Understanding and Improving Failure Tolerant Training for Deep Learning Recommendation with Partial Recovery

Kiwan Maeng[1 2], Shivam Bharuka[1], Isabel Gao[1], Mark C. Jeffrey[1], Vikram Saraph[1], Bor-Yiing Su[1], Caroline Trippel[1], Jiyan Yang[1], Mike Rabbat[1], Brandon Lucia[2], Carole-Jean Wu[1]

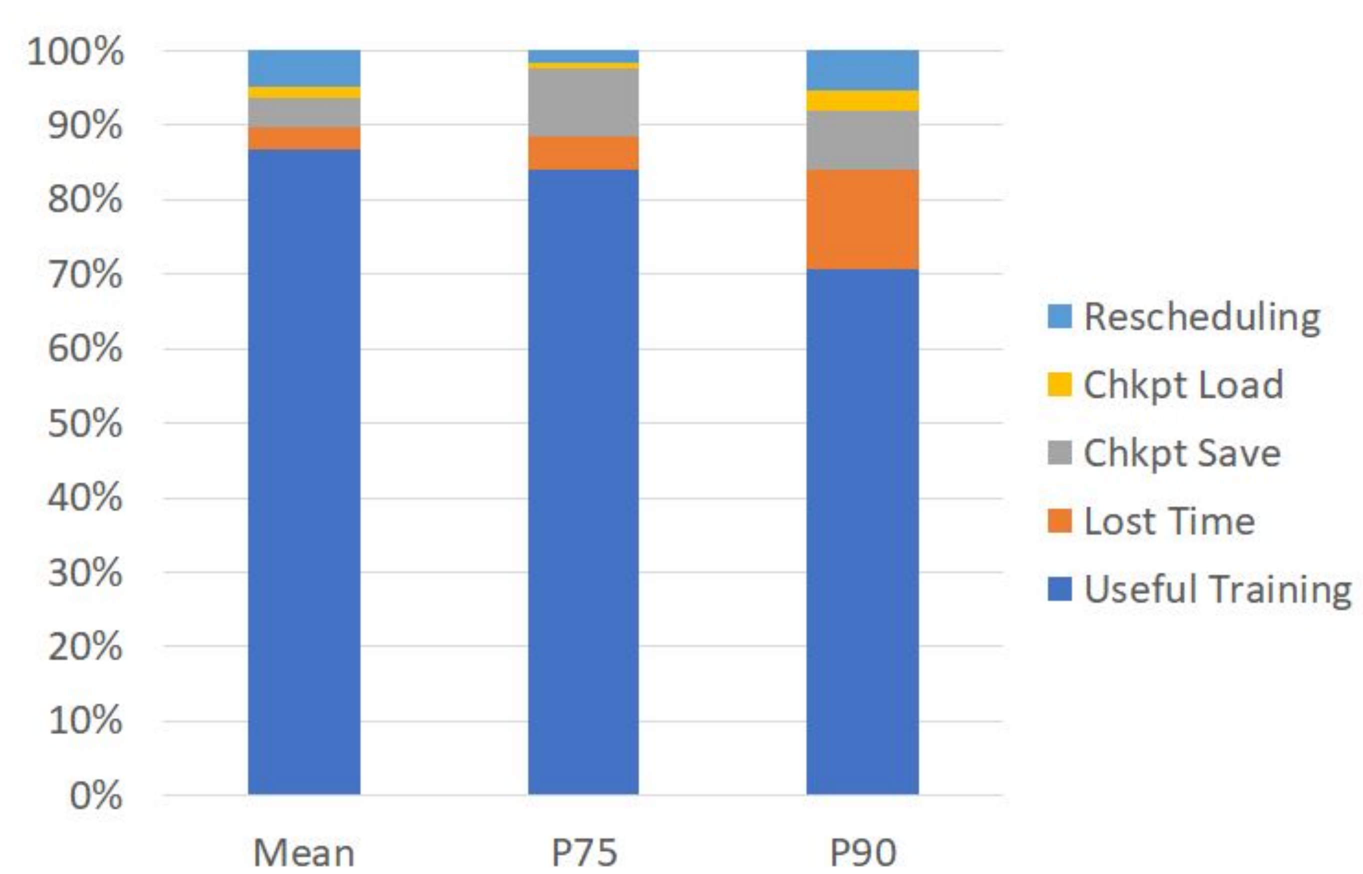
[1] Facebook AI   [2] Carnegie Mellon University

Motivation

- Deep learning recommendation systems (RecSys) consumes significant resources in real-world datacenters.
  - 50% of all AI training cycles in Facebook
  - 80% of all AI inference cycles in Facebook
- RecSys training involves training GBs-TBs sized embedding tables, requiring several tens to hundreds of nodes.
- Failure handling for RecSys training incurs non-negligible overhead, ranging from 13% on average to over 30% on P90 case.
- We designed CPR (**C**heckpointing with **P**artial recovery for **R**ecommendation model training), a training system that balances model quality and checkpointing overhead using partial recovery.
- CPR removes over 90% of the failure-handling overhead on production-scale cluster, while showing only negligible accuracy degradation.

