{CE}\big(f_c^{(k)}(\boldsymbol{W}_c^{(k)}; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), y_i^{(k)}\big) + \ell_{KD}\big(\boldsymbol{z}_s^*, \boldsymbol{z}_c^{(k)}\big) \tag{10}$$

Client

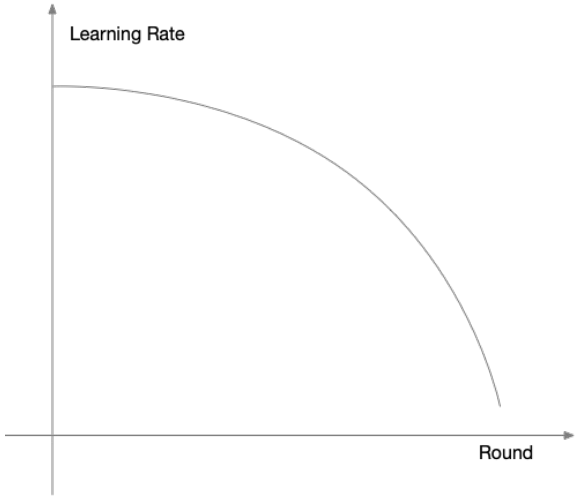
$$\textit{where } \boldsymbol{z}_c^{(k)} = f_c^{(k)}(\boldsymbol{W}_c^{(k)}; \underbrace{f_e^{(k)}(\boldsymbol{W}_e^{(k)}; \boldsymbol{X}_i^{(k)})}_{\boldsymbol{H}_i^{(k)}}), \textit{and } \boldsymbol{z}_s^* = f_s(\boldsymbol{W}_s^*; \boldsymbol{H}_i^{(k)}) \tag{11}$$

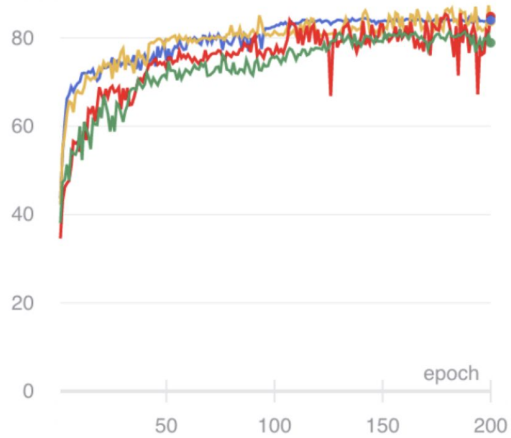
Method: Distributed Optimization Methods

1. Local SGD with Momentum

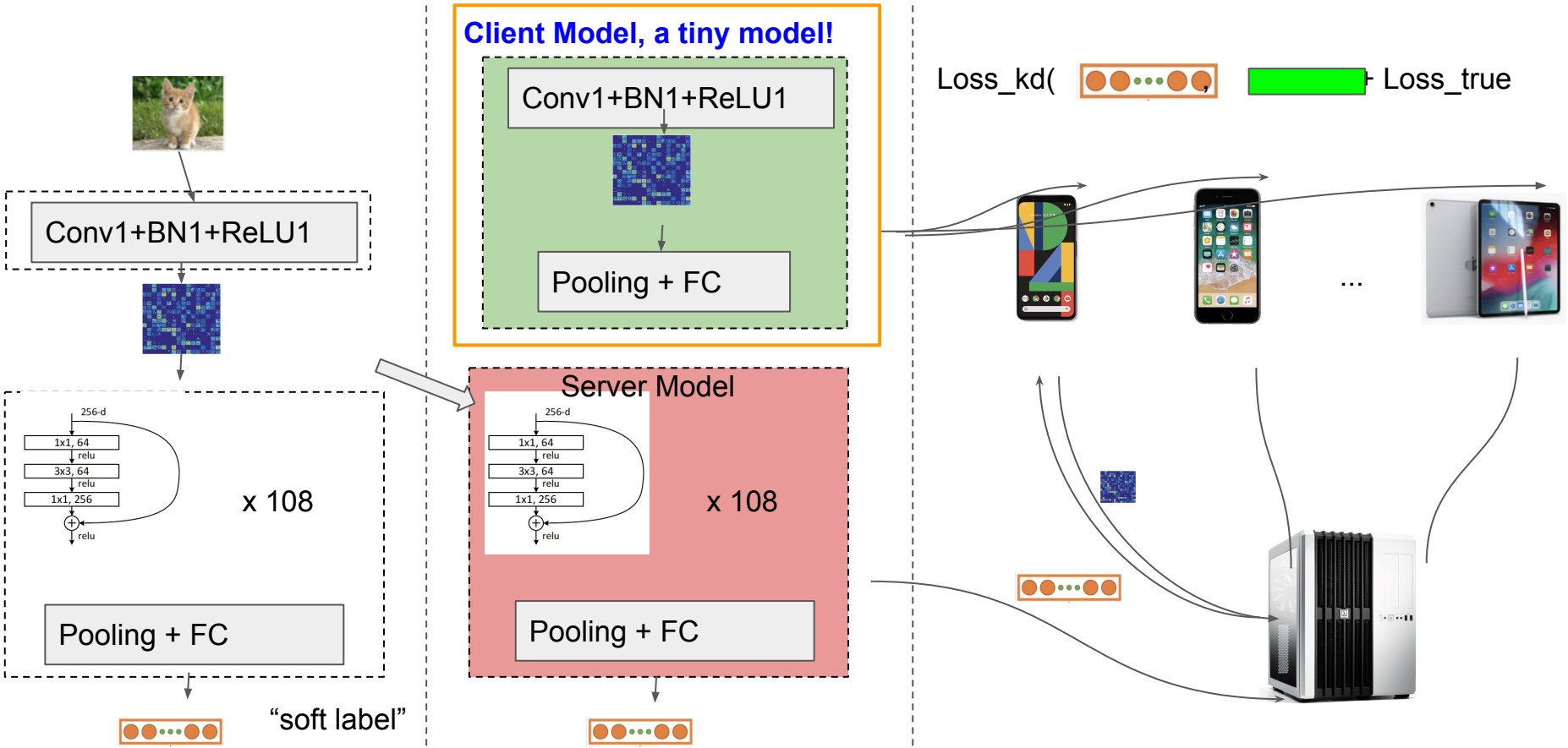
$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

