Table of Contents

Self-Introduction

Shilong Lei

Self-Introduction

- Education
  - Purdue University, Department of Computer Science
    - Ph.D. student (3rd year)
  - Tsinghua University, School of Information Science and Technology
    - B.E. in Automation
- Internship
  - ByteDance Inc.
    - Research Scientist Intern, Seed Group (ML System)
      - Supported a LLM training simulator to simulate the training latency of LLM
      - Supported memory profiling and distributed training simulation in frameworks
      - Supported various parallelism techniques in Bytedance opensource distributed machine learning framework.

Self-Introduction

- Research Area
  - Efficient machine learning systems (LLMs)
    - LLM training/inference speedup
  - Autonomous drones & efficient mobile vision system

Project Report

Shilong Lei

SimpleFSDP: Automatic Selective Unsharding

Shilong Lei
Mentor: Shuai Yang
Date: Aug 21, 2025

Motivation

- **Training large models = balancing communication, computation, memory**
- SimpleFSDP exposes AG/RS ops as explicit PT2 graph nodes → enables graph-level optimization
- Current limitations:
  - **Fixed bucket size → communication bubbles**
  - **Binary policies: free-all vs keep-all**
    - Free-all: minimal memory but redundant AG in backward
    - Keep-all: no redundant comm but huge memory footprint
- **Goal:**
  Use *idle GPU memory* to **avoid redundant all-gathers** and **improve training throughput**.

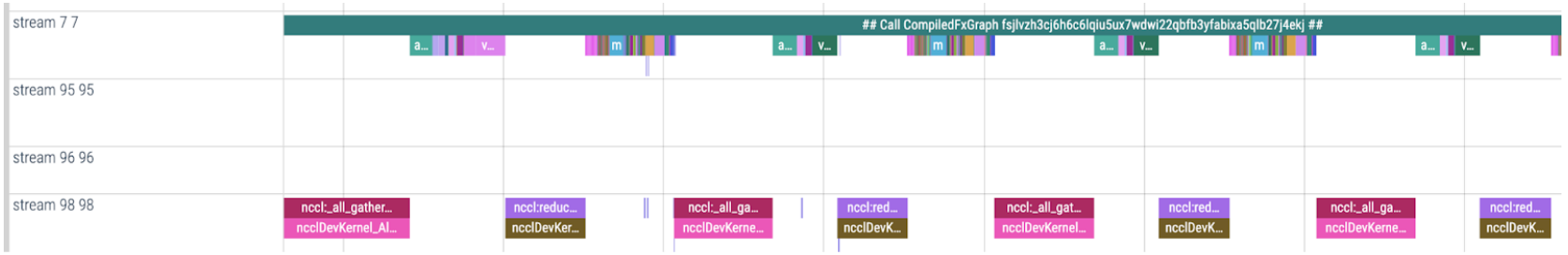
Key Idea

- **Selective Unsharding Framework**
  - “Retain only the parameters that yield the best communication savings per unit memory.”
  - Benefits:
    - Reduces backward AG communication
    - Uses spare memory intelligently
    - Works with PT2/compile-friendly SimpleFSDP
- Modes:
  1. **Manual** — user specifies modules to unshard
  2. **Automatic** — system selects modules based on memory budget & granularity

SimpleFSDP Enhancements

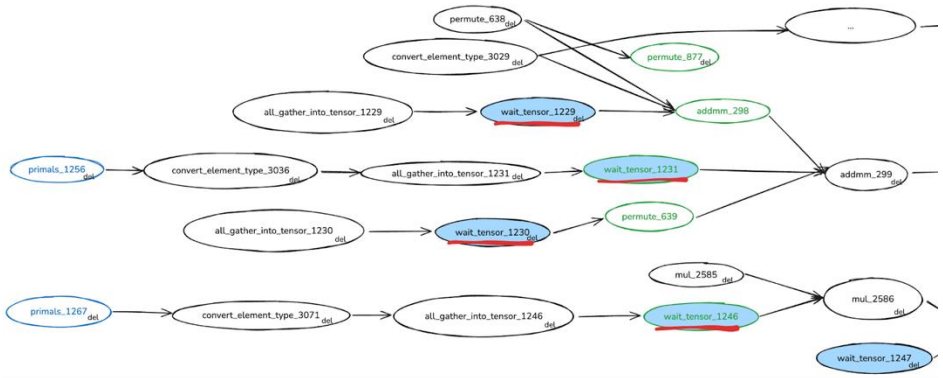
1. Configurable Bucket Size

- Bucket size can be set per-layer or global
- Helps align AG/RS with compute workloads
- Reduces comm bubbles
- APS-friendly configuration
- *Example:*
  - all_gather_bucket_cap_mb: {"default": 2000, 2: 1000}



Hierarchical Module Memory Profiling

- **Why needed?**
  - PT2 compile breaks module → graph-node mapping
  - Need memory footprint & mapping to decide unsharding
- **Our solution**
  - **tlparse-based hierarchical memory profiler**
  - Records:
    - module <=> graph node mapping
    - memory cost per module/submodule
  - Supports profiling only all_gather_into_tensor nodes
- **Enables:**
  - memory-aware unsharding
  - recursive splitting
  - automatic greedy selection

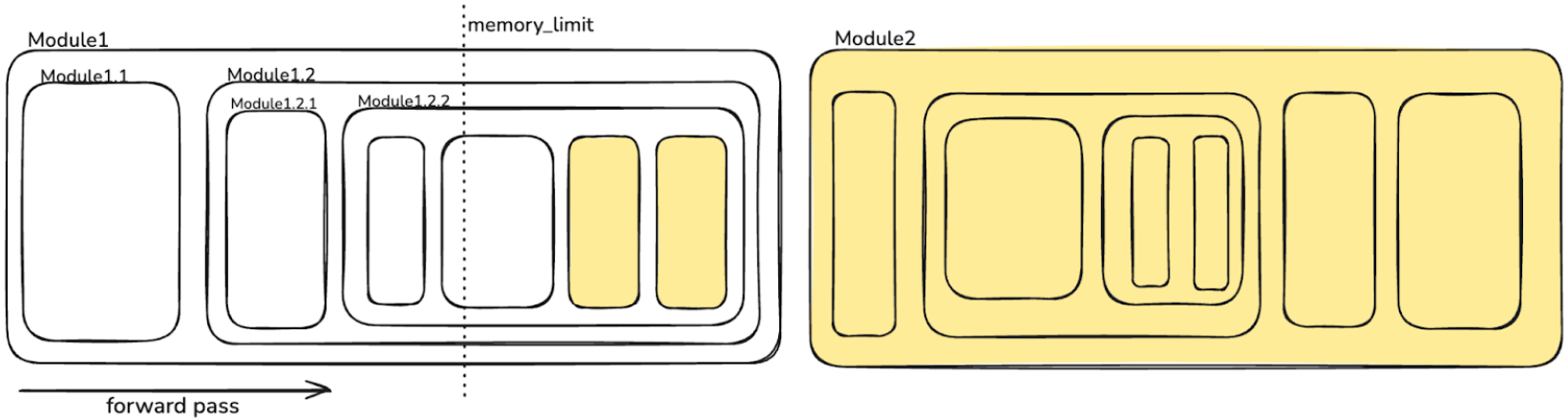


Selective Unsharding Framework

- **Manual Mode**
  - automatic: false
  - User directly provides target modules
  - Deterministic, expert-controlled
- **Automatic Mode**
  - automatic: true
  - Input: memory_limit, granularity, target modules
  - Greedy reverse traversal:
    - unshard late modules first
    - split large modules if needed
    - stop when reaching budget

Automatic Unsharding Algorithm (Simplified)

- **Reverse-order greedy selection:**
  1. Try to fit a module into memory budget
  2. If too large → recursively split
  3. Stop when no more progress possible
- **Rationale:**
  Later-forward modules = shorter residency → cheaper to unshard.



