SHADE: Enable Fundamental Cacheability for Distributed Deep Learning Training

Redwan Ibne Seraj Khan[1],
Ahmad Hossein Yazdani[1], Yuqi Fu[2], Arnab K. Paul[3], Bo Ji[1],
Xun Jian[1], Yue Cheng[2], Ali R. Butt[1]

Virginia Tech[1], University of Virginia[2], BITS Pilani Goa[3]

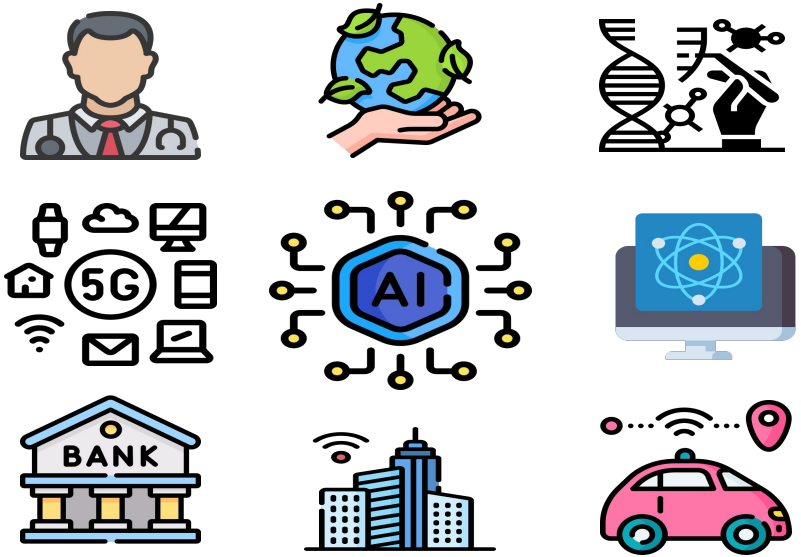








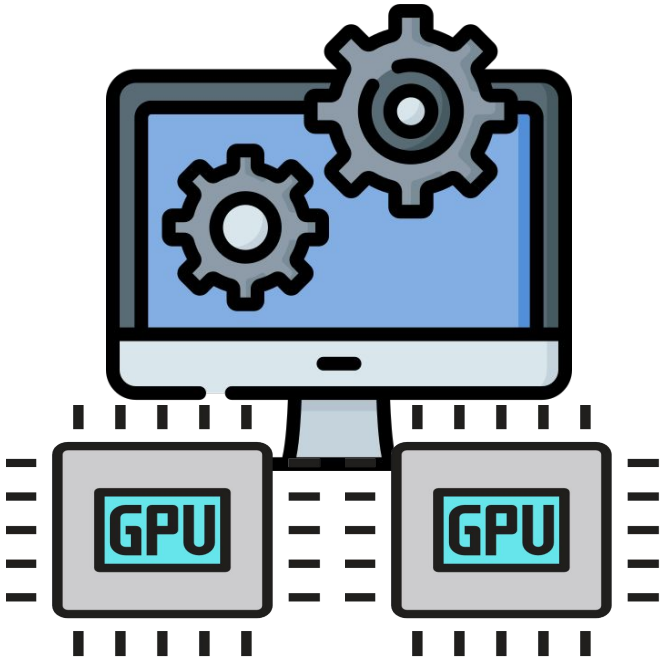
Deep Learning (DL) is ubiquitous

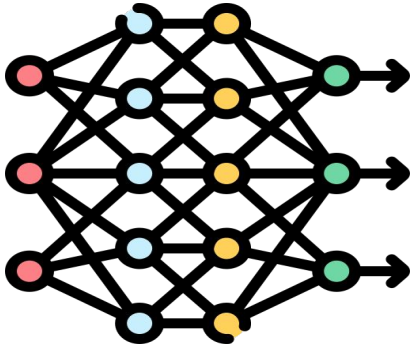


DL is getting used in everywhere to make life easier.
Market expected to reach 12 billion dollars by 2025!
Exponential growth ~19% annually

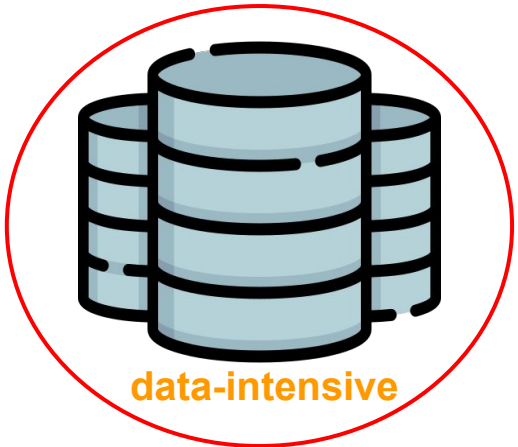
Global Conversational AI Market Report 2021. https://tinyurl.com/Global-AI-Market-Report-2021

Challenge: Matching DL Needs with System Resources

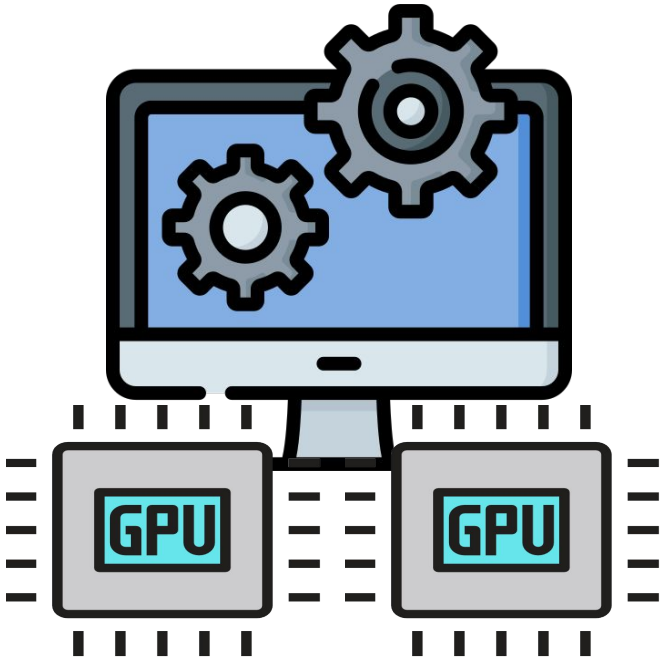




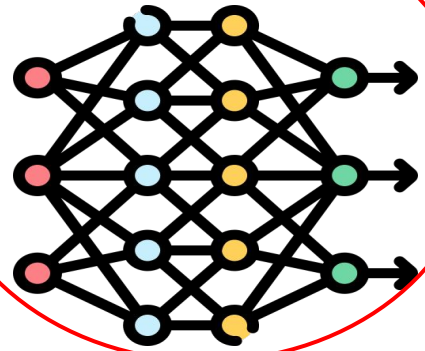
Deep Learning (DL)



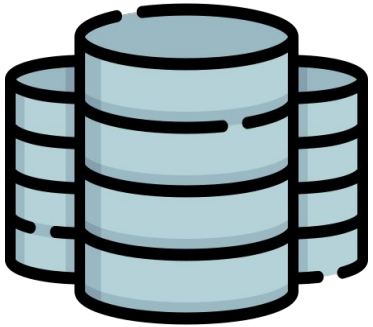
Challenge: Matching DL Needs with System Resources



compute-intensive

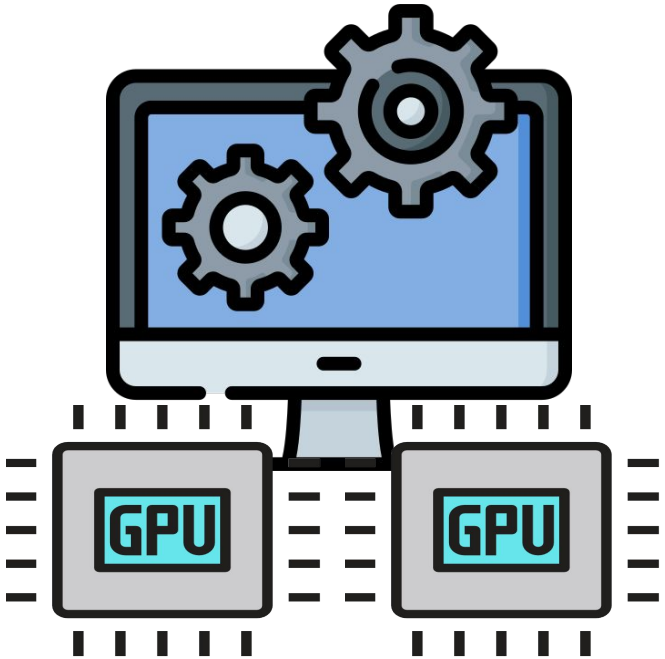


Deep Learning (DL)



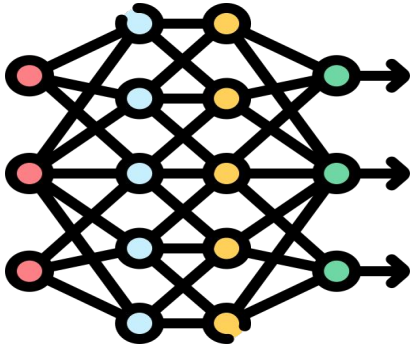
data-intensive

Challenge: Matching DL Needs with System Resources

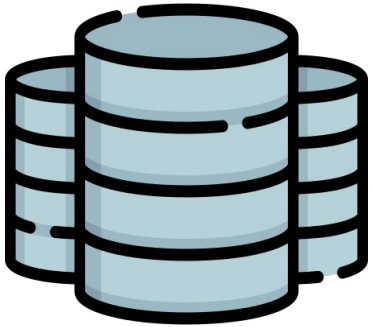




compute-intensive



Deep Learning (DL)

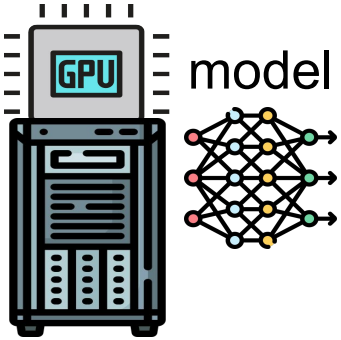


data-intensive

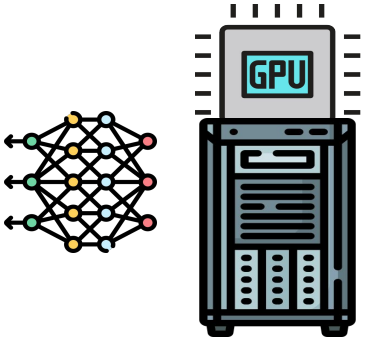
Data Storage Impacts the Performance of DL Training