Overhead of Failure Handling in RecSys Training

- Traditional system uses checkpointing with full recovery:
  - all nodes periodically save checkpoints, and if at least one node fails,
  - every nodes load the last checkpoint and re-execute from there.
- Full recovery has four major overheads, adding up to 13% on average:
  - checkpoint saving overhead,
  - checkpoint loading overhead,
  - re-execution of the lost computation after a failure,
  - and rescheduling the job running on the failed node.
- Checkpointing the embedding tables are the bottleneck in checkpointing.



CPR Design Choice 1. Adopting Partial Recovery

- Partial recovery only loads the checkpoint for the failed node, incurring model inconsistency to eliminate lost computation overhead.
- Unlike full recovery, partial recovery introduces an unexplored tradeoff between the final model quality and checkpoint-related overheads.

	Full recovery	Partial recovery
Time Overhead	Chkpt save, Chkpt load, Node rescheduling, Lost computation $O_{save}\frac{T_{total}}{T_{save}} + (O_{load} + O_{resched} + 0.5T_{save})\frac{T_{total}}{T_{fail}}$ Chkpt saving interval; Mean time between failures	No re-execution of the lost computation $O_{save}\frac{T_{total}}{T_{save}} + (O_{load} + O_{resched})\frac{T_{total}}{T_{fail}}$
Model Quality	Remains the same	$f(T_{save}, T_{fail}) = ?$

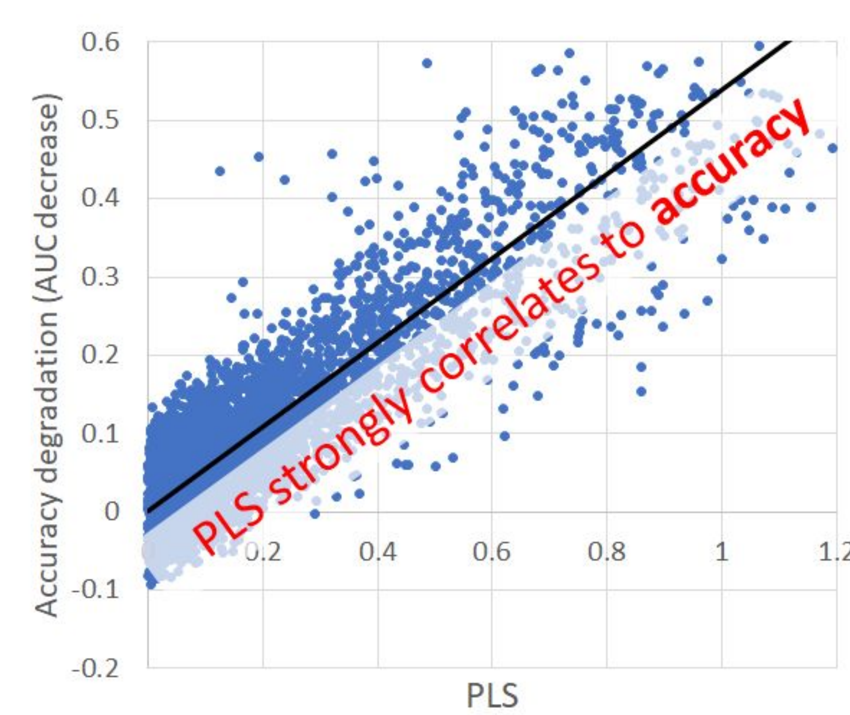
CPR Design Choice 2. Using PLS Metric

- CPR uses a metric called PLS (**P**ortion of **L**ost **S**amples) to predict the model quality degradation if partial recovery is used.
- CPR selects the appropriate checkpoint saving interval to contain the accuracy degradation to a user-specified level, while maximizing performance.

