NYU

Optimizing Transformer-based Sequential Recommender Systems

PRESENTED BY:

Mahima Sachdeva - ms15523@nyu.edu

Tanvi Takavane - tt2884@nyu.edu

05/08/2025

Executive Summary

Goal:
To evaluate how model scaling and optimization strategies affect training efficiency and performance of BERT4Rec for sequential recommendation.

Approach:
Scaled BERT4Rec from 5M to 430M parameters and applied multiple optimization techniques across three hardware platforms (NVIDIA A100, V100, L4).

Value / Benefit:
Demonstrated how modern ML optimization techniques can dramatically reduce training time and resource cost while improving throughput for large models.

Technical Challenges

- Model instability with scaling (sensitive to LR, exploding gradients) => Hyperparameter sweeps
- Memory bottlenecks during training larger models (430M) => Distributed training
- Instability in training with mixed precision (AMP) => use GradScaler for gradient scaling
- Slower training on V100, L4 => did not use L4 for the largest model
- GPU availability on HPC
- Tuning iLoRA

Approach

Baseline Setup

- **Dataset**: MovieLens-20M (initial), Amazon Electronics (for scaled up variants)
  - Preprocessed to filter users/items with fewer than 5 interactions
  - Created input sequences for masked item prediction (like MLM)

- **Base model**: BERT4Rec (originally 5M parameters, scaled up to 85M for new baseline)

- **Scaled variants**: 130M, 430M

- **Framework**: PyTorch 2.1
  - Mixed Precision using torch.cuda.amp
  - Model Optimization using torch.compile, torch.jit
  - iLoRA integration via manual LoRA injection into transformer layers

- **Baseline metrics**: Hit Rate at 10, epoch time, throughput

- **Experiment Tracking & Profiling**: Weights & Biases (WandB), PyTorch Profiler

- **Multi-GPU scaling**: DistributedDataParallel

NYU

Approach

Optimization Techniques

- Hyperparameter sweeps: LR, dropout, weight decay
- Mixed precision training: AMP
- torch.compile & torch.jit: runtime graph optimizations
- iLoRA (Instance-wise LoRA): parameter-efficient tuning
- Metrics recorded: Accuracy, Time/epoch, Throughput (samples/sec)

NYU

Summary of Main Results

Model Size	Accuracy (Hit Rate@10)	Hardware (GPU)	Time/Epoch	Throughput (samples/sec)
5M : original model on MovieLens-20M	0.0745	T4	~17 min	3378.37
85.11 M	0.0818	A100	~2.5 min	19659.98
130.10 M	0.0855	A100	~4 min	11630.68
429.15 M	0.0910	A100	~9 min	3805.26

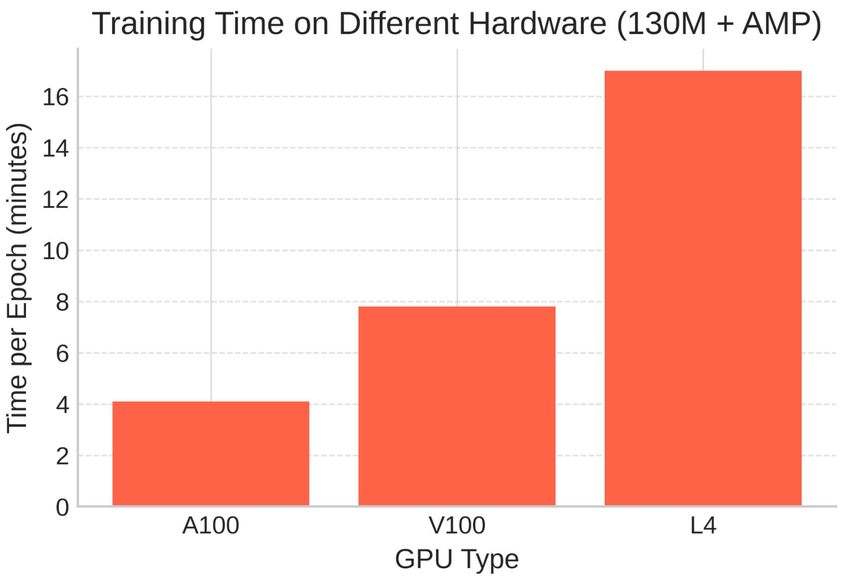
- AMP & torch.compile halved training time for larger models
- iLoRA achieved similar accuracy with <1% of trainable params

NYU

Experimental Evaluation:

