Fixing the critical flaw in Mixtral of Experts (8x7b)

0. Paper under Review

Mixtral of Experts, 2024 (by Mistral AI): https://arxiv.org/pdf/2401.04088
Cited by 1510

1. Introduction - What is the Flaw?

The Mixture of Experts (MoE) architecture was popularized by Mixtral of Experts to the generic public. Mistral AI (the company which developed it) open-sourced the model, and it performed on par with the then-opensource demon Llama-2-70b.

The core idea behind MoE is to leverage a large number of experts - each handling a subset of the input - to achieve conditional computation. This allows the overall model capacity to scale without a linear increase in computation cost per token. Hence, some of the biggest deep neural nets with 1T+ parameters have been MoE. However, this strength also introduces a critical challenge:

Load Imbalance: Some experts may be overloaded (or underutilized), creating bottlenecks during parallel execution. This causes latency in inference since some experts in the FFN finish their forward pass early while others hold up the process.

I'll explain this with an example: suppose we have 100 tokens (more precisely, their latent representations) freshly enriched from the self-attention layer. They are then passed through the FFN layer, where a gating function distributes these 100 tokens across experts. This distribution is non-uniform, and the next step (the add and layernorm) will only execute when every token has passed through its assigned expert. Hence, load imbalance across experts can cause the most loaded expert to delay the entire operation. What if we could achieve a more even distribution?

In this technical report, I explore the underlying bottlenecks of MoE, review the evolution of expert routing -from the original MoE papers to recent advances like Mixtral and Deepseek V3- and describe my approach for mitigating these imbalances during inference.

1.1 Expert Parallelism in Mixtral of Experts

Mixtral of Experts utilizes **Expert Parallelism** to compute latent representations in the FFN layers. This operation is highly parallelizable, yet the load imbalance issue still persists.

1.2 Deepseek V3 and Redundancy-Based Solutions

Deepseek V3 addresses the load imbalance by duplicating an overloaded expert in memory, effectively creating redundancy to absorb extra tokens. Although this technique balances the load, it increases memory usage and introduces computational redundancy. My aim is to address these issues **without duplicating expert weights in memory**, thereby preserving efficiency and reducing resource overhead.

2. Proposed Approach: Inference-Time Token Rerouting

Rather than extensively training the gating function with additional penalties - which may hurt overall performance -I focus on inference-time tweaks. My method uses a **token drop and rerouting mechanism**. This approach is inspired by a very recent paper on the Straggler Effect (though the authors did not release any code, so I had to implement it myself :)

2.1 Core Ideas

- **Capacity-Aware Token Drop:**
  Each expert is allocated a fixed capacity (determined by the total number of tokens, top-k routing, number of experts, and a configurable capacity factor). If an expert is assigned more tokens than its capacity, the tokens with the lowest routing confidence are dropped.
- **Token Rerouting:**
  Dropped tokens are not simply discarded; they are collected and then rerouted to fallback experts that have spare capacity. This dynamic reassignment helps to mitigate the straggler effect while maintaining overall performance.

2.2 Pseudocode Overview

Below is a high-level pseudocode outline of the capacity-aware token drop and reroute mechanism:

```
# For each expert, check if it got too many tokens.
dropped = {}  # Holds dropped tokens for each expert

for expert in range(num_experts):
    tokens = expert_assignments[expert]  # Tokens assigned to this expert
    cap = compute_capacity(batch_size, seq_length, top_k, num_experts, capacity_factor)

    if len(tokens) > cap:
        # Sort tokens by their routing score (low score = drop)
        sorted_tokens = sort_tokens_by_routing_weight(tokens, routing_weights)
        expert_assignments[expert] = sorted_tokens[:cap]  # Keep the best tokens
        dropped[expert] = sorted_tokens[cap:]

# Reroute dropped tokens to any expert with free capacity.
for token_list in dropped.values():
    for token in token_list:
        for expert in range(num_experts):
            if len(expert_assignments[expert]) < cap:
                expert_assignments[expert].append(token)
                break
```