Remote storage

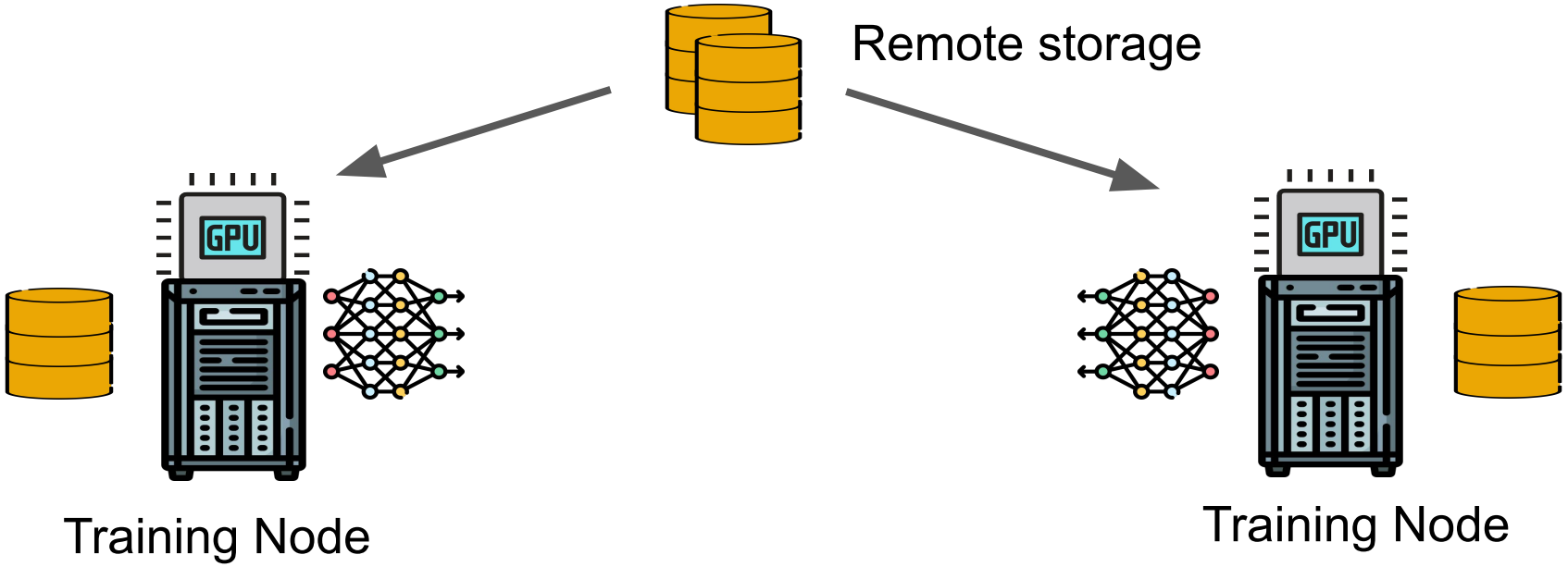


Training Node

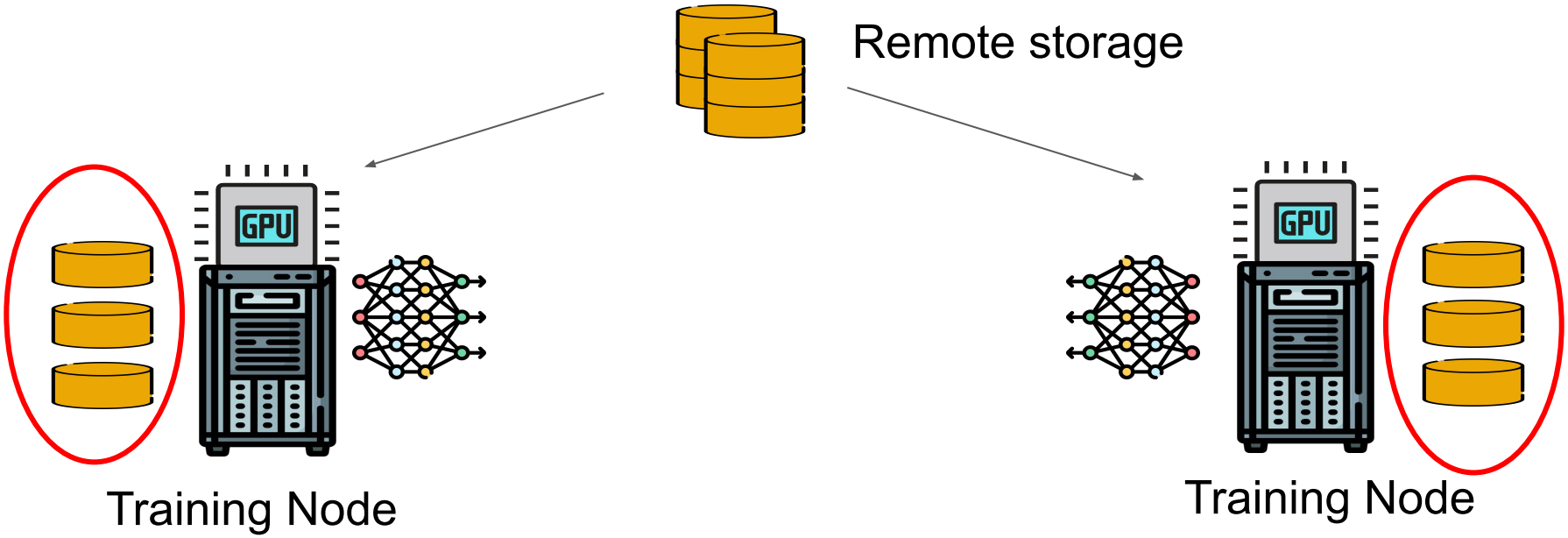


Training Node

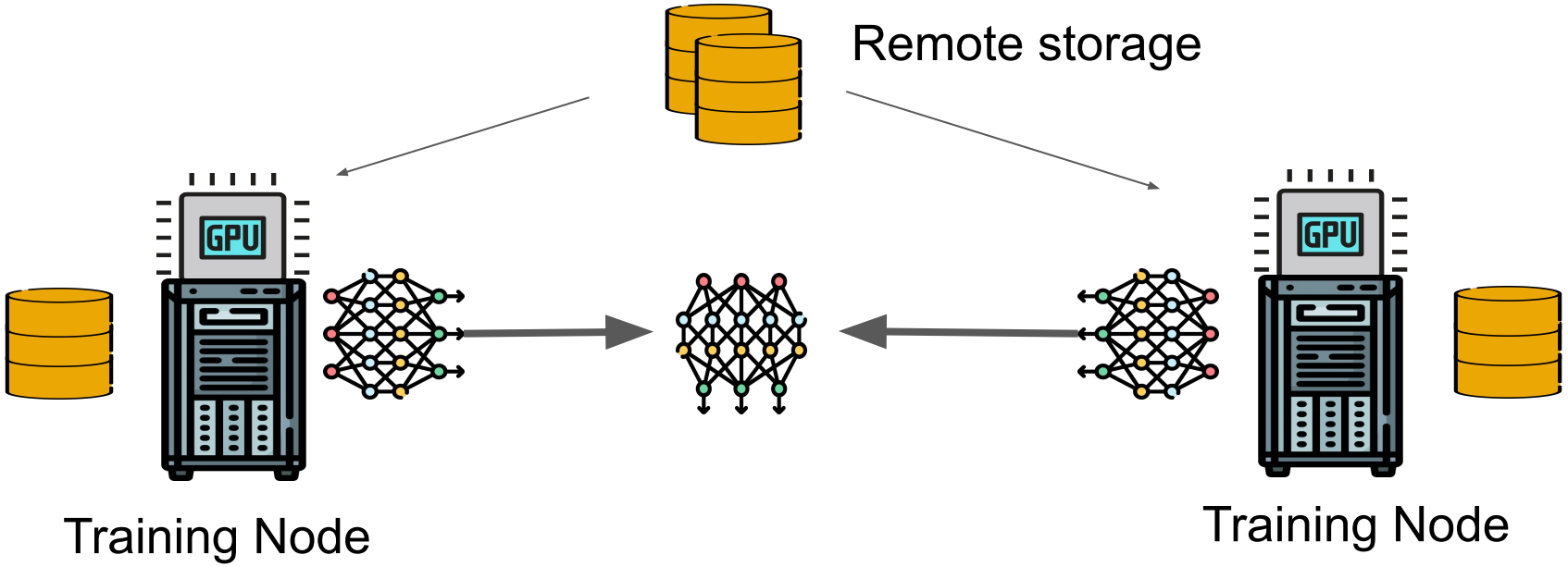
Data Storage Impacts the Performance of DL Training



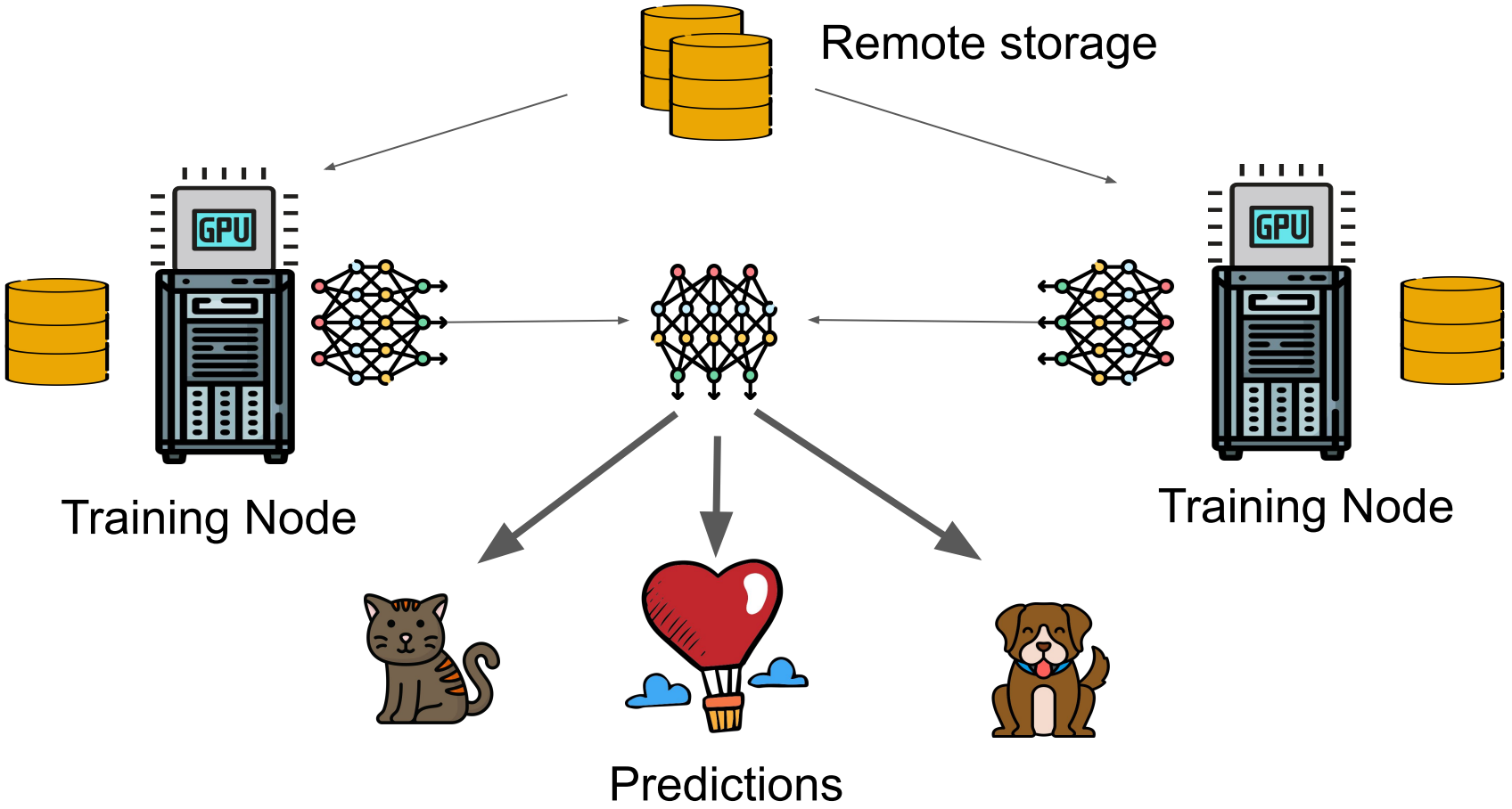
Data Storage Impacts the Performance of DL Training



