DGIST DataLab

Data-Intensive Computing Systems Laboratory

NDPipe: Exploiting Near-data Processing in Storage Clusters for Scalable Inference and Continuous Training

Jungwoo Kim, Seonggyun Oh, Jaeha Kung*, Yeseong Kim, Sungjin Lee

DGIST and Korea Univ.

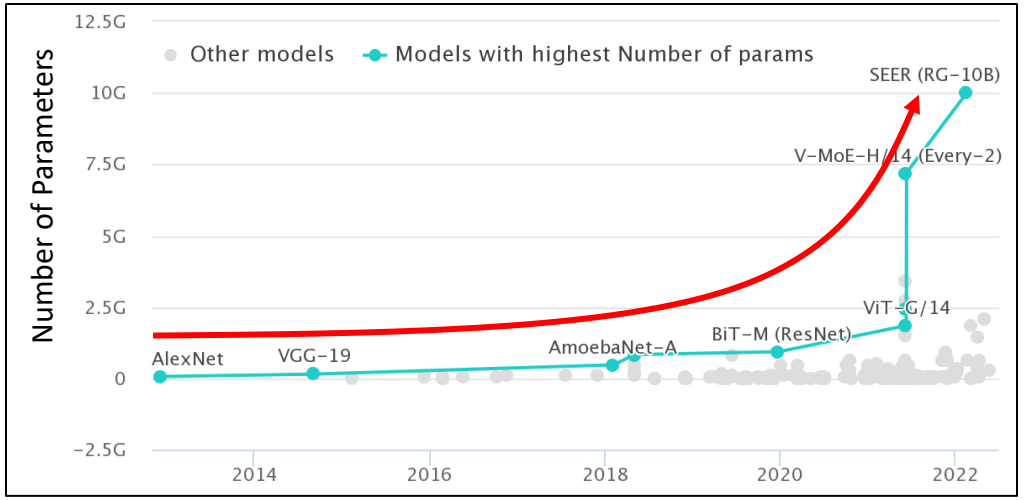
2024 Operating System Support for Next Generation Large Scale NVRAM

(Presented at ASPLOS'24)

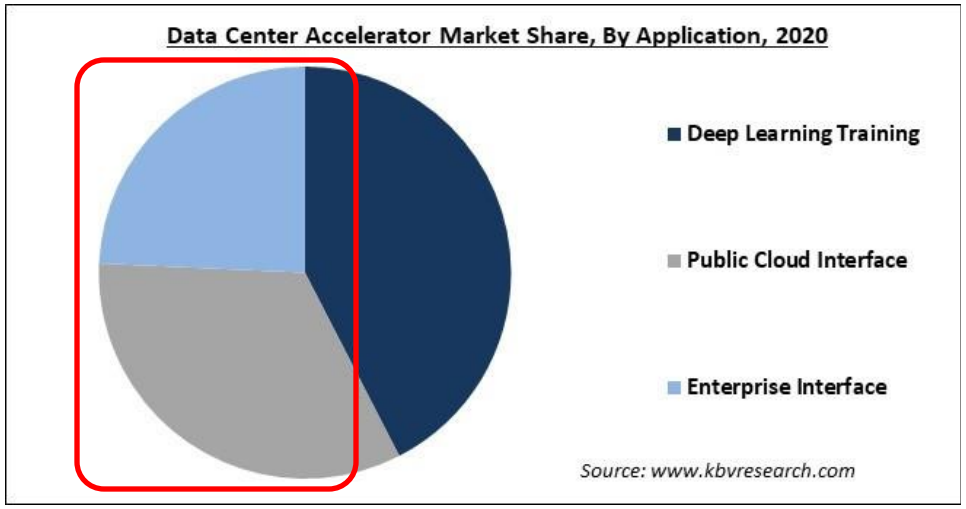
Long Training Time and High Inference Cost

- Deep learning (DL) algorithms have evolved to have larger models to improve the quality of deep neural network (DNN) models

- As more applications rely on DL algorithms, DNN inference is becoming a major operation in datacenters



DNN model size



Market portion of inference accelerators in datacenters

Long Training Time and High Inference Cost

- **Deep learning (DL) algorithms have evolved to have larger models to improve the quality of deep neural network (DNN) models**

  → ***Large model requires very long training time***

- **As more applications rely on DL algorithms, DNN inference is becoming a major operation in datacenters**

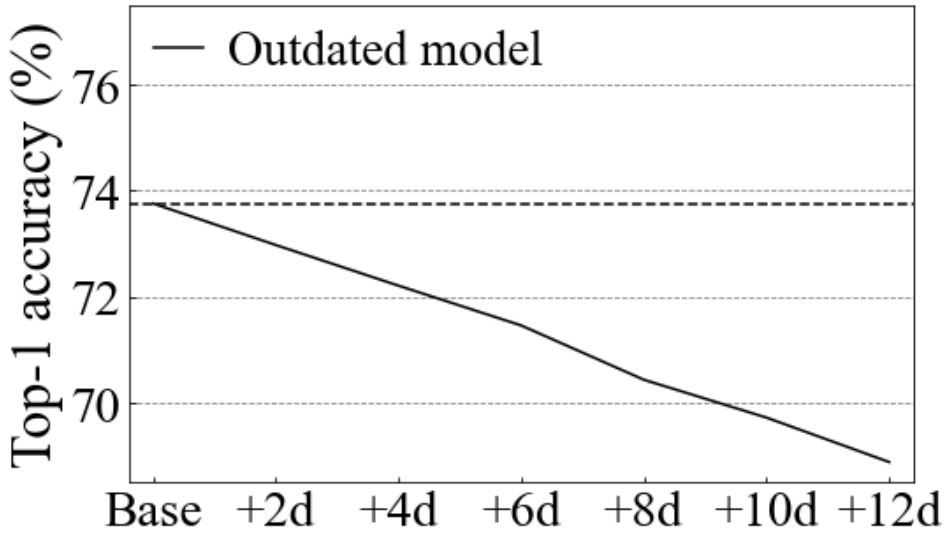
  → ***Inference involves lots of data movements***

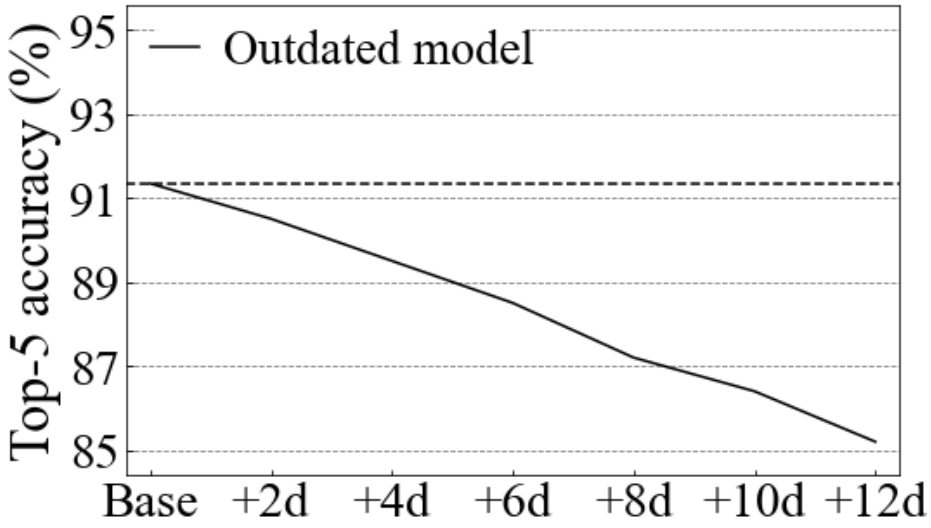
These pose two technical challenges when deploying DNN models:
(1) *outdated model problem* and (2) *data relabeling problem*

Technical Challenge #1:

Outdated Model Problem

- **Creating large DNN models is costly, in terms of computation and energy, and takes very long time (e.g., 2—6 weeks)**
- **Long training time makes it difficult to create up-to-date DNN models timely, which negatively affects model accuracy**
- **The model's accuracy declines over time on changing data due to *drift***

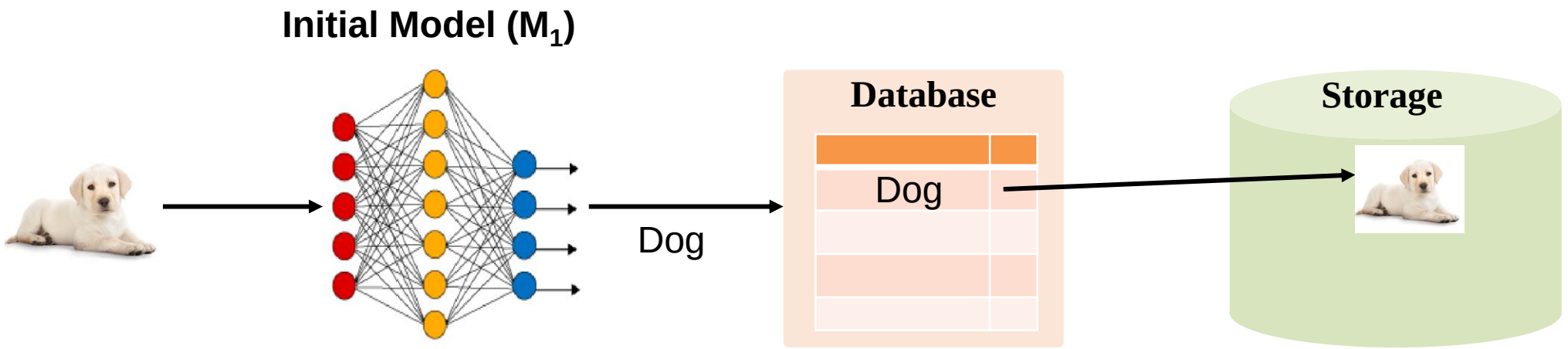




Technical Challenge #2:

