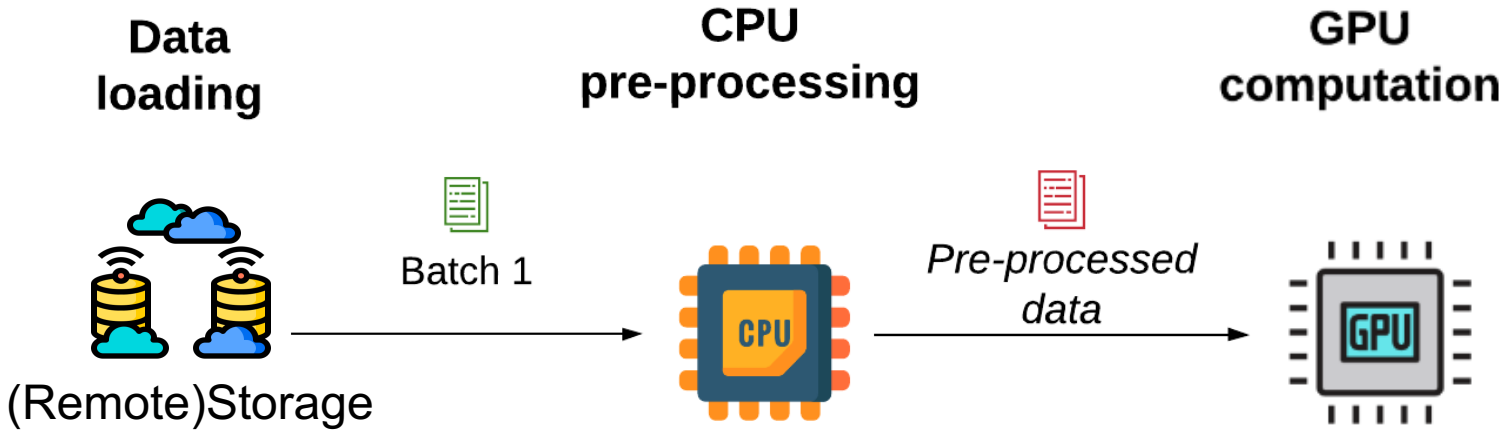
A Selective Preprocessing Offloading Framework for Reducing Data Traffic in DL Training

Meng Wang, Gus Waldspurger,

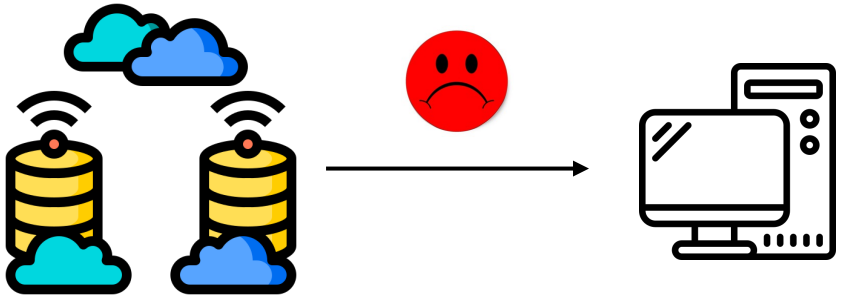
Swaminathan Sundararaman

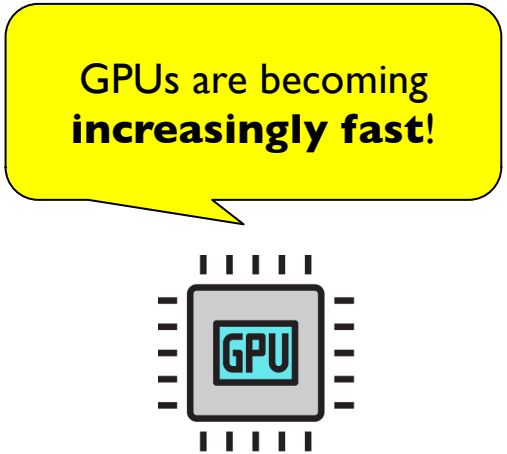
Introduction

- Deep Learning (DL) is widely used
  - Training is important for good accuracy
- DL training has intense requirements for storage, CPUs, and GPUs.



