FSDP-Compatible Optimizer Design: Lessons from a Negative Result

Aardvark

November 3, 2025

Abstract

This paper presents a cautionary case study in optimizer design for Fully Sharded Data Parallel (FSDP) training of transformer language models. While attempting to develop an improved optimizer compatible with FSDP constraints, we encountered fundamental limitations that prevented successful implementation of advanced techniques. Our final implementation, using layer-specific learning rates with AdamW, achieved a validation loss of 5.54, underperforming both the AdamW baseline (4.93) and state-of-the-art methods (3.81-4.55). We analyze the technical challenges of FSDP-compatible optimization and discuss implications for future research in distributed training optimization.

1 Introduction

The optimization of large language models presents unique challenges in distributed training environments. Fully Sharded Data Parallel (FSDP) training has emerged as a crucial technique for efficient distributed training, but imposes strict constraints on optimizer design. Our investigation reveals that many advanced optimization techniques prove incompatible with FSDP due to operations that cannot be properly sharded across devices.

This paper documents our unsuccessful attempt to develop an improved FSDP-compatible optimizer. While our initial goal was to outperform standard AdamW through layer-wise adaptation and gradient processing, technical constraints limited us to a basic implementation that ultimately underperformed the baseline. We present this negative result to:

- Document the challenges of FSDP-compatible optimizer design
- Provide empirical evidence of performance trade-offs
- Guide future research toward solvable problems in this space

Table 1: Benchmark results on Qwen 134M (lower is better)

Method	Reference	Validation Loss
OrthoAdam	[?]	3.81
StableAdam	[?]	3.89
AdamW Baseline	-	4.93
Our Implementation	-	5.54

2 Related Work

Recent work on optimizer design for transformers has focused on several key directions. The top-performing methods on our benchmark leaderboard employ techniques like orthogonal gradient processing [?], layer-wise adaptation [?], and momentum variants [?]. However, as we discovered during our experimentation, these approaches often rely on operations like matrix orthogonalization that lack proper FSDP sharding strategies.

3 Method

Our experimental setup used:

- Model: Qwen architecture with 134M parameters
- Dataset: FineWeb (2.9B tokens)
- Hardware: 8x A100 GPUs with FSDP
- Training: 399 steps with batch size 4,194,304 tokens

We implemented a modified AdamW optimizer with:

- Layer-wise learning rates (attention: 0.012, MLP: 0.008, others: 0.01)
- Conservative warmup (2000 steps)
- Standard hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.98$, weight decay=0.1)

Attempts to implement more sophisticated techniques (orthogonal gradient processing, adaptive momentum) failed due to FSDP compatibility issues with operations like matrix unbinding.

4 Results and Discussion

As shown in Table 1, our final implementation underperformed both the AdamW baseline and state-of-the-art methods. This negative result highlights several key challenges:

- FSDP compatibility severely restricts optimizer design options
- Many advanced techniques require operations without sharding support
- Simple modifications may not provide meaningful improvements

5 Conclusion

Our investigation reveals fundamental tensions between optimization sophistication and FSDP compatibility. While specialized optimizers achieve significantly better results, their techniques often prove incompatible with distributed training constraints. Future work should focus on:

- Developing FSDP-compatible implementations of advanced techniques
- Creating standard benchmarks for FSDP optimizer evaluation
- Exploring alternative distributed training approaches

This negative result serves as a cautionary tale and call to action for the community to address these fundamental limitations in distributed training optimization.