Experiment Setup

- **Hardware:** 8 × GPUs (GRANDTETON)
- **Model:** OmniFM v2 (pruned)
- **Batch size:** 128
- **Iterations:** 1000

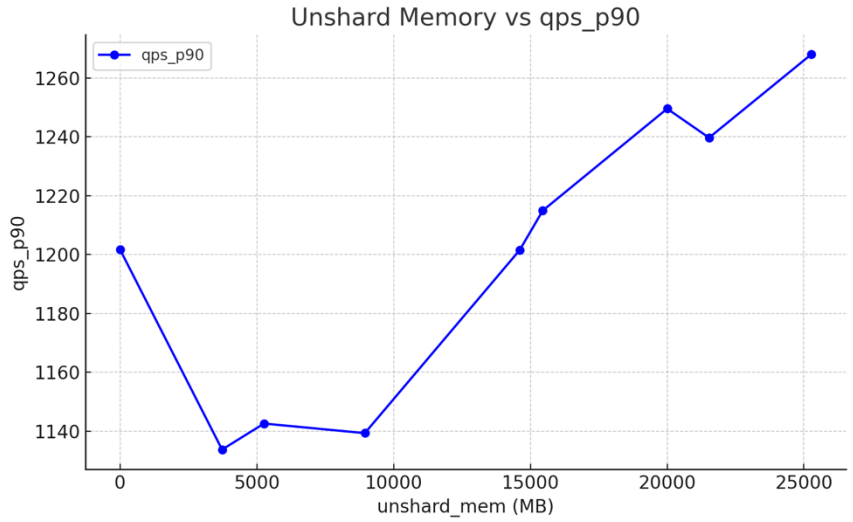
- Experiments:

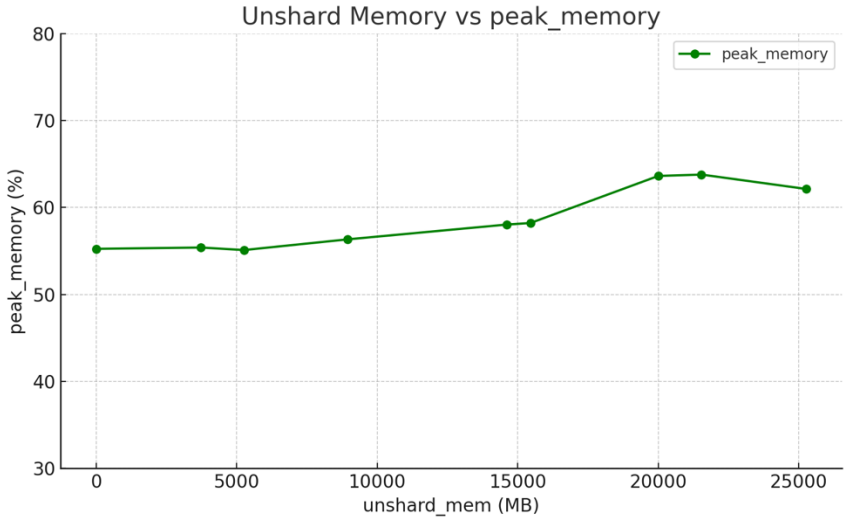
  1.Manual mode: progressively unshard later → earlier modules

  2.Auto mode: sweep memory_limit from small → large

Manual Mode Results

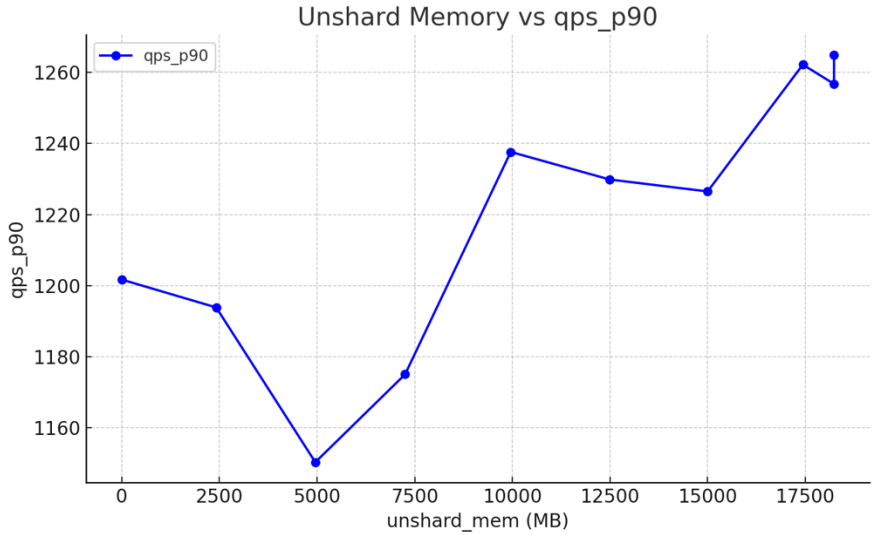
- Observation:
  - QPS **increases** as unshard size grows
  - GPU memory usage rises accordingly
  - Best configuration: ~**5.5% QPS improvement**
  - Confirms: *idle memory → better efficiency*

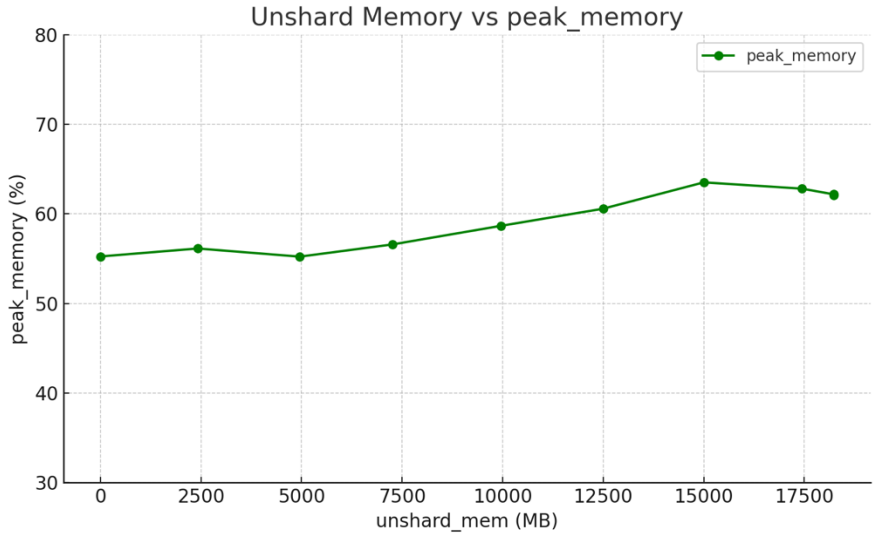




Auto Mode Results

- Observation:
  - As memory_limit increases:
    - Selected unshard size increases
    - QPS improves (not strictly monotonic)
  - Achieved +**5% QPS increase** with large budgets
  - No manual expertise needed → easier for users





Discussion

- **Why some partial unsharding regresses?**
  - Some AGs are not exposed or not on the critical path
  - Removing non-critical AGs does not reduce latency
  - **Takeaways:**
  - Memory-aware selection is key
  - Communication savings vary per module
  - Future: identify critical-path AGs

Future Work

- **1. Smarter Unshard Policies**
  - Prioritize modules with exposed AGs
  - Cost–benefit ranking instead of pure greedy
- **2. Dynamic & Adaptive Bucketing**
  - Auto-tuning bucket sizes
  - Runtime adaptation based on memory pressure or comm patterns
- **3. Broader Evaluation**
  - More models, deeper architectures
  - Expect larger gains for bigger models
- **4. Runtime Integration**
  - Tighter coupling with PT2 compiler & memory manager
  - Fully automated memory–communication co-optimization

Conclusion

- Introduced **Selective Unsharding** for SimpleFSDP
- Trades idle GPU memory for reduced communication
- Achieved **~5.5% QPS improvement** on OmniFM v2
- Flexible: manual + automatic modes
- Provides a *new design dimension*:
  memory → communication → performance

DiT Training Simulator

Shilong Lei, Zhihao Bai (mentor), Yanghua Peng (mentor), Haibin Lin

2024 May – 2024 Aug

Design / Workflow

1. Get model forward graph and backward graph. (PyTorch.fx)
2. Get nodes and ops and corresponding inputs tensor meta (shape, dtype)
3. Run each ops to get each kernel latency.
4. Calculate memory constraints.
5. Construct a timeline which can be viewed by Perfetto.ui & tensorboard.
6. Support: data parallel, pipeline parallel, tensor parallel, sequence parallel.