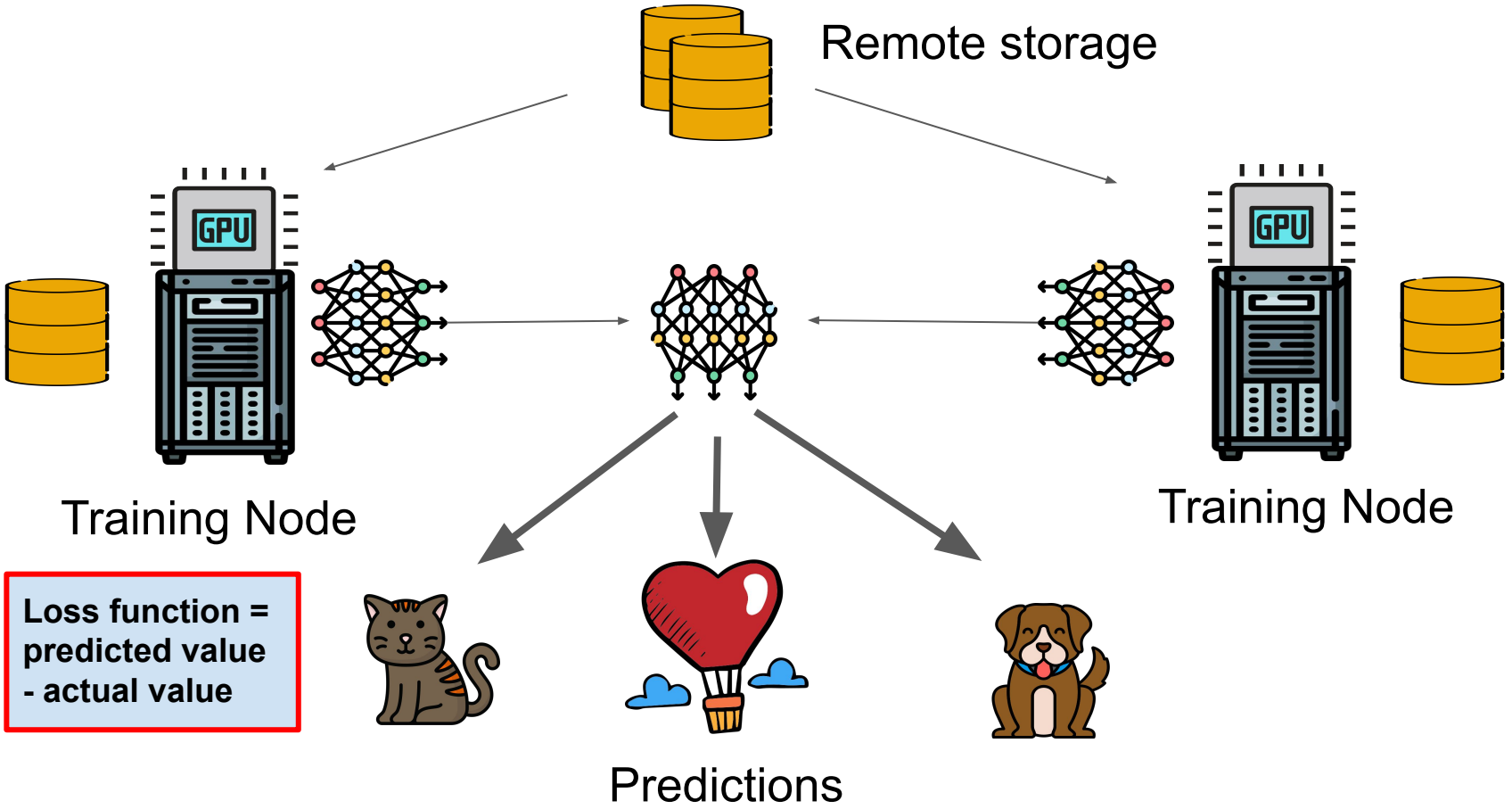
Data Storage Impacts the Performance of DL Training



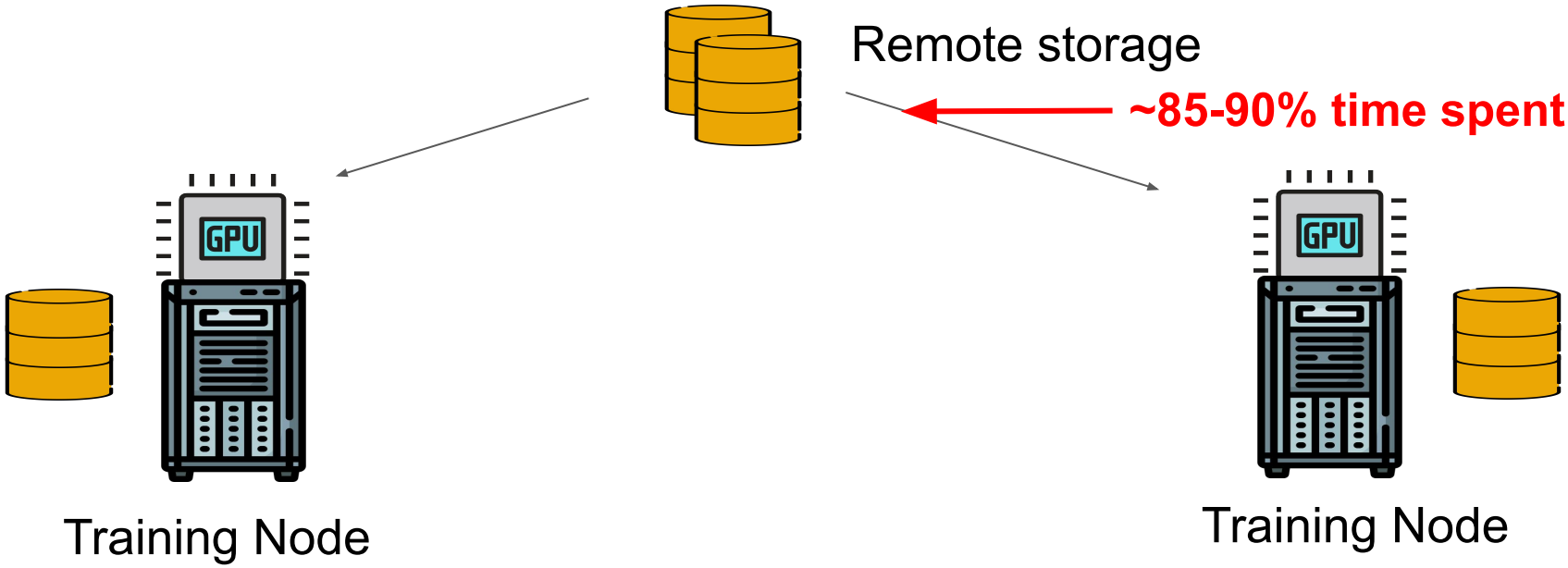
Data Storage Impacts the Performance of DL Training



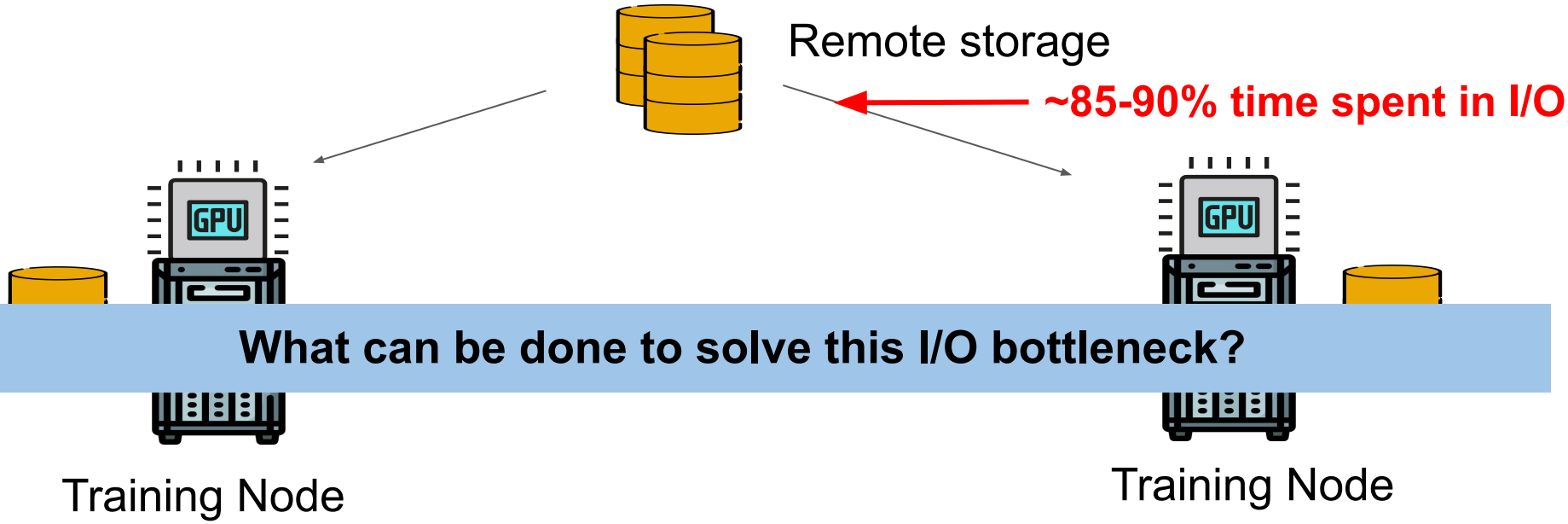
Data Storage Impacts the Performance of DL Training



I/O is the Bottleneck in Distributed DL Training

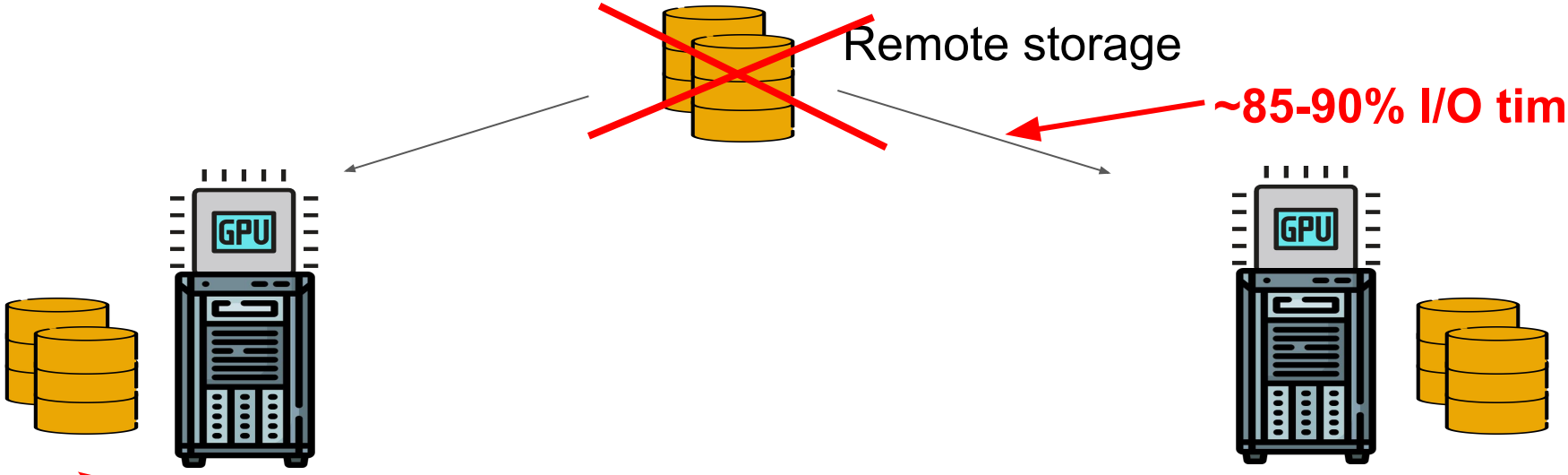


I/O is the Bottleneck in Distributed DL Training



What can be done to solve this I/O bottleneck?

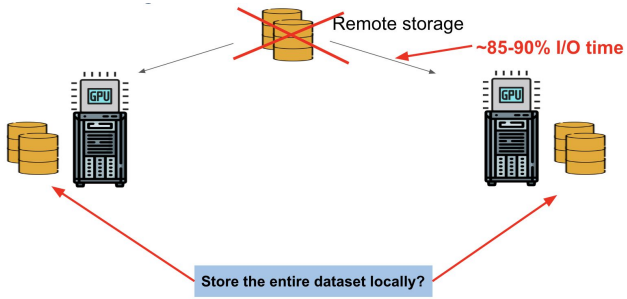
A Possible Solution - Local Storage



Store the entire dataset locally?

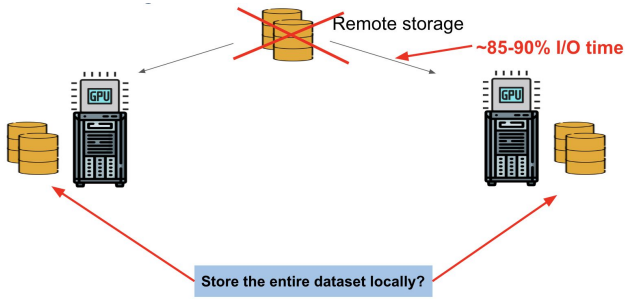
Problems with Storing Data Locally for DL Training

1. GPU VMs can lead to loss of local storage state



Problems with Storing Data Locally for DL Training

1. GPU VMs can lead to loss of local storage state
2. RAM size is very small



Can We Cache for DL Training?

