FuncPipe: A Pipelined Serverless Framework

For Fast and Cost-Efficient Training of Deep Learning Models

Authors & Affiliations: YUNZHUO LIU, Shanghai Jiao Tong University, China; BO JIANG, Shanghai Jiao Tong University, China; TIAN GUO, Worcester Polytechnic Institute, U.S.; ZIMENG HUANG, Shanghai Jiao Tong University, China; WENHAO MA, Shanghai Jiao Tong University, China; XINBING WANG, Shanghai Jiao Tong University, China; CHENGHU ZHOU, Chinese Academy of Sciences, China

Presenter: Muhan Zhang

Background & Motivation

Serverless Computing Benefits

- True pay-as-you-go pricing
- No infrastructure management
- High scalability
- Quick deployment

Current Challenges

- Memory limits (10GB in AWS Lambda)
- Limited bandwidth (~70 MB/s)
- No direct inter-function communication
- Resource configuration complexity

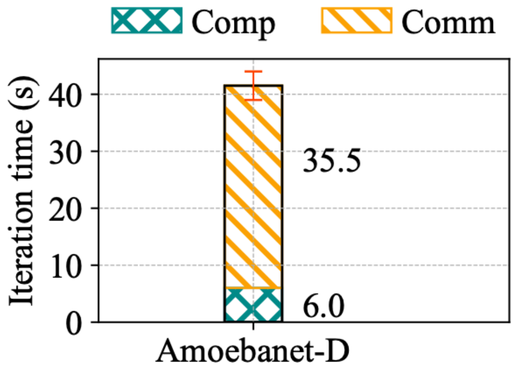
Problem Quantification

Case Study

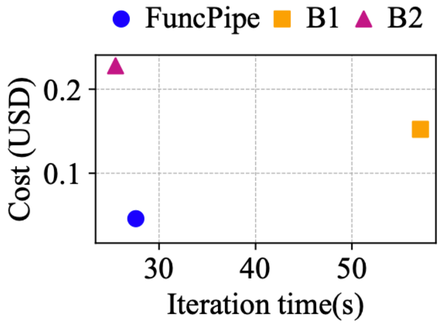
- AmoebaNet-D (900MB model) training metrics
- Computation: 6s/iteration
- Communication: 35.5s/iteration (6x slower!)

Key Issues

- Excessive communication overhead
- Memory constraints limiting batch sizes
- Low resource utilization
- Inefficient existing solutions



(a) LambdaML performance.



(b) Training with three configurations.

FuncPipe Overview

Key Innovations:

1. Model Partitioning Strategy

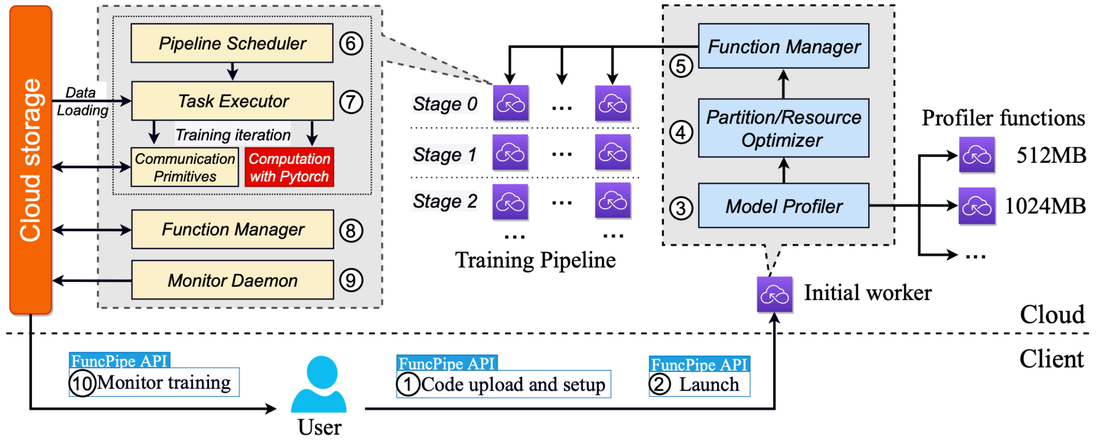
- Memory constraint solution
- Communication pattern optimization