Contribution

1. Successfully support several DiT video generation models into the simulator.
    1. forward latency / backward latency (95%+ accuracy)

2. Support pytorch.FSDP zero2/3 timeline with pipeline and memory constraints.
    1. AllGather latency / ReduceScatter latency (90%+ accuracy)
    2. FSDP differenct strategy in timeline and memory.
    3. Memory profiling and calculation
    4. Add AG/RS to timeline.

3. WIP: Simulated in large scale (eg. 100 ~ 1k GPUs) to verify.

4. Assisting to supported MoE model and pytorch.FSDP to ByteDance opensource training framework veScale.

OOGC-LLM: An Efficient Out-of-GPU-Core LLM Training System

Shilong Lei, Yikang Yue, Yuxuan Liu, Xuehai Qian

2023 May – 2024 Jan

Background

- ZeRO-Offloading: LLM -> GPU memory -> CPU memory / NVMe

- Current system fails to maximize the value of fast GPU memory and CPU memory. (**C2G and G2C are bottleneck**)

- We want to develop a **more efficient offloading system** for groups with insufficient computing resources

Background

- DeepSpeed's ZeRO3

- Fail to maximize the value of GPU / CPU memory.

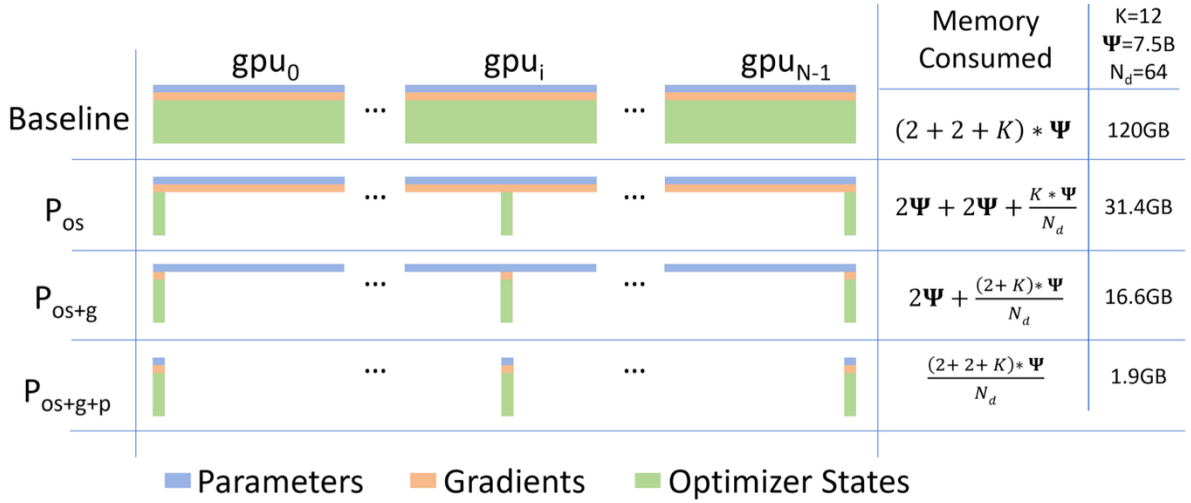


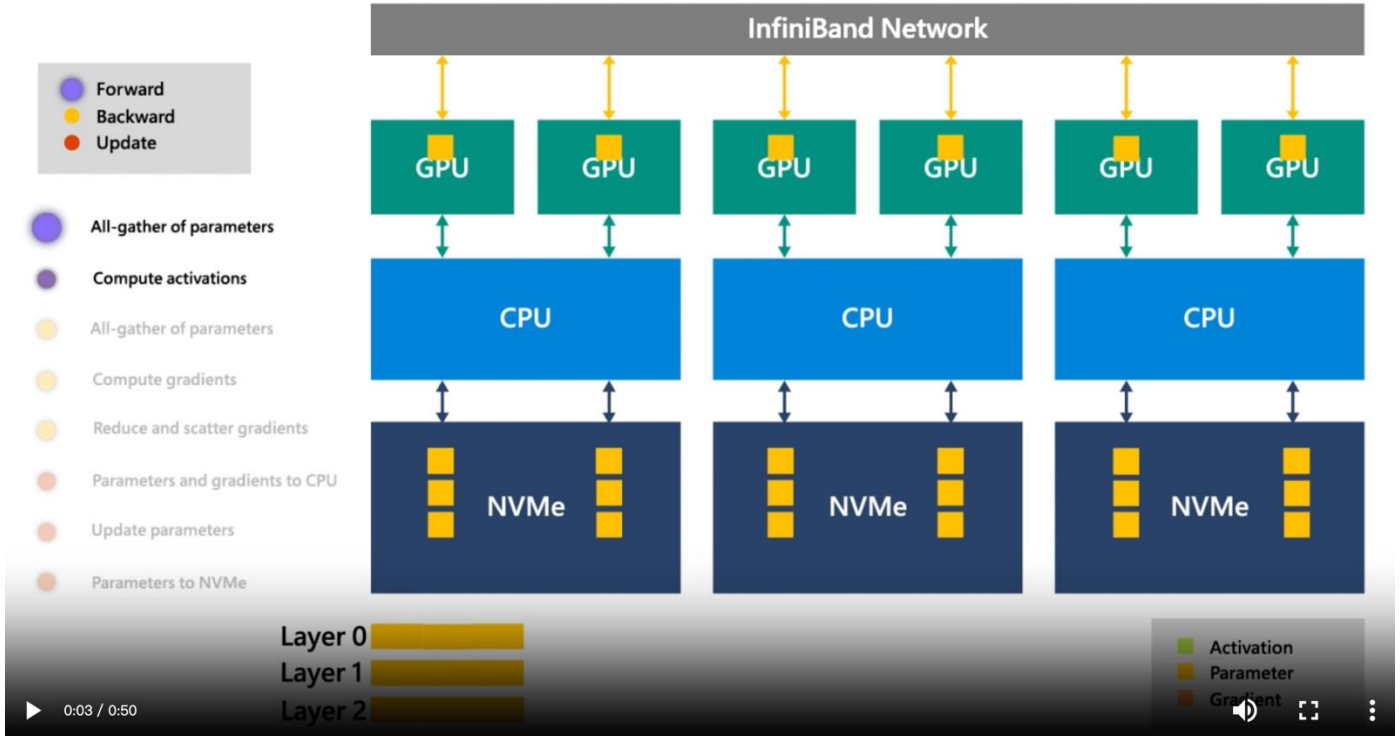
Figure 1: Comparing the per-device memory consumption of model states, with three stages of *ZeRO*-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.

https://www.microsoft.com/en-us/research/blog/zero-infinity-and-deepspeed-unlocking-unprecedented-model-scale-for-deep-learning-training/

Background

- ZeRO Offload

- Fail to maximize the value of GPU / CPU memory.