Goal: Improve performance using a small working set size (WSS)

Can We Cache for DL Training?

Goal: Improve performance using a small working set size (WSS)

Caching DL workloads is non-trivial

- Data samples are fetched randomly
- All samples are accessed every epoch

Can We Cache for DL Training?

Goal: Improve performance using a small working set size (WSS).

Caching DL workloads is non-trivial.

- Data samples are fetched randomly
- All samples are accessed every epoch

Which samples should we cache?

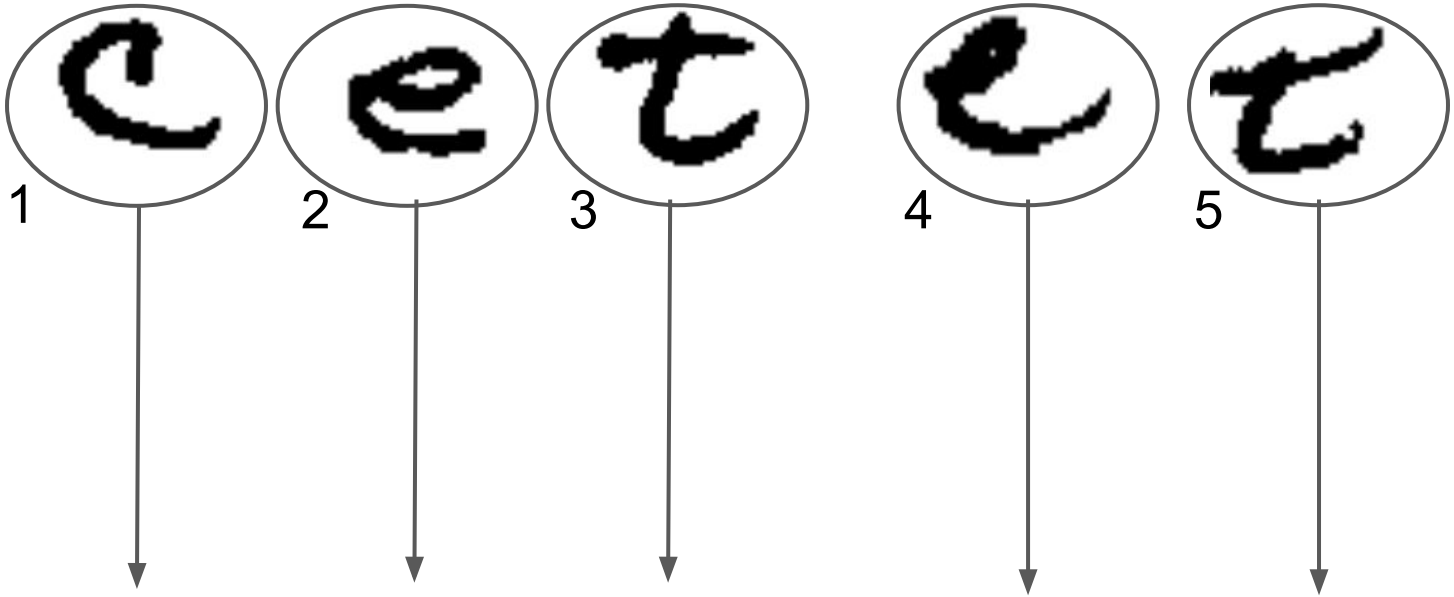
Key Observation - Not All Samples are Equal

Not all samples are equal[1,2]. Some are more important than others as they contribute more towards increasing the accuracy of a DL model.

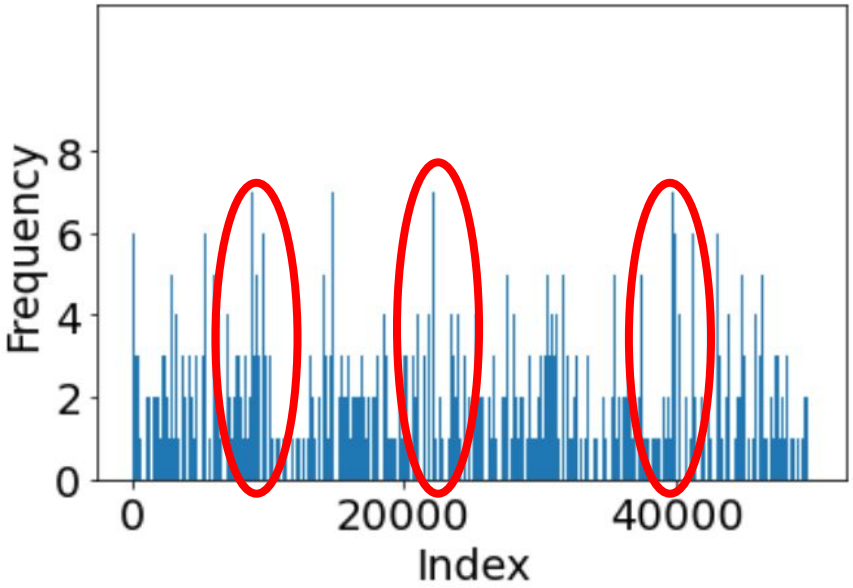
1. Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 2525–2534. PMLR, 10–15 Jul 2018
2. Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. arXiv preprint arXiv:1511.06343, 2015.

Inequality Among Sample Contribution

Hard-to-learn samples are the most important
Example: FEMNIST dataset



Repetitive Access Patterns are Amenable to Caching



Observations

~26% samples accessed more than once
~10% samples accessed more than three times

Key Insight of SHADE

DL Training can treat different samples differently. If we can predict the I/O accesses, we can cache the samples to fundamentally improve the I/O efficiency.

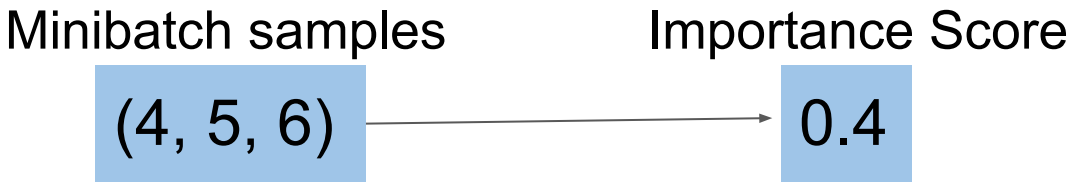
Making Important Samples Cacheable is Challenging

1. Identify per-sample importance
2. Avoid making the model biased
3. Track importance scores



Challenge 1: Coarse-grained Importance Scores

Importance scores are generally coarse-grained and thus inaccurate.

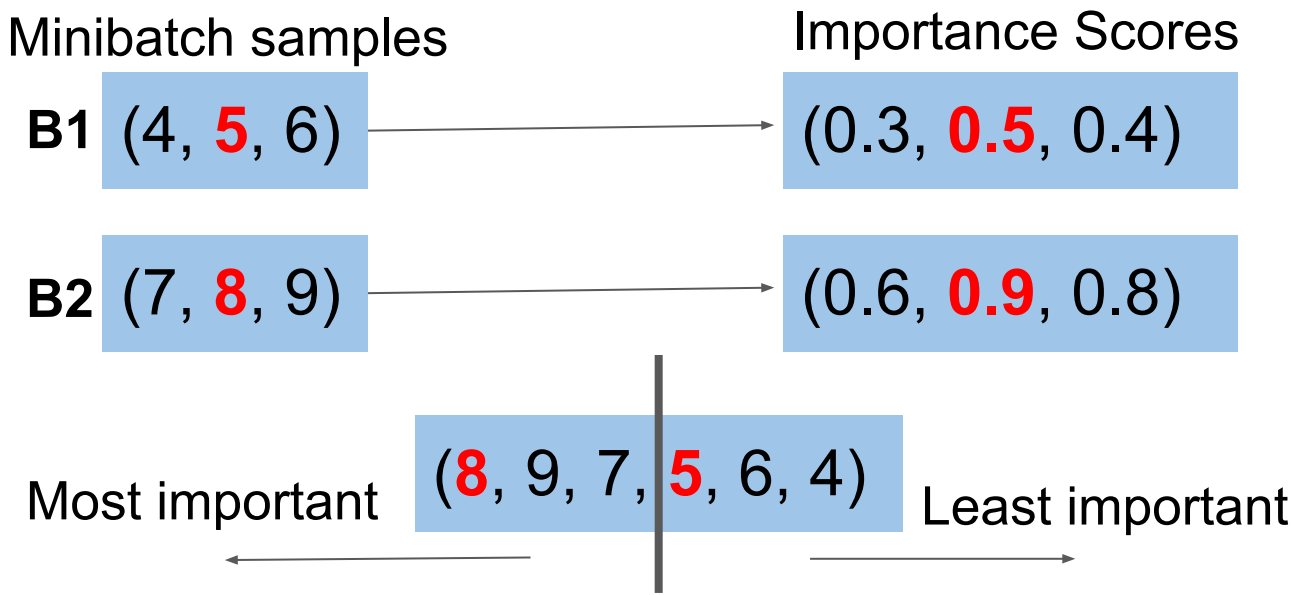


Score(4) = Score(5) = Score(6) → Not accurate

Individual samples have their own contribution towards improving the accuracy of the model.

Challenge 1: Coarse-grained Importance Scores

How can we compare the importance across minibatches?



Relative comparison across minibatches become difficult.

Challenge 2: Biased Models

Minibatch samples

(6, **8, 8, 8**, 9)

Importance Scores

(0.5, 0.8, 0.8, 0.8, 0.7)

Let’s use repeated samples to get more hits!



Challenge 2: Biased Models

Minibatch samples

(6, **8, 8, 8**, 9)

Importance Scores

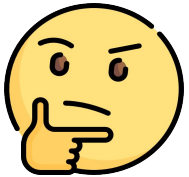
(0.5, 0.8, 0.8, 0.8, 0.7)

Let’s use repeated samples to get more hits!



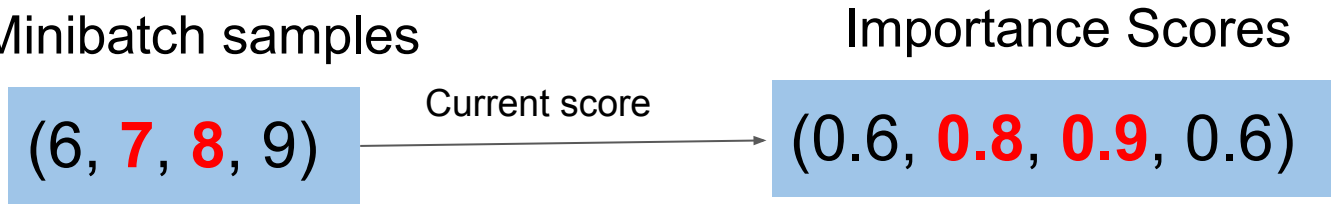
But, wait a sec!

Would training on repeated samples lead to biased results?



Challenge 3: Constant Change of Importance Scores

Importance scores are dynamic



Let’s put 7 and 8 in cache!

Challenge 3: Constant Change of Importance Scores

Importance scores are dynamic

Minibatch samples

(6, 7, 8, 9)

Current score

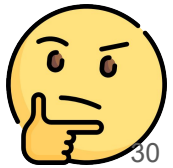
Importance Scores

(0.6, **0.8**, **0.9**, 0.6)

Let's put 7 and 8 in cache!

Future score

(**0.5**, 0.3, 0.4, **0.6**)

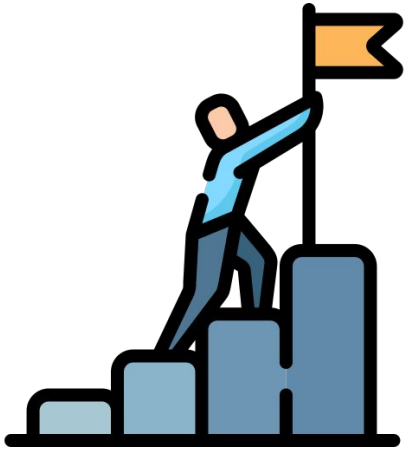


If cache is not updated, most important samples will not be in cache.

