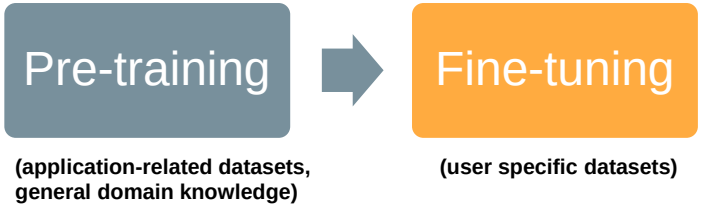
Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism

Saar Eliad, Ido Hakimi, Alon De Jager, Mark Silberstein, Assaf Schuster

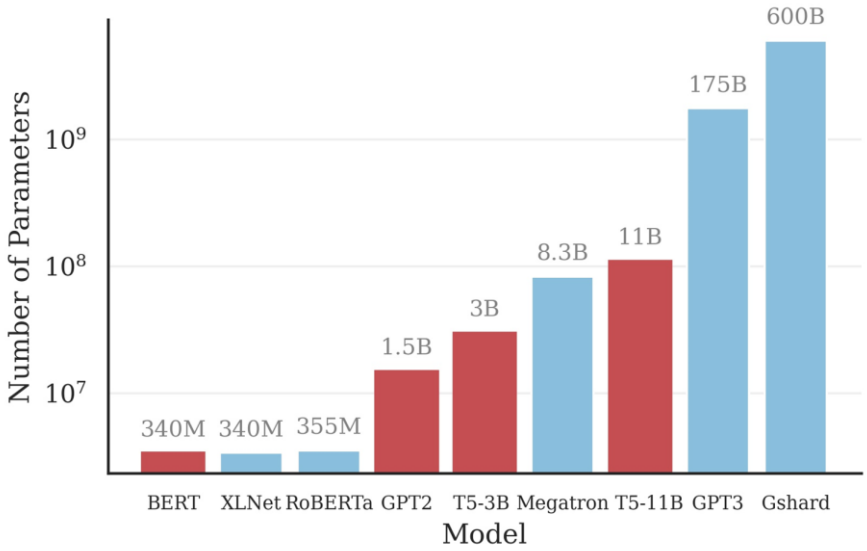
USENIX ATC21

Presented by Saar Eliad

Background and Motivation



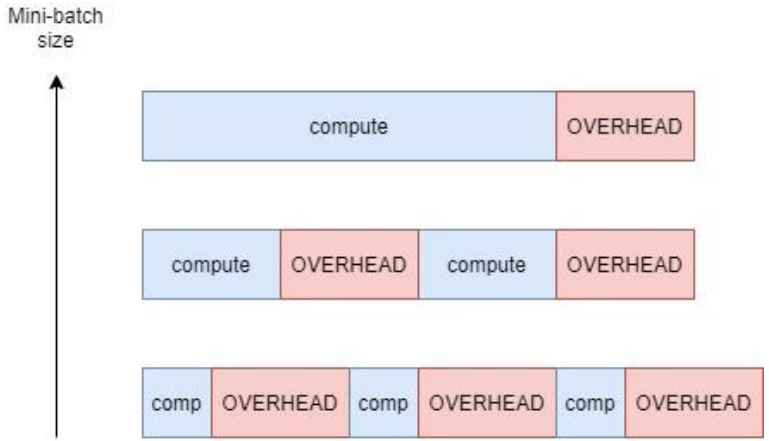
- Fine-tuning is becoming more prevalent
- DNNs are getting larger
- Huge memory demands



No framework for efficient fine-tuning of giant DNNs on commodity GPUs

Memory reduction methods have high overheads

- Swapping or Sharding
- Fine-tuning scenario: even higher overheads
- **Small batch size => harder to hide the overheads**



Example from T5 (Colin et al.)