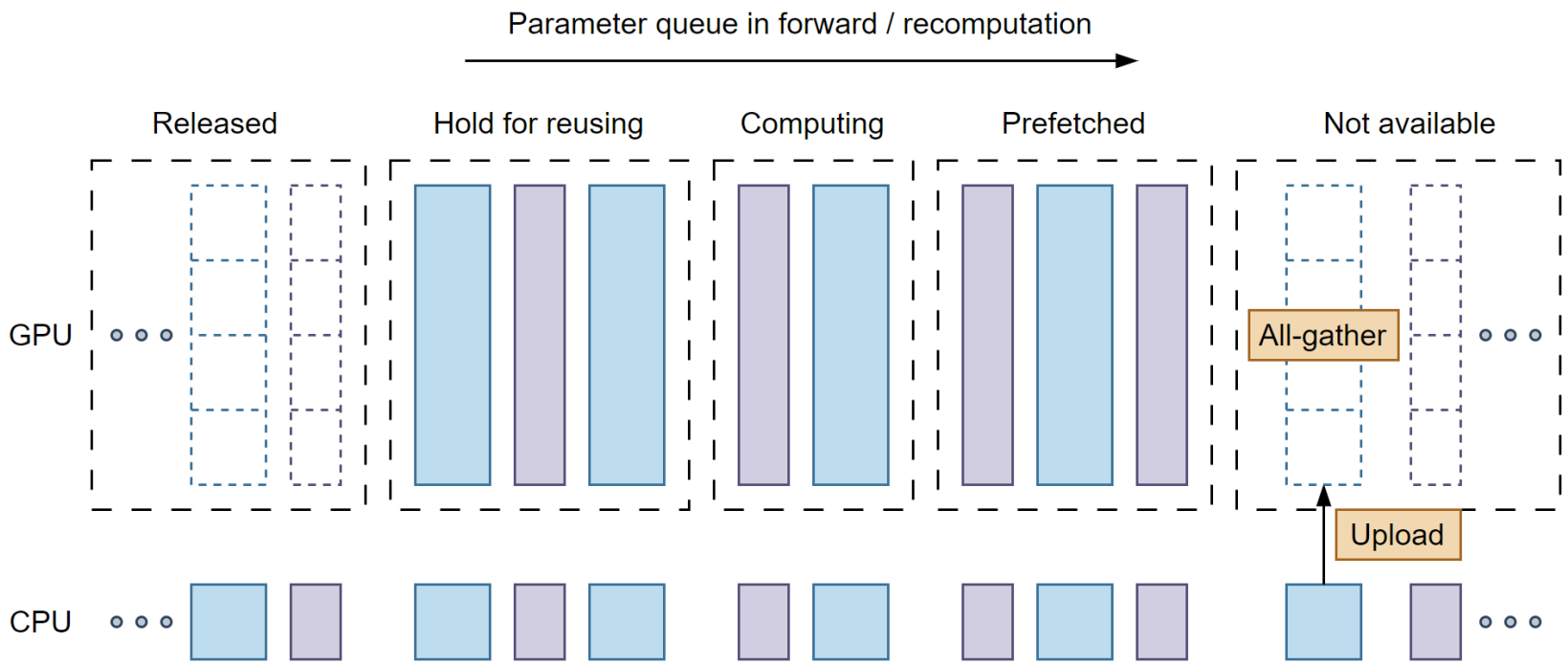
https://www.microsoft.com/en-us/research/blog/zero-infinity-and-deepspeed-unlocking-unprecedented-model-scale-for-deep-learning-training/

Background

- ZeRO Offload Drawbacks:
  - The GPU memory and CPU memory is underutilized.
    - Just a few params reside in GPU memory
  - While the CPU is active, the GPU remains idle, and vice versa.
    - GPU: fwd / bwd CPU: step
  - Does not fully support Mixture-of-Experts (MoE) model
    - Optimization opportunities from imbalance expert

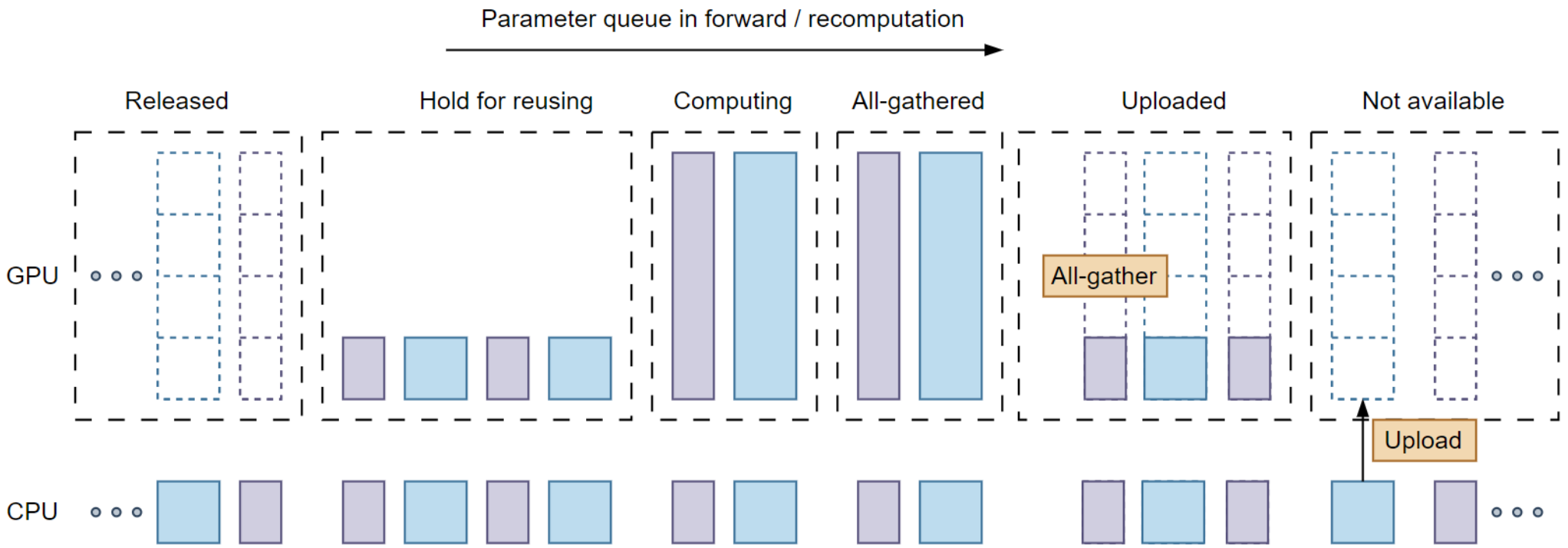
Methodology

- Multi-layer prefetching and lazy all-gather
  - ZeRO:



Methodology

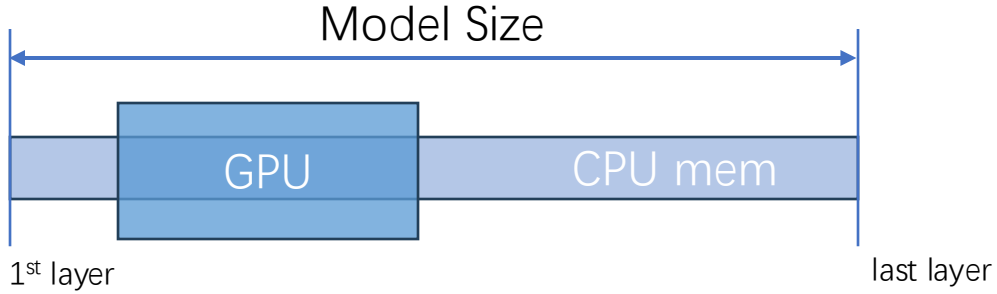
- Multi-layer prefetching and lazy all-gather
  - Ours:



  - Improves GPU memory utilization, reduce C2G / G2C

Methodology

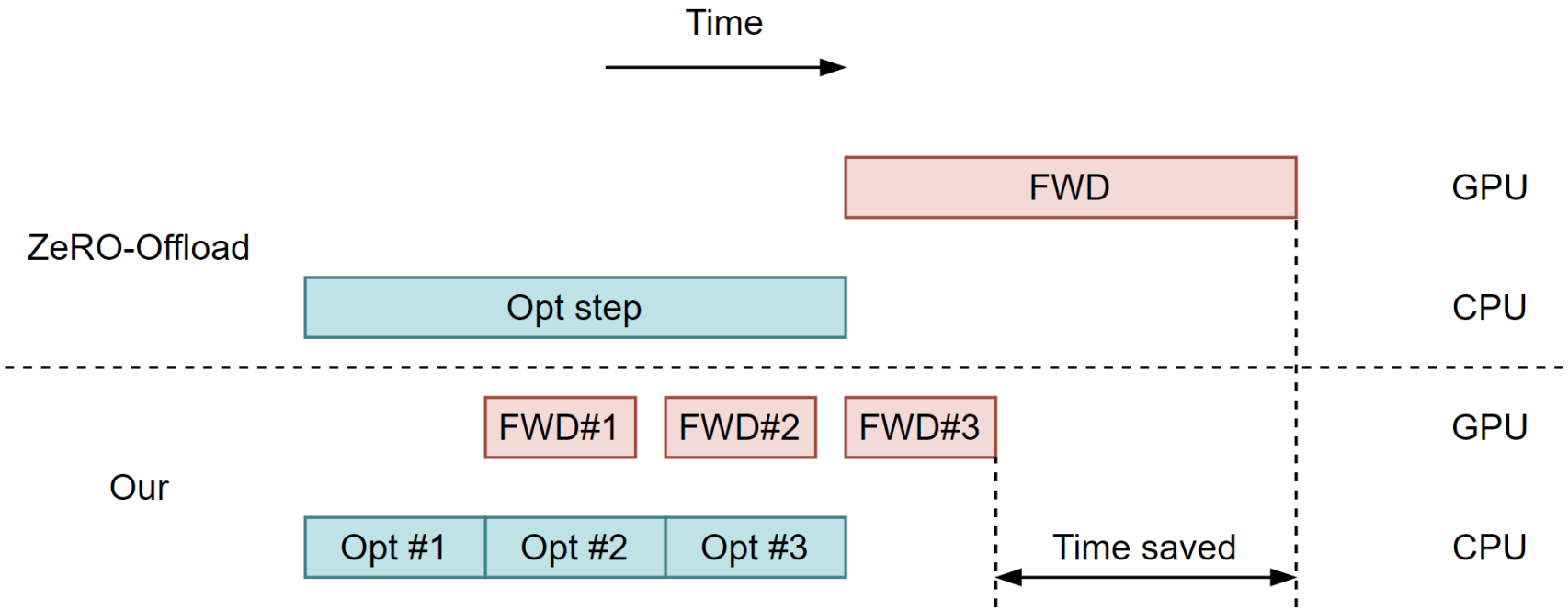
- Memory Management
  - ZeRO:
    - Static: size need to be configured by user otherwise Out of Memory
    - Prefetch **just next** module from CPU memory or NVMe
      - Just a few modules reside in GPU, which makes GPU memory underutilized
  - Ours:
    - **Dynamic**: CUDA swapper (GPU/CPU) + CPU swapper (CPU/NVMe) fits user's memory
    - **Sliding window**



    - **Eviction planner**
      - optimal strategy: Discard the least commonly used parameters
      - optimizer states -> gradients -> parameters

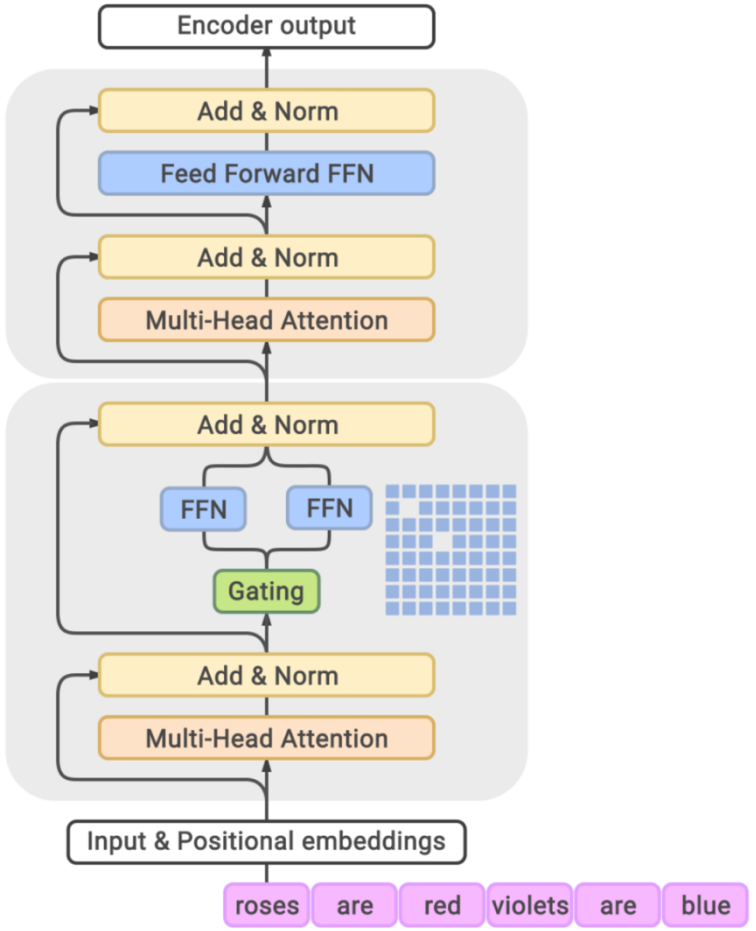
Methodology

- Pipelined optimizer step
  - ZeRO:
    - CPU Step after GPU forward/backward
  - Ours:



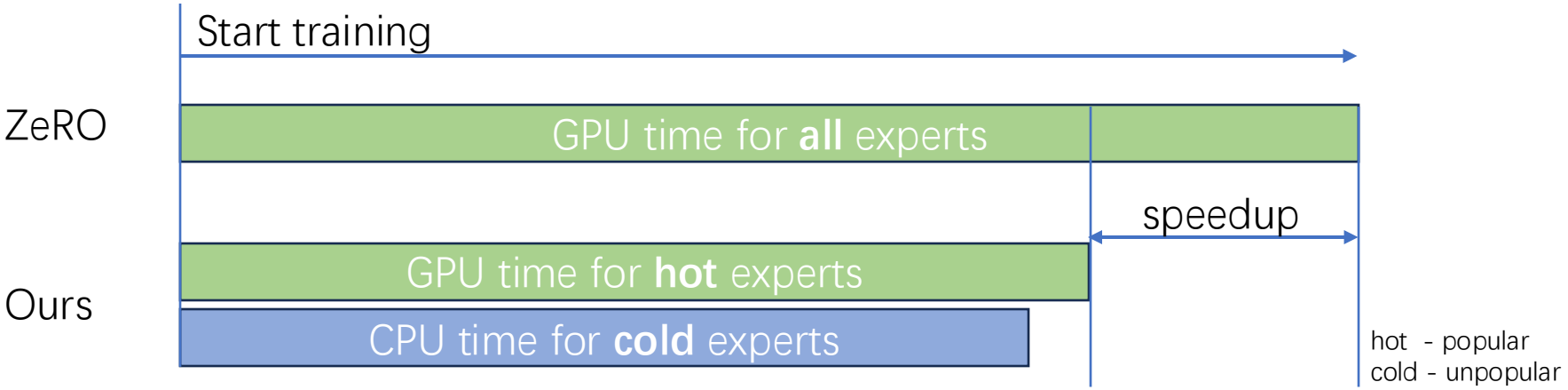
Methodology

- MoE Model
  - Backgroud
    - Unbalanced activated expert models
    - Similar activation **patterns** between iterations
    - Very large model size
  - ZeRO:
    - Does not fully support MoE, especially Parameter offload



Methodology

- MoE Model
  - Ours:
    - Support MoE model
    - Forward / Backward: CPU for unpopular experts / GPU for popular experts
      - Reduce 20% CPU to GPU communication and GPU computation.
      - Intuition: for unpopular experts, C2G + GPU + G2C is slower than CPU.



    - Activation checkpointing: remove unpopular expert recomputation, since CPU memory is large enough.

Evaluation

- Memory management system and asynchronized stepping resulting in a 20% end-to-end speedup.

- MoE part can yield an improvement of around 15% (estimation)
  - Based on optimal prediction.

- All parts weren’t fully integrated. The whole workflow hasn’t been evaluated.