Problem Statement





- Training data too large to fit in local storage
  - For example, OpenImages totals 18TB.
- Need to be fetched from **remote storage**

*What if remote **data** fetch rate **<** **GPU** compute rate?*

GPU underutilized!

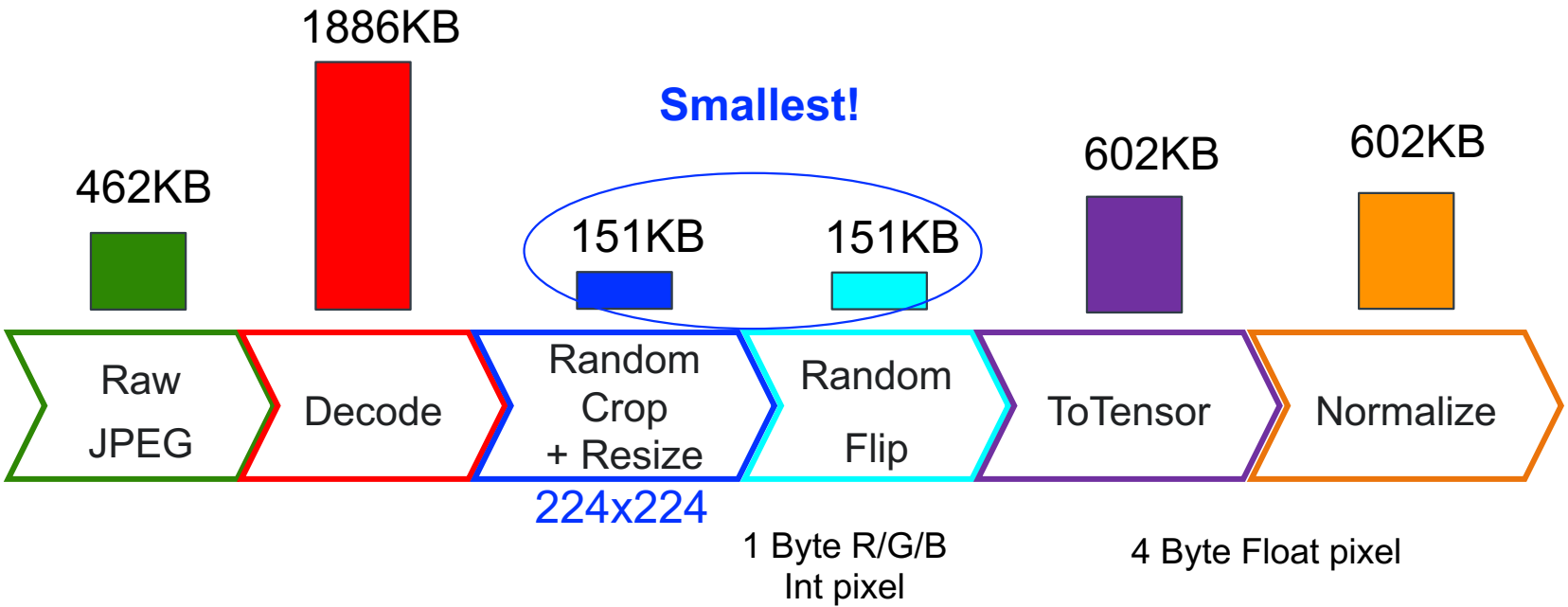
Longer training time!