Data Relabeling Problem

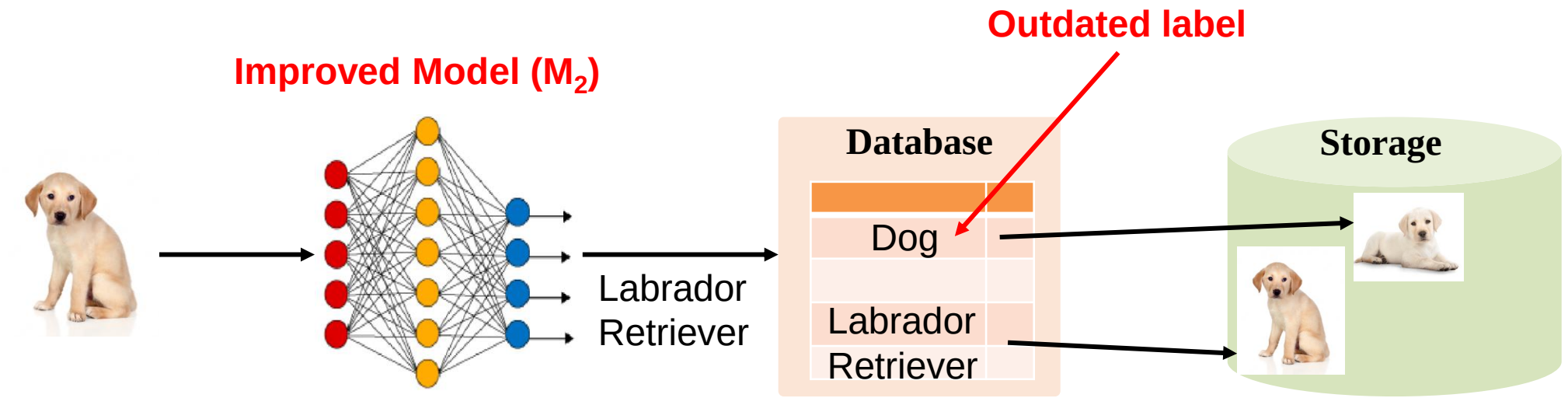
- **Many DL-based services extract labels from incoming data using DNN models**
  - e.g., extracting image classes from incoming user images
- **Extracted labels are kept in a database to serve user requests quickly**
- **Labels for previously stored data become obsolete whenever new DNN models are released**



Technical Challenge #2:

Data Relabeling Problem

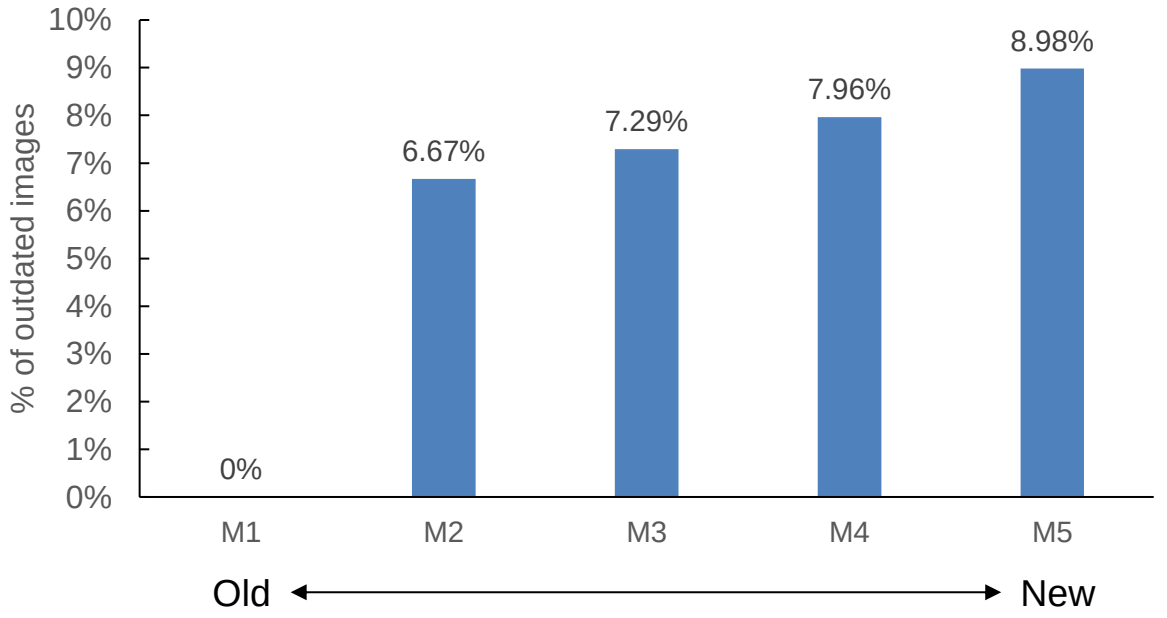
- **Many DL-based services extract labels from incoming data using DNN models**
  - e.g., extracting image classes from incoming user images
- **Extracted labels are kept in a database to serve user requests quickly**
- **Labels for previously stored data become obsolete whenever new DNN models are released**



Technical Challenge #2:

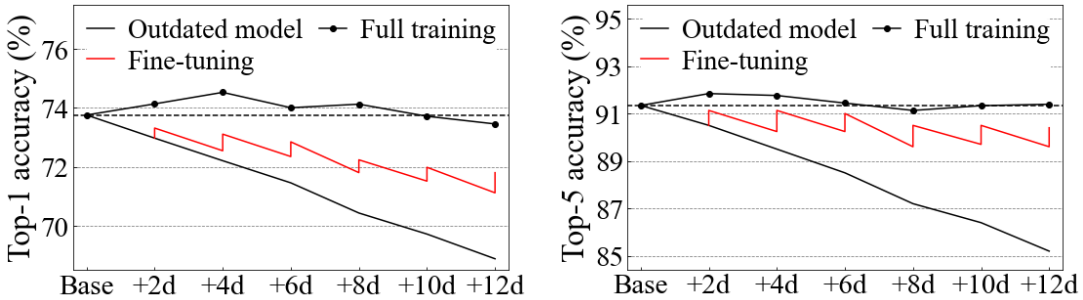
Data Relabeling Problem (Cont.)

- **Many DL-based services extract labels from incoming data using DNN models**
  - e.g., extracting image classes from incoming user images
- **Extracted labels are kept in a database to serve user requests quickly**
- **Labels for previously stored data become obsolete whenever new DNN models are released**



Any Mitigations?

- **Outdated Model Problem**
  - Online learning or incremental learning
  - Regular full training
  - Fine tuning



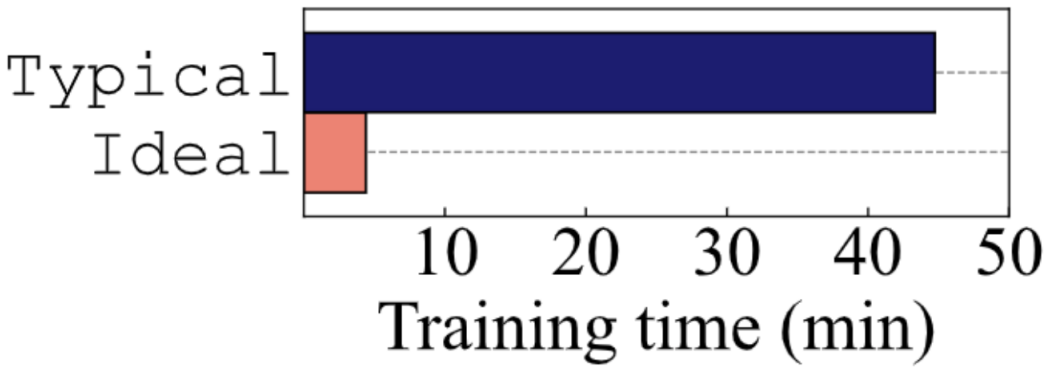
- **Data Relabeling Problem**
  - Fast image labeling
  - Refine labels by similarity
  - Offline inference