2. Cross-round Learning Rate Scheduler



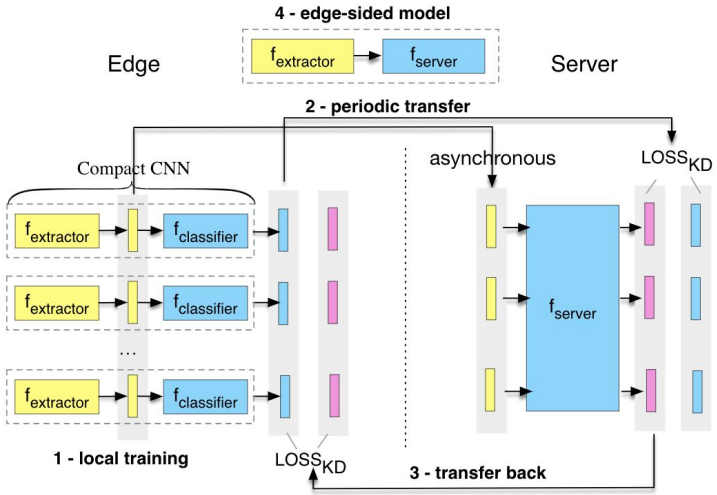


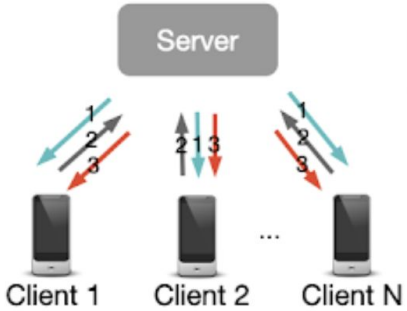
Method: Model Architecture



System Design and Implementation

FedGKT is a new distributed training framework, and requires specific communication protocol (*soft labels, hidden features*, etc) and training paradigms (*training models in client and server alternatively*).





Step 1: Model weight

Step 2&3: Hidden Representation, Logits (soft labels)

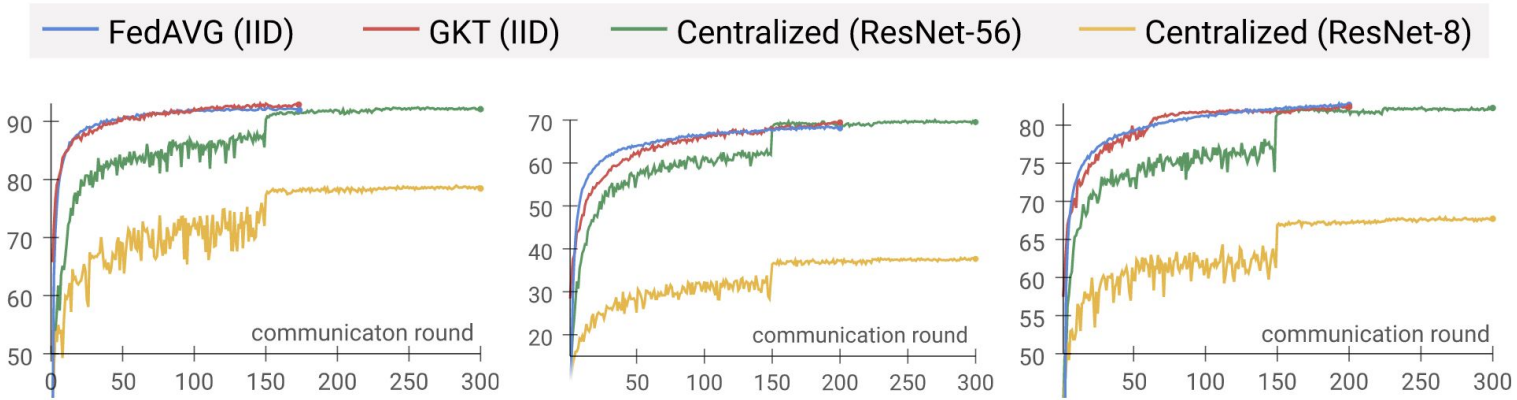
System Design and Implementation

With the help of FedML library, we can easily implement FedGKT algorithm and conduct various experiments with different datasets and models.

Homepage: https://fedml.ai

(Best Paper Award, NeurIPS 2020, FL Workshop)

Experimental Results: Test Accuracy



(a) ResNet-56 on CIFAR-10 (b) ResNet-56 on CIFAR-100 (c) ResNet-56 on CINIC-10

Figure 3: The Test Accuracy of ResNet-56 (Edge Number = 16)

Experimental Results: Non-IID

Table 1: The Test Accuracy of ResNet-56 and ResNet-110 on Three Datasets.