Existing Solutions

- We want to reduce data traffic from remote storage to compute node
- Local cache
  - Selectively cache data in local storage or memory
  - Can be limited by local storage/memory capacity 
    - Datasets are still increasing in size
- Store preprocessed data
  - Store preprocessed data in remote storage for repeated use
  - Risks compromising training accuracy 
    - Online preprocessing is important for training accuracy
  - Data size might even increase after preprocessing

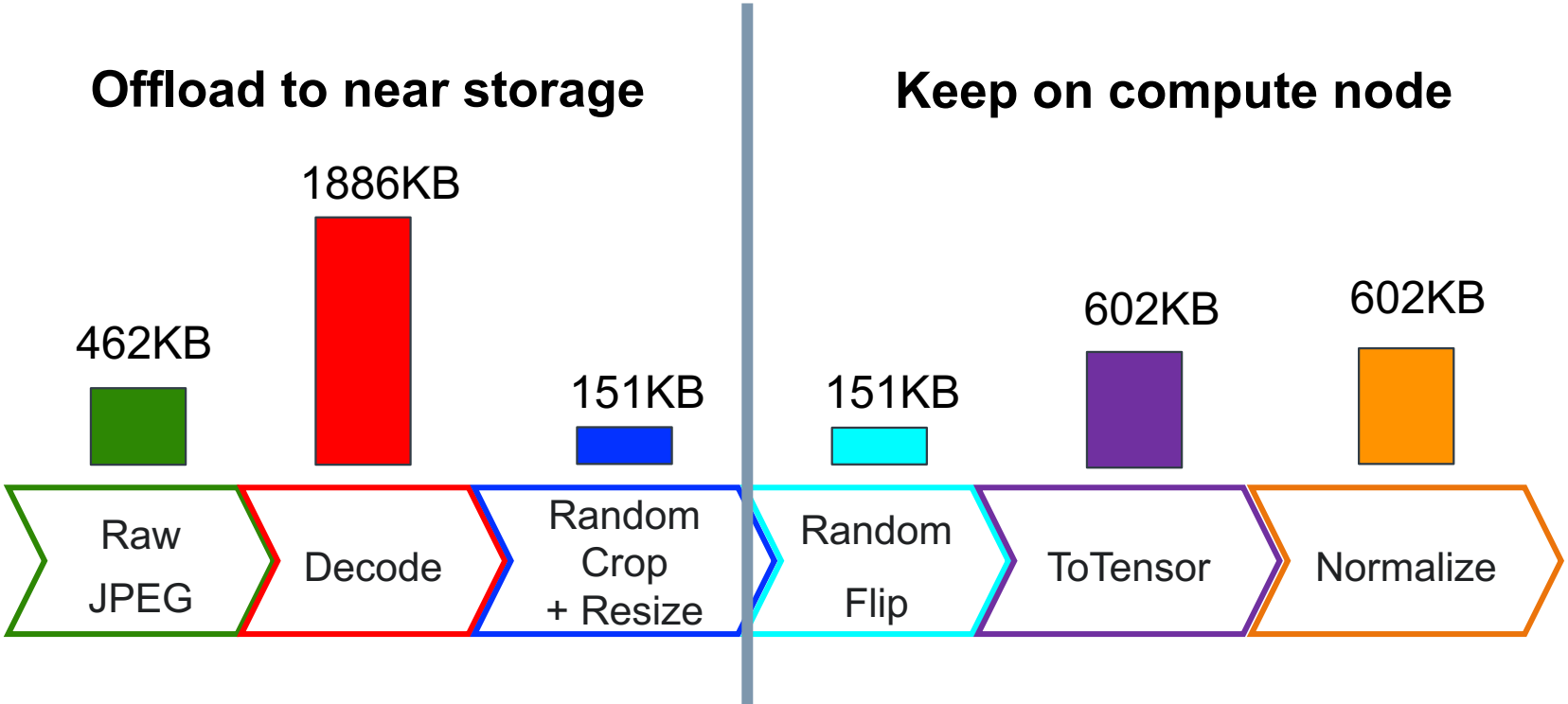
Opportunities in Data Preprocessing

- Many samples' sizes decrease in the middle of preprocessing
- Case study: ImageNet classification