	Pre-training	**Fine-tuning**
Minibatch size	2048	8 (most tasks), 128

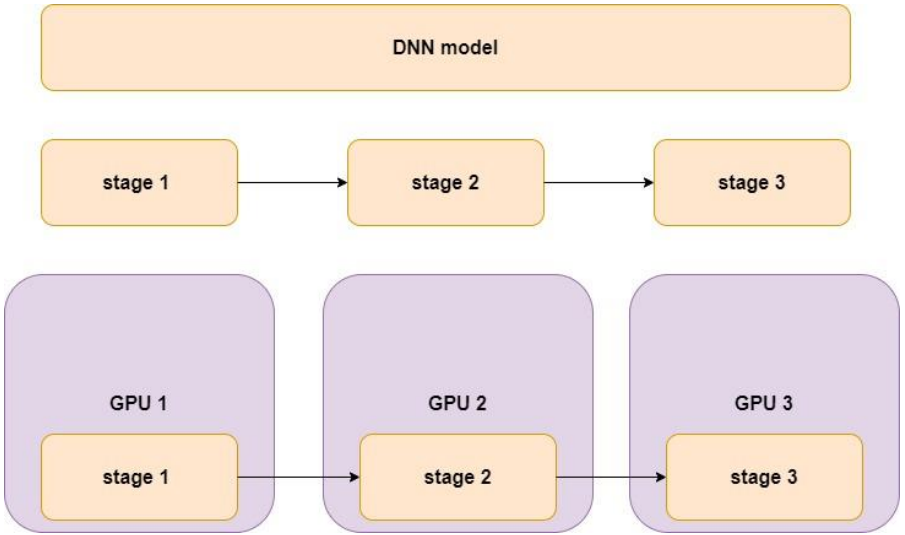
Pipeline model parallelism is promising!

- Splits models layer by layer across GPUs
- Low communication volume for giant models
- Enables communication-computation overlap

Example of network I/O for T5-3B:

- Data-parallel: 12GB ***per worker***.
- Pipeline-parallel: 458MB ***total***. (and full overlap)

(minibatch 4, full-precision)

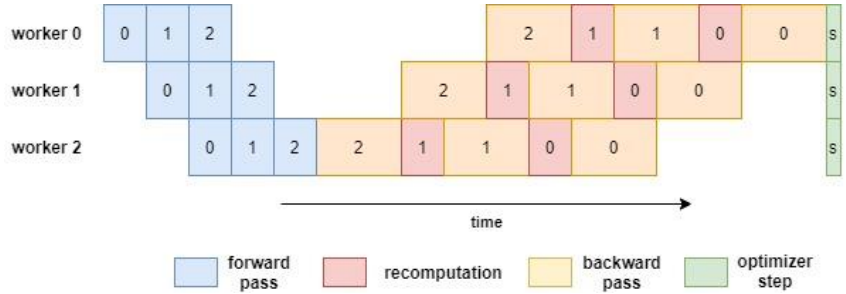


State-of-the-art pipeline frameworks

GPipe: synchronous

Parallelization across Micro-batches (bubbles)

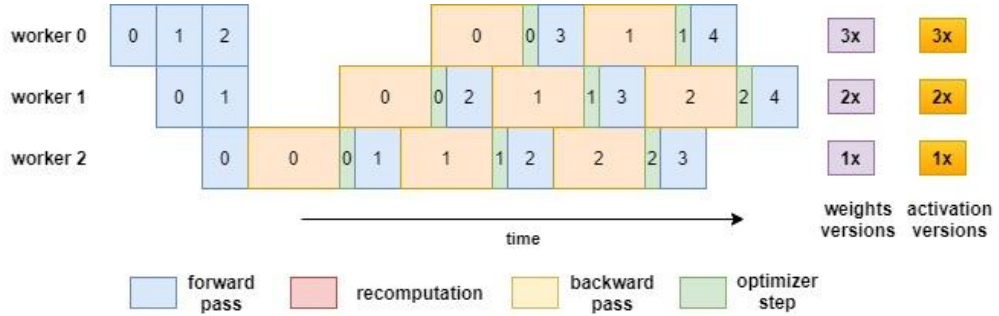
Checkpointing/Recomputation allows Memory reduction