I. Hardware Comparison: NVIDIA A100, V100, L4

- A100: Fastest
- V100: Takes double the time.
  2x V100 equal to 1x A100 in time.
- L4: Takes almost 4x of A100



NYU

Experimental Evaluation:

II. JIT + AMP

- torch.compile reduced Python overhead significantly
- AMP improved speed almost 2x with negligible loss in accuracy

III. iLoRA

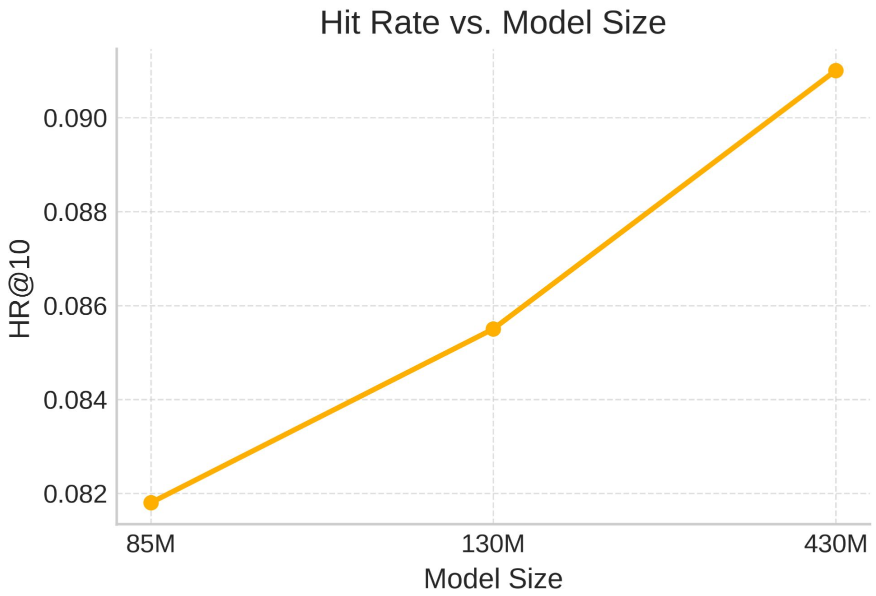
- Applied on 130M (trainable 0.55%) & 430M (trainable 0.96%) models
- Retained accuracy while reducing trainable parameters by 99.4%
- Tradeoff: iLoRA bit slower per epoch, but saves memory and compute

Experimental Evaluation

Performance vs Model size:

Hit Rate@10: a measure of evaluating the performance of recommender systems.

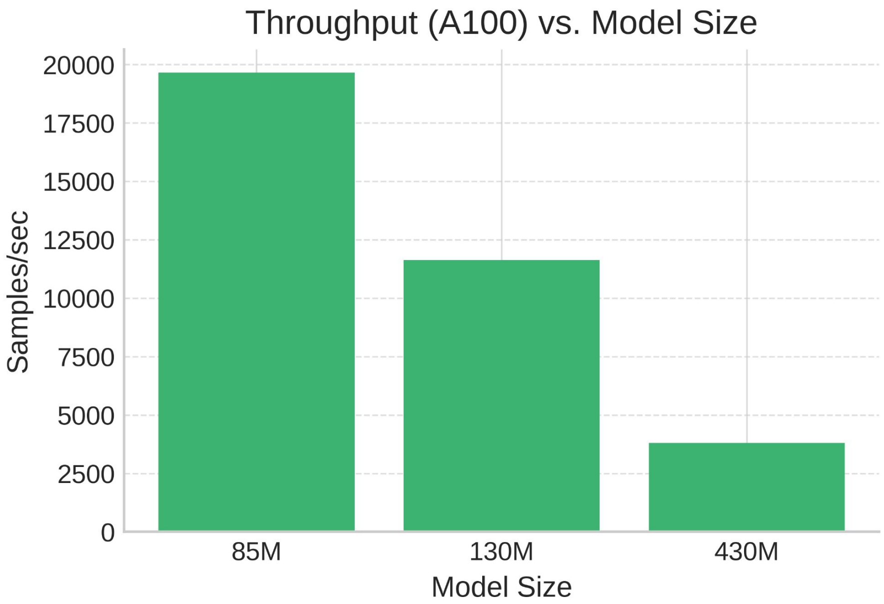
We observe that the hit rate improves with increasing model size due to higher capacity of the model to learn dependencies.