Making Important Samples Amenable to Caching

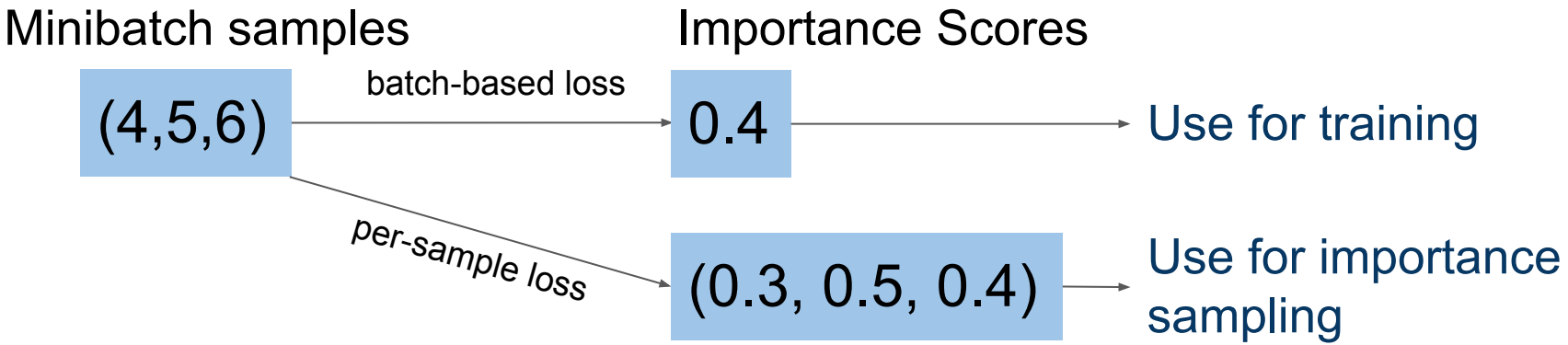
- Loss Decomposition
- Rank-based Score
- Priority-based Adaptive Sampling (*PADS*)
- Adaptive Priority-aware Prediction (*APP*) Cache

→ SHADE



Technique 1: Loss Decomposition

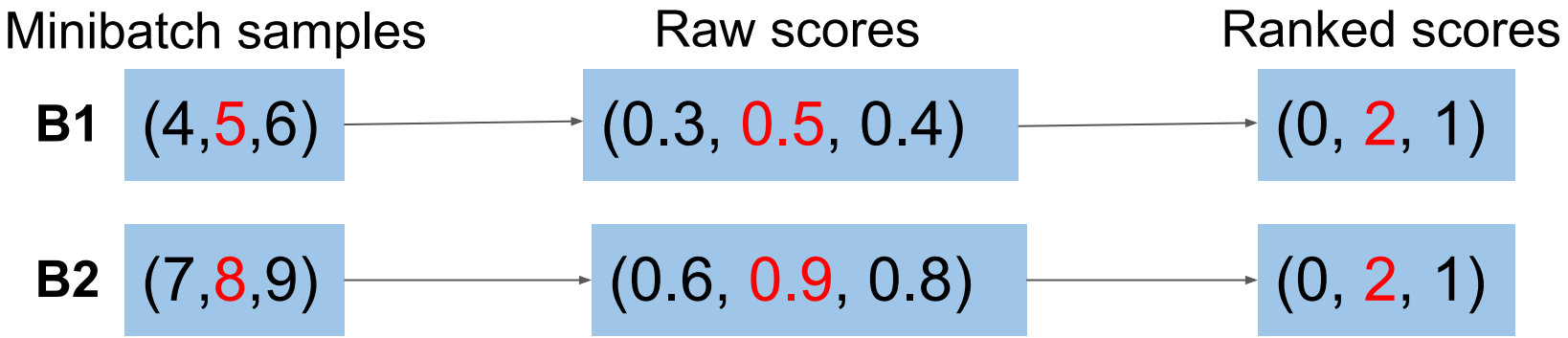
Tackling coarse-grained nature of importance scores



Use coarse-grained loss for training, fine-grained loss for importance sampling

Technique 2: Rank-based Importance Score

Adapting importance scores for priority-based caching



Rank fine-grained losses to detect relative importance.

Technique 3: Priority-based Adaptive Sampling (*PADS*)

Adapting sampling based on loss/accuracy change to avoid bias

Samples

4, 7, 3, 5, 3, 5, 2, 2, 1, 10 → Create samples list with repetitions based on importance.
Higher repetitions → Higher importance

3, 3, 5, 5, 2 → Send optimal list for training and caching
Cache → (3,5)
Hits → (3,3,5,5); Hit rate → 80%

Monitor the loss + accuracy in real time

Increase hit rate at will while keeping the accuracy improvement in check

Technique 4: Adaptive Priority-aware Prediction (*APP*)

Keep the most important samples in cache

Priority Queue(PQ) → importance of currently cached samples
Ghost Cache → importance of all trained samples.

Compare(importance of cached sample, importance of incoming sample)



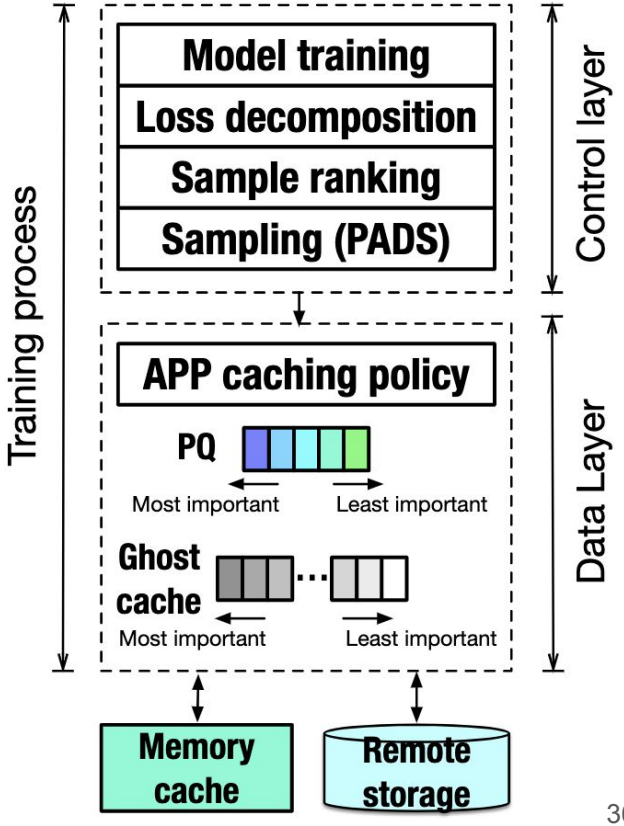


Get from PQ

Get from ghost cache

Compare importance scores to keep most important samples in cache

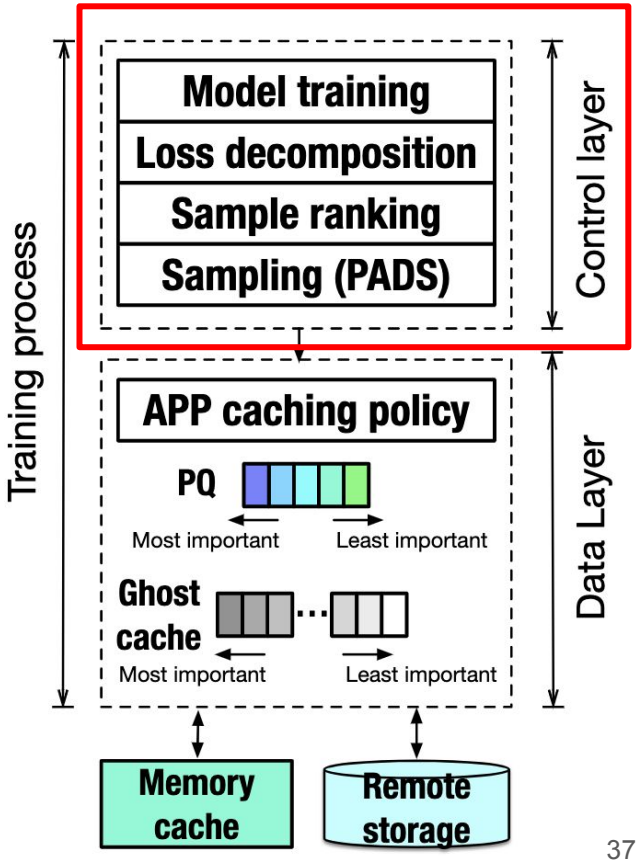
SHADE Architecture



SHADE Architecture

SHADE Control Layer

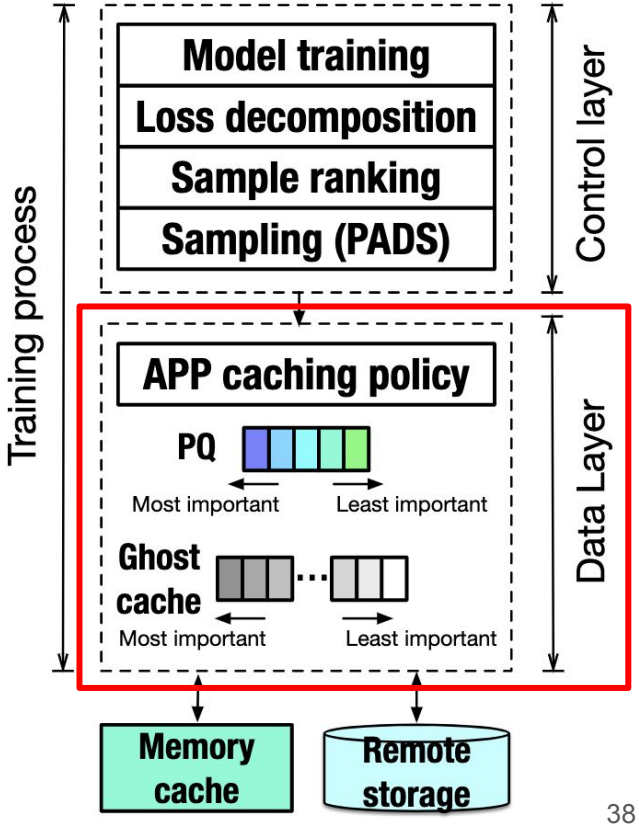
- Finding fine-grained importance, ranking them, and sampling



SHADE Architecture

SHADE Control Layer
- Finds fine-grained importance, ranks, and samples data

SHADE Data Layer
- Stores and retrieves from cache
- Updates the cache



SHADE Implementation

ShadeDataset

- Contains *APP* Cache logic
- Uses Redis as in-memory pooled cache

ShadeSampler

- Communicates with training processes using APIs
- Contains *PADS* logic

SHADE Training Process

- Contains Loss Decomposition + Ranking Logic

SHADE Analysis Framework

- Built-in framework to facilitate experimentation

SHADE Feature: Ease of Deployment

Default PyTorch

```
train_dataset =
datasets.ImageFolder(
            train_directory,
            transform_function)
val_dataset =
datasets.ImageFolder(
            val_directory,
            transform_function)
train_sampler =
DistributedSampler(train_dataset)
Train()
Validate()
```

SHADE

```
train_dataset = ShadeDataset(
            train_directory,
            transform_function,...)
val_dataset = ShadeValDataset(
            val_directory,
            transform_function)
train_sampler =
ShadeSampler(train_dataset)
Train()
Validate()
```