Model	Methods	CIFAR-10		CIFAR-100		CINIC-10	
		I.I.D.	non-I.I.D.	I.I.D.	non-I.I.D.	I.I.D.	non-I.I.D.
ResNet-56	**FedGKT (ResNet-8, ours)**	**92.97**	**86.59**	**69.57**	**63.76**	**81.51**	**77.80**
	FedAvg (ResNet-56)	92.88	86.60	68.09	63.78	81.62	77.85
	Centralized (ResNet-56)	93.05		69.73		81.66	
	Centralized (ResNet-8)	78.94		37.67		67.72	
ResNet-110	**FedGKT (ResNet-8, ours)**	**93.47**	**87.18**	**69.87**	**64.31**	**81.98**	**78.39**
	FedAvg (ResNet-110)	93.49	87.20	68.58	64.35	82.10	78.43
	Centralized (ResNet-110)	93.58		70.18		82.16	
	Centralized (ResNet-8)	78.94		37.67		67.72	

Experimental Results: Efficiency

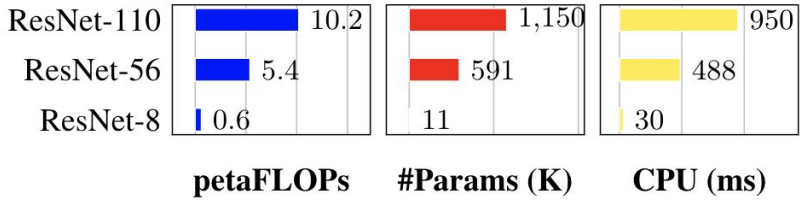


Figure 4: Edge Computational Efficiency (CIFAR-100)

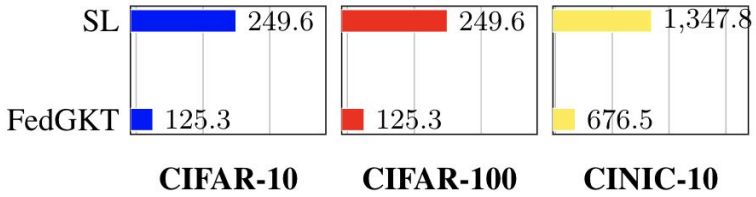
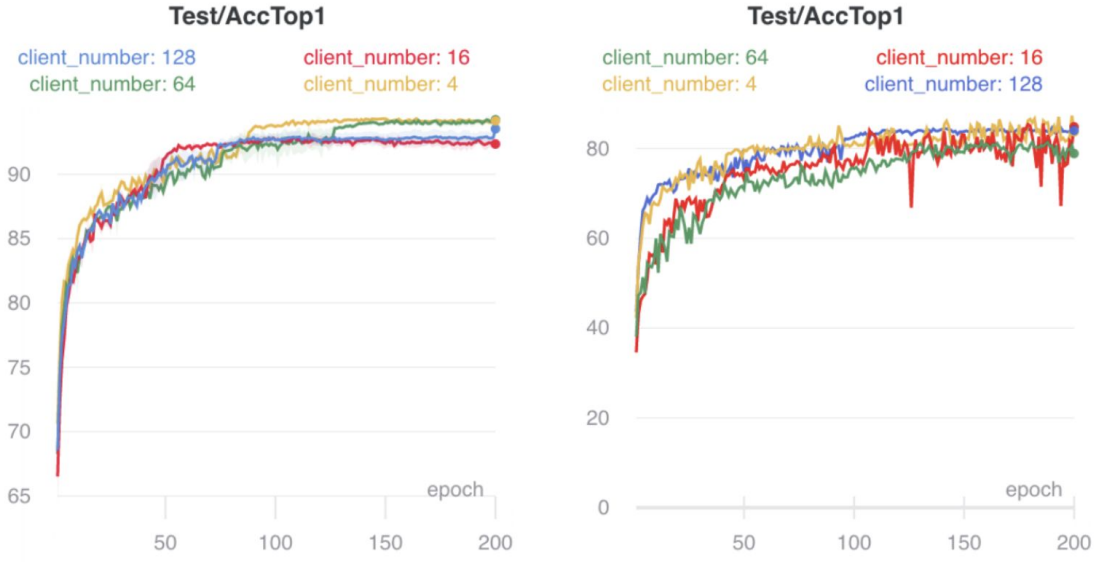


Figure 5: Communication Efficiency (ResNet-56)

FedGKT demands **9 to 17 times** less computational power (FLOPs) on edge devices and requires **54 to 105 times** fewer parameters in the edge CNN