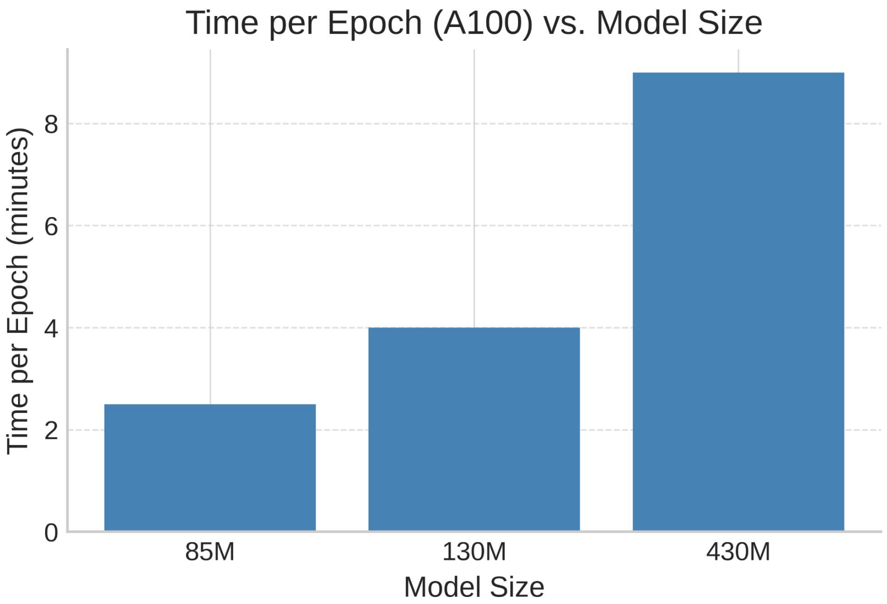


NYU

Training efficiency vs Model size:

- Throughput drops as model size increases due to higher compute and memory demand.
- Training time increases sharply as model size grows, especially beyond 130M.





Profiling Insights with PyTorch Profiler: 130M model

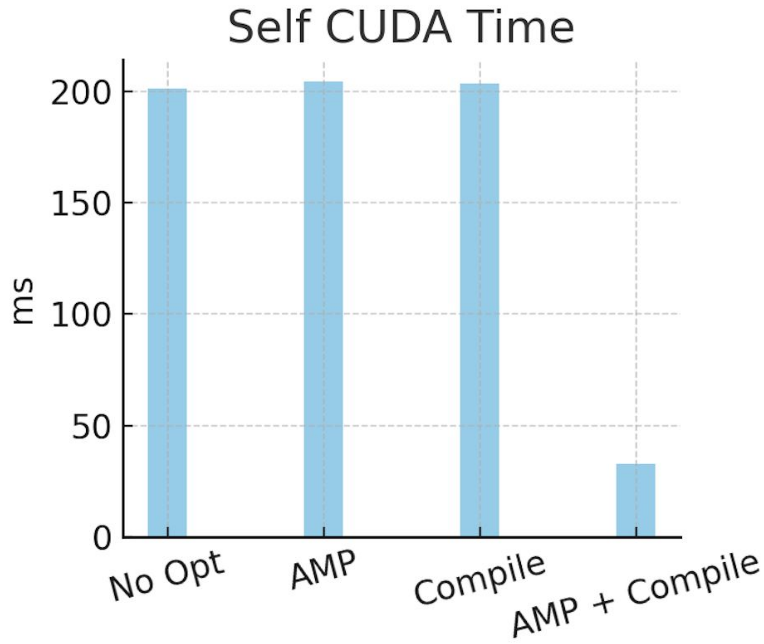
Metric	No Opt	AMP Only	Compile Only	AMP+Compile
Self CUDA Time Total	201.2 ms	204.4 ms	203.5 ms	32.6ms
Peak CUDA Memory	~11.2GB	~7.2GB	~11.2GB	~7.2GB
Top Op by CUDA %	aten::mm	aten::mm	aten::mm	CompiledFunction + mm
Launch Overhead	High	High	Lower	Lowest