Evaluation Setup

Cluster

- 4 nodes each having 2 P100 GPUs
- HDD-based NFS server as a remote storage

Datasets

- ImageNet-1K
- CIFAR-10

Models

- AlexNet, ResNet-18, ResNet-50, VggNet

Baseline

- LRU incorporated PyTorch Distributed Training

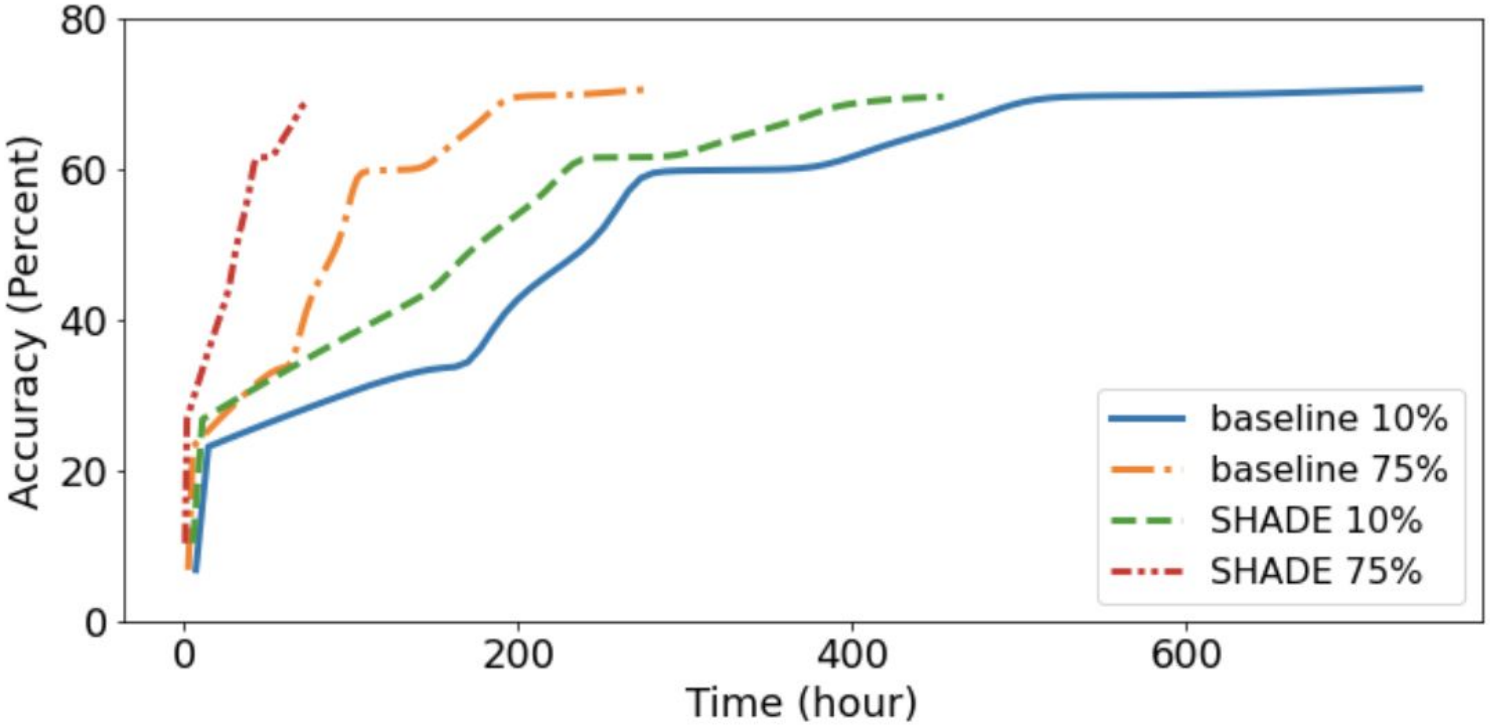
Evaluation Objectives

1. Accuracy vs. Time
2. Throughput
3. Hit Ratio
4. Minibatch Load Time

Evaluation Objectives

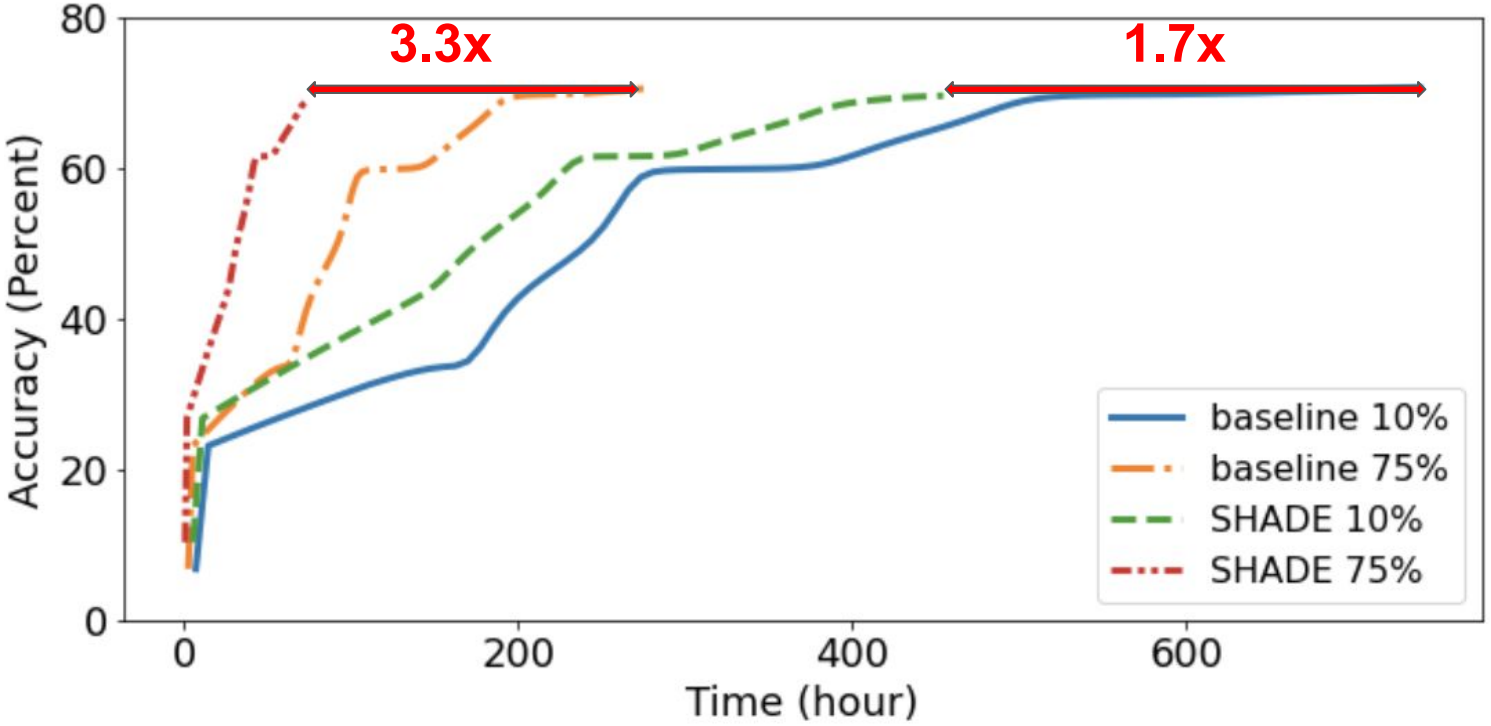
1. Accuracy vs. Time
2. Hit Rates
3. Throughput
4. Minibatch Load Time