Experimental Results: Ablation Study



(a) GKT on IID dataset

(b) GKT on non-IID dataset

Figure 5: GKT result on CIFAR10

Experimental Results: Ablation Study

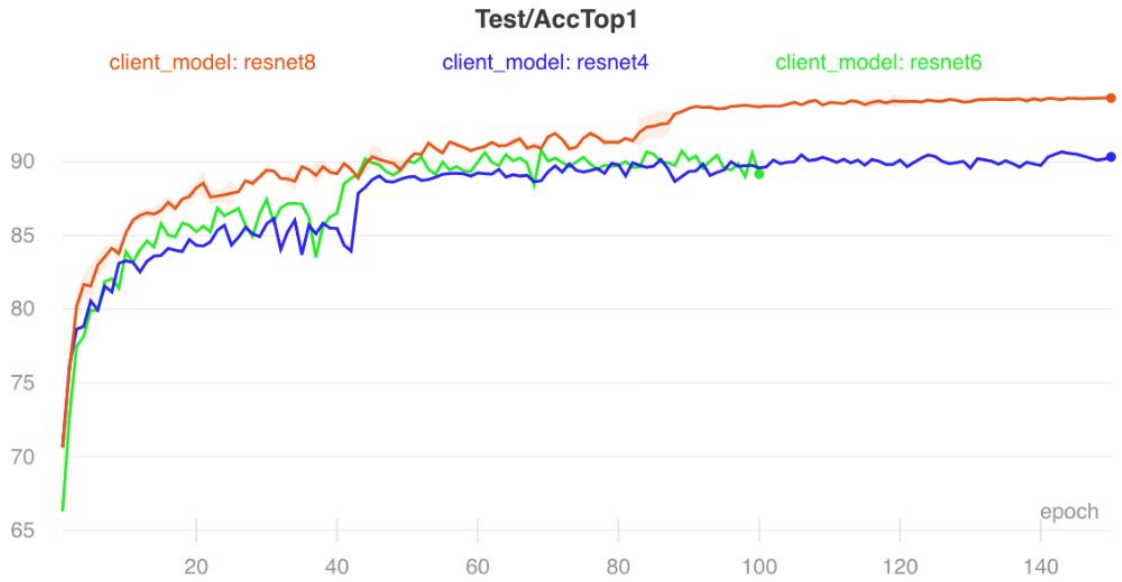


Figure 6: Client Model Architecture Exploration

Experimental Results: Ablation Study

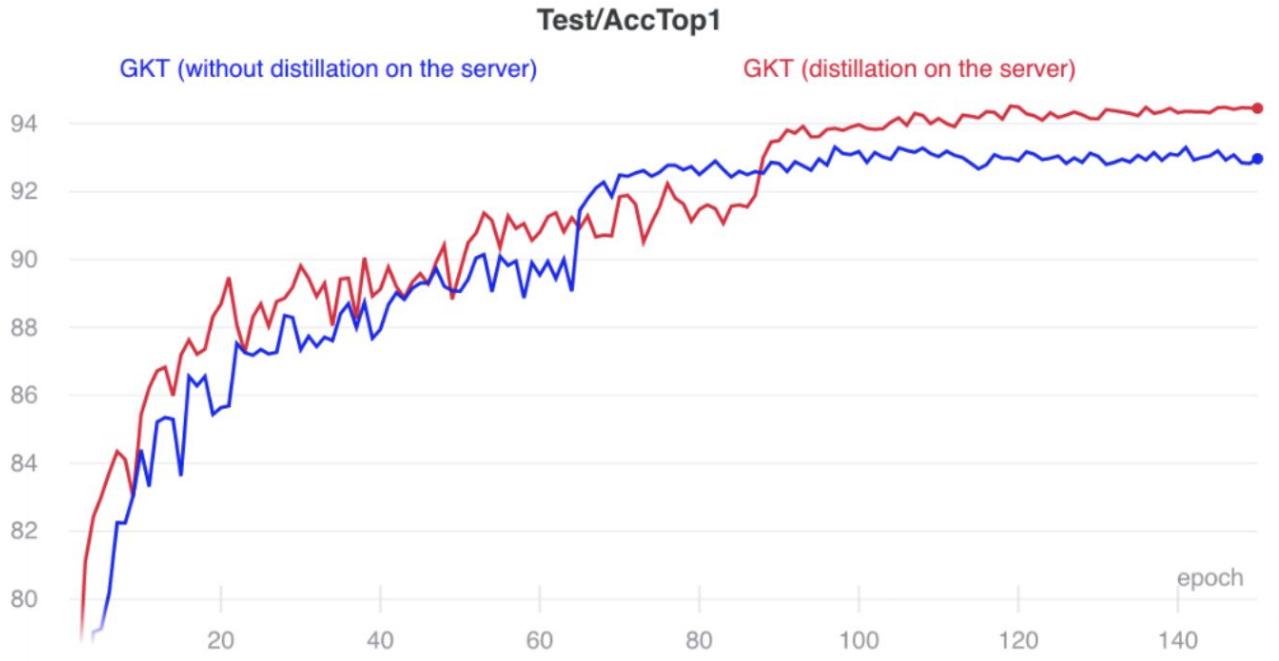


Figure 4: Evaluation on the distillation loss on the server side

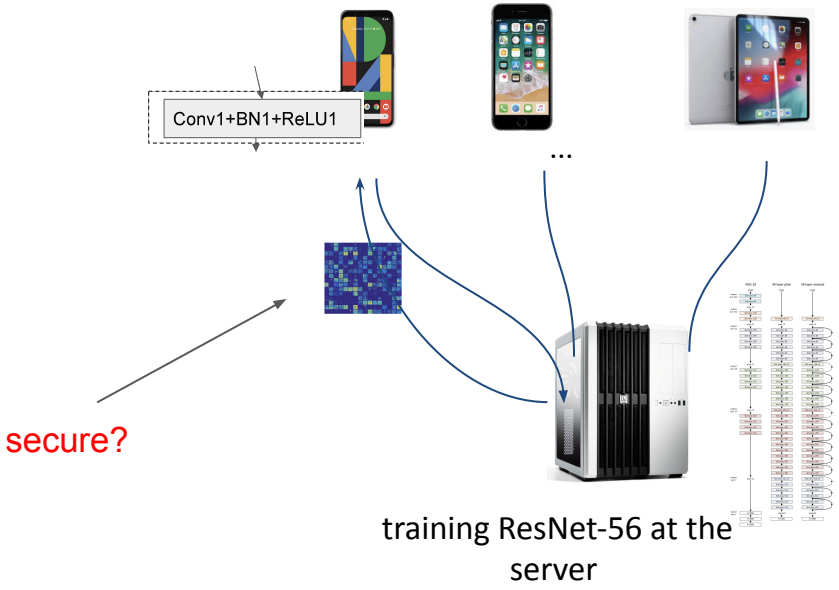
Contributions

FedGKT is the early work that explores federated deep learning and transfer knowledge (not data and model) on edge devices.