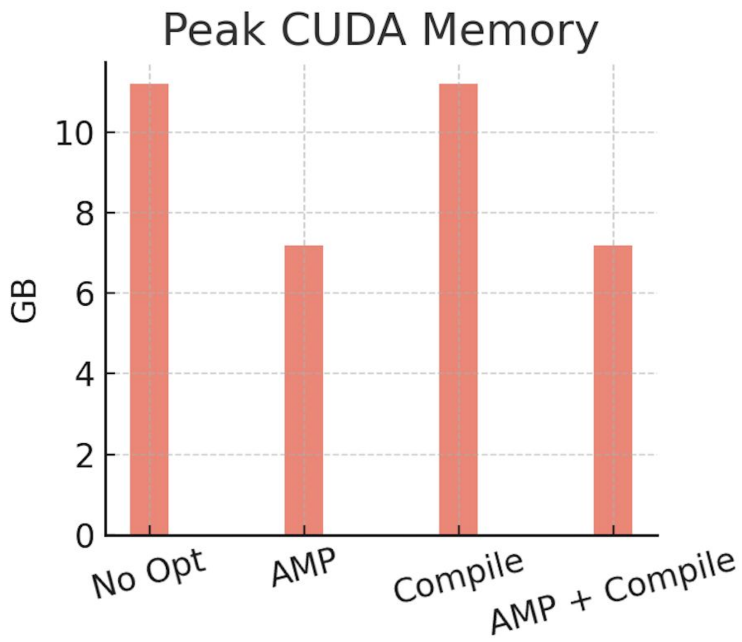
*these experiments were done on the 130M model on A100 GPU

- **AMP + Compile drastically reduces Self CUDA time** (6× faster), due to op fusion + FP16 acceleration.
- **AMP alone reduces memory usage** (~4 GB saved), enabling larger batch sizes.
- torch.compile introduces minor CPU overhead but eliminates redundant kernel launches.
- aten::mm remains dominant, but fused CompiledFunction replaces many smaller ops.

NYU

Profiling Insights with PyTorch Profiler





Combining AMP and torch.compile reduces step time by 75% and memory by 40%, giving the most optimized BERT4Rec training setup

Profiling Insights with PyTorch Profiler

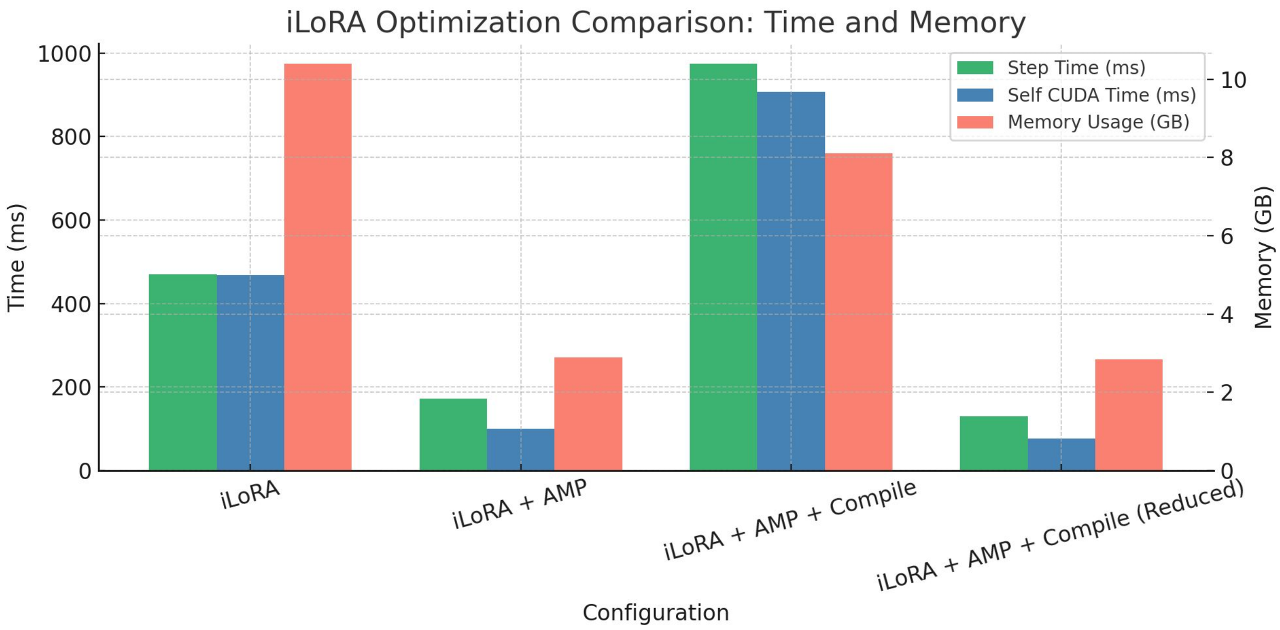
Configuration	Step Time	Self CUDA	Memory
iLORA	~470ms	~470ms	10.4GB
iLORA+AMP	~173ms	100.7ms	2.9GB
iLORA+AMP+Compile	~975ms	908ms	8.1GB
iLORA+AMP+Compile(Reduced)	~130ms	~76ms	2.85GB

*these experiments were done on the 430M model on A100 GPU

- The **baseline iLoRA** shows the slowest execution, with high memory demands and over 470 ms per training step.
- Adding **AMP** drastically reduces both step time and memory consumption, dropping usage from ~10.4 GB to ~2.9 GB.
- However, using **default torch.compile** with AMP introduces significant overhead due to dynamic adapter logic, increasing step time to nearly 1 second.
- Switching to **torch.compile(mode='reduce-overhead')** recovers performance — achieving the **fastest iLoRA configuration**, with ~130 ms step time and only 2.85 GB memory usage.