Chkpt saving interval

$$\mathbb{E}[PLS] = \frac{0.5T_{save}}{T_{fail}N_{emb}}$$

Mean time between failures   Number of Emb PS

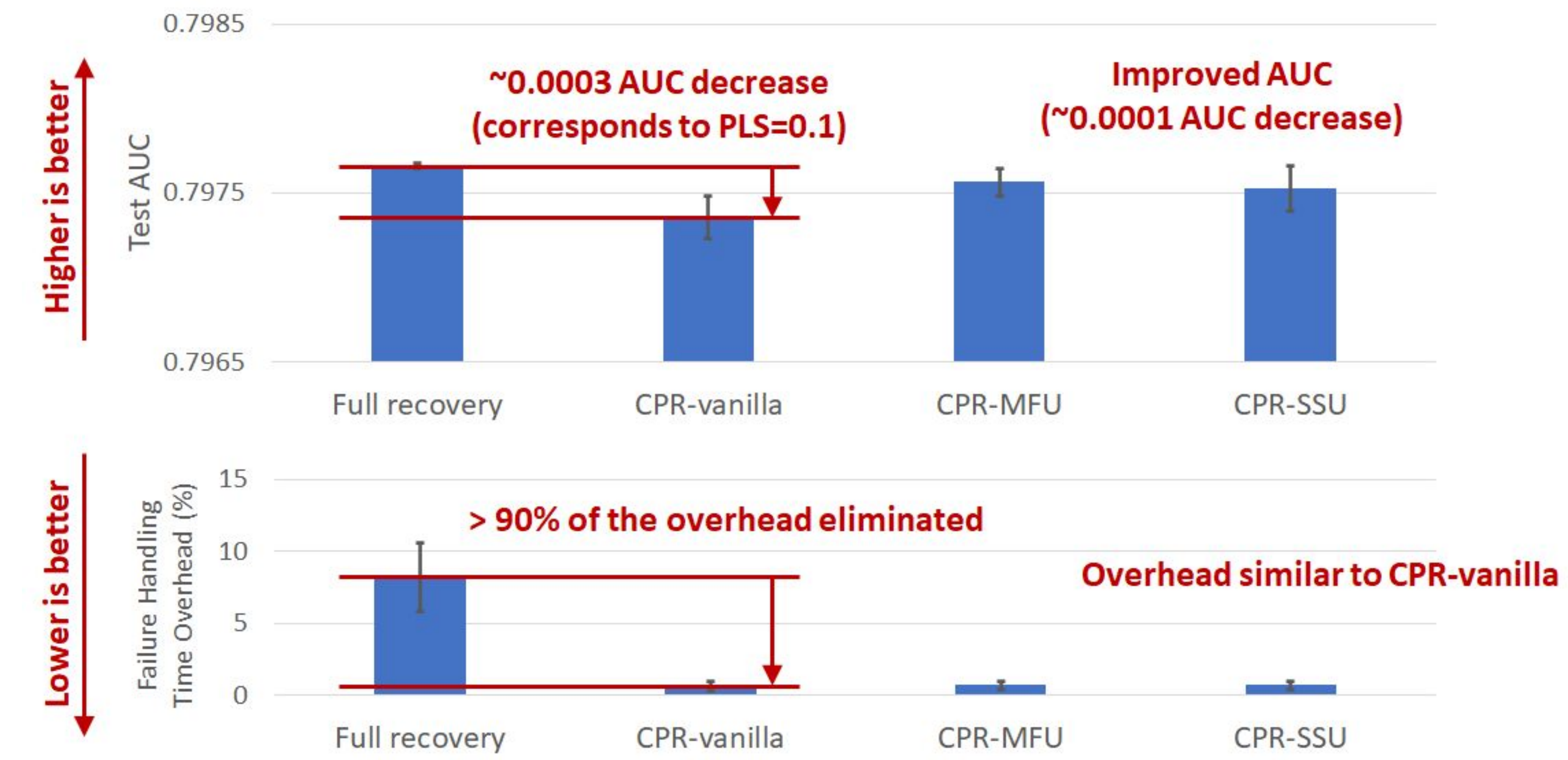


CPR Design Choice 3. MFU/SSU Optimizations

- CPR prioritizes saving the more frequently updated vectors in the embedding table.
- MFU (**M**ost **F**requently **U**sed): allocates a counter per row in the embedding tables to track the Top-k most frequently accessed rows
- SSU (**S**ub-**S**ampled **U**sed): randomly subsample inputs and select rows the inputs access to proxy MFU efficiently