- FedGKT is **memory and computation efficient**, similar to SL
- FedGKT can train in a local SGD manner like FedAvg to **reduce the communication frequency**
- Exchanging hidden features as in SL, as opposed to exchanging the entire model as in FedAvg, **reduces the communication bandwidth requirement**
- FedGKT is **public data-free** knowledge transfer method
- FedGKT naturally supports **asynchronous** training, which circumvents the severe synchronization issue in SL

Future Works

- Exchanging hidden features provides “some” privacy guarantees to the users
  - Providing strong privacy/security guarantees for FedGKT is an interesting next step
  - Leveraging Secure Aggregation in FedGKT?



Future Works

- Extend FedGKT to Transformer Models

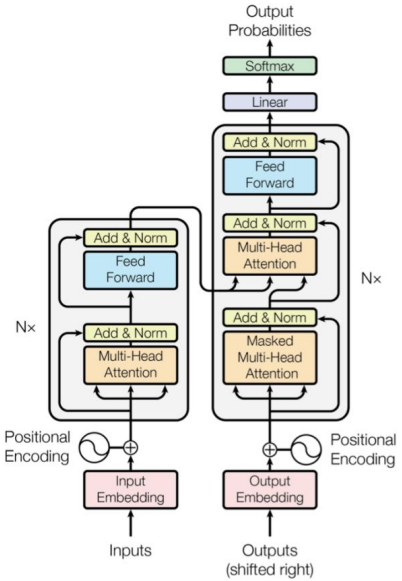


Figure 12. Transformer Model Architecture (Vaswani et al., 2017)

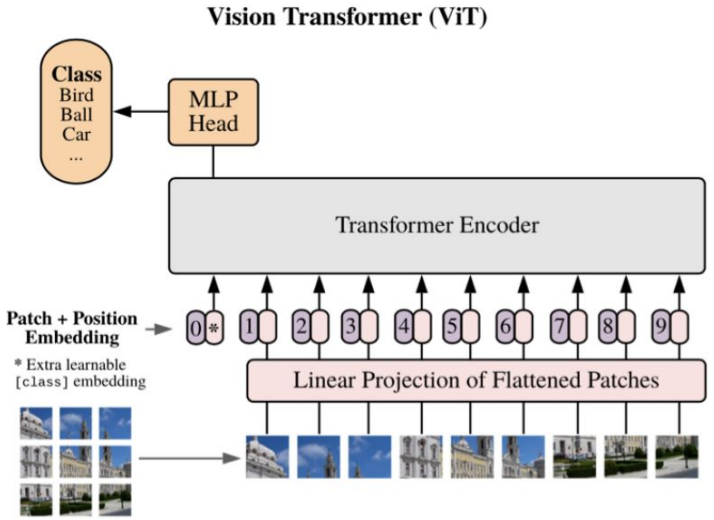
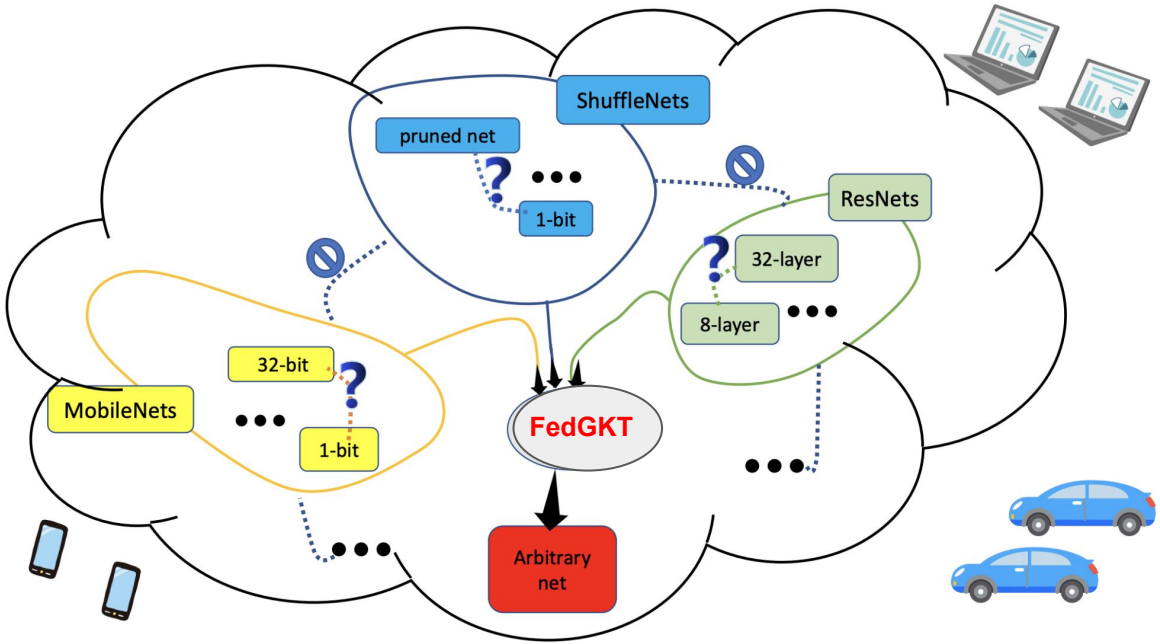