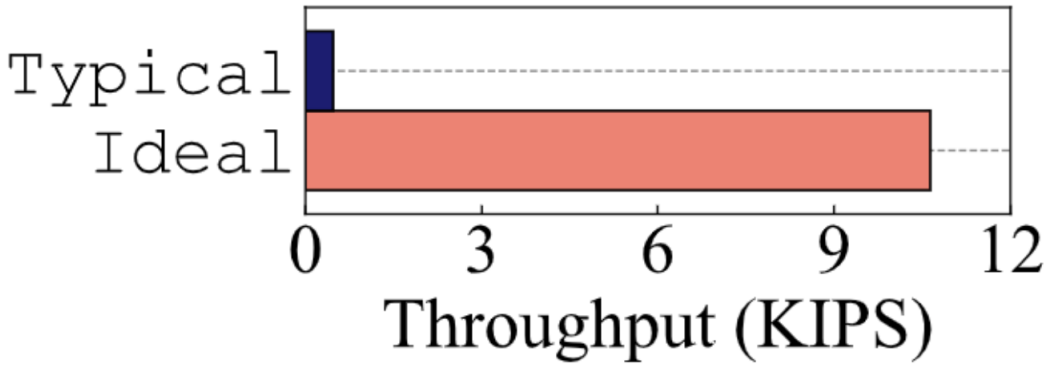
Fine-tuning and Offline Inference

- **Performance evaluation of fine-tuning and offline inference**
- **Compare two different setups: Typical and Ideal**
  - Typical: a typical system setup where compute and storage are disaggregated
    - Storage servers keep images
    - Compute servers perform fine-tuning and offline inference by loading images from storage servers over network
  - Ideal: an ideal setup where compute servers contain all images locally
    - No data movement for fine-tuning and offline inference

Data Movement is a Bottleneck

- Fine-tuning and offline inference require relatively low computing power that low-end GPUs can run
- A large amount of data transferred between storage and compute servers is a major bottleneck





- ***How to eliminate data movement? Use near-data processing!***

Index

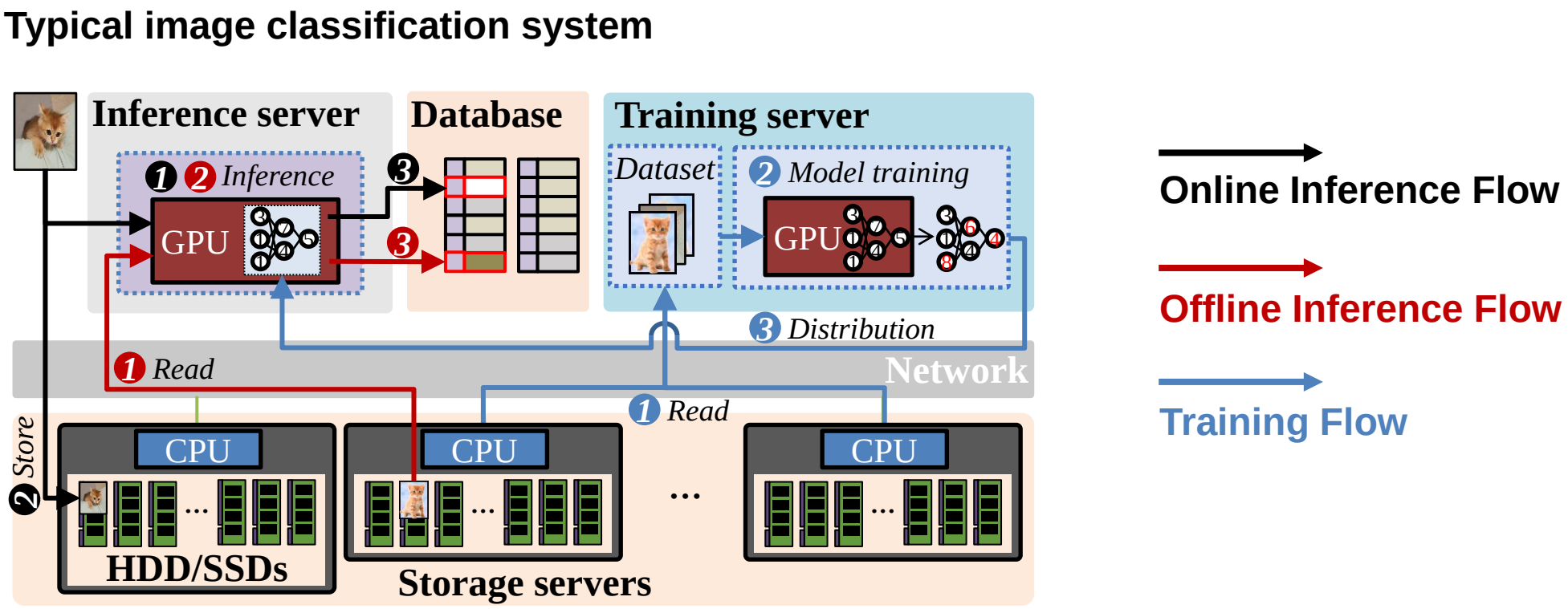
- Introduction
- Overall Design of NDPipe
- Optimization of NDPipe
- Experimental Results
- Conclusion

NDPipe's Approach

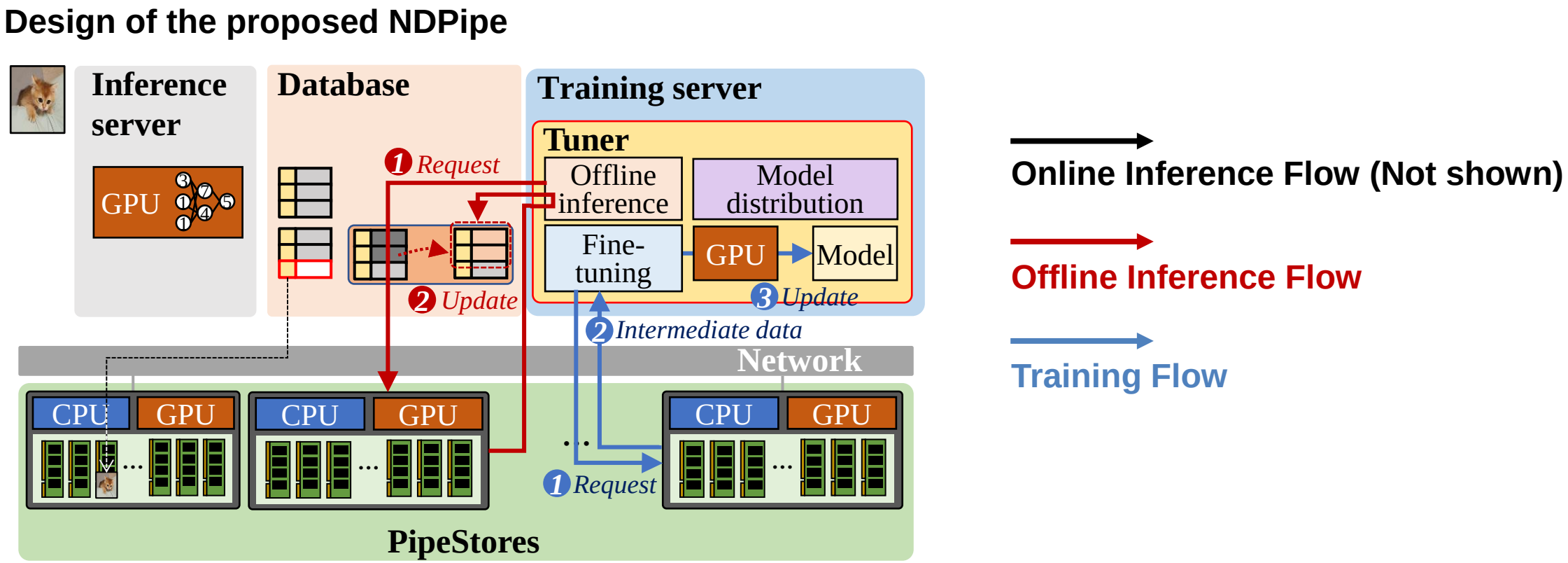
- **Design goal of NDPipe**
  - Improve fine-tuning and offline inference by leveraging near-data processing at the level of storage clusters where data resides nearby

- **Our approach**
  1. Deploy cost- and power-effective commodity GPUs in storage servers
  2. Execute fine-tuning and offline inference on storage servers and return only labels or intermediate results to compute servers → Eliminate almost all data transfers required in fine-tuning and inference
  3. Accelerate the fine-tuning and inference processes by fully utilizing high aggregate throughputs of low-end GPUs in multiple storage servers