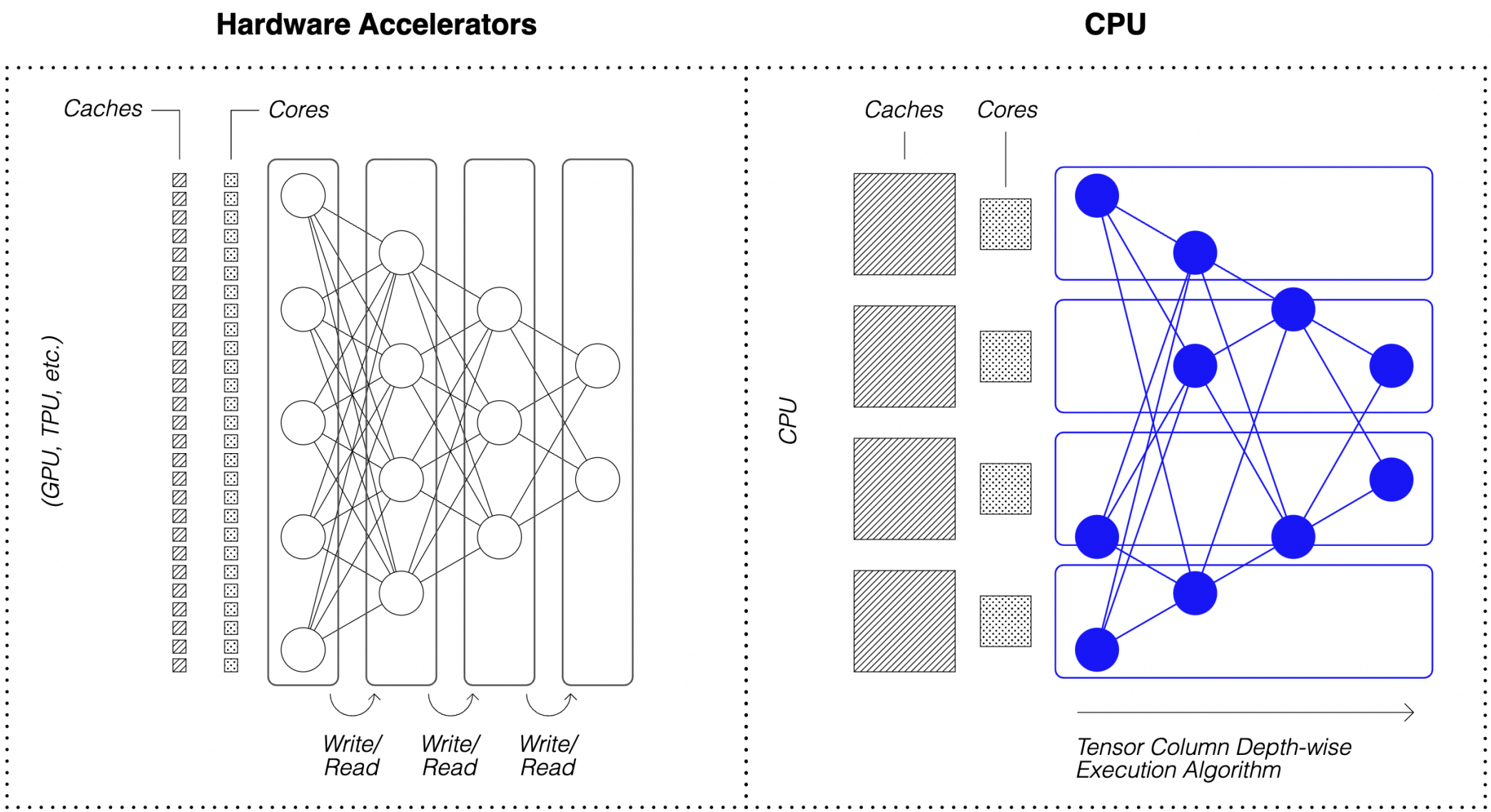
My Role

- Leader of the project. Mentored 2 interns.

- Proposed optimization strategies
  - Multi-layer prefetching and lazy all-gather. - Yikang
  - Memory Management. - Yuxuan
  - Pipelined optimizer step. - Shilong
  - MoE optimization. - Shilong

