Batch Inference Load Balancing for Mixture-of-Experts LLMs*

Peter Lin

*Idea originally from group research project for CS 498 ML Systems

2/16/2026

Project Overview: Optimizing Machine Learning Systems

Background and Motivation: What is MoE? Why use batch inference?

Problem: How does batching impact expert load balance?

Proposed Solution: How does the proposed algorithm address the core of the problem?

Experimental Setup: Hardware details and evaluation dataset

Initial Evaluation: What are the baseline statistics?

Final Evaluation: How does the optimized model perform compared to baseline?

Conclusion: Final thoughts and takeaways

Background: Mixture-of-Experts (MoE) Models are Sparse

In dense models, all parameters are activated during generation

- Computationally expensive

In sparse models, only a subset of parameters are activated

- Computationally efficient
- Uses a router to assign specific tokens to a subset of (k) experts
    - Top-k experts chosen based on score

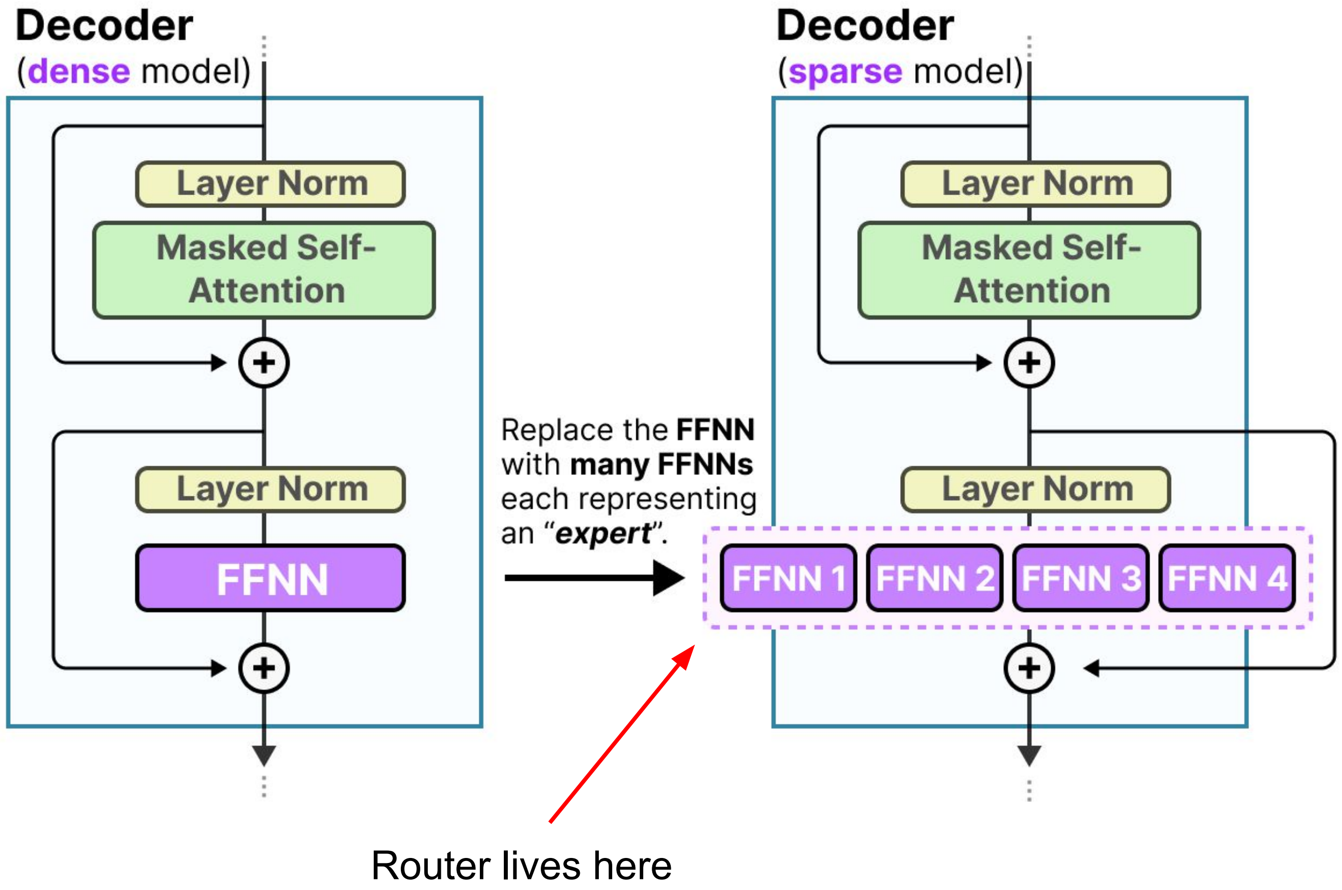


Image: https://www.maartengrootendorst.com/blog/moe/

Background: Qwen3 MoE Architecture

Qwen3-30B-A3B has decoder-only architecture

- 48 decoder layers
- 128 unique experts per layer
- k = 8 for top-k selection

We can interact with intermediate computations

- e.g. hook into gate module to see router's top-k expert selection
- e.g. replace the forward function in the mlp module with a custom forward function

```
Qwen3MoeForCausalLM(
  (model): Qwen3MoeModel(
    (embed_tokens): Embedding(151936, 2048)
    (layers): ModuleList(
      (0-47): 48 x Qwen3MoeDecoderLayer(