Overall Architecture of NDPipe



Overall Architecture of NDPipe



Technical Challenges

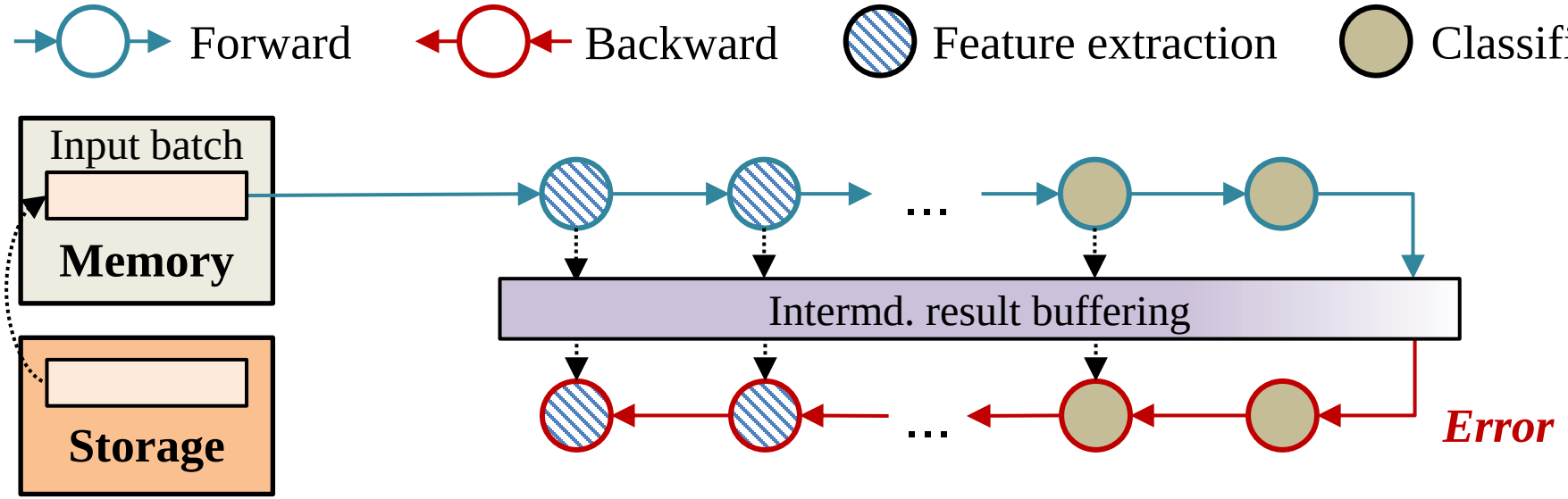
- **Challenge #1: How to efficiently execute (fine-tuning-based) training tasks over many storage servers without serious synchronization cost**
  - *Fine-tuning-based data and model parallelism (FT-DMP)*
- **Challenge #2: How to make individual PipeStores with a commodity GPU fast and efficient for fine-tuning and inference**
  - *Near-data processing engine (NPE)* optimized with various optimizations (e.g., pipelining, task offloading, quantization, and so on)
- **Challenge #3: How to redistribute the latest DNN models to storage servers**
  - *Version-aware model redistribution (VAMR)* to minimize model distribution cost

Index

- Introduction
- Overall Design of NDPipe
- Optimization of NDPipe
- Experimental Results
- Conclusion

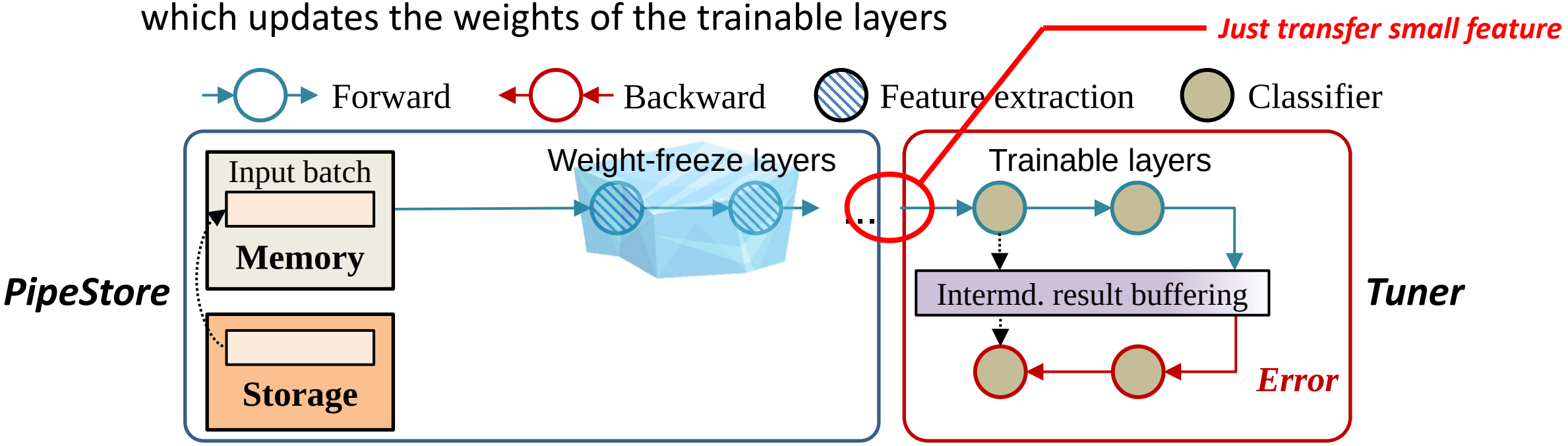
Fine-tuning-based data and model parallelism (FT-DMP)

- **FT-DMP exploits the unique property of fine-tuning where the model's layers are split into weight-freeze and trainable layers**
  - Assign a replica of weight-freeze layers to PipeStores, executing multiple workers
  - Assign trainable layers to Tuner and run a single worker on Tuner
  - The individual PipeStores extract features for local batches and deliver them to Tuner which updates the weights of the trainable layers

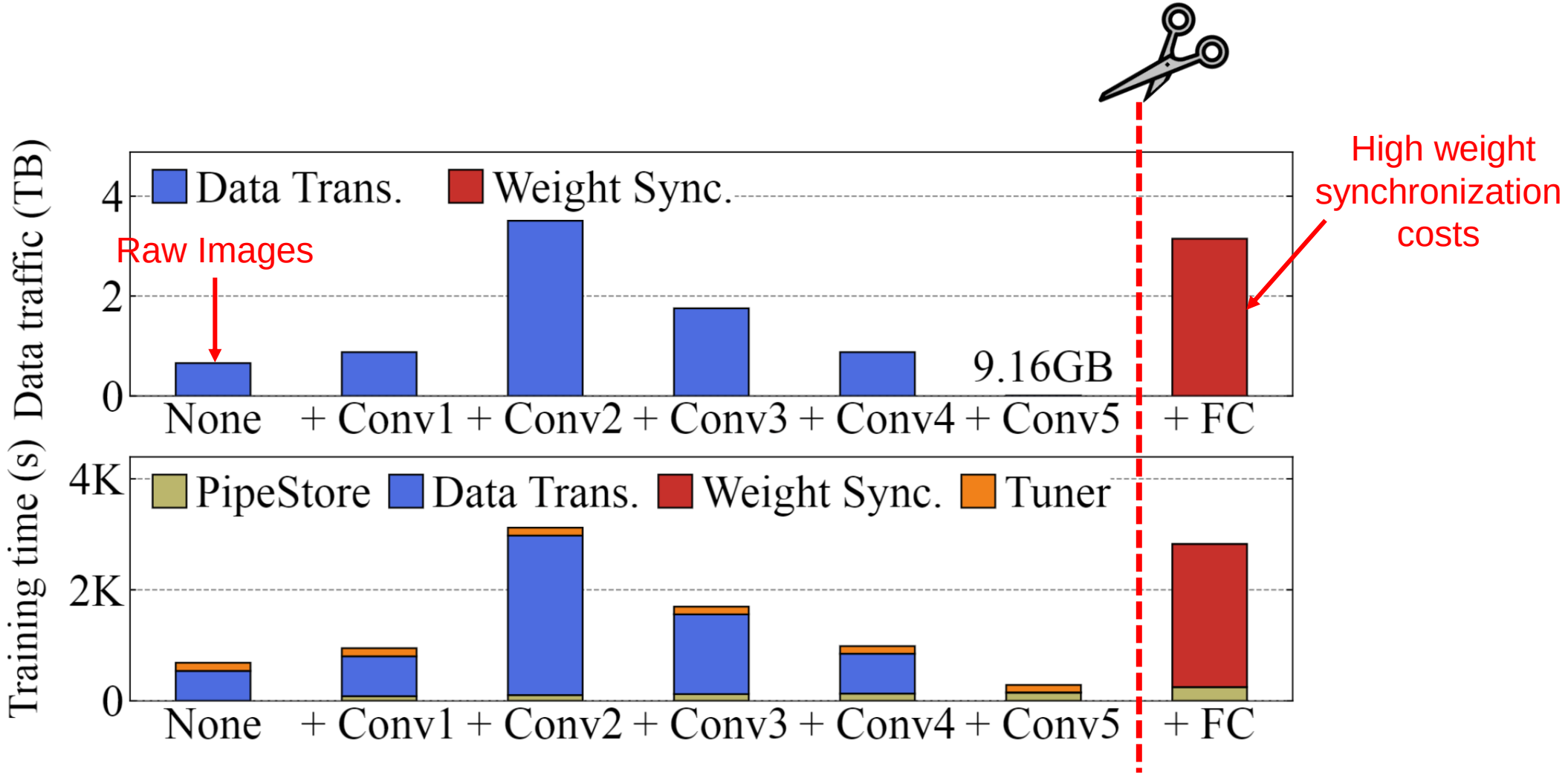


Fine-tuning-based data and model parallelism (FT-DMP)

- **FT-DMP exploits the unique property of fine-tuning where the model's layers are split into weight-freeze and trainable layers**
  - Assign a replica of weight-freeze layers to PipeStores, executing multiple workers
  - Assign trainable layers to Tuner and run a single worker on Tuner
  - The individual PipeStores extract features for local batches and deliver them to Tuner which updates the weights of the trainable layers