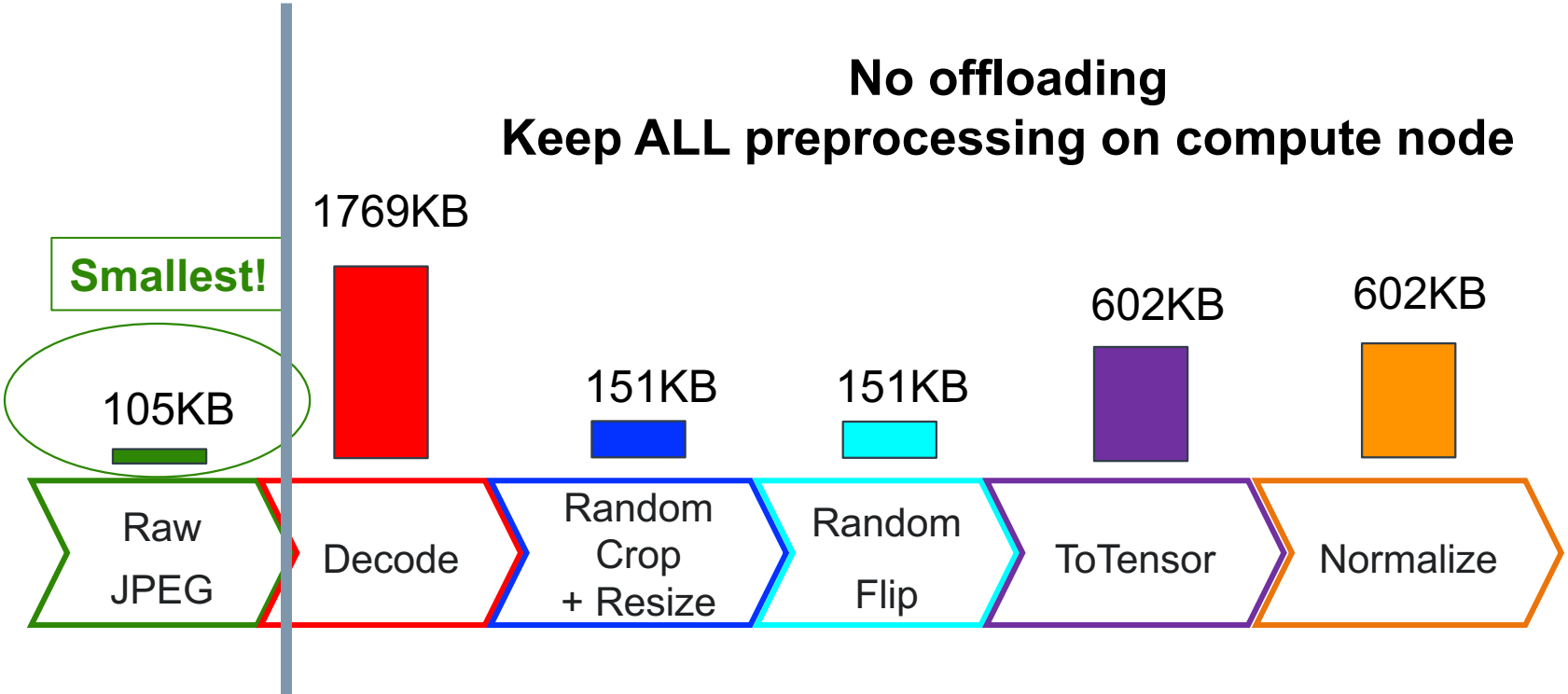
Solution: Preprocessing Offloading

- Solution: Selectively offload preprocessing steps to storage side



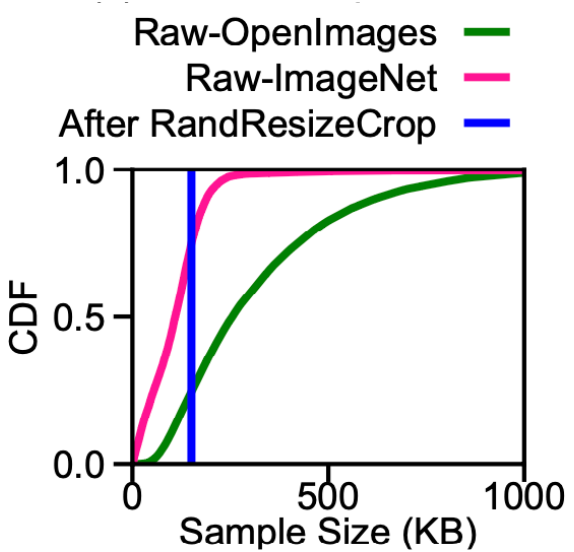
Challenges in Preprocessing Offloading

- Finding 1: NOT all samples can benefit from preprocessing offloading



Challenges in Preprocessing Offloading

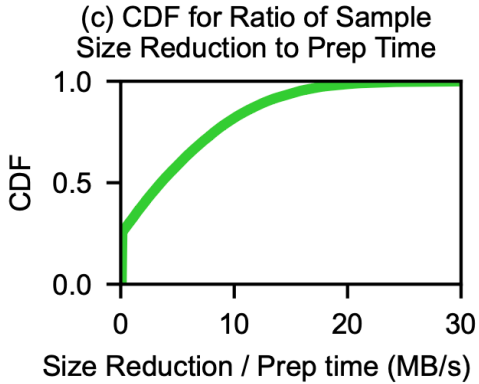
- Finding 2: Different datasets/samples benefit differently from offloading
- Case studies:
  - ImageNet
    - 26% samples can benefit
  - OpenImages
    - 76% samples can benefit
  - Larger raw images benefit more



CDF of Sample size

Challenges in Preprocessing Offloading

- Finding 3: Offloading introduces CPU overhead to storage node
  - Storage cluster usually has limited CPU capabilities
  - Tradeoff between traffic reduction vs. CPU overhead
- Offloading efficiency:
  - Ratio of size reduction to offloaded preprocessing time
  - Different samples have different offloading efficiency