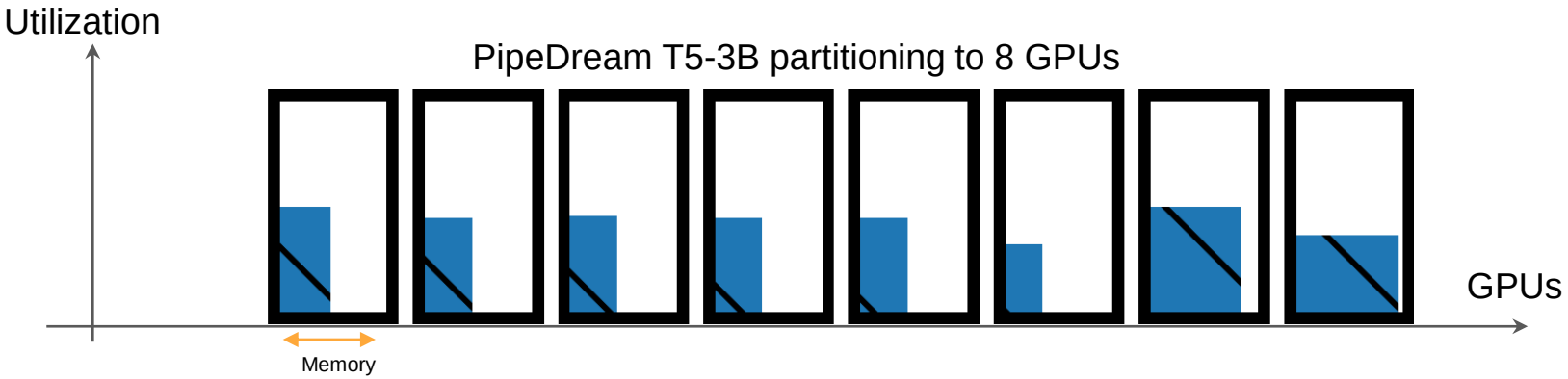
PipeDream: asynchronous

Staleness example:
parameter versions for minibatch 1.

Weight stashing staleness mitigation

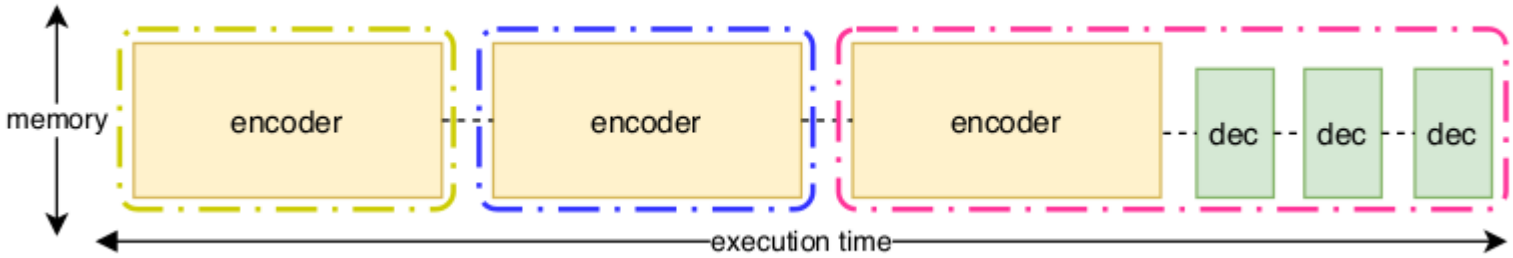


Problem: load imbalance in complex DNNs



Root cause for imbalance: topological constraints

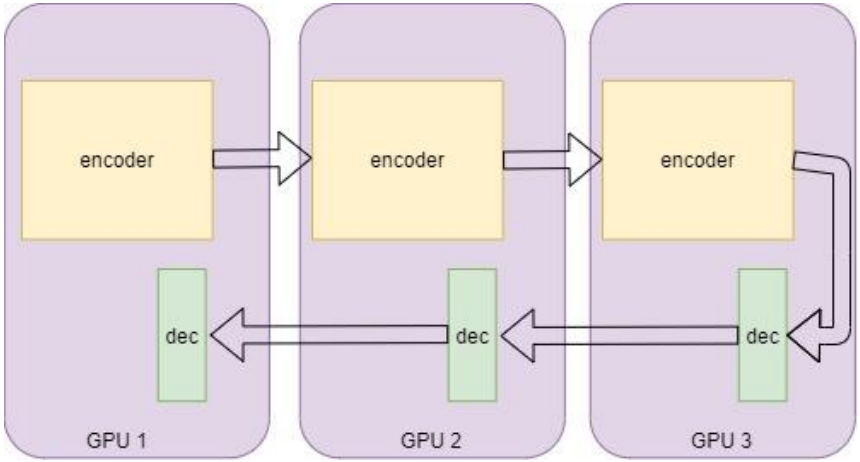
Example: NLP model partitioning across 3 GPUs
If each GPU may run only consequent layers, there is **no balanced partition**