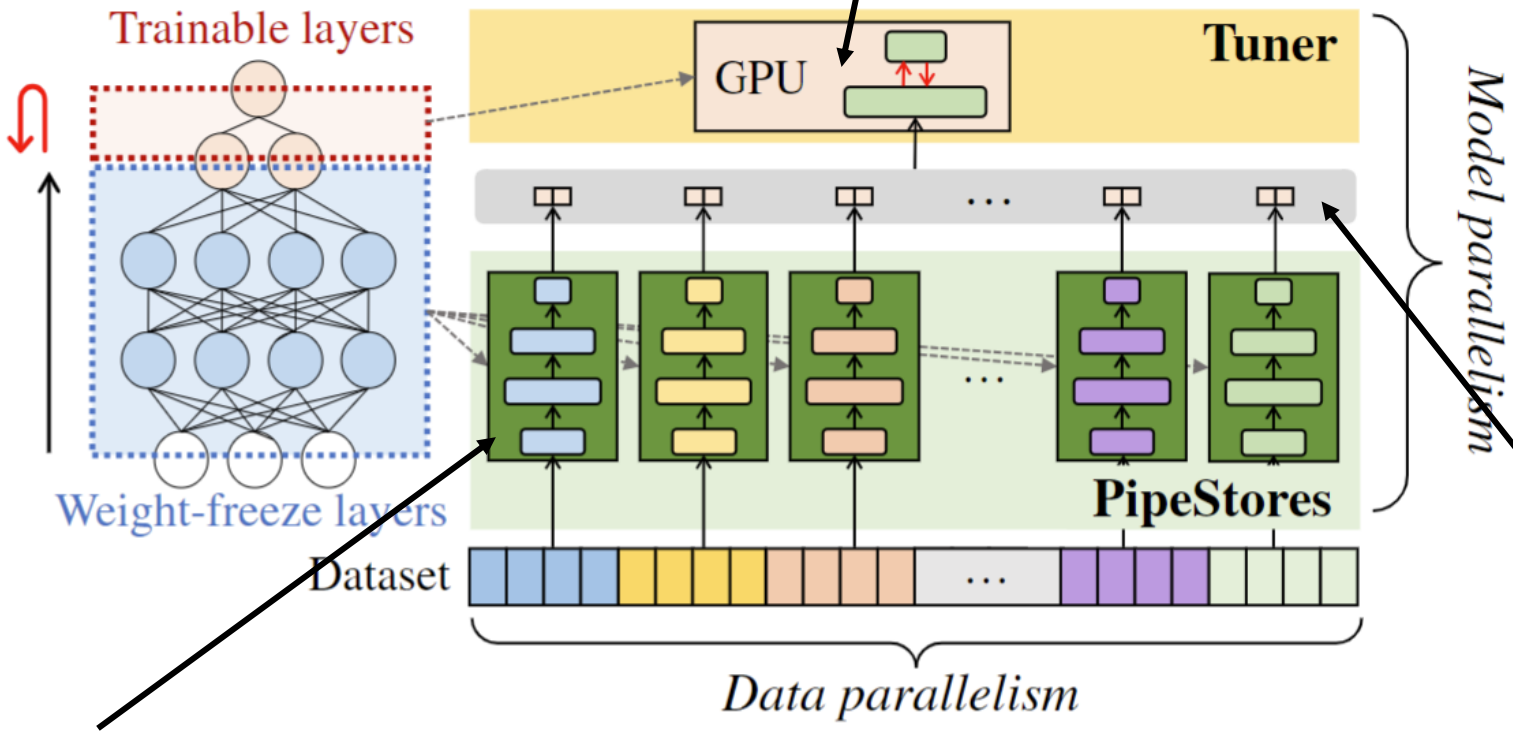


Impact of layer offloading and data traffic



Fine-tuning-based data and model parallelism (FT-DMP)

Weight synchronization run locally on Tuner → up-to-date models can be produced with little synchronization costs

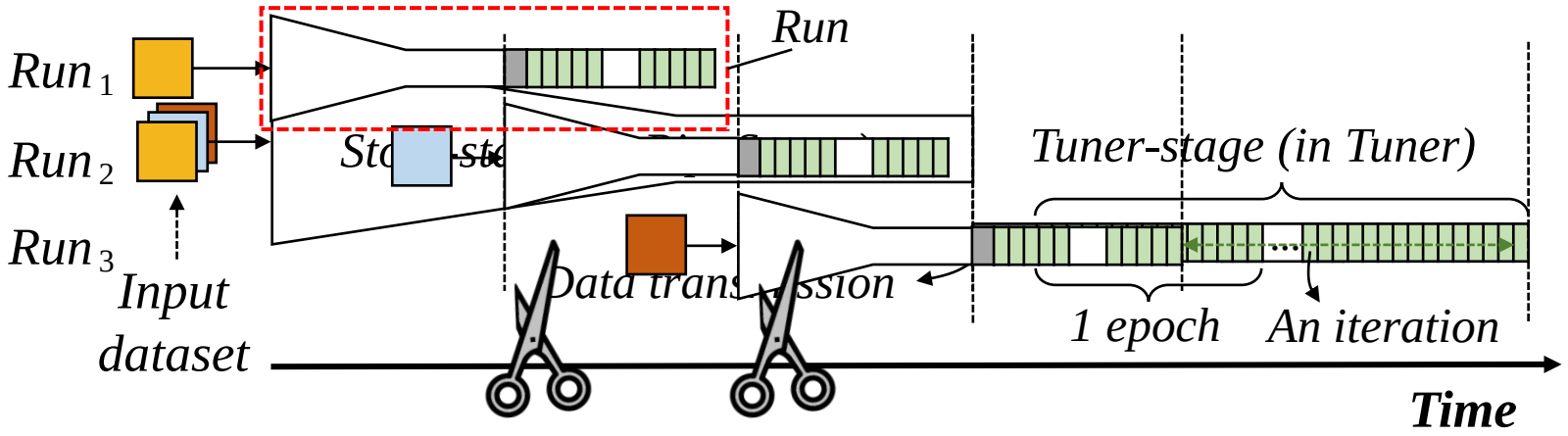


The weight-freeze layers only require processing the forward pass of the network → low-end GPUs provide sufficient compute capability

Small-sized outputs from the last weight-freeze layer are delivered to Tuner → minimum network traffic

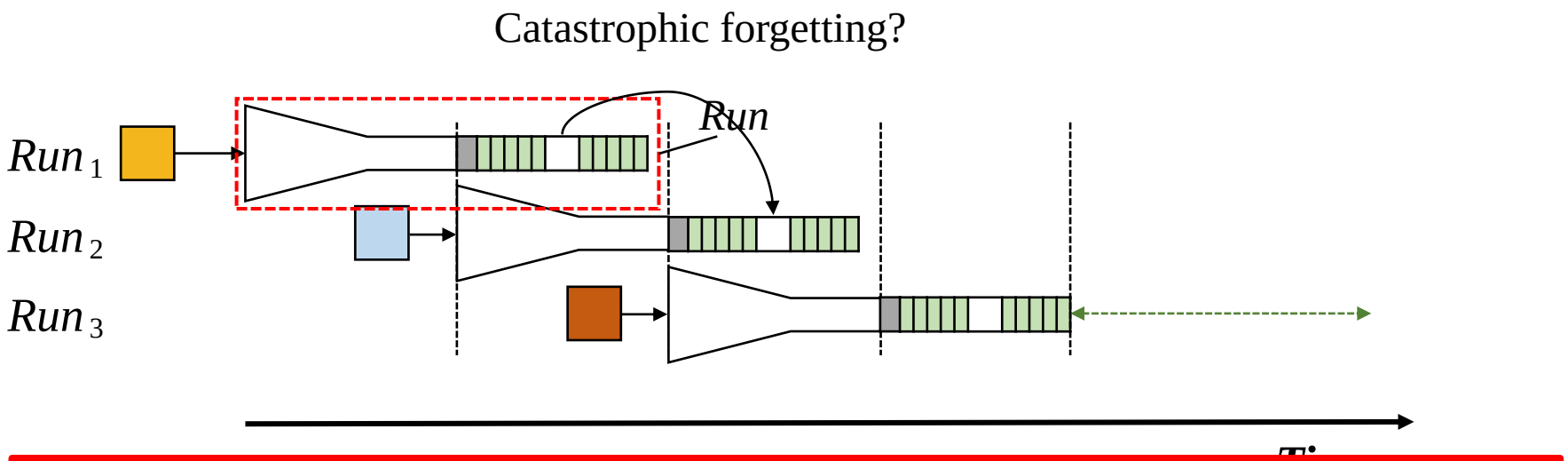
Pipelined FT-DMP

- In NDPipe, all the batches are distributed across multiple PipeStores
  - Following the DNN training procedure that requires the entire training data, Tuner may wait for the intermediate results corresponding to all data samples from the participated PipeStores
  - This results in the serial execution of PipeStores (Store-stage) and Tuner (Tuner-stage)
  - Inspired by pipelined model parallelism that executes split partitions concurrently, we propose a pipelined training strategy



Pipelined FT-DMP (Cont.)