        (self_attn): Qwen3MoeAttention(
          (q_proj): Linear(in_features=2048, out_features=4096, bias=False)
          (k_proj): Linear(in_features=2048, out_features=512, bias=False)
          (v_proj): Linear(in_features=2048, out_features=512, bias=False)
          (o_proj): Linear(in_features=4096, out_features=2048, bias=False)
          (q_norm): Qwen3MoeRMSNorm((128,), eps=1e-06)
          (k_norm): Qwen3MoeRMSNorm((128,), eps=1e-06)
        )
        (mlp): Qwen3MoeSparseMoeBlock(
          (experts): Qwen3MoeExperts(
            (act_fn): SiLUActivation()
          )
          (gate): Qwen3MoeTopKRouter()
        )
        (input_layernorm): Qwen3MoeRMSNorm((2048,), eps=1e-06)
        (post_attention_layernorm): Qwen3MoeRMSNorm((2048,), eps=1e-06)
      )
    )
    (norm): Qwen3MoeRMSNorm((2048,), eps=1e-06)
    (rotary_emb): Qwen3MoeRotaryEmbedding()
  )
  (lm_head): Linear(in_features=2048, out_features=151936, bias=False)
)
```

All 128 experts live here

Router lives here

Motivation: Batching Inputs Leverages GPU Parallelism

Without batching, individual inputs are processed sequentially

Offline batching inputs enables parallel processing but introduces pad tokens (**focus of this project**)

Continuous batching packs inputs tighter but introduces infra overhead (e.g. vLLM)

Image: https://www.baseten.co/blog/continuous-vs-dynamic-batching-for-ai-inference/

Problem: Offline Batch Inference Induces Expert Load Imbalance

Heatmaps plot token distribution across every expert in the model

- All 128 experts in all 48 layers (6,144 unique experts)