2. Pipeline Parallelism

- Resource utilization improvement
- Idle time reduction

3. Pipelined Scatter-Reduce

- Bi-directional bandwidth utilization
- Synchronization overhead reduction



4. Joint Optimization Framework

- Automatic partitioning
- Dynamic resource allocation

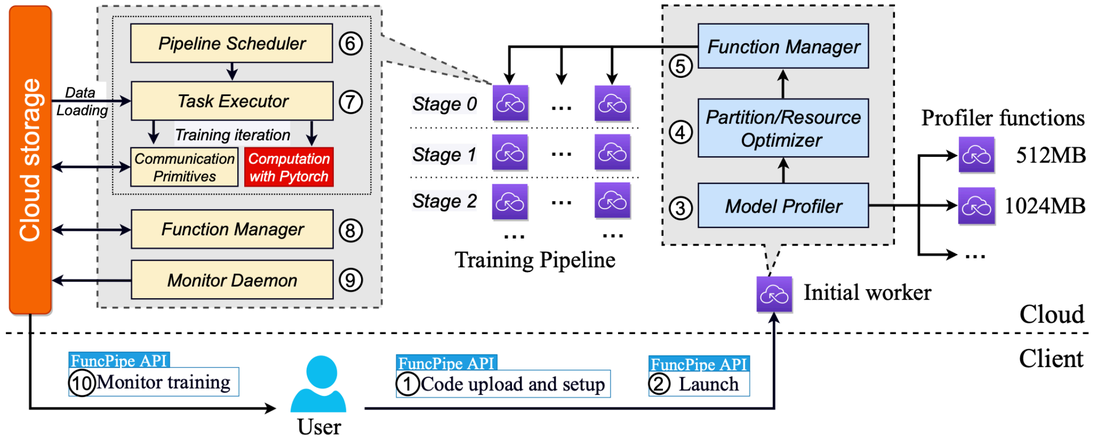
System Architecture

Client-side APIs

- User interface
- Configuration management
- Monitoring capabilities

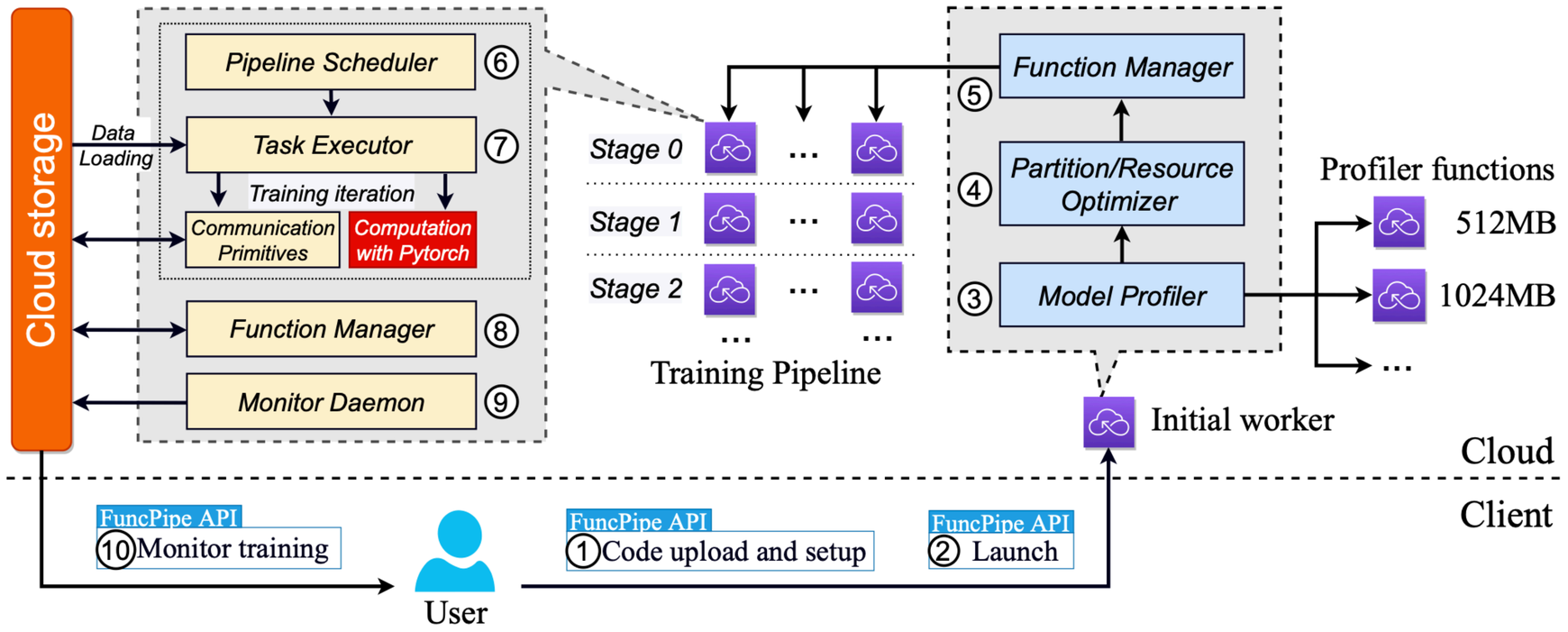
Startup Components

- Model Profiler
- Partition/Resource Optimizer
- Function Manager



Runtime Components

- Pipeline Scheduler
- Task Executor
- Communication Primitives



FuncPipe system architecture and workflow.

Pipeline Design

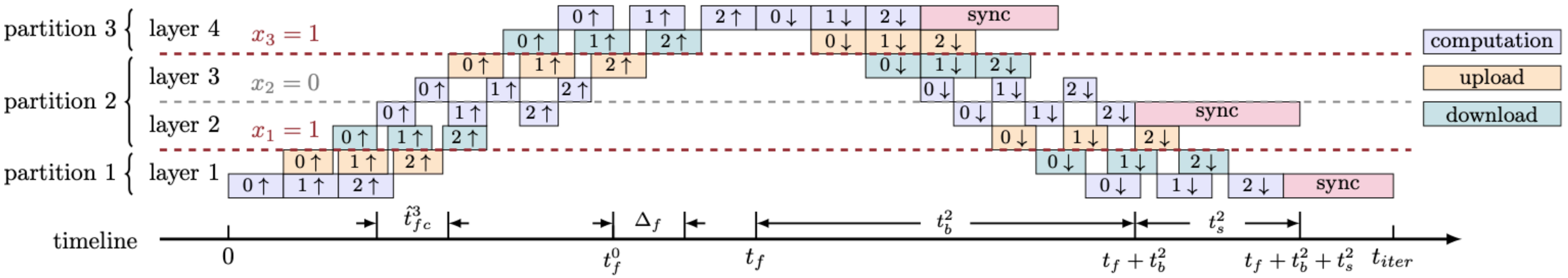
GPipe-based Pipeline:

- Micro-batch processing
- Forward/Backward phases
- Synchronous training

FuncPipe Extensions:

✦ Communication as Pipeline Stage

✦ Computation-Communication Overlap



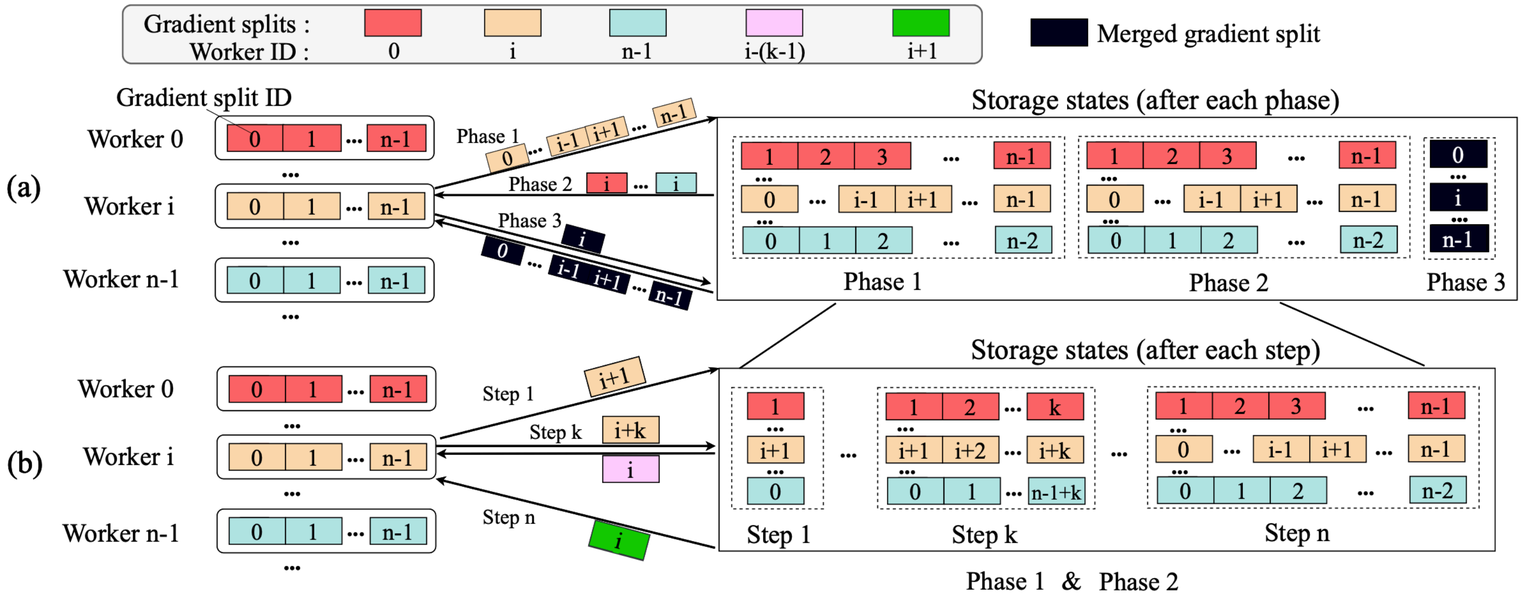
Scatter-Reduce Optimization

Traditional Approach

- Serial communication
- Low bandwidth utilization
- High synchronization overhead

FuncPipe Optimization

- Bi-directional bandwidth usage
- Pipelined design
- Efficient synchronization



Example of Traditional Approach

Phase 1:

Worker 0:

- Splits gradients into {G0_0, G0_1, G0_2}

- Uploads G0_1, G0_2 to storage

Worker 1:

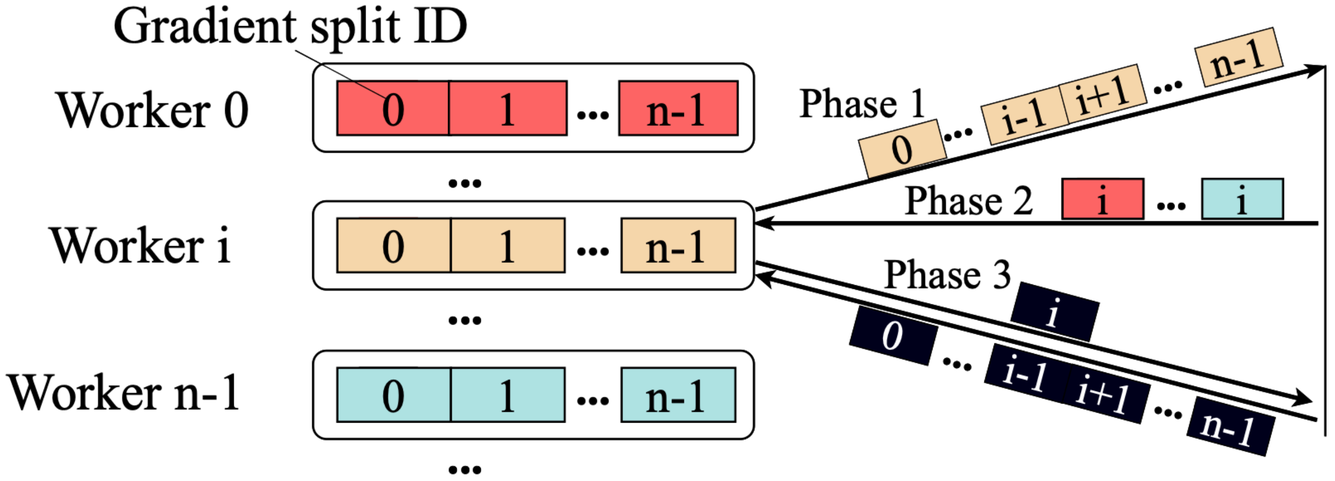
- Splits gradients into {G1_0, G1_1, G1_2}

- Uploads G1_0, G1_2 to storage

Worker 2:

- Splits gradients into {G2_0, G2_1, G2_2}

- Uploads G2_0, G2_1 to storage



Example of Traditional Approach

Phase 2:

Worker 0:

- Upload split 2

- Simultaneously download split 0 (uploaded by Worker 2)

Worker 1:

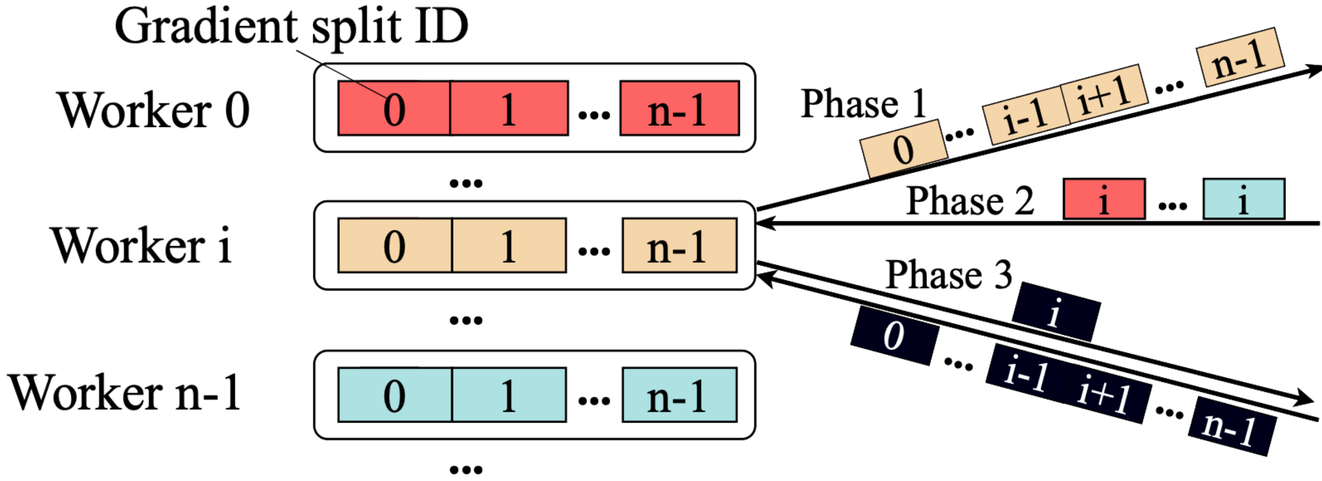
- Upload split 0

- Simultaneously download split 1 (uploaded by Worker 0)

Worker 2:

- Upload split 1

- Simultaneously download split 2 (uploaded by Worker 1)



Example of Traditional Approach

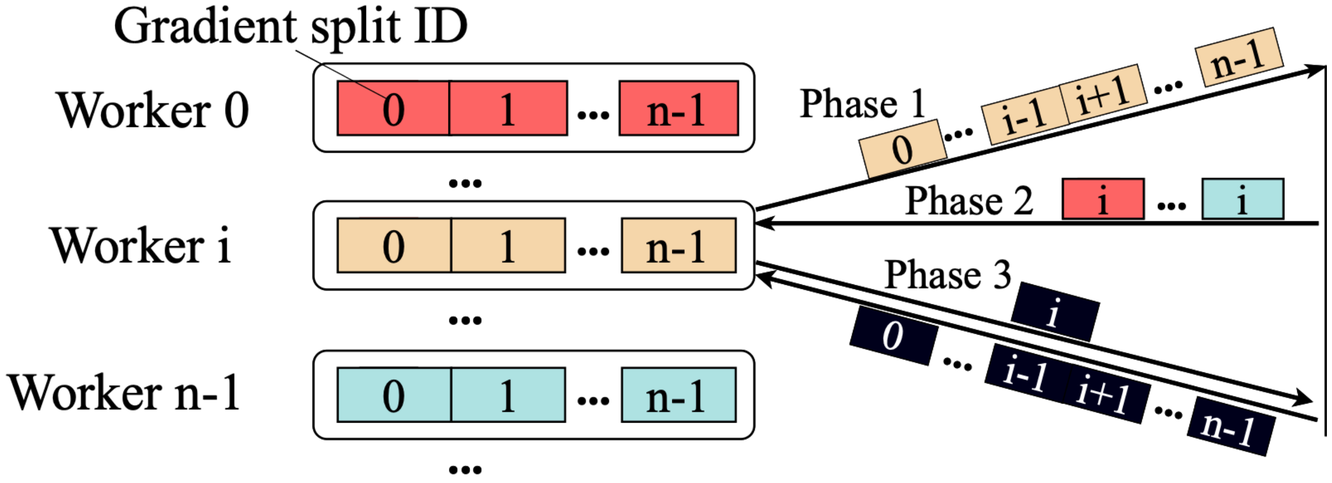
Phase 3:

All workers:

- Upload their merged results (M0/M1/M2)

- Download merged results from other workers

- Finally each worker has complete merged gradients {M0, M1, M2}



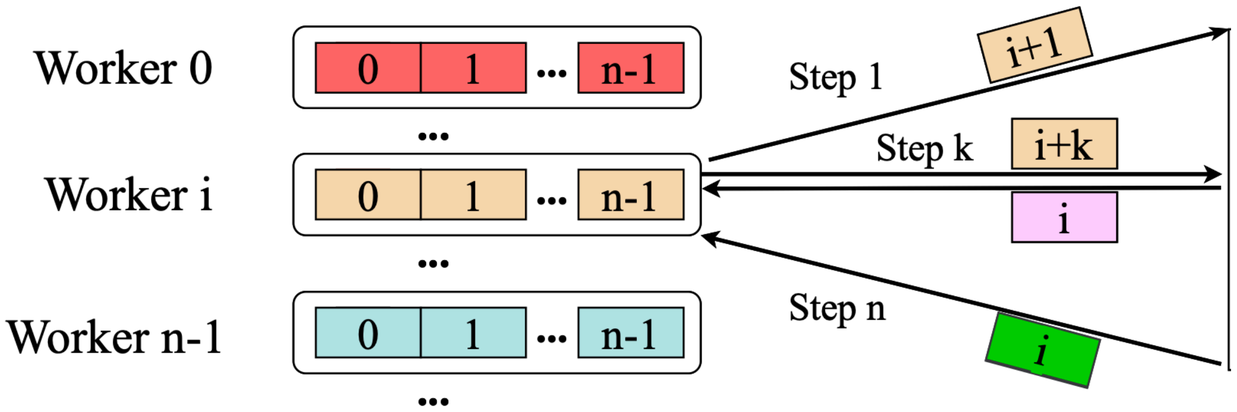
Example of FuncPipe Optimization

Step 1:

Worker 0: Upload split 1 to storage

Worker 1: Upload split 2 to storage

Worker 2: Upload split 0 to storage



Example of FuncPipe Optimization

Step 2:

Worker 0:

- Upload split 2

- Simultaneously download split 0 (uploaded by Worker 2)

Worker 1:

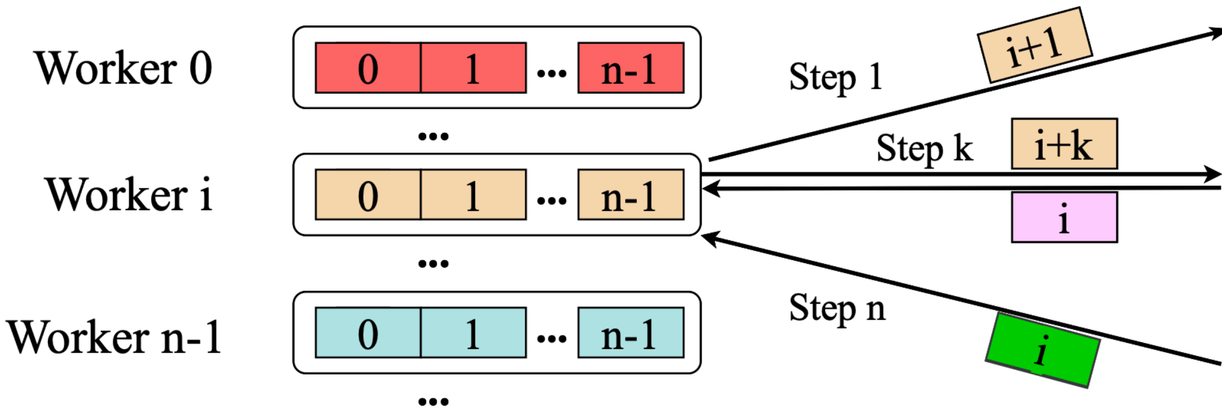
- Upload split 0

- Simultaneously download split 1 (uploaded by Worker 0)

Worker 2:

- Upload split 1

- Simultaneously download split 2 (uploaded by Worker 1)



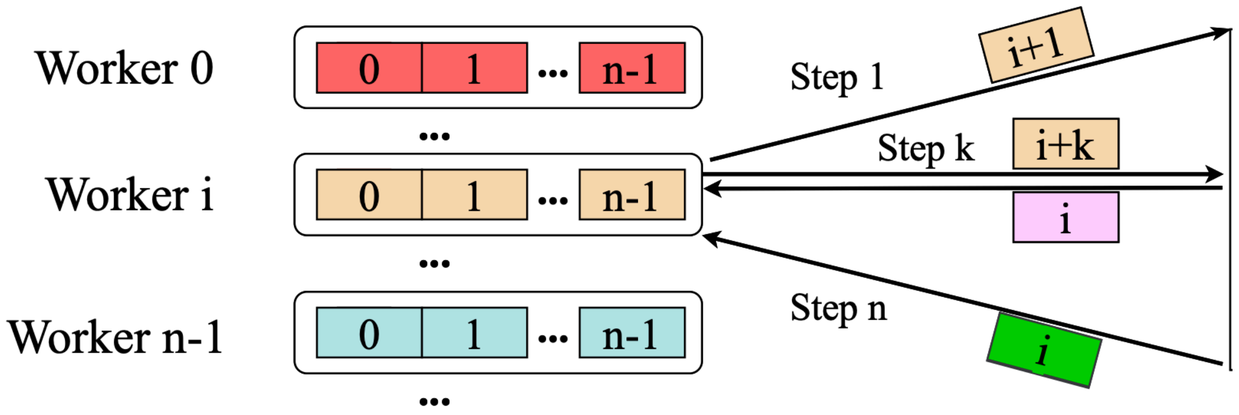
Example of FuncPipe Optimization

Step 3:

Worker 0: Download split 0 (uploaded by Worker 1)

Worker 1: Download split 1 (uploaded by Worker 2)

Worker 2: Download split 2 (uploaded by Worker 0)



Co-optimization in FuncPipe

Challenge & Motivation

- Individual optimization is suboptimal
- Model partition affects resource needs
- Resource allocation impacts performance
- Strong coupling between decisions

Key Components

- Model Partition
  - Split DL model into stages
  - Assign layers to workers
  - Balance computation load
- Resource Allocation
  - Memory per worker
  - Data parallelism degree
  - Worker configurations

Co-optimization in FuncPipe

Optimization Problem

Objective:

- $\min(\alpha_1 \cdot c\_iter + \alpha_2 \cdot t\_iter)$

  where:

  c_iter = cost per iteration

  t_iter = time per iteration

  α_1, α_2 = trade-off weights

Key Constraints:

- Memory usage ≤ allocation
- Consistent resources within partition
- Valid configurations

Co-optimization in FuncPipe

Solution Method

MIQP Transformation

- Convert to Mixed Integer Quadratic Programming
- Linearize non-linear constraints
- Use layer merging for scalability