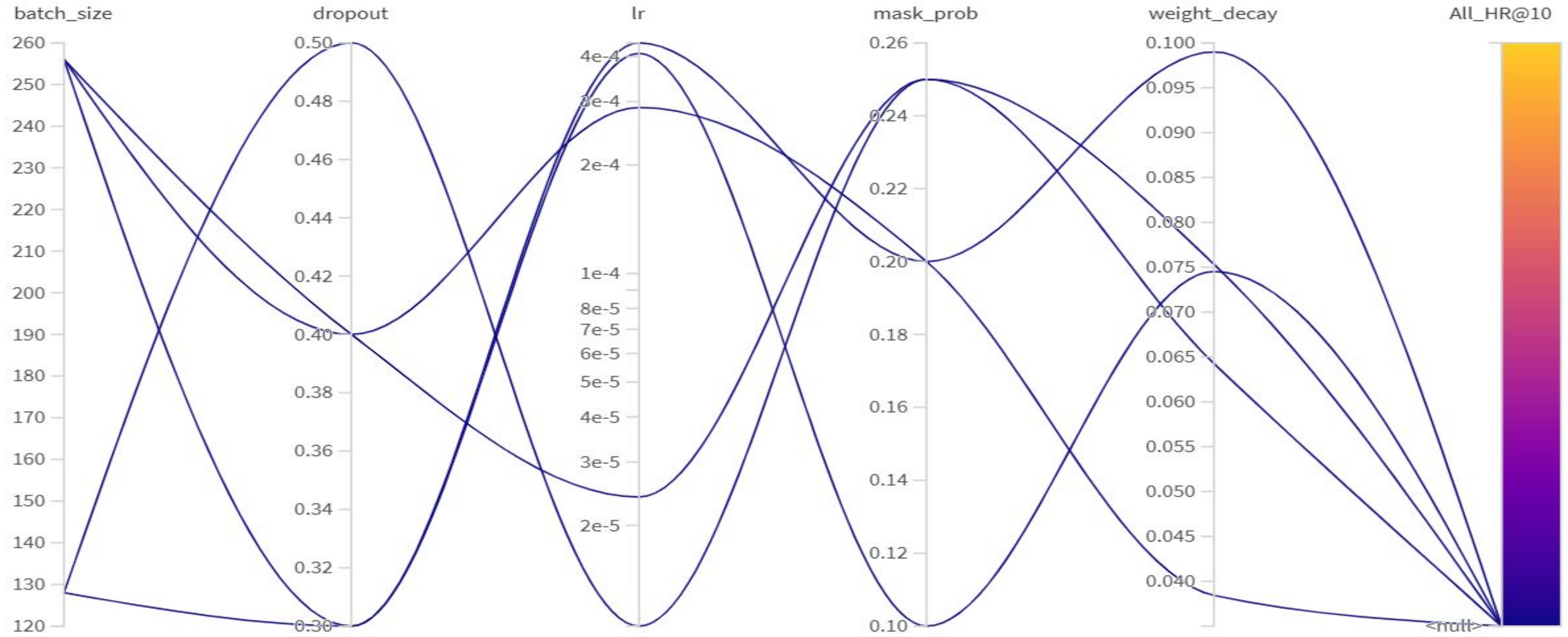
NYU

Profiling Insights with PyTorch Profiler



By switching to reduce-overhead, we retain AMP's memory and speed gains while avoiding compile overhead — achieving the best iLoRA performance.

WandB Sweeps



Observation and Conclusion

- Scaling BERT4Rec improves performance but at high compute cost
- AMP + torch.compile offers best speedup across all sizes
- iLoRA is a promising technique for training large models with limited resources
- Optimization needs to be hardware-aware for best results

GitHub Repository

GitHub:
https://github.com/mahi397/Optimizing-Transformer-based-Sequential-RecommenderSystems

WandB: https://wandb.ai/ms15532-new-york-university/BERT4Rec

WandB: https://wandb.ai/tt2884-new-york-university/BERT4Rec

THANK YOU

NYU