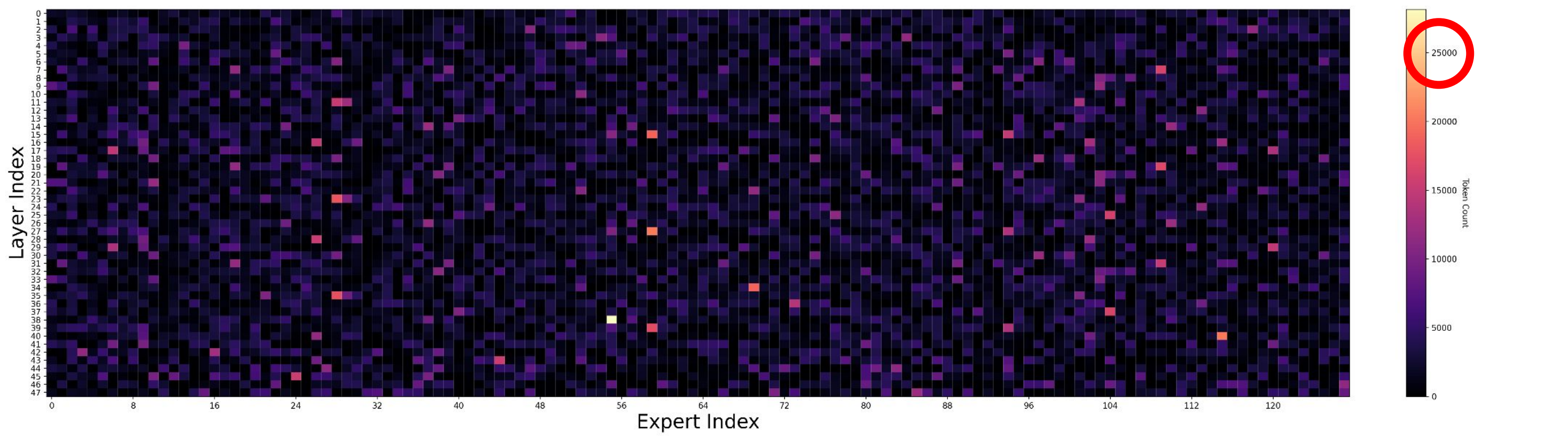
No batching

- Tokens are relatively evenly distributed

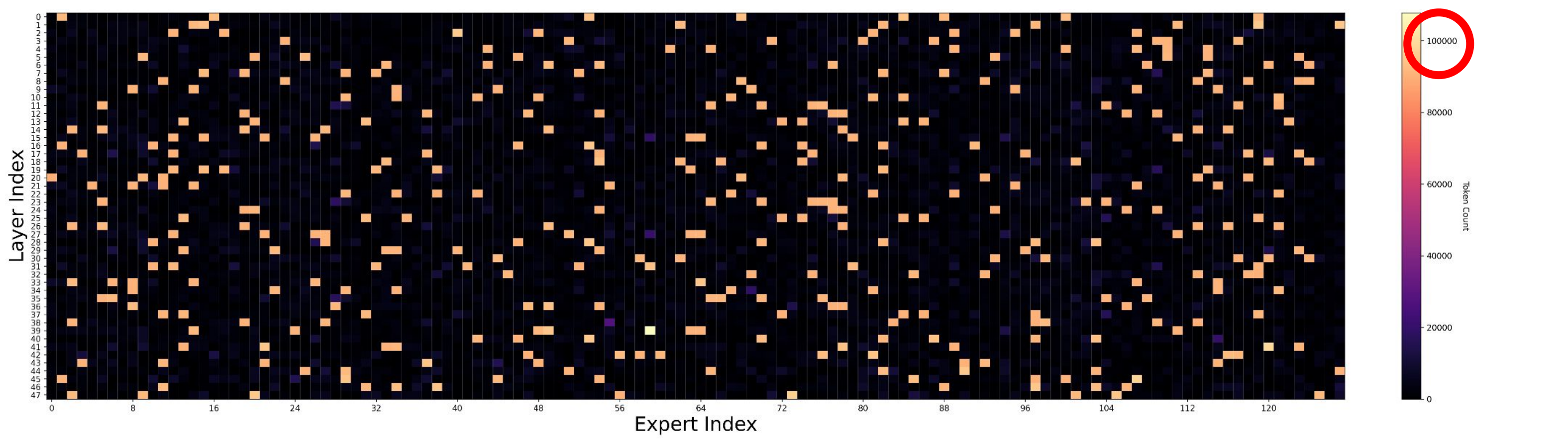
With batching

- Pad tokens clearly overpower the distribution

Token Distribution Heatmap (No Batching)



Token Distribution Heatmap (With Batching, Same Data)