Performance Model

- Provides cost estimates
- Guides optimization
- < 12% prediction error

Experimental Setup

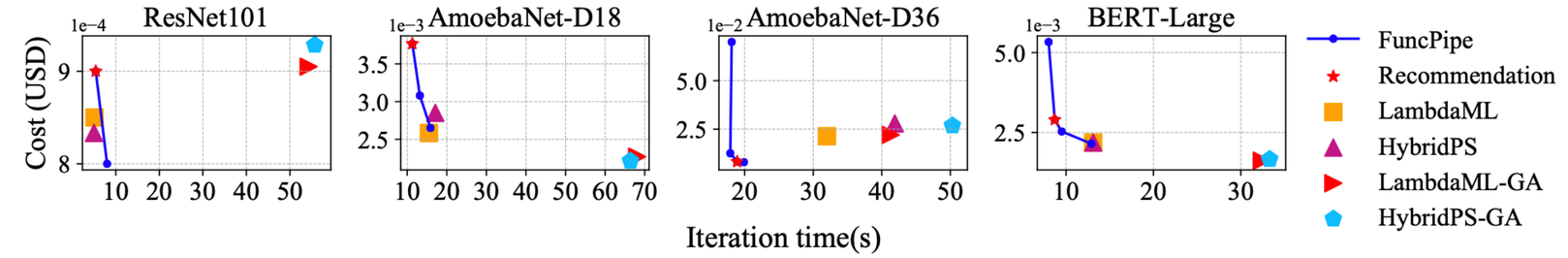
Cloud Serverless Testbed

- AWS Lambda (max 10GB memory)
- Alibaba Cloud Function Compute (max 32GB memory)
- Most evaluations on AWS Lambda

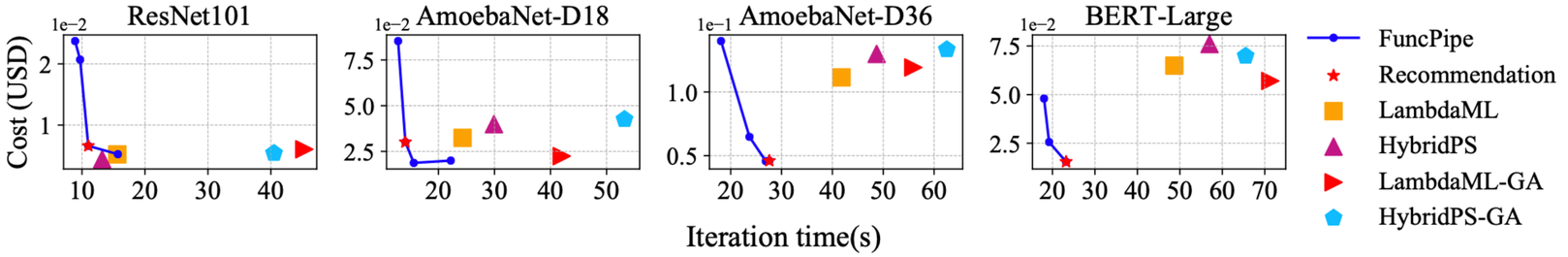
Models & Datasets

- Models: ResNet101, AmoebaNet-D18, AmoebaNet-D36, BERT-Large
- Datasets: CIFAR-10 (CNN models), Wikitext-2 (BERT)
- Training metrics: iteration time and cost

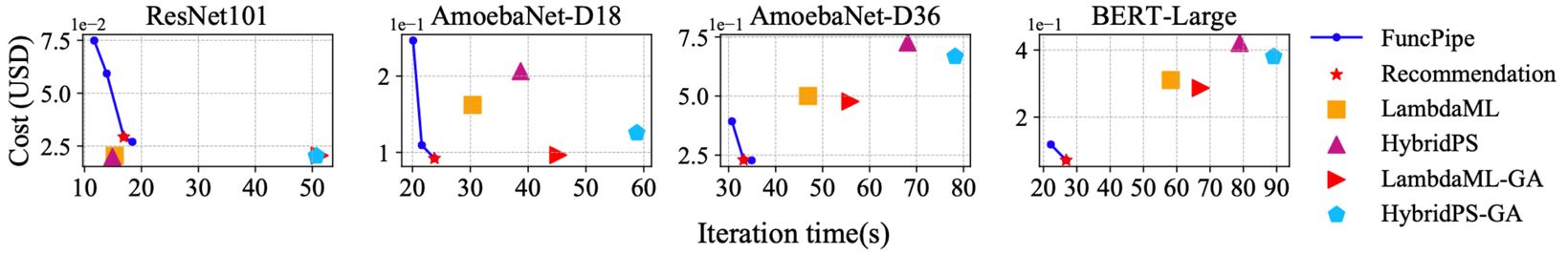
Experimental Results



(a) Batch size = 16



(b) Batch size = 64



(c) Batch size = 256

Overall Performance

Key Results:

- 1.3X-2.2X speedup
- 7%-77% cost reduction
- Better scaling with model size

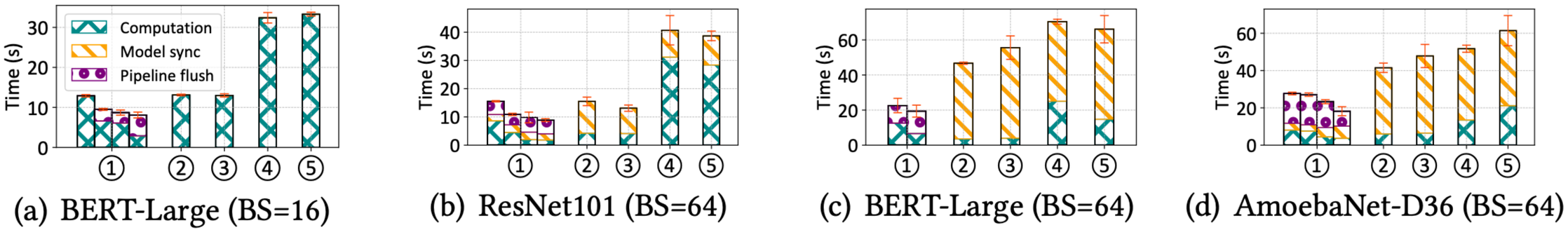
Models Tested:

- - ResNet101
- - AmoebaNet-D
- - BERT-Large

Time Breakdown Analysis

Key Findings:

- Communication overhead significantly reduced
- Better computation-to-communication ratio
- Pipeline effectively hides communication latency



Training time breakdown.