- The pipelined training executes multiple runs simultaneously over separate sub-datasets
  - The pipelined training starts the training in Tuner for the current run, while PipeStores are processing local batches for the next run
  - In our case, the model would be more fitted to the training samples used in the last run



Our theoretical analysis confirms that the pipelined training can still guarantee the convergence of training and not seriously affect the final model quality with a reasonable number of pipeline runs, N_{run}

Pipelined FT-DMP (Cont.)

starts with the same initial loss of ϵ. We denote by $l^1(T_1)$ the loss converged at iteration T_1 during the first run, and $l^1(T_1) < \epsilon_1 (= \epsilon)$. With the conditions above, we claim that the second run starting with initial loss $l^1(T)$ also converges:

Theorem 5.1. *Assume that the gradient descent is initialized such that the weights have a deficiency margin [13], $c > 0$ and are δ-balanced. Also, suppose that the second run starts with the weights converging in the first run by $l^1(T_1)$ and the inter-run loss difference, Δ. Then, with the minimum learning rate requirement for η in Eq. 7 of [13], $\forall \epsilon_2 > 0$ and*

$$T_2 \geq \frac{1}{\eta \cdot c^{2(N-1)/N}} \cdot \log\left(\frac{l^1(T_1) + \Delta}{\epsilon_2}\right), \tag{1}$$

the loss at iteration T_2 is no greater than ϵ_2. In other words, the second run is guaranteed to converge with the loss of ϵ_2.

See our paper for more details!

Theorem 5.1 applies to both the first and second runs, simplifying the explanation, but it also ensures convergence for all other runs through its inductive step. That is, as long as a p-th run converges at T_p with the final loss $l^p(T_p)$, any $(p+1)$-th run starting with the weights trained in the previous run will also converge. Generalizing Theorem 5.1 with $l^1(T_1) = l^p(T_p)$ and $l^2(T_2) = l^{p+1}(T_{p+1})$ accomplishes the induction, and this completes the proof of the convergence of our pipelined training. To establish the claim, we describe the inter-run loss difference Δ in the following lemma.

Lemma 5.2. *For a given confidence level θ,*

$$l^2(0) \leq l^1(T_1) + \Delta$$

where $\Delta = \sqrt{\frac{1}{2m} log(\frac{2P}{\theta})}$, P is the number of total weights in the model, and m is the number of training dataset samples.

Proof of Lemma 5.2. By Hoeffding's inequality [45], the final loss of the first run and the initial loss of the second run satisfy the following relationship with a probability bound ε:

$$\mathbf{P}(|l^2(0) - l^1(T_1)| \geq \varepsilon) \leq 2\exp(-2m\varepsilon^2). \tag{2}$$

Near-data processing engine (NPE)

- **PipeStore handles both fine-tuning and offline inference, which require six common steps**
  - Image loading
  - Decoding
  - Resizing
  - Normalization
  - Feature extraction (FE)
  - Classification (CL)

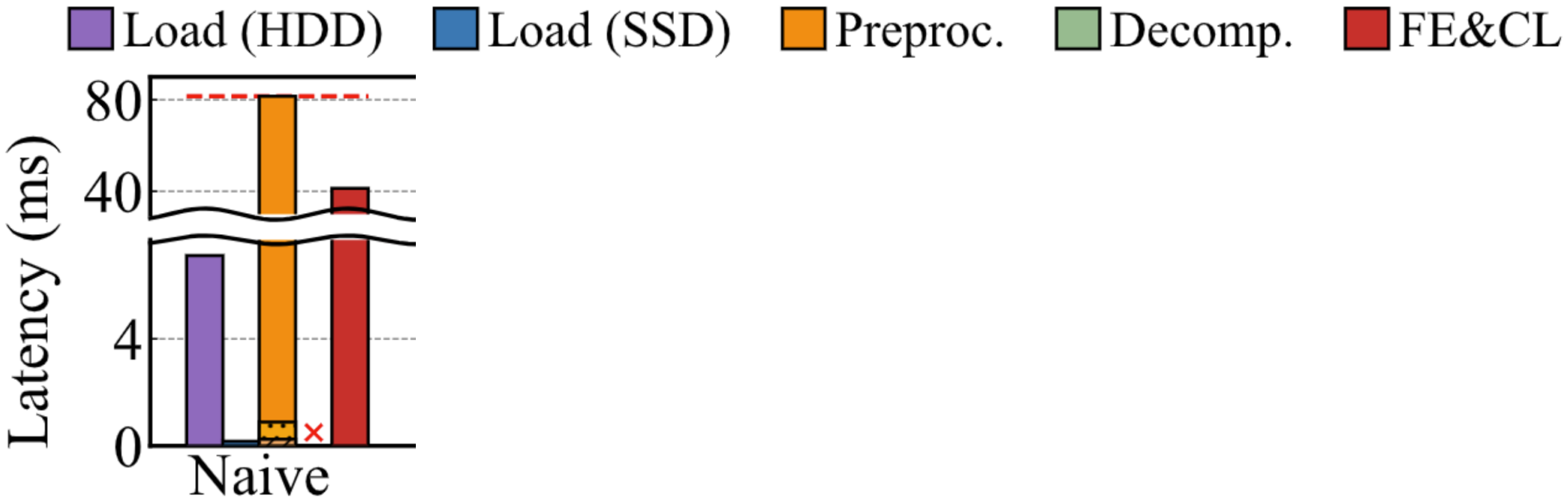
I/O operations by HDDs or SSDs

Preprocessing by CPU

FE & CL by GPU

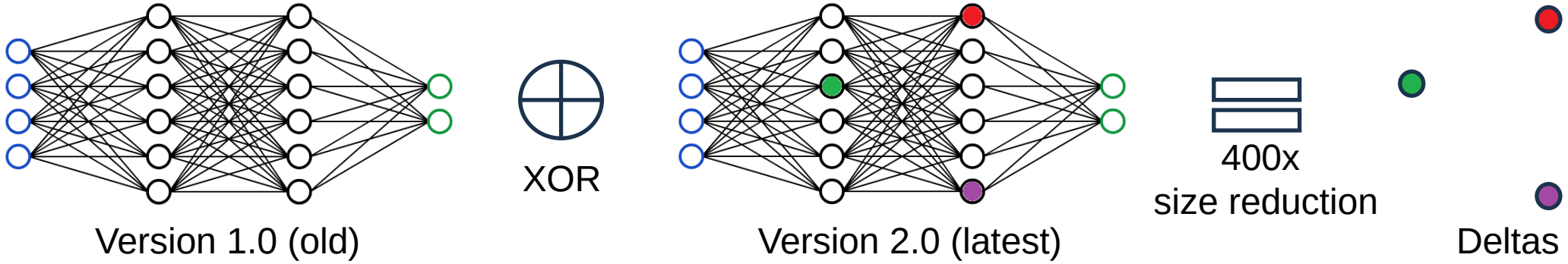
Near-data processing engine (NPE)

- **Optimize each step by applying (1) pipelining and (2) offloading and (3) by optimizing inference engine with quantization, layer fusion, and enlarged batch size**