Limitations of Prior Offloading Works

- We need a fine-grained, data-selective offloading approach
  - **Data selective**: carefully select which samples to offload based on dataset's characteristics.
- Existing preprocessing offloading works
  - Focuses on CPU bottlenecks
    - Offload preprocessing to remote CPU workers
    - Not designed to reduce remote data traffic
  - NOT data selective
    - Coarse-grained
    - Fail to exploit heterogeneous size behavior of samples

	Operation Selective	Data Partial	Data Selective	To Near Storage
tf.data svc	–	–	–	–
GoldMiner	✔	–	–	–
FastFlow	–	✔	–	–
cedar	✔	–	–	–

Our Solution: SOPHON

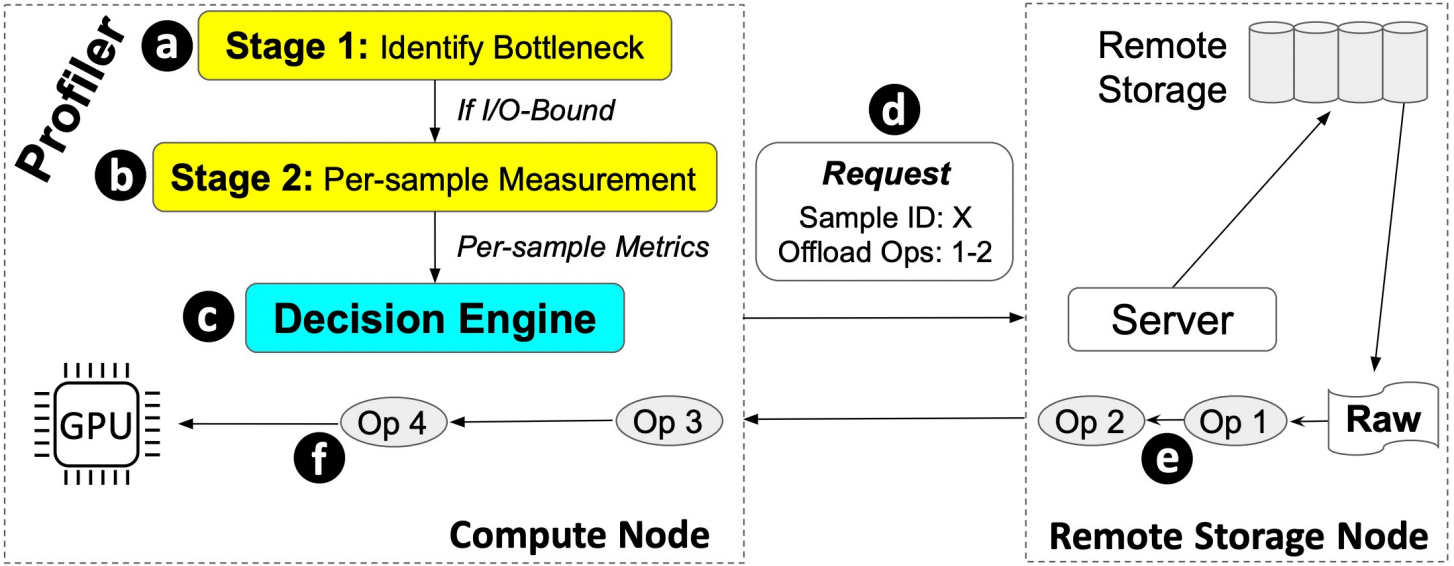
- **SOPHON**: **S**electively **O**ffloading **P**reprocessing with **H**ybrid **O**perations **N**ear-storage
  - Per-sample per-operation granularity
    - First data-selective offloading for DL training
  - Minimizes remote data traffic
  - Balances offloaded CPU overhead and traffic reduction