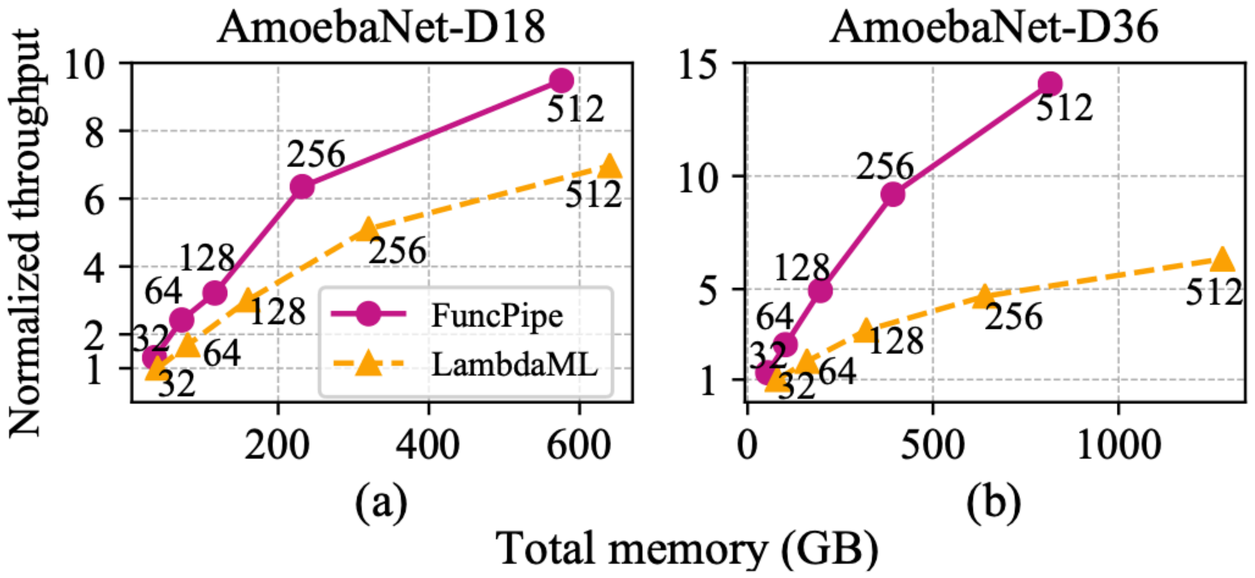
System Scalability

Experiments:

- Varied total memory allocation
- Different batch sizes
- Multiple worker configurations

Results:

- Better scaling than existing solutions
- 180% higher throughput at 800GB total memory
- More robust to bandwidth contention

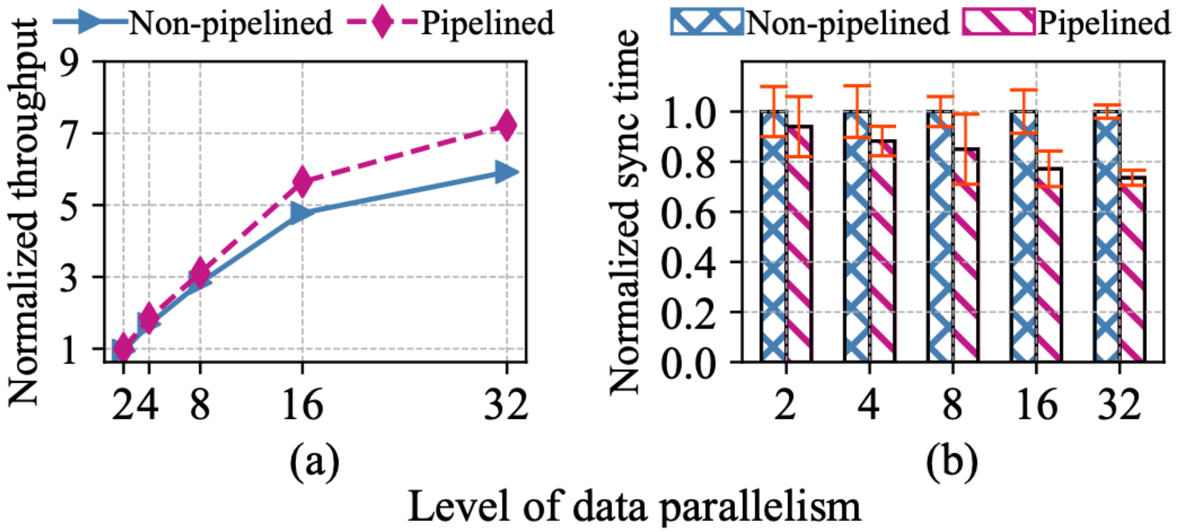


System scalability test.

Communication Optimization

Pipelined Scatter-Reduce Results:

- 2%-22% higher training throughput
- 6%-26% lower synchronization time
- Benefits increase with data parallelism level



Performance of our pipelined scatter-reduce method.