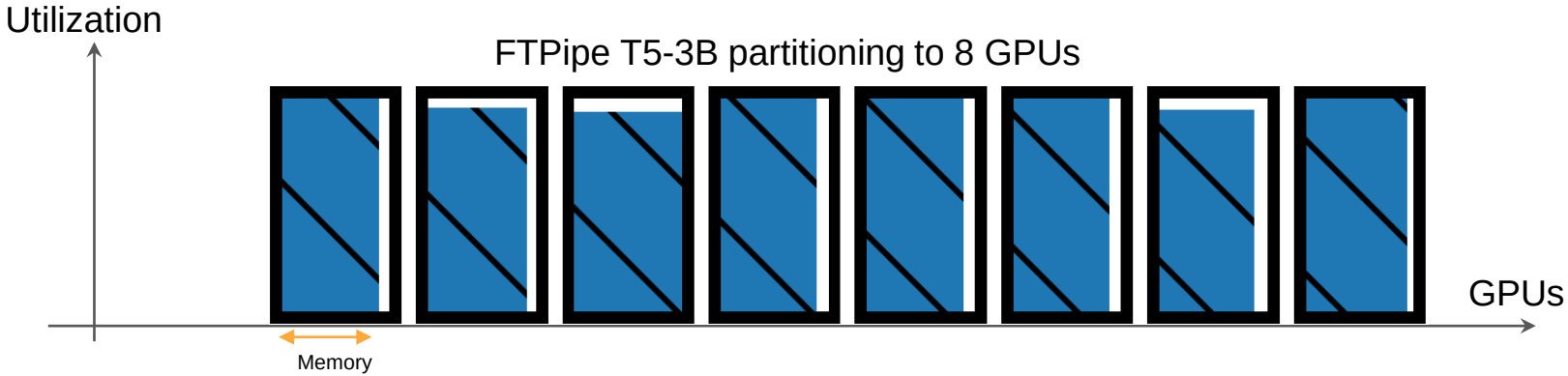


Topological partitioning is limited!

Mixed-Pipe: Balanced solution

- Mixed-Pipe: A general partitioning scheme that significantly improves memory and compute load balancing while hiding communication overheads.

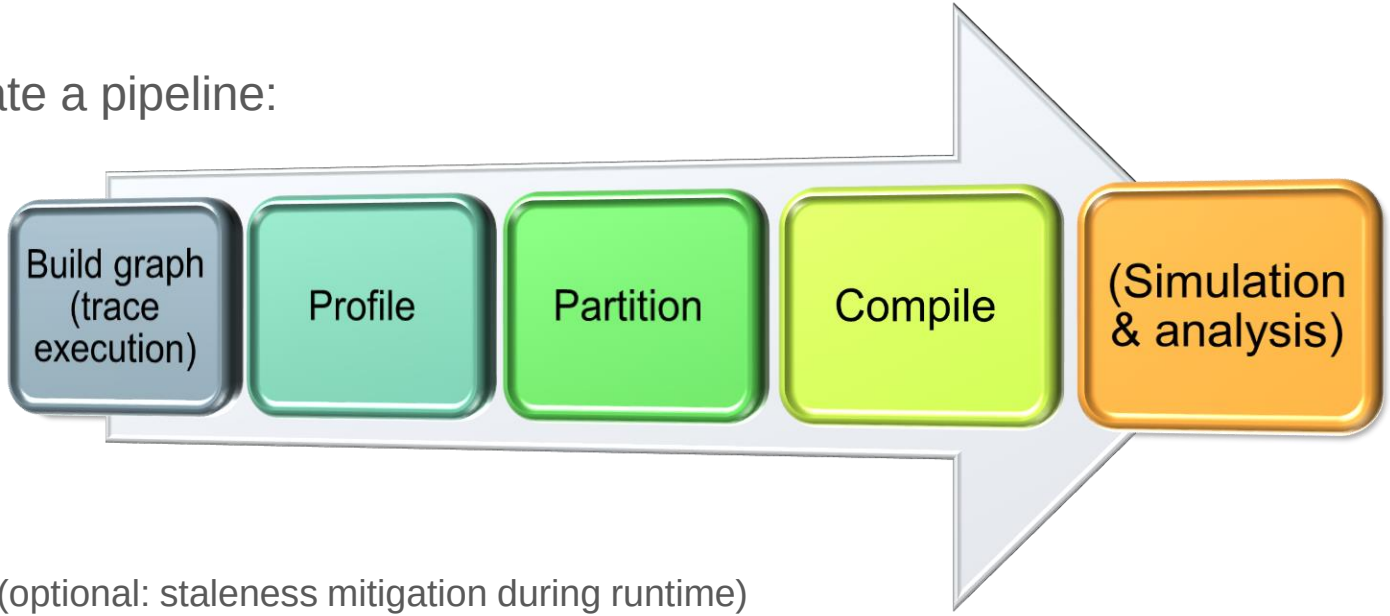




FTPipe system

Input: model, dataset, hardware configuration, and
training configuration{batch size, optimizer, …}