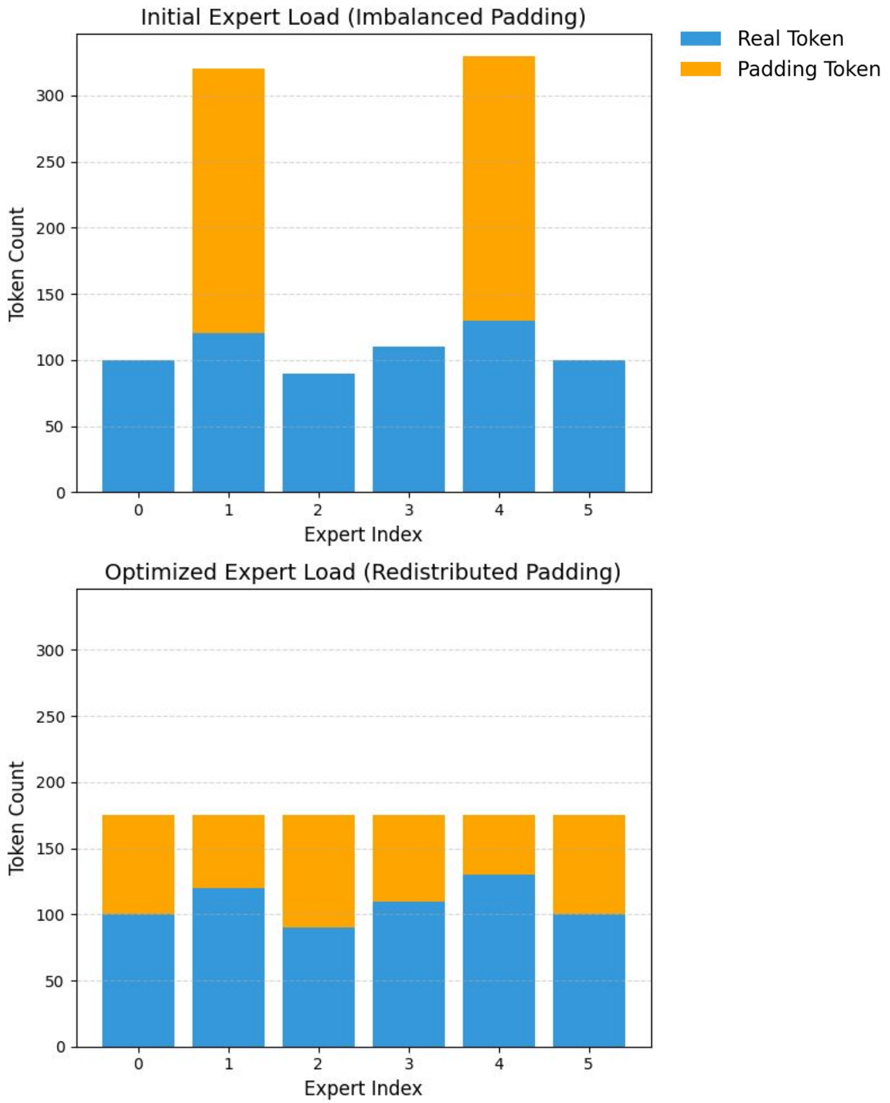


Solution: Reroute Padding Tokens to Underutilized Experts

Optimized forward function:

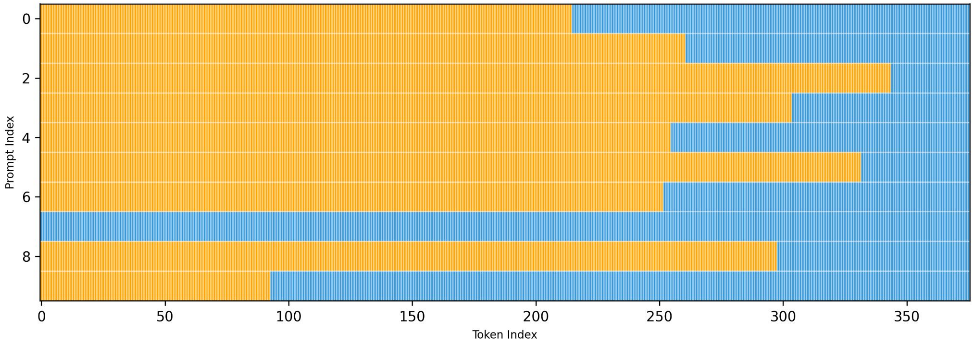
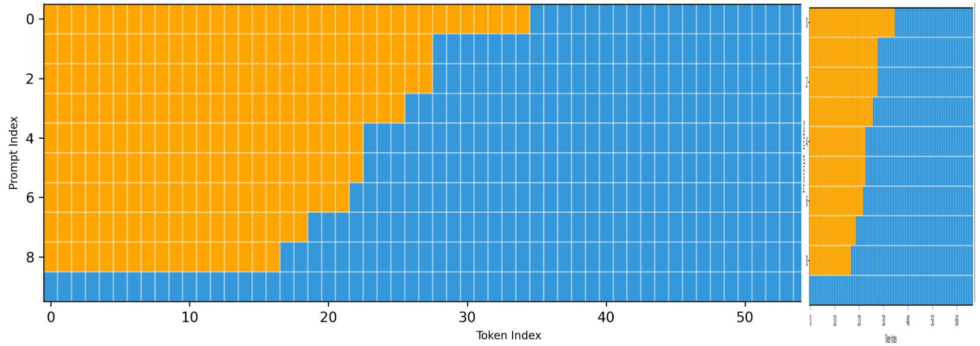
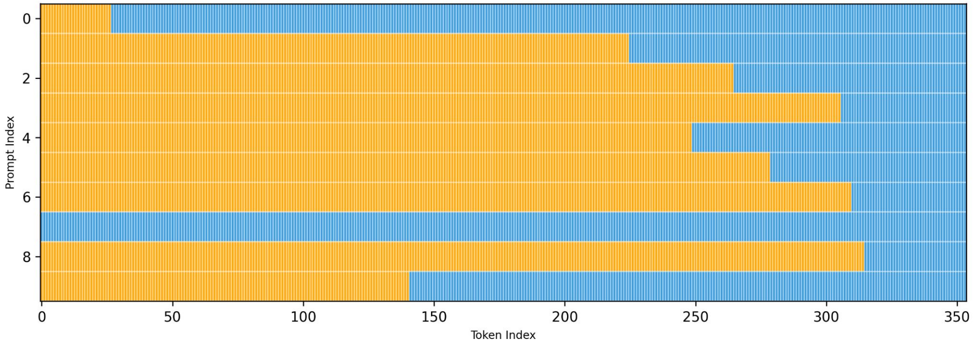
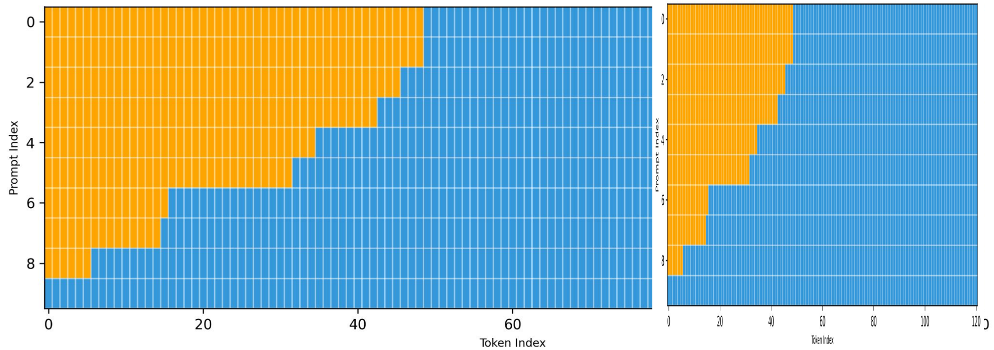
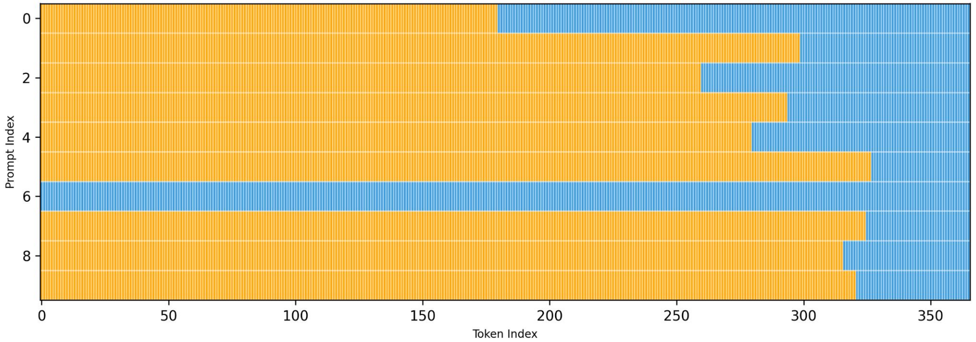
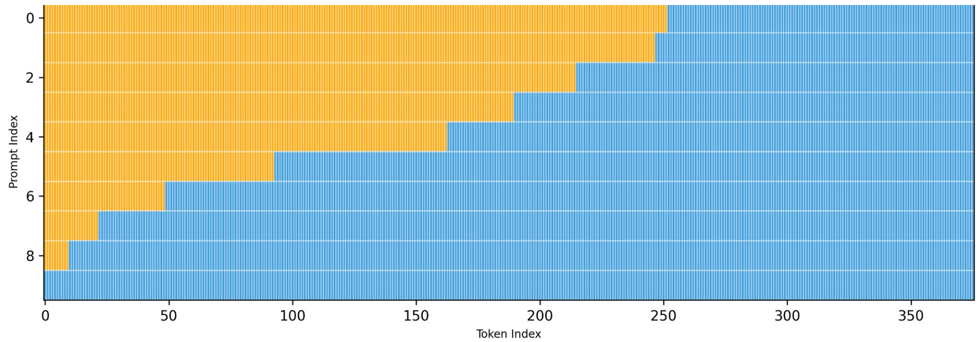
1. Get padding mask (1 if padding 0 otherwise)
2. Get expert selection from gate network
3. Compute padding indices and number of pad tokens
4. If number of pad tokens is positive:
    a. Redistribute every pad token's expert assignments to bring distribution closer to average "real" token count
    b. Update expert selection
5. Compute expert outputs using expert selection
6. Return final tensor