	Operation Selective	Data Partial	Data Selective	To Near Storage
tf.data svc	–	–	–	–
GoldMiner	✔	–	–	–
FastFlow	–	✔	–	–
cedar	✔	–	–	–
SOPHON	✔	✔	✔	✔

Table 1. Existing Offloading [32–35] vs. SOPHON.

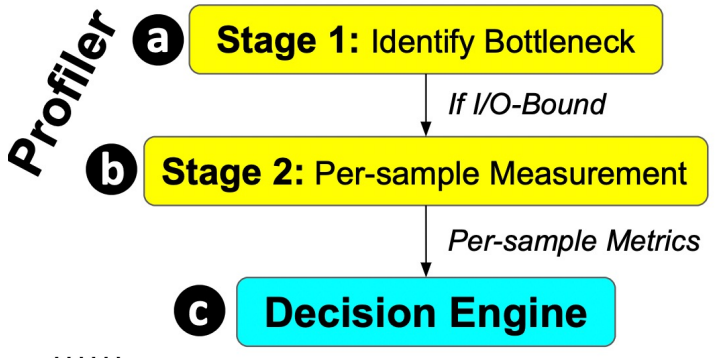
SOPHON Design

- Lightweight two-stage profiler
- Per-sample offloading decision engine



SOPHON: Two-Stage Profiler

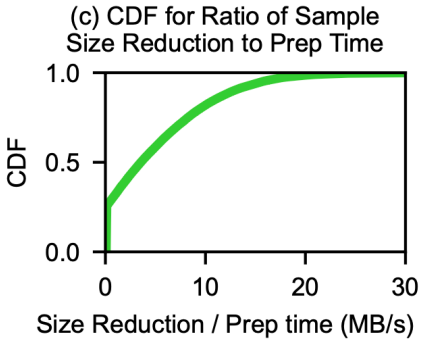
- **Stage 1**: Identify bottlenecks (Inspired by DataStall@VLDB '21)
  - Throughput measurement
    - GPU throughput
    - I/O throughput
    - CPU throughput
  - Proceed ***only if I/O-bound***



- **Stage 2**: Per-sample profiling
  - Measure *time* and *size* per sample per preprocessing step
  - *Online* measurement during *first epoch* without offloading
  - Assumes homogeneous CPUs

SOPHON: Offloading Policy

- Compute *offloading efficiency* for each sample
  - Ratio of size reduction to offloaded CPU preprocessing time
- Sort samples by offloading efficiency
  - **Prioritize** samples with **higher offloading efficiency**
- Pick samples for offloading and update throughput until:
  - I/O is no longer bottleneck, or
  - All samples with positive efficiency have been picked
- Balances traffic reduction and offloaded CPU overhead