Figure 13. Vision Transformer (Dosovitskiy et al., 2020)

Future Works

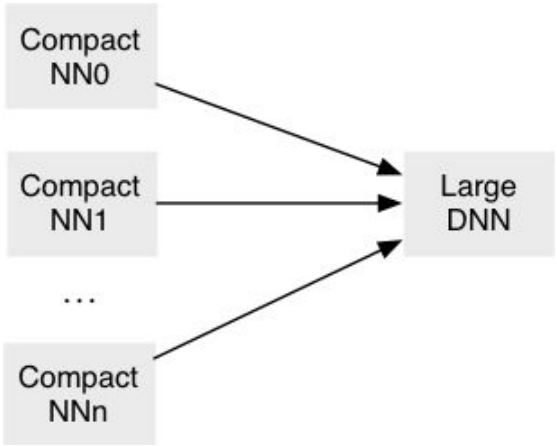
- FedGKT also allows for different models to be used at each user
  - Providing model adaptation and personalization at each user is also another next step



Long-term Goal

1. Transfer Data (Centralized Training)
2. Transfer Model (Federated Learning)

→ 3. Can we only transfer “knowledge”?

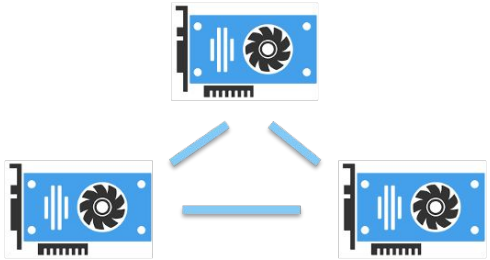


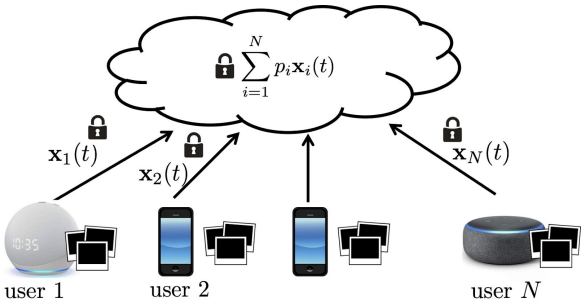
FedGKT is a starting point of our goal towards “Knowledgeable Communication”:

1) AI Agent (ML model) in Internet can freely exchange/transfer/share their “knowledge” without disclosing a large amount of raw data or model;

2) ML models can be equipped with communication networking interface to exchange neural representation with other models in a decentralized Internet.

Distributed Training v.s. Federated Learning





In essence, three methodologies are useful to both:

1. Dynamic system and stochastic ML lead to complicated interaction between these objectives.

PipeTransformer, ICML 2021

2. Trade-off among multiple objectives including accuracy, efficiency, security, and privacy.

Group Knowledge Transfer, NeurIPS 2020

3. On demand: focus on the key demand for different use cases (ML applications), optimizing one of the objectives but not deteriorating the others.

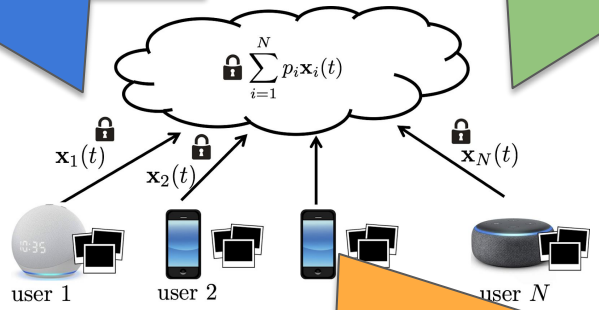
Federated Learning

Foundations of Algorithm Design

- ❑ User selection, optimal scaling
- ❑ Heterogeneity, personalization, and fairness
- ❑ Unsupervised federated learning

System Bottlenecks/Opportunities

- ❑ Resource-constrained FL (small edge models, large server models)
- ❑ Scalability: 1K users → 1M users
- ❑ Federated neural architecture search



Trustworthy

- ❑ Secure and resilient model aggregation
- ❑ Adversarial users (data/model poisoning)
- ❑ Leveraging trusted computing environments