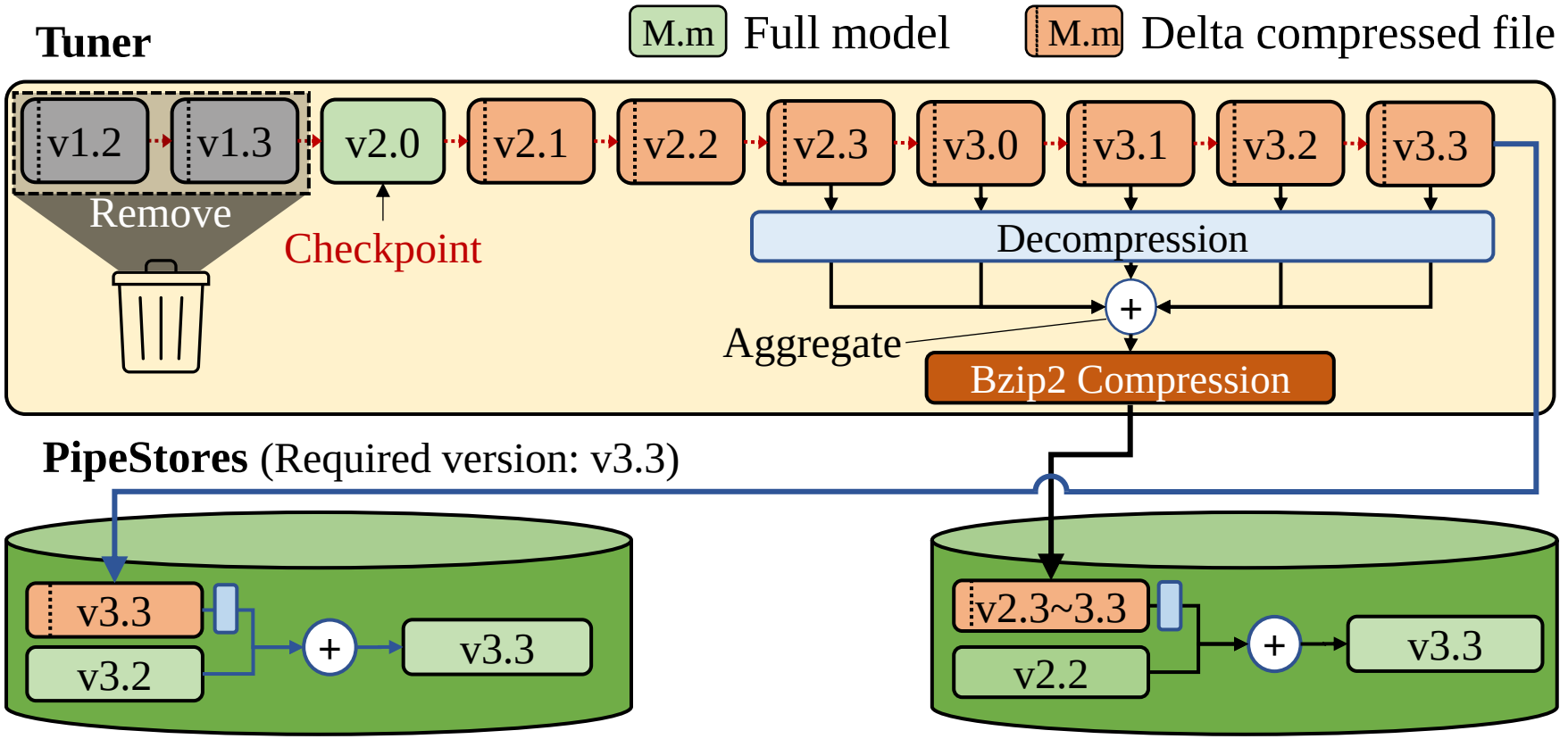
Version-aware Model Redistribution

- **Each PipeStore must keep up-to-date models for fine-tuning and inference**
  - Delivering new models to many PipeStores is costly
- **VAMR reduces the network traffic to deliver up-to-date models**
  - Send only small deltas by exploiting similarity between adjacent models



  - Send the latest model to PipeStores where inference and fine-tuning take place
    - Manage different versions of models each PipeStore has

Version-aware Model Redistribution (Cont.)



Index

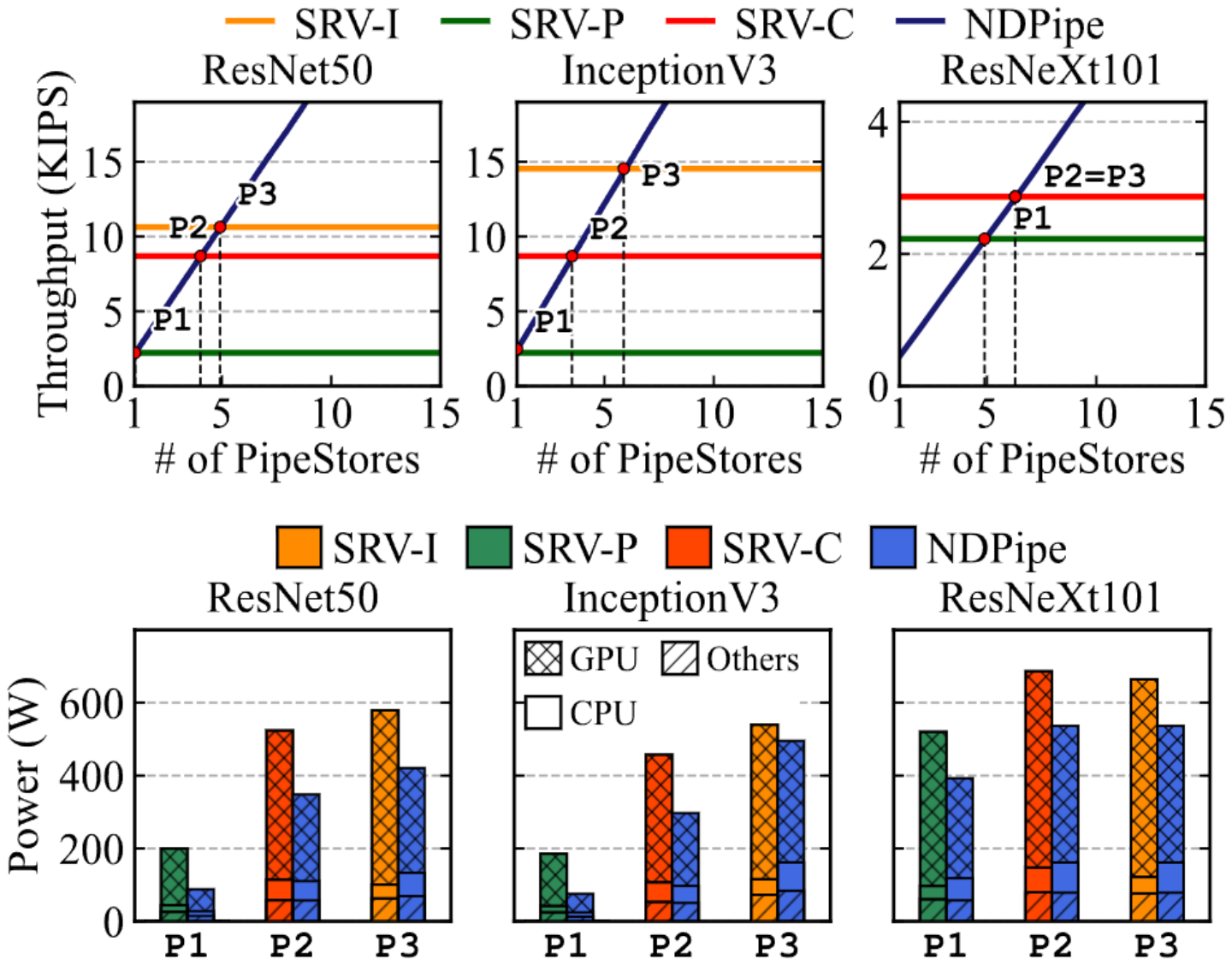
- Introduction
- Overall Design of NDPipe
- Optimization of NDPipe
- Experimental Results
- Conclusion

Evaluation Setup

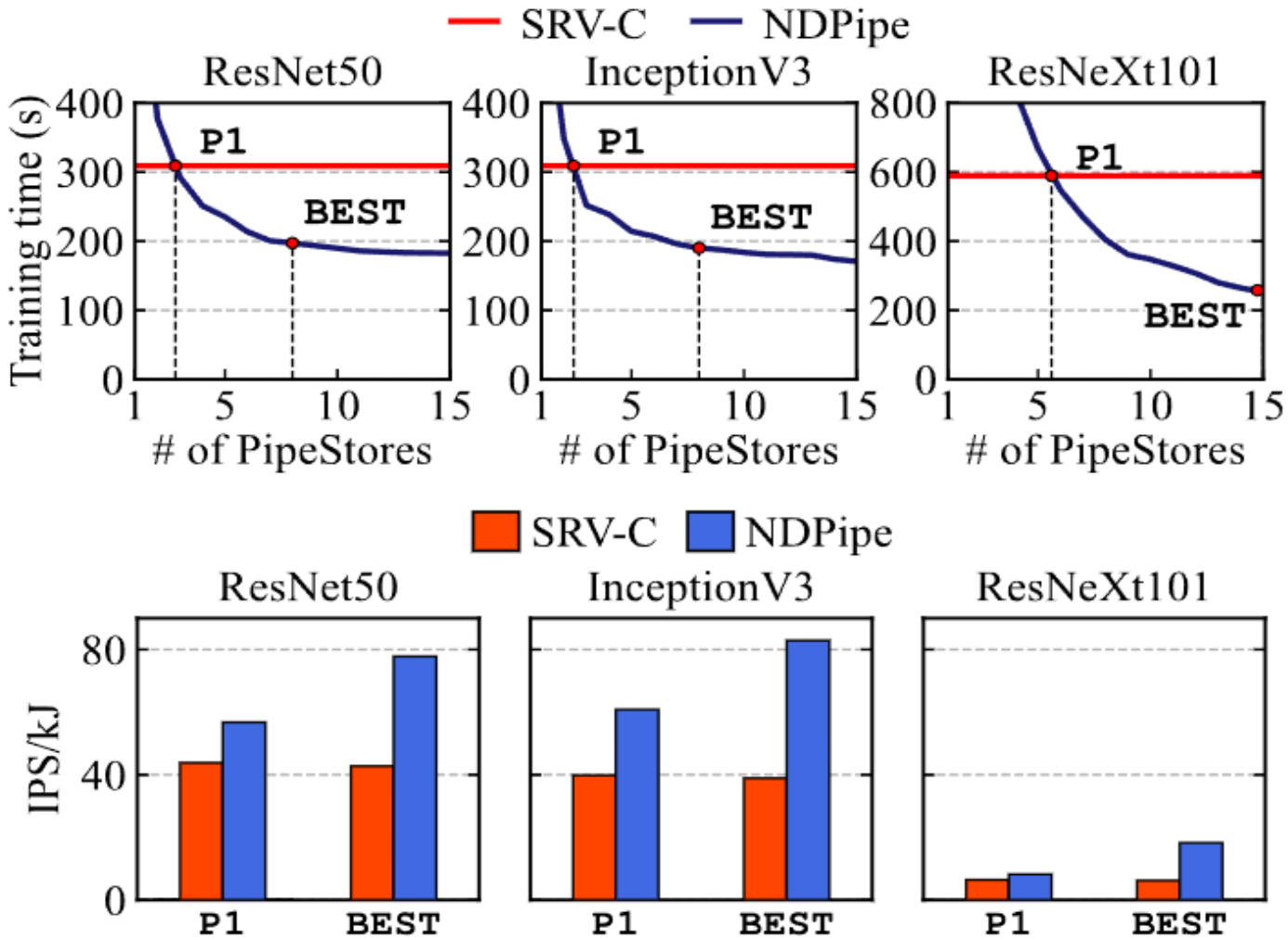
- We implement the prototype of NDPipe in Amazon Web Services (AWS)
- We use ImageNet 2012 ILSVRC as an input dataset
- Our experiments are conducted using four image classification models (InceptionV3, ResNet50, ResNeXt101-32x4d, and ViT)