e.g. rerouting pad tokens from experts 1 and 4 across all 6 experts



Observation: Sorting Before Batching Reduces Padding

Real Token
Padding Token

Padding Percentage	3 Batches Without Sorting	3 Batches After Sorting by Length	Padding Percentage
62.61% (2,354 tokens)			32.99% (221 tokens)
59.80% (2,117 tokens)			24.05% (291 tokens)
71.09% (2,602 tokens)			33.01% (1,241 tokens)

Experimental Setup

Hardware: 1xA100 GPU (Google Colab, High-RAM)

Model: Qwen3-30B-A3B

Dataset: Massive Multitask Language Understanding (MMLU)

Baseline evaluation on sorted and unsorted batches

Final evaluation comparing baseline with optimization

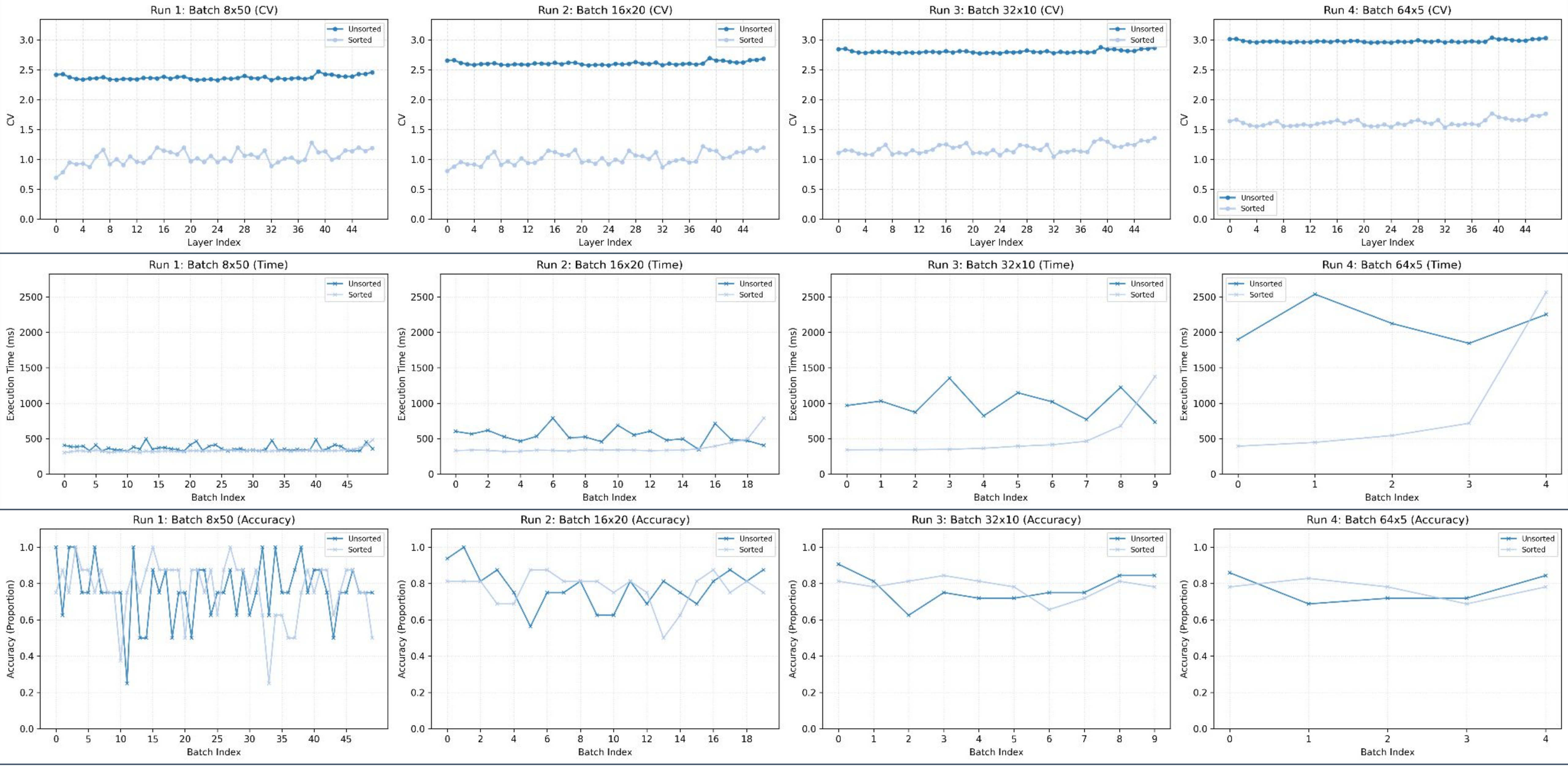
For each evaluation:

- Measure load balance with coefficient of variation (CV), defined as std dev / mean
- Measure execution time based on time taken to complete processing a single batch
- Measure model output quality using MMLU accuracy