Ongoing Works 1: Self-supervised Federated Learning

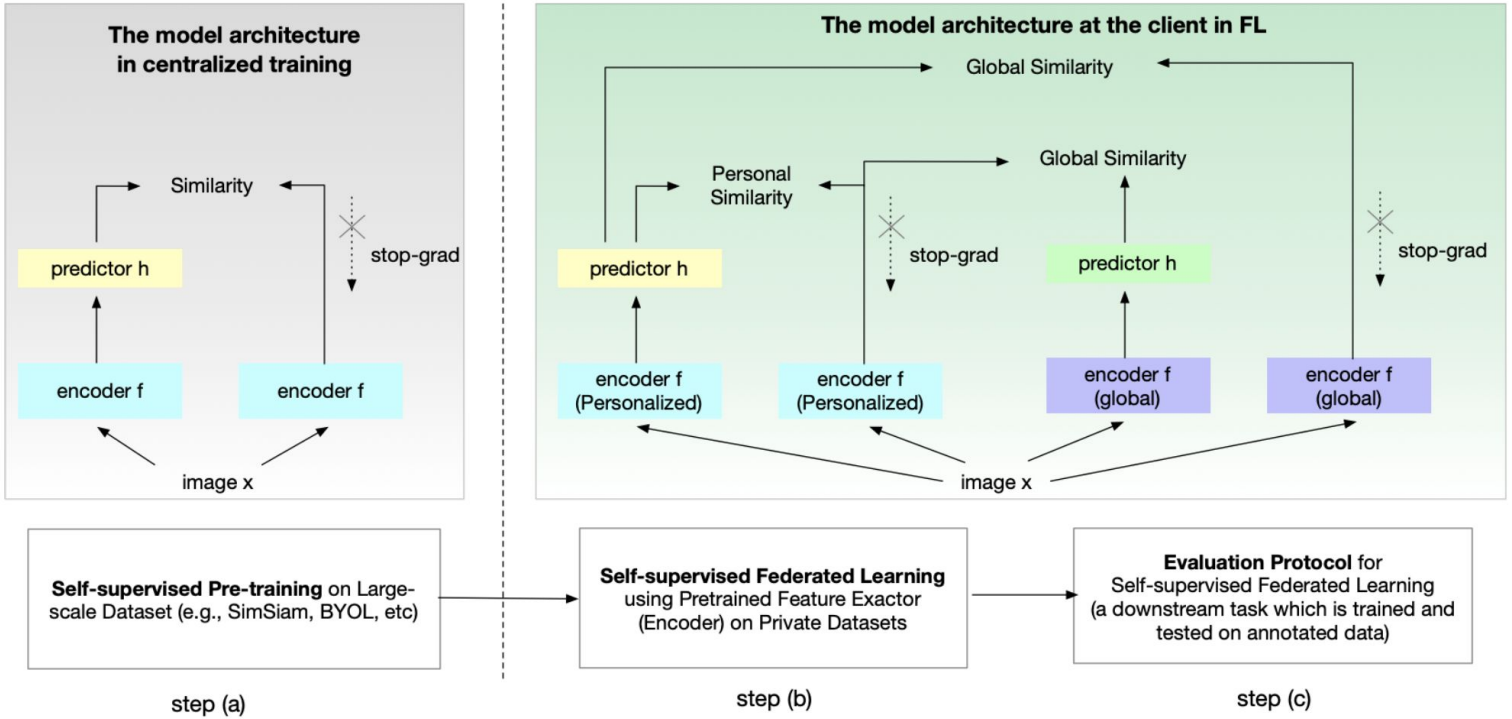
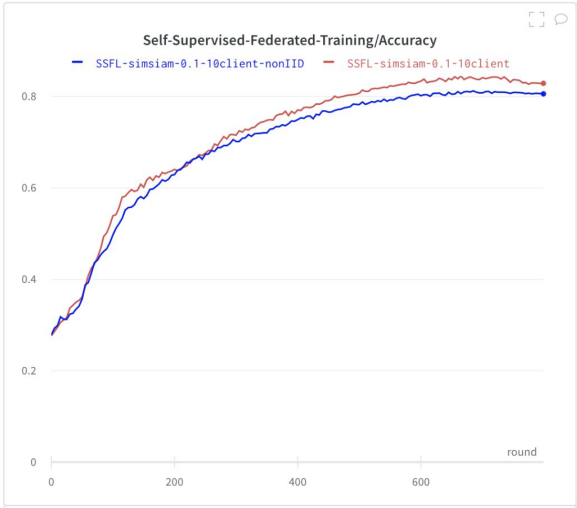
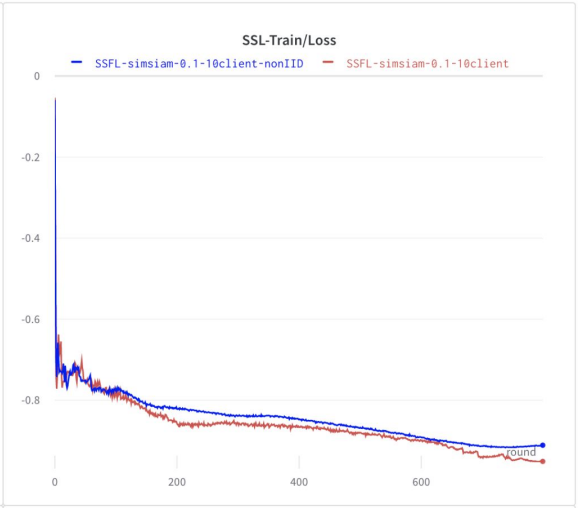
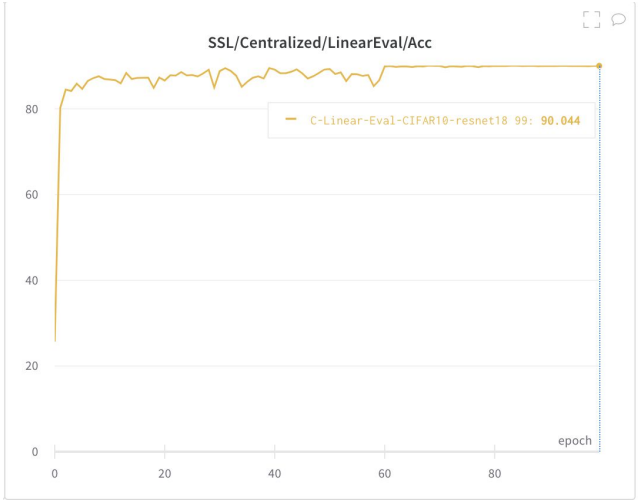


Fig. 1: Personalized Self-supervised Federated Learning

Ongoing Works 1: Self-supervised Federated Learning







Elementary Result: Contrastive Loss with SimSiam Framework in FL setting (Weighted Averaging) can archive top-1 **accuracy 91%** in CIFAR-10 (non-IID) using ResNet-18!