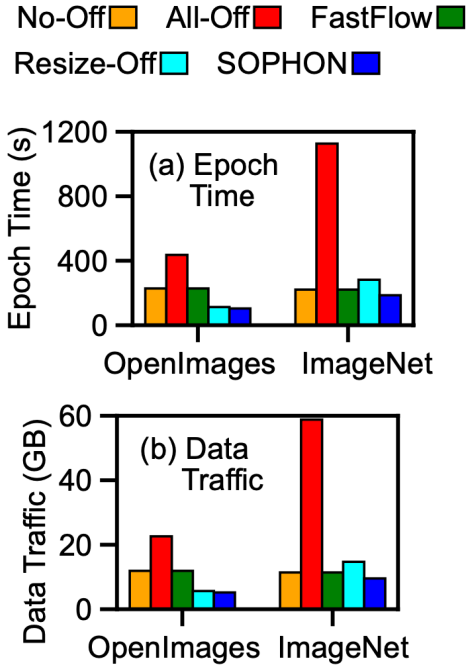


Implementation & Experiment Setup

- Implemented on top of PyTorch
- Small-scale experiments to mimic real-world scenarios
- Two nodes
  - A GPU (RTX6000) node
  - A storage node
- Benchmarks:
  - OpenImages (12GB subset)
  - ImageNet (11GB subset)
- Limited bandwidth
  - Downscaled to 500Mbps
- Five offloading policies
  - No-Off: offload nothing
  - All-Off: offload everything
  - FastFlow: designed for CPU bottlenecks
  - Resize-Off: offload until RandomCropResize
  - SOPHON

Evaluation: Ample CPUs on Storage

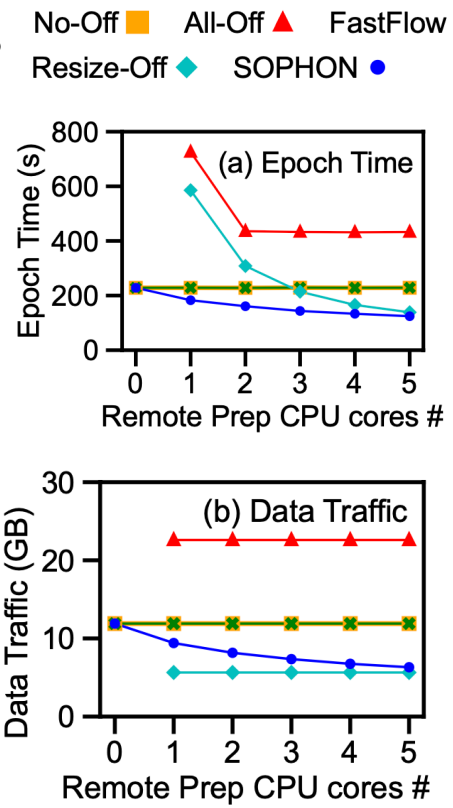
- No-Off: Baseline
- All-Off: Longest training time
  - Many samples become larger after all prep
- FastFlow: choose to offload nothing
  - Coarse-grained and detect All-Off is bad
- Resize-Off:
  - 2x reduction for OpenImages
  - 1.3x increase for ImageNet
- SOPHON:
  - 2.2x reduction for OpenImages
  - 1.2x reduction for ImageNet



Evaluation: Limited CPUs on Storage

- No-Off: Baseline
- All-Off: Longest training time
  - Even longer time when <2 cores due to CPU bottleneck
- FastFlow: choose to offload nothing
- Resize-Off:
  - Least traffic **BUT** longer training time
  - Can cause CPU bottleneck on storage node
- SOPHON:
  - Least training time 

OpenImages



Conclusion & Future Work

- **SOPHON**: **S**electively **O**ffloading **P**reprocessing with **H**ybrid **O**perations **N**ear-storage for DL training
  - Two-stage profiler to collect essential metrics
  - Per-sample offloading decision engine to balance traffic reduction and CPU overhead
- Future work:
  - Selectively compress preprocessed data
  - Extend support to heterogeneous CPUs
  - Study more DL workloads
  - Conduct more realistic evalutions

Thanks!

Q&A