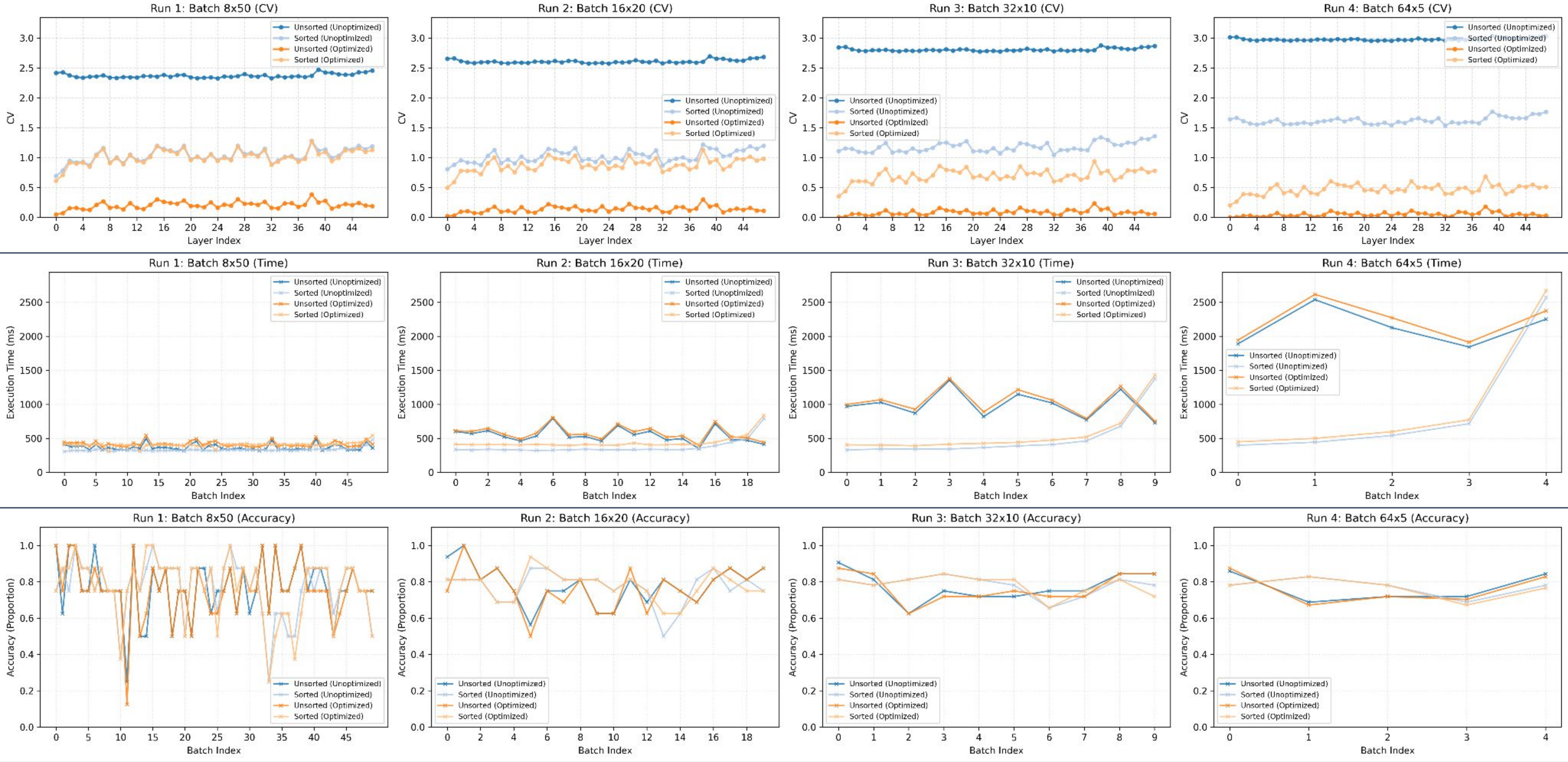
Measure stats for four runs (batch size x batch count):

- 8x50, 16x20, 32x10, 64x5

Initial Evaluation: Baseline (Unsorted and Sorted Batches)



Final Evaluation: Optimization vs Baseline



Conclusion: Rerouting Padding Tokens Improves Load Balance

As batch size increases, coefficient of variation gets lower when padding tokens are rerouted

The load-balance-optimized forward function is efficient; execution time only slightly increases and accuracy is comparable to baseline

The next step is to test the algorithm on a multi-GPU node and measure latency improvements

Questions?