Important optimization tricks: 1) local SGD with momentum 2) cross-round learning rate scheduler

Ongoing Work 2 - Joint Adaptation to Data and System Heterogeneity with FedNAS

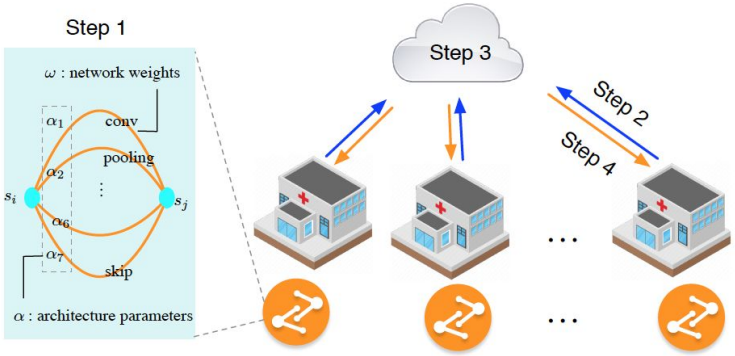


Figure 1: Federated Neural Architecture Search (*step 1*: search locally; *step 2*: sending the gradient of α and ω to the server side; *step 3*: merge gradient to get global α and ω; *step 4*: synchronize updated α and ω to each client.)

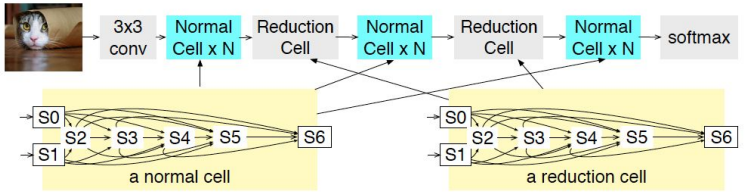


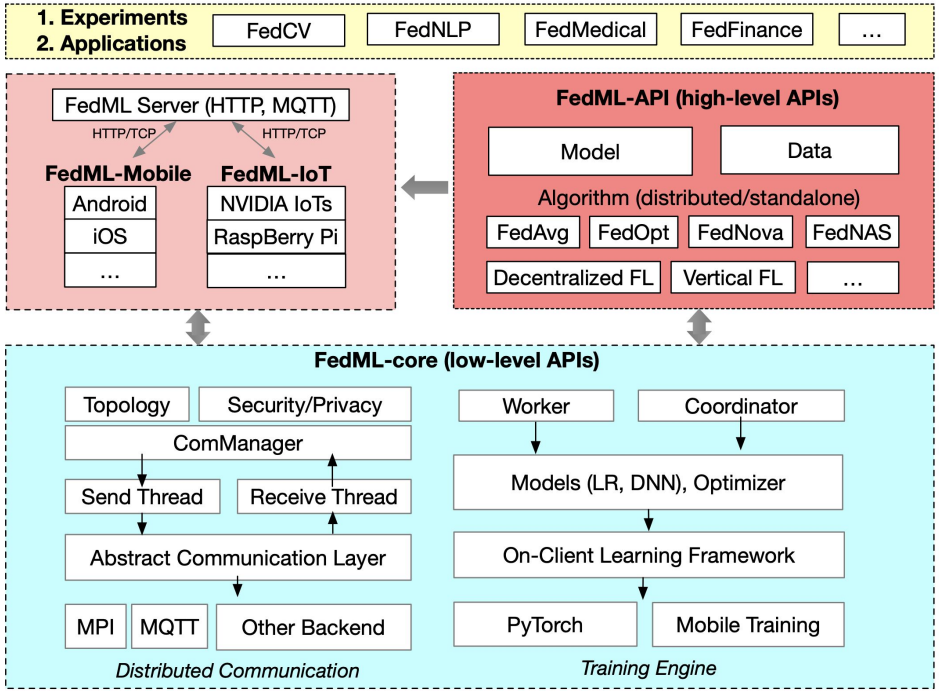
Figure 2: Search Space

$$\bar{o}^{(i,j)}(x) = \sum_{k=1}^{d} \underbrace{\frac{\exp(\alpha_k^{(i,j)})}{\sum_{k'=1}^{d} \exp(\alpha_{k'}^{(i,j)})}}_{p_k} o_k(x) \qquad (3)$$

FedML and DistML Ecosystem

USC University of Southern California

FEDML
MAKING COLLABORATION INTELLIGENT



PyTorch

Application/Model

FedCV , **FedNLP** (preparing for EMNLP 2021) **FedGraphNN** (ICLR 2021 workshop, MLSys 2021 workshop)

Algorithm/Theory

FedGKT (NeurIPS'2020): resource constrained FL

FedNAS, MiLeNAS (CVPR'2020): data heterogeneity and automation

OnlinePushSum (NeurIPS'2020 Workshop): Single-sided Trust in Decentralized Topology

System/Infrastructure

FedML.ai (NeurIPS 2020, FL Workshop, Best Paper Award): fundamental system, an open source library for FL research

PipeTransformer (ICML'2021): elastic distributed training for giant models (Transformers)

Vision/Review

Advances and Open Problems in Federated Learning
FnTML (Foundation and Trend in Machine Learning), Vol3, 2021