Accuracy vs. Time



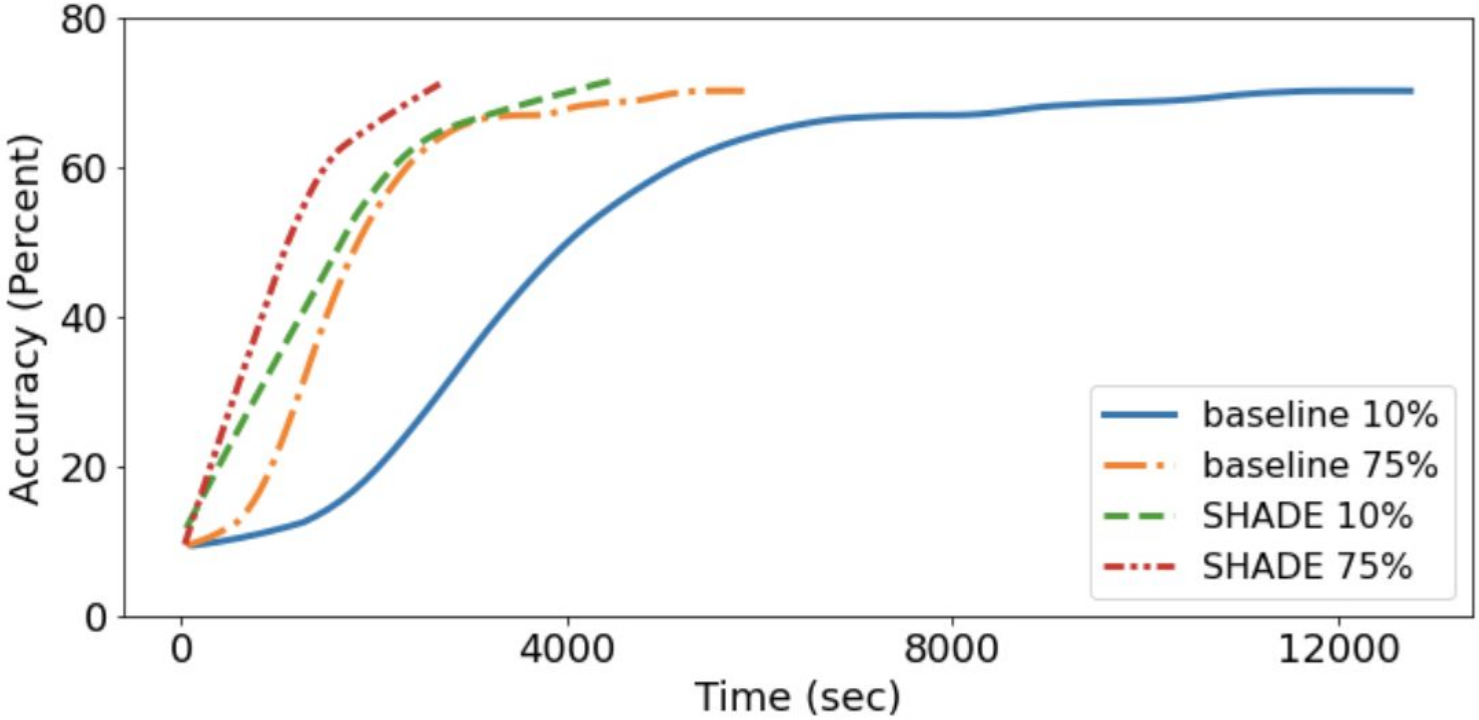
ResNet-50 model + ImageNet Dataset

Faster Accuracy Convergence



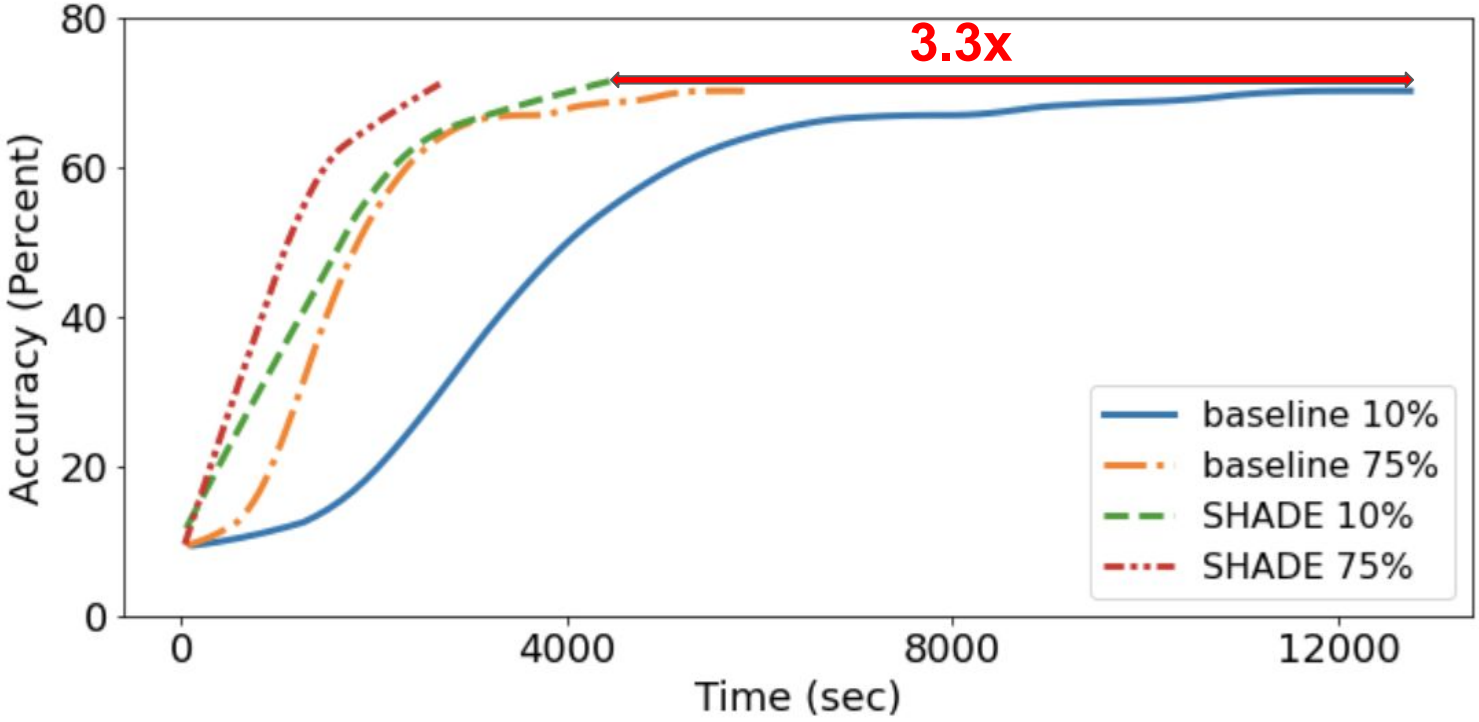
ResNet-50 model + ImageNet Dataset

Accuracy vs. Time



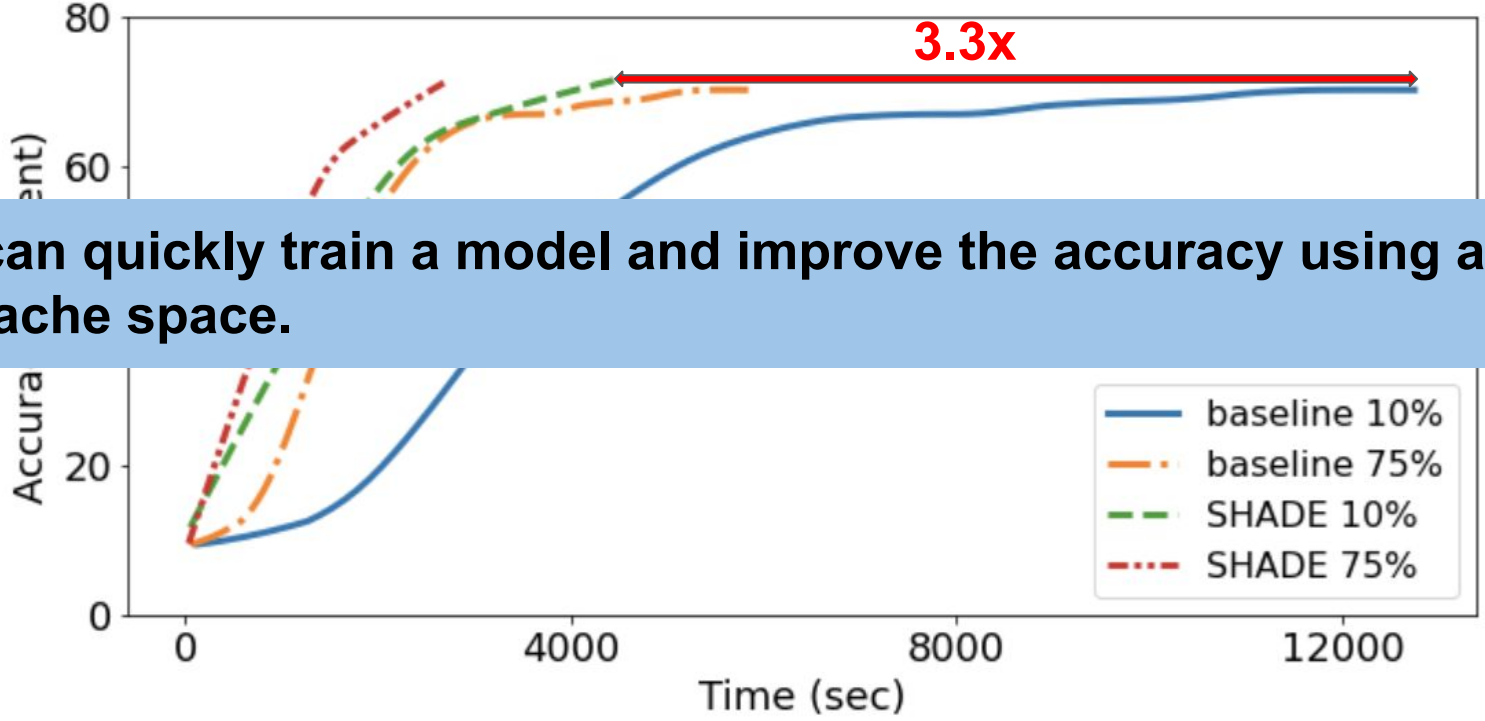
AlexNet model + CIFAR-10 Dataset

Faster Accuracy Convergence



AlexNet model + CIFAR-10 Dataset

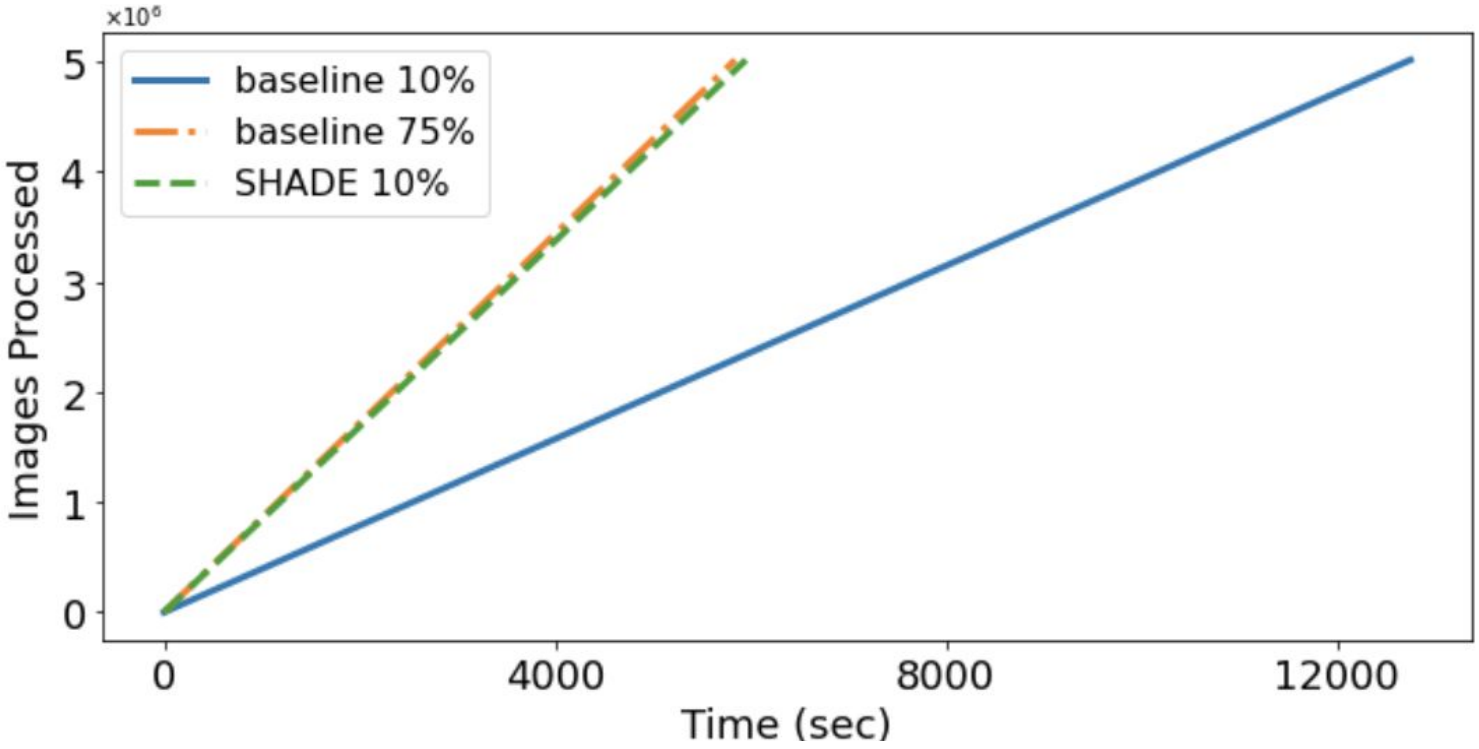
Faster Accuracy Convergence



SHADE can quickly train a model and improve the accuracy using a limited cache space.