Inference System for Machine Learning Models on CPU

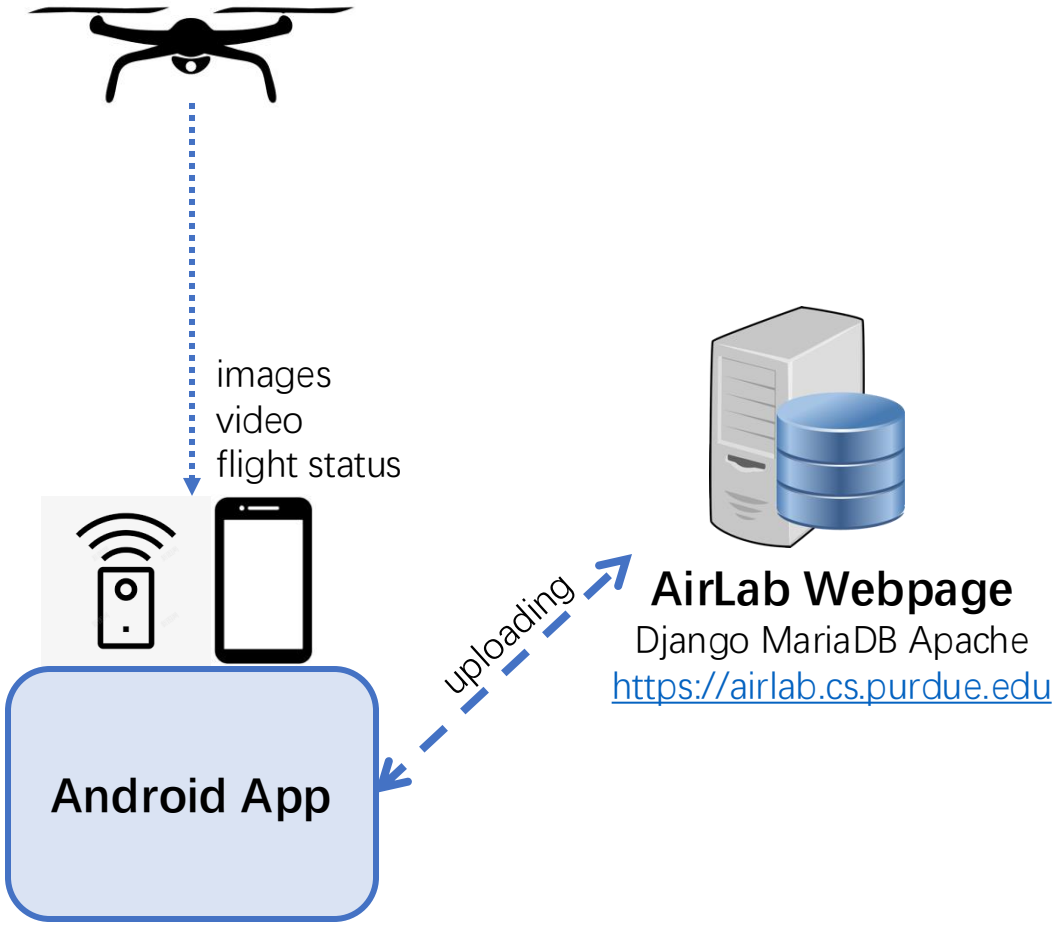
Shilong Lei, Gengyu Rao, Xuehai Qian

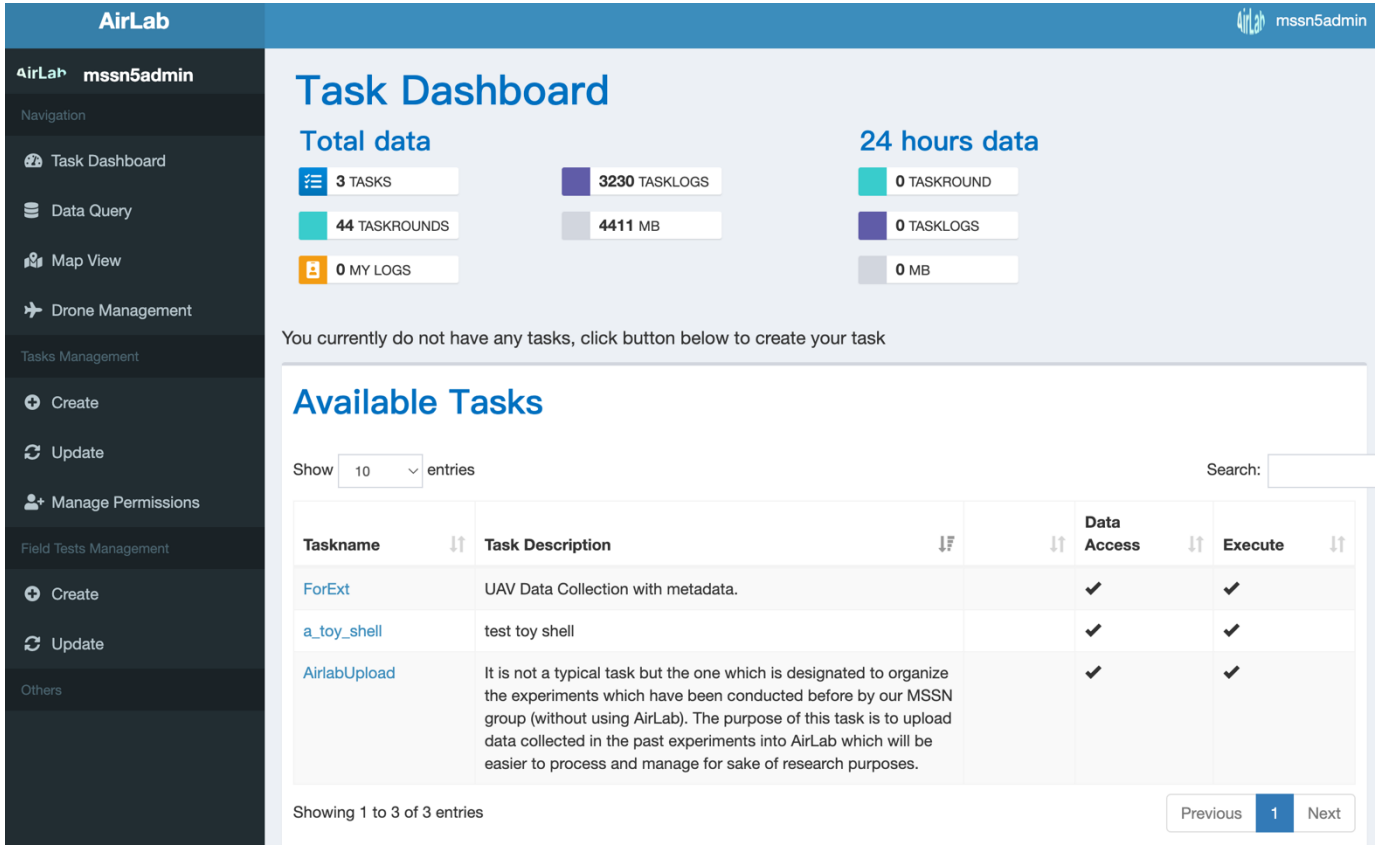

Hardware Accelerators
CPU
Caches
Cores
(GPU, TPU, etc.)
Write/
Read
Write/
Read
Write/
Read
Caches
Cores
CPU
Tensor Column Depth-wise
Execution Algorithm

AirLab Platform

Shilong Lei, Jiaxin Du, Chunyi Peng

Framework


images
video
flight status
uploading
AirLab Webpage
Django MariaDB Apache
https://airlab.cs.purdue.edu
Android App


AirLab
mssn5admin
Navigation
Task Dashboard
Data Query
Map View
Drone Management
Tasks Management
Create
Update
Manage Permissions
Field Tests Management
Create
Update
Others
mssn5admin
Task Dashboard
Total data
3 TASKS
3230 TASKLOGS
44 TASKROUNDS
4411 MB
0 MY LOGS
24 hours data
0 TASKROUND
0 TASKLOGS
0 MB
You currently do not have any tasks, click button below to create your task
Available Tasks
Show 10 entries
Search:
Taskname
Task Description
Data Access
Execute
ForExt
UAV Data Collection with metadata.
a_toy_shell
test toy shell
AirlabUpload
It is not a typical task but the one which is designated to organize the experiments which have been conducted before by our MSSN group (without using AirLab). The purpose of this task is to upload data collected in the past experiments into AirLab which will be easier to process and manage for sake of research purposes.
Showing 1 to 3 of 3 entries
Previous
1
Next

Framework

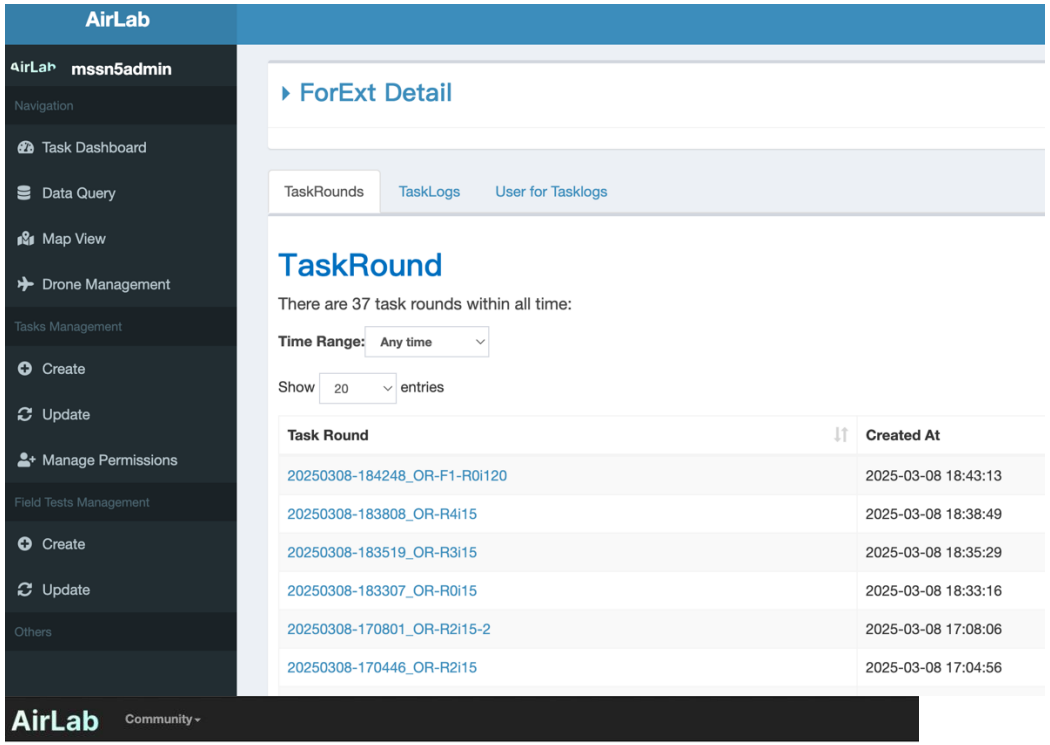
- The raw flight status will be parsed and processed into DB.
  - Can be viewed in map view.