	Baseline (SRV)		NDPipe	
	Fine-tuning	Offline inference	Fine-tuning	Offline inference
Host	**p3.8xlarge** • 2 V100 GPUs • 32 vCPUs (2.3GHz) • 244GB Memory • $12.24 per hour		**p3.2xlarge** • 1 V100 GPU • 8 vCPUs (2.3GHz) • 61GB Memory • $3.06 per hour	
Storage server	**g4dn.4xlarge (GPU disabled)** • 16 vCPUs (2.5GHz) • 64GB Memory • $0.688 per hour		**g4dn.4xlarge** • 1 T4 GPU • 16 vCPUs (2.5GHz) • 64GB Memory • $1.204 per hour	

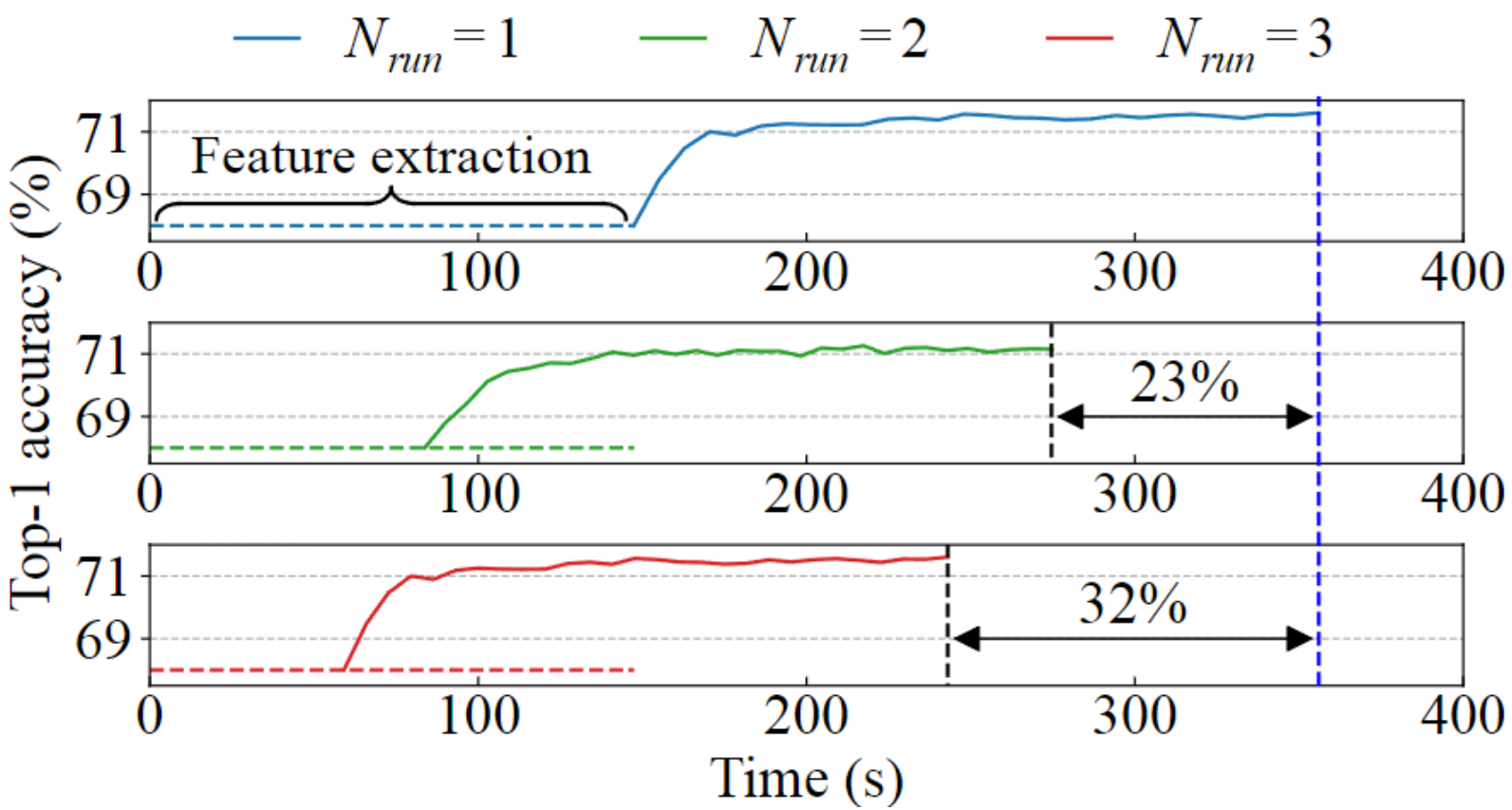
Inference Throughput & Power Efficiency



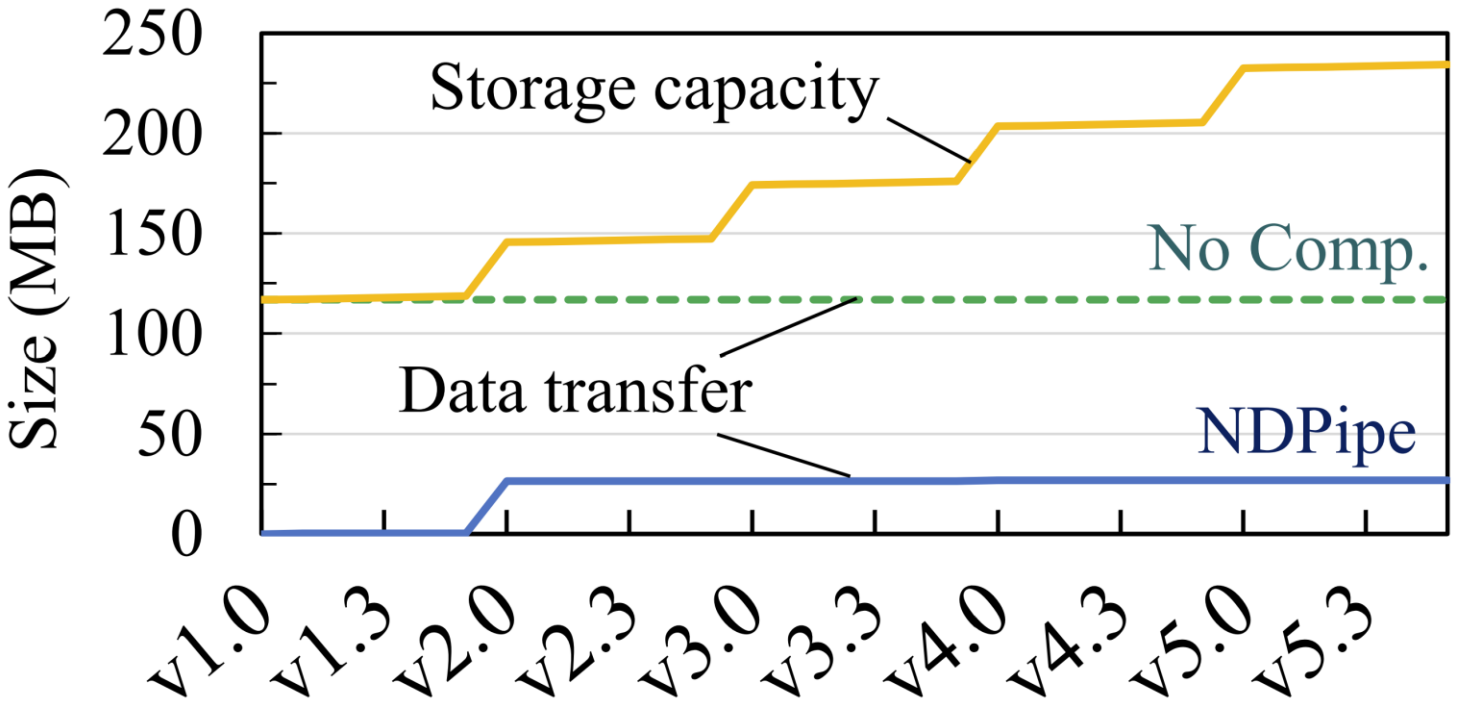
Training Time and Energy Efficiency



Pipelined Training



Model Redistribution Cost



We assume the worst-case scenario where all the PipeStores have the model v1.0

Conclusion

- We proposes a novel deep learning (DL) system called NDPipe, which accelerates training and inference performance by leveraging near-data processing in storage servers

- NDPipe distributes storage servers with inexpensive commodity GPUs in a data center and uses their collective intelligence to perform inference and training near data

- Our results show that, NDPipe exhibits 1.8x faster training speed with 1.7x lower AWS cost

Thank You! Any Question?