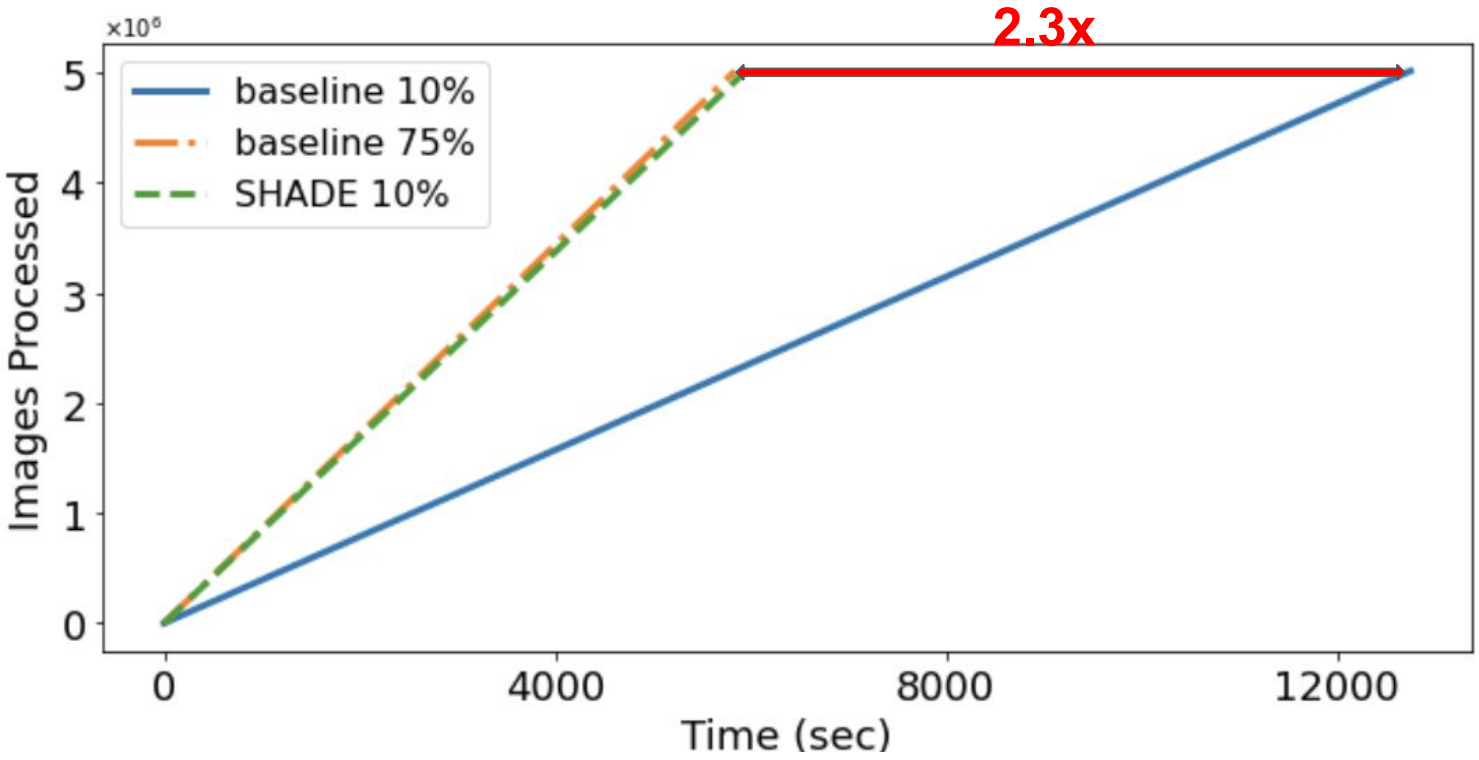
AlexNet model + CIFAR-10 Dataset

Impact on Throughput



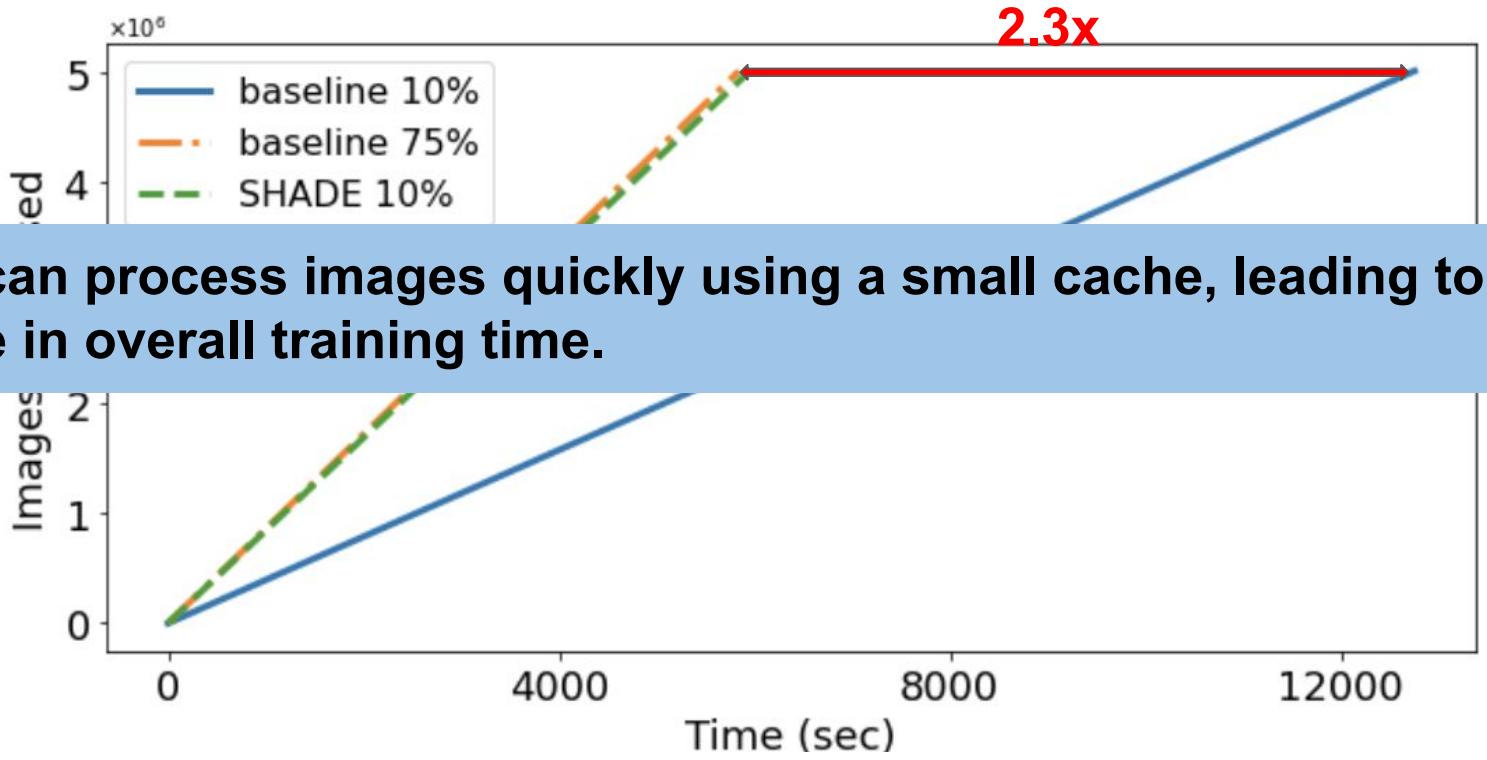
AlexNet model + CIFAR-10 Dataset

Higher Throughput



AlexNet model + CIFAR-10 Dataset

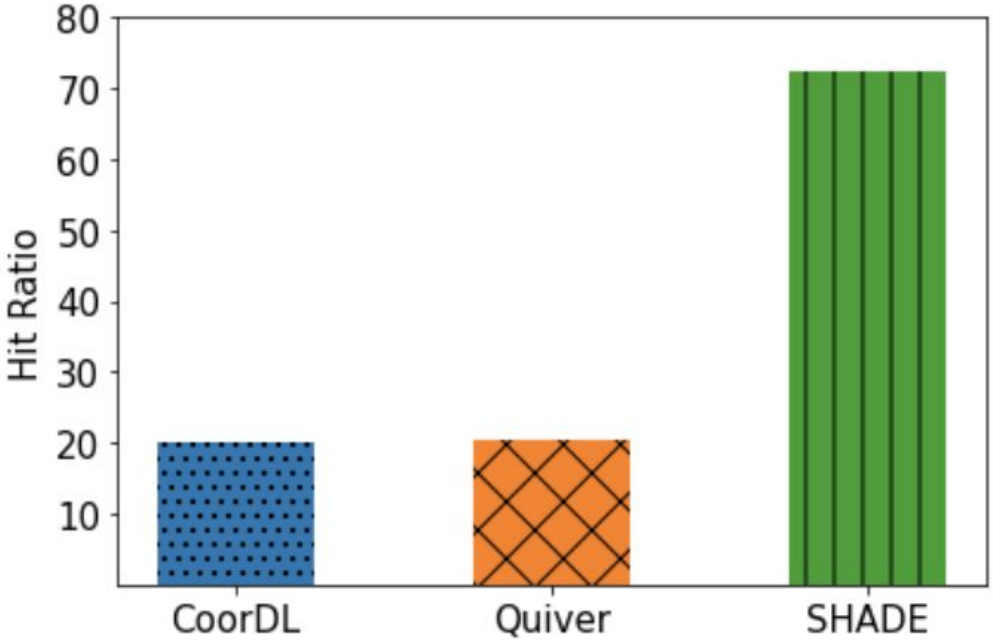
Higher Throughput



SHADE can process images quickly using a small cache, leading to a decrease in overall training time.

AlexNet model + CIFAR-10 Dataset

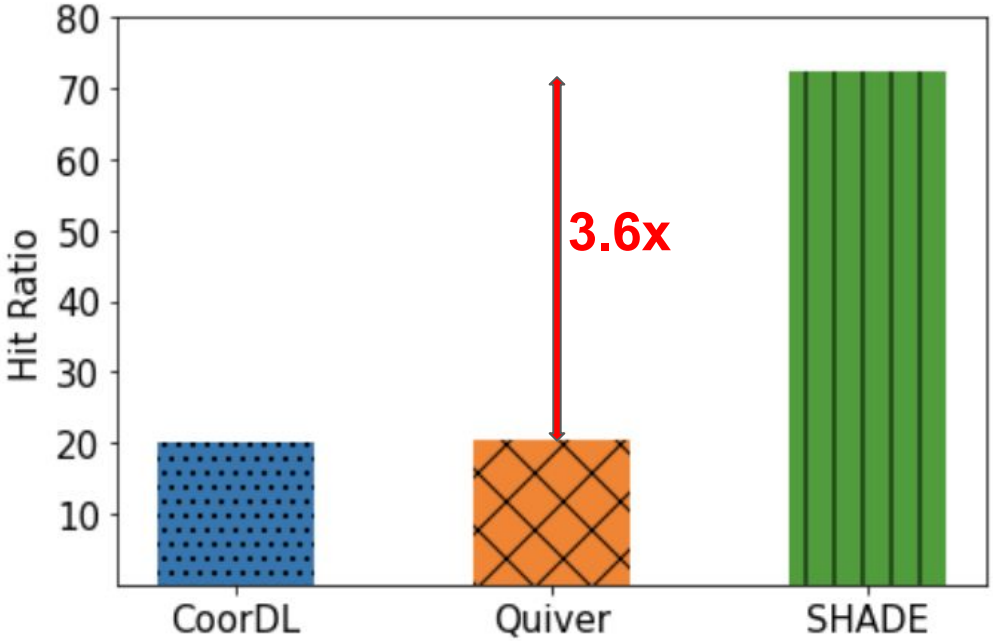
Impact on Hit Ratio



20% WSS of CIFAR-10 + ResNet-18

1. Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An informed storage cache for deep learning. In 18th USENIX Conference on File and Storage Technologies (FAST 20), pages 283–296, 2020.
2. Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. arXiv preprint arXiv:2007.06775, 2020

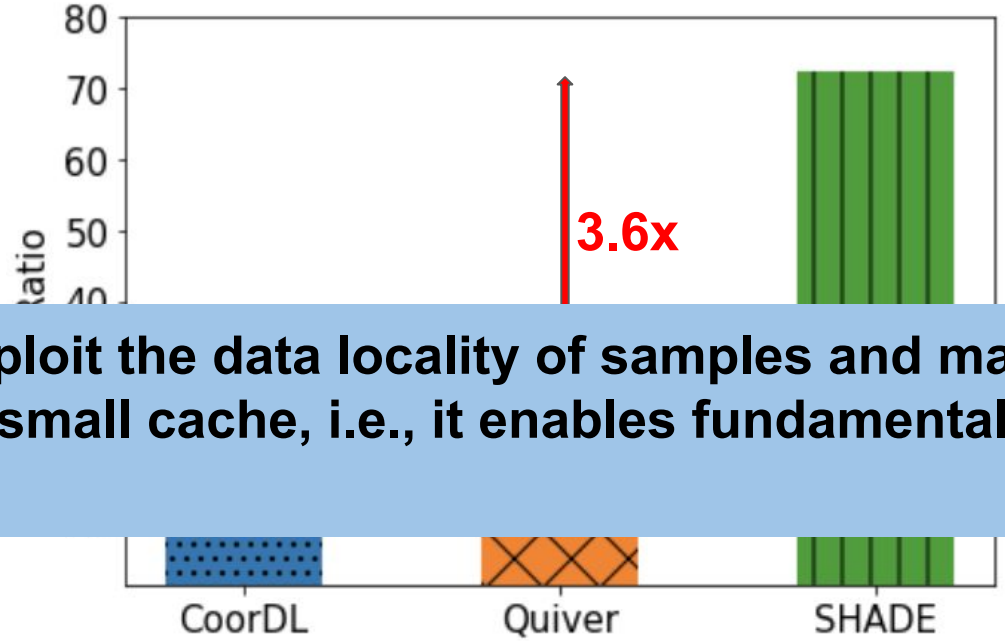
Higher Exploitation of Data Locality



20% WSS of CIFAR-10 + ResNet-18

1. Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An informed storage cache for deep learning. In 18th USENIX Conference on File and Storage Technologies (FAST 20), pages 283–296, 2020.
2. Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. arXiv preprint arXiv:2007.06775, 2020

Higher Exploitation of Data Locality



SHADE can exploit the data locality of samples and make the best utilization of a small cache, i.e., it enables fundamental cacheability of DL workloads.

20% WSS of CIFAR-10 + ResNet-18

1. Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An informed storage cache for deep learning. In 18th USENIX Conference on File and Storage Technologies (FAST 20), pages 283–296, 2020.
2. Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. arXiv preprint arXiv:2007.06775, 2020

Summary and Conclusion

SHADE is a caching system for DL workloads.

- Provides the ability to train more on hard-to-learn samples (Ranking fine-grained importance + *PADS policy*)
- Retains the most important samples in cache (*APP* Cache)
- Increases hit rate in cache leading to faster accuracy convergence (~3.3x) and increased throughput (~2.3x)
- Enables fundamental cacheability of DL Training



SHADE is available at:
https://github.com/R-I-S-Khan/SHADE

Contact:
Redwan Ibne Seraj Khan
redwan@vt.edu
My webpage:
https://sites.google.com/vt.edu/redwanibneserajkhan/home



Distributed Systems and Storage Laboratory @ Virginia Tech
https://dssl.cs.vt.edu