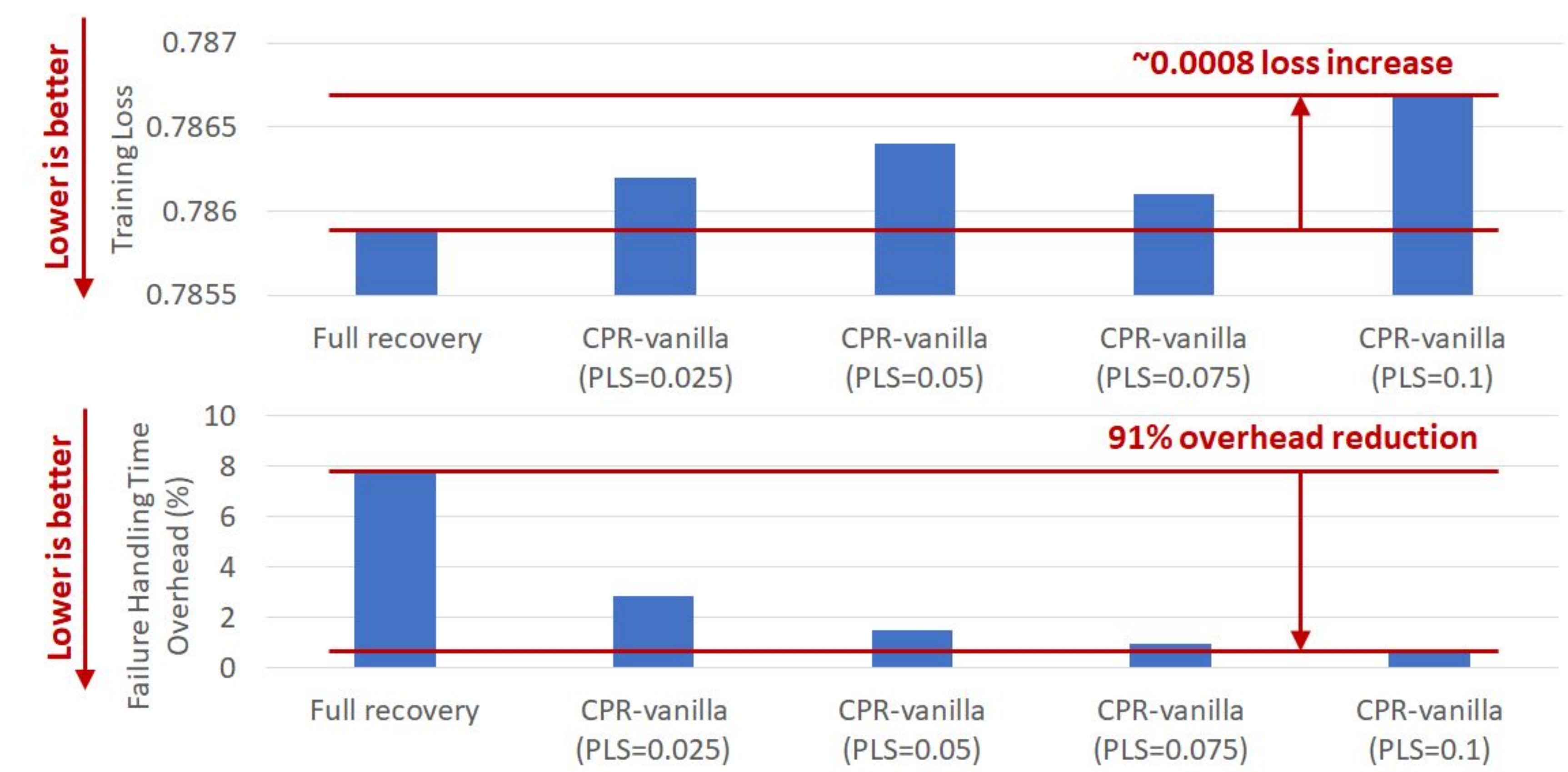
Evaluation 1. Open-source Emulation Setup

- Criteo Kaggle [1] and Terabyte [2] datasets, DLRM [3] model
- Trained on a single GPU, emulating multi-node failures by injecting random failures that clears 12.5%, 25%, or 50% of the embedding tables.
- Overhead numbers modeled after the production-scale statistics.
- CPR eliminated over 90% of the failure-related overheads, while sacrificing only 0.0001-0.0003 AUC.



Evaluation 2. Production-scale Setup

- 20 MLP trainers, 18 embedding parameter servers: each Intel 20-core, 2GHz processors, 25Gbit Ethernet.
- 50-hour training, injected 5 failures that failed randomly selected 4 embedding parameter servers.
- CPR eliminated over 91% of the overheads, while sacrificing only 0.0008 loss.



References

[1] https://www.kaggle.com/c/criteo-display-ad-challenge

[2] https://labs.criteo.com/2013/12/download-terabyte-click-logs.

[3] Deep Learning Recommendation Model for Personalization and Recommendation Systems. Naumov et al. CoRR-2019.