- create a pipeline:

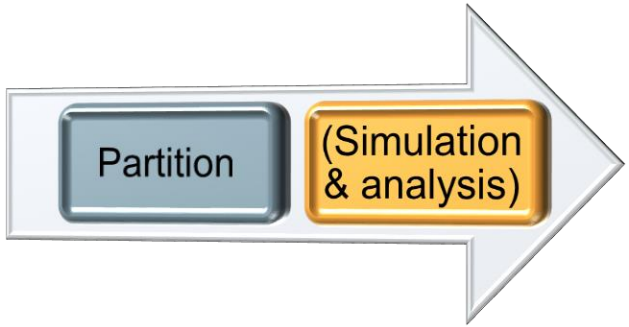


- Run
  - (optional: staleness mitigation during runtime)

FTPipe partitioning

- Seq pipe (exhaustive) - (Problem: may take too long)
- Seq pipe (greedy)
- Mixed-pipe

- **Run what is feasible, take the best.**



Mixed-Pipe Hierarchical graph partitioning

- Hierarchical graph partitioning is a general problem:
  Given a Graph G(V={compute demands, memory demands}
  ,E=communication demands) find a balanced partition to P processors while minimizing communication cost

- The general problem **does not capture** the staleness of asynchronous pipelines
- The general problem ignores communication overlapping (inaccurate objective)

Mixed-pipe high level partitioning algorithm

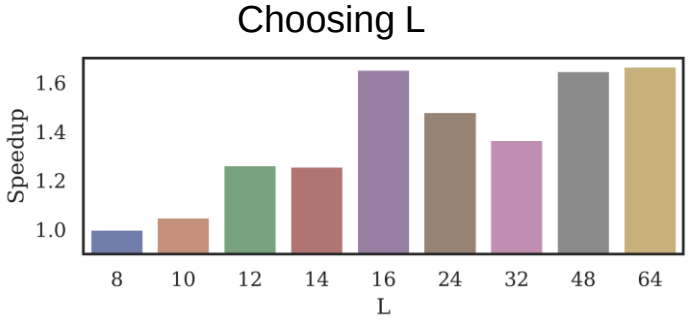
Input: a graph with compute, memory, and I/O requirements

1. **Coarsening:** Create **L** pipeline stages. (L is also chosen to limit staleness).
2. **Initial Assignment:** Assign stages to **P** GPUs (s.t. memory constraint hold) while optimizing load balance.
3. **Uncoarsening, Refinement**: shift blocks across adjacent stages while improving throughput objective or I/O

Outcome: balanced partitions with low I/O requirements allowing compute-I/O overlap

Mixed-Pipe: **coarsening**

- create $\boldsymbol{L} > P$ pipeline stages from N nodes. (typically – thousands of nodes to small L, e.g., $L = 2P, 3P$)
  - Our main experiments: P=8, L=16, N=4000
- We do so by **coarsening** the graph – merging nodes by edge contraction
  - Merge zero weight nodes and optionally outlier edges
  - **CCO property** "computation overlaps with -communication". Starting from smallest node, merge until CCO.
  - Coarsen by type
  - Coarsen around centers



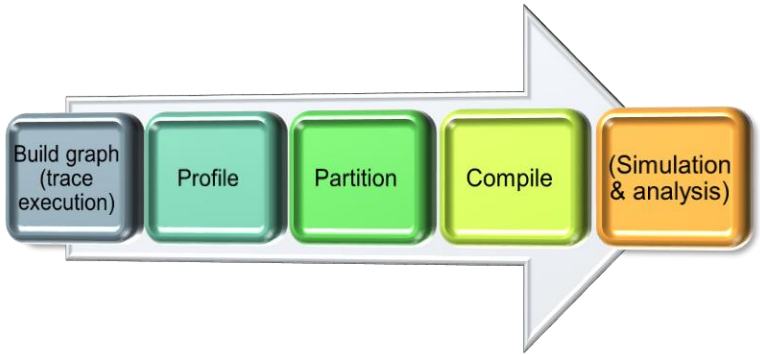
$$T^i_{comm_{fwd}} - C_i \leq 0, \qquad T^i_{comm_{bwd}} - C_i \leq 0$$