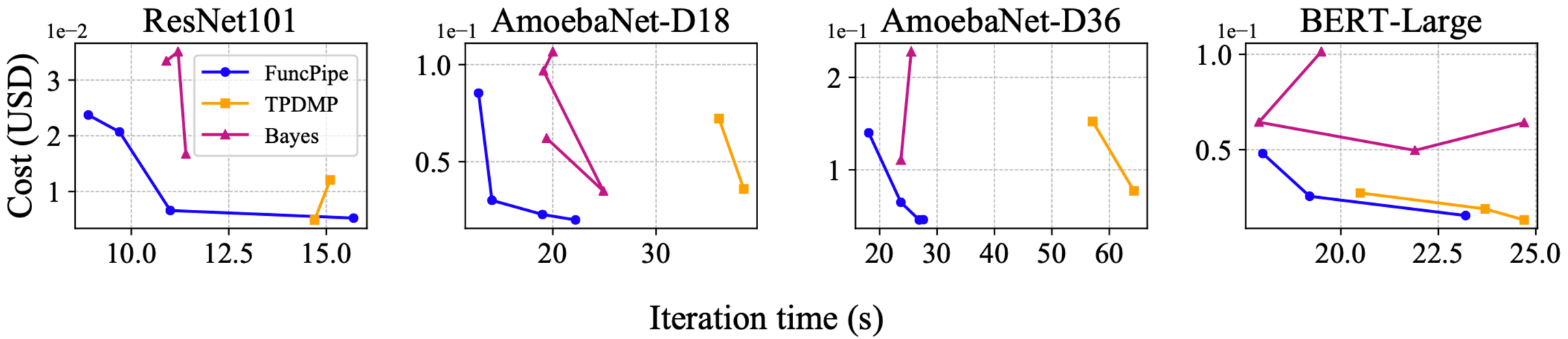
Co-optimization Performance

Comparison with:

- TPDMP: Graph-based model partition
- Bayes: Black-box optimization

Results:

- 7% higher training speed
- 55% lower average cost
- Reasonable solution time (minutes)

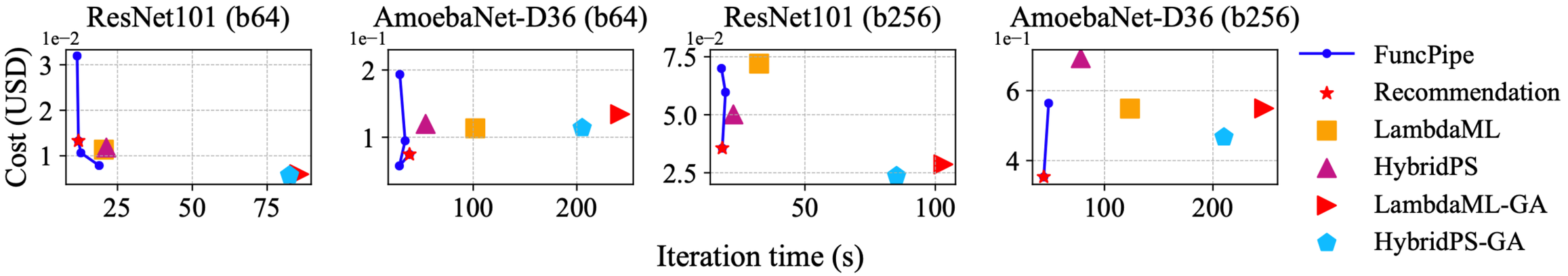


Co-optimization performance evaluation.

Impact of Resource Availability

Cross-Platform Performance:

- Similar benefits on Alibaba Cloud
- Up to 1.8X speedup
- 49% cost reduction
- Effective under bandwidth limits

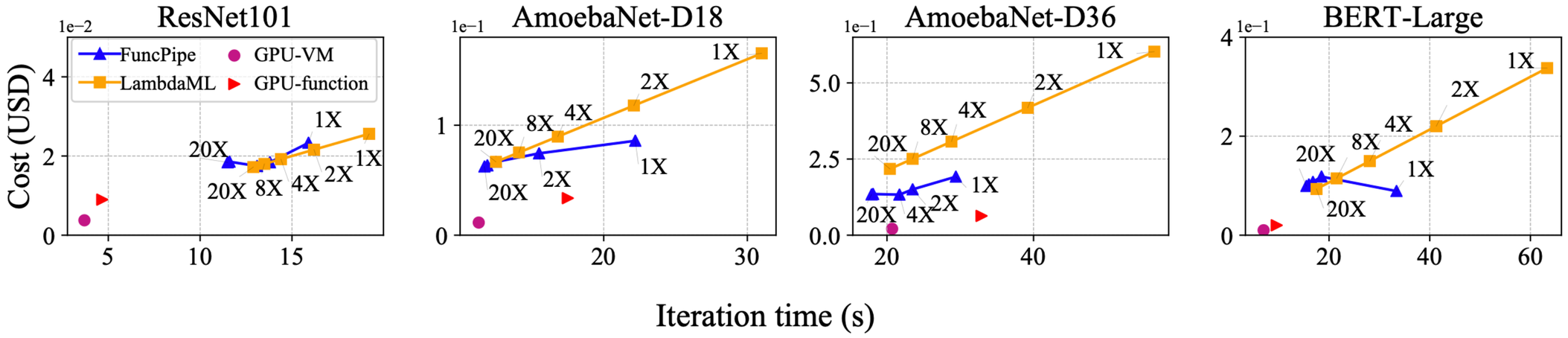


Performance on Alibaba Cloud.

Network Bandwidth Impact

Study with Increased Bandwidth:

- Simulated up to 20x current bandwidth
- Maintains performance advantages
- More robust to different network settings
- Cost benefits persist even with high bandwidth



Iteration time and cost with the increase of network bandwidth.

Limitations & Future Work

Current Limitations:

- Cannot handle layers exceeding max serverless memory
- Performance interference in multi-worker scenarios

Future Directions:

- Tensor parallelism support
- VM-based storage integration
- GPU function support when available

Conclusion

Key Contributions:

- Novel pipelined serverless training framework
- Efficient communication design
- Co-optimization approach
- Comprehensive evaluation on real platforms