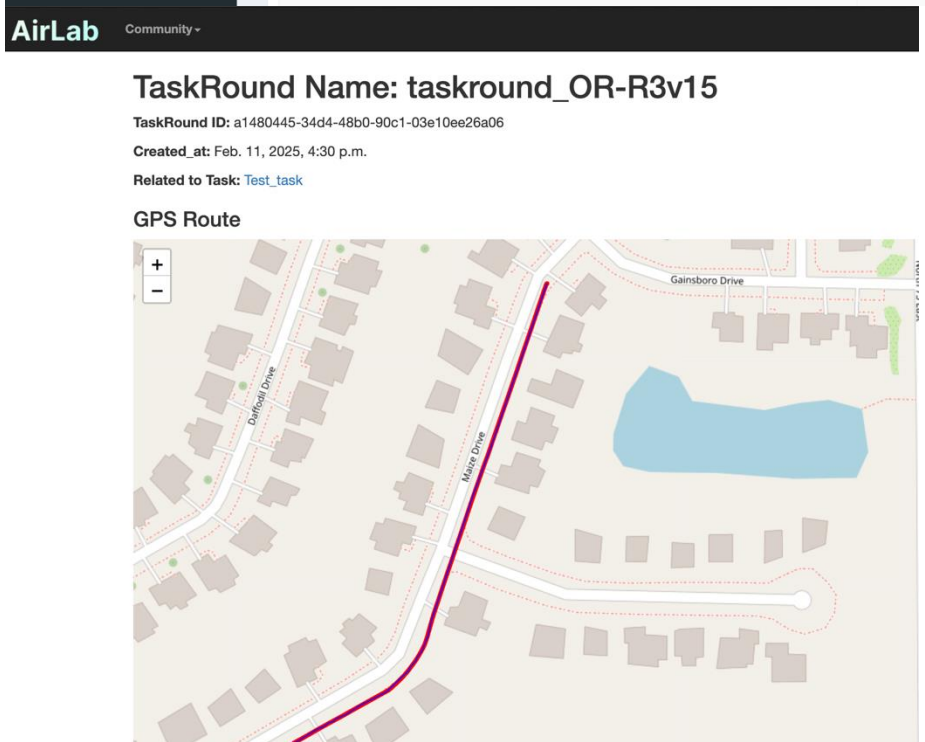
- Data Management:
  - Task
    - Taskround
      - Tasklogs

- Drones Management
  - Add/edit drones; Status; User;
  - Checkout/Expected return time

- Field Tests Management
  - Start time & end time; Drones; User

- WIP:
  - Labeling
  - Training
  - Inferencing





D-Air-Patrol Project

(**mobiCom'24 Best Poster Award**)

Jiaxin Du, Shilong Lei, Chunyi Peng

2024 Jan – 2024 May

Framework

1. A drone controlled by phone app monitoring the road traffic.
2. Frames are transmitted to an edge server for speed detection.
3. Motion detection is enhanced by 2 layers method: base layer + motion layer.

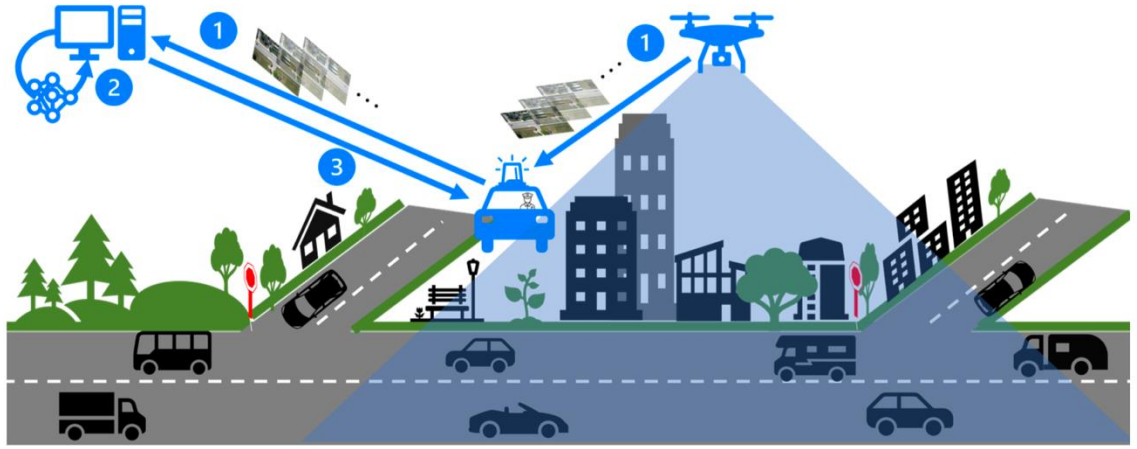


Figure 1: Illustration of air patrol using drones to monitor traffic and detect traffic law violations from the sky.

Design

1. Base Layer: U-Net semantic segmentation to locate static road region.
2. Motion Layer: tracking vehicles within the road region rather than processing the entire image.
    1. YOLOv8
    2. Optical Flow
    3. ByteTrack (enhance continuous tracking)

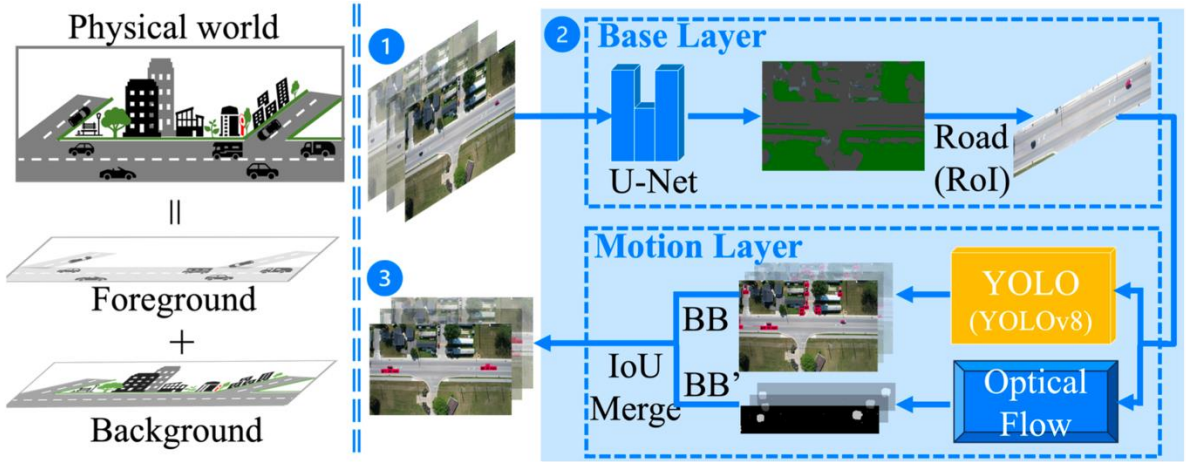


Figure 2: Overview of D-AirPatrol (BB: bounding box).

Evaluation

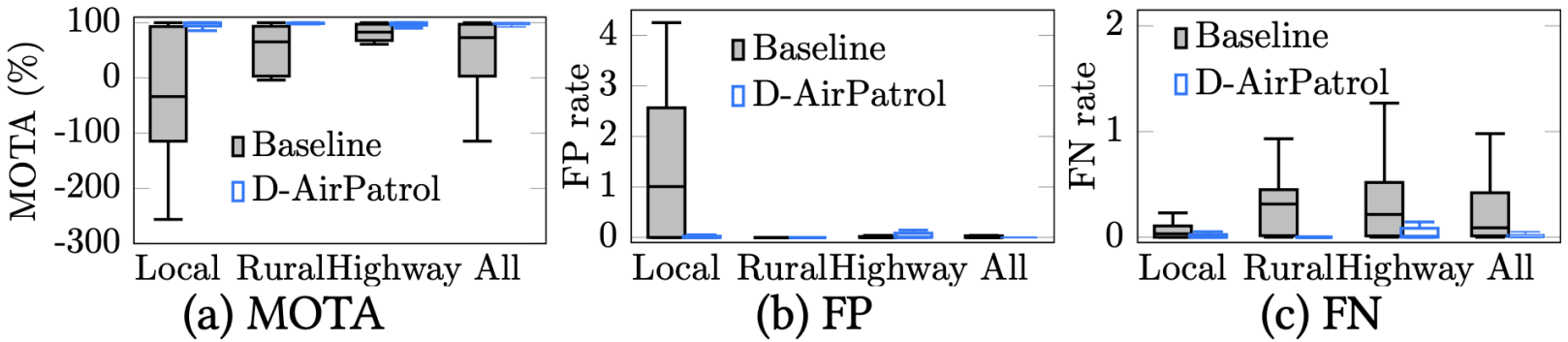


Figure 3: Evaluation over three types of roads.