Example: coarsening by type

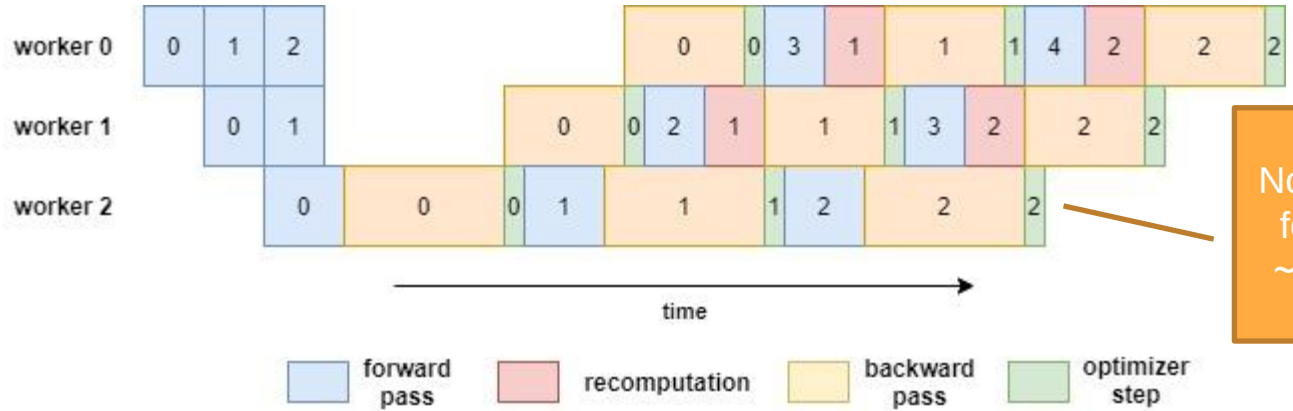
- Recorded at model tracing:

'T5ForConditionalGeneration/T5Stack[encoder]/T5Block[23]/T5LayerSelfAttention[0]/T5Attention[SelfAttention]/Linear[q]'
'T5ForConditionalGeneration/T5Stack[encoder]/T5Block[23]/T5LayerSelfAttention[0]/T5LayerNorm[layer_norm]'
'T5ForConditionalGeneration/T5Stack[decoder]/T5Block[23]/T5LayerFF[2]/T5DenseReluDense[DenseReluDense]/Linear[wo]’

- common prefixes:
  - **T5Block** will merge all 3,
  - **T5LayerSelfAttention** will merge first 2



FTPipe asynchronous pipeline



Note: careful profiling for last stage gives ~14% improvement

Generalized pipeline: dataflow-graph
We allow weight sharing, concurrent stages, smart data-loading and data propagation.

Results Evaluation: All improvements:

Model	Size	Task	Dataset	Accuracy	Partitioning	Pipeline	Speedup over GPipe Seq-pipe Epoch time	TTA
T5	3B	WSD	WiC	74.92%	mixed	Async	2.48×	[3]11.54×
						Sync	1.19×	1.19×
		NLI	RTE	90.97%	mixed	Async	2.71×	4.1×
						Sync	1.2×	1.2×
		QA	BoolQ	89.05%	mixed	Async	2.13×	2.88×
						Sync	1.08×	1.08×
		QA	MultiRC	85.6 F1, 59.3 EM	mixed	Async	3.32×	3.32×
						Sync	1.11×	1.11×
GPT2	1.5B	LM	WikiText2	12.02 perplexity	sequential	Async	1.6×	1.6×
Bert	340M	QA	Squad	93.3 F1, 87.2 EM	sequential	Async	2.04×	2.04×

FTPipe achieved faster training without accuracy loss compared to GPipe

Mixed-Pipe vs Seq-Pipe (T5-3B)

Mixed-Pipe is useful for both sync and async pipelines
Mixed-Pipe Improved load balance, no accuracy loss

FTPipe Summary

- A system for fine-tuning giant DNNs with limited resources.
- Mixed-Pipe overcomes topology constraints
  - Better load balance – no accuracy loss!
- Asynchronous fine-tuning: Staleness was not a problem when fine-tuning.
- Up to 3x faster fine-tuning of giant Transformers with billions of parameters

https://github.com/saareliad/FTPipe

Contact:
saareliad@gmail.com

“Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism”

- Mixed-Pipe partitioning achieves better load balance by relaxing topology constraints
- Staleness causes no accuracy loss in fine-tuning tasks

Thank You!

Saar Eliad, Ido Hakimi, Alon De Jager, Mark Silberstein, Assaf Schuster

saareliad@gmail.com

https://github.com/saareliad/FTPipe

https://www.linkedin.com/in/saar-eliad-314668151/