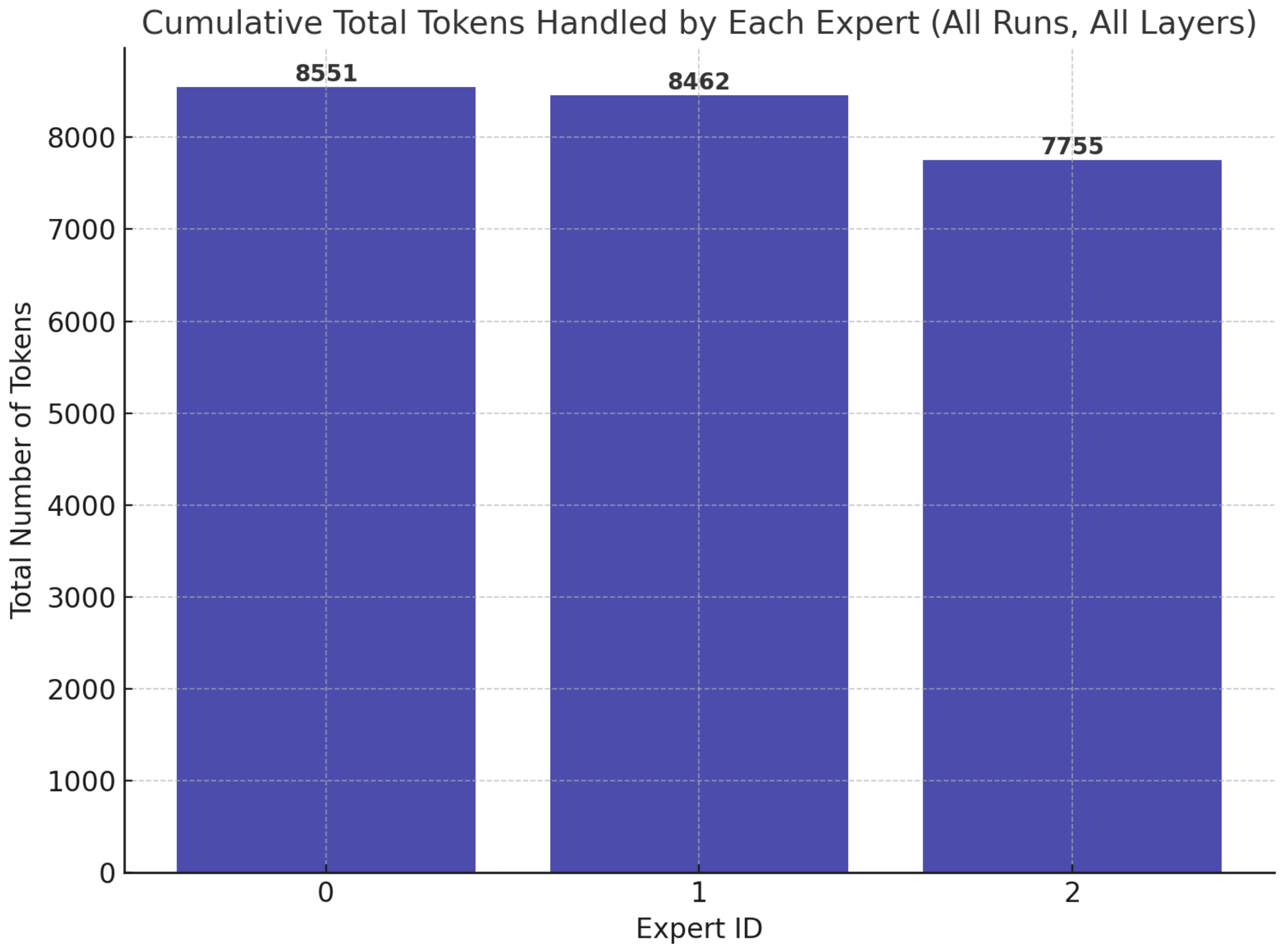
2.3 Implementation Details

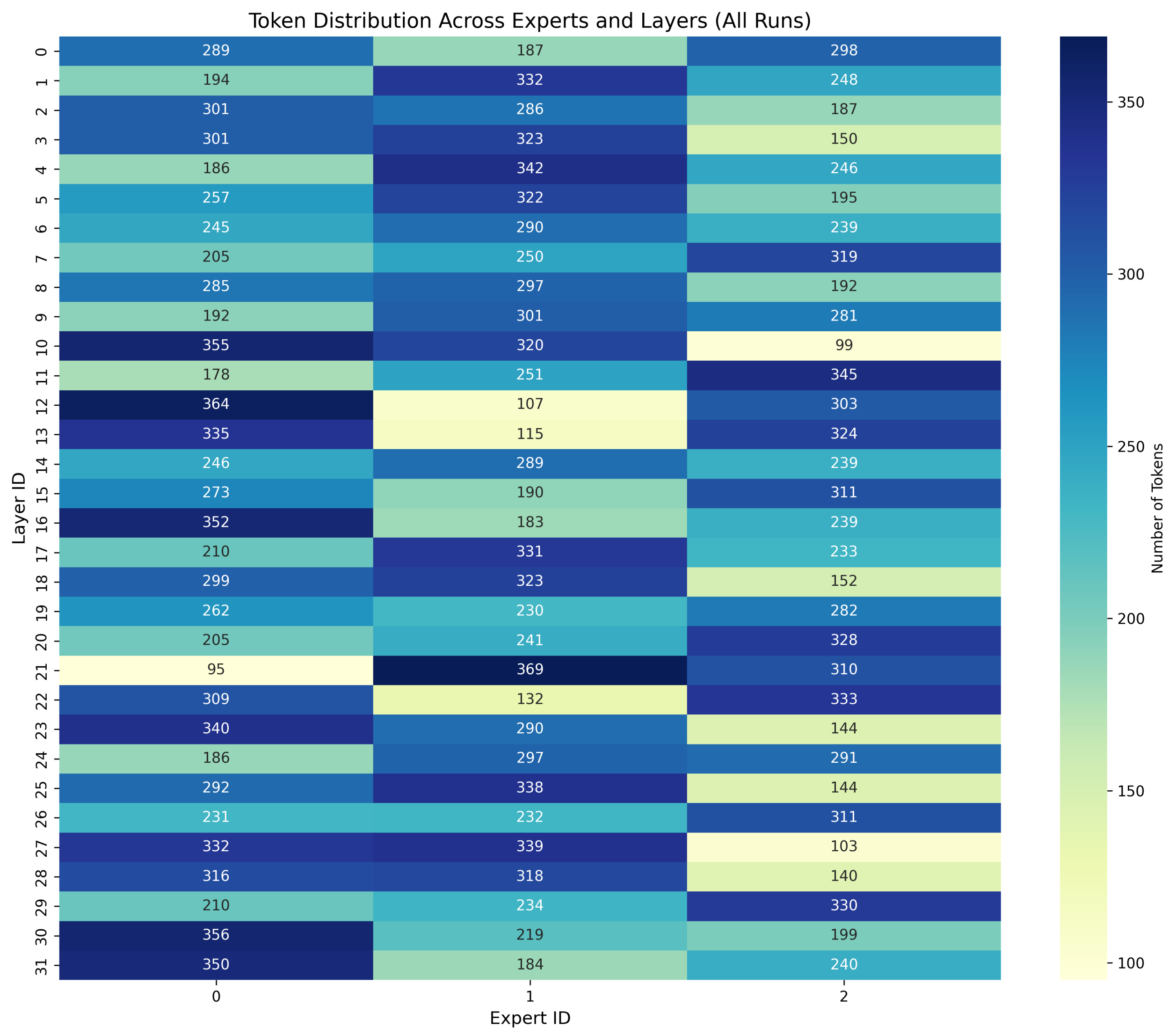
- **What Did I Actually Modify?**
  If you understand the high-level structure of Hugging Face's transformers library, you'll notice that our file of concern is `transformers/src/transformers/models/mixtral/modeling_mixtral.py`. The changes are integrated directly into the `MixtralSparseMOEBlock` module. By cloning the transformers codebase and running in development mode, every modification is immediately reflected during inference.
- **Hardware Constraints and Model Used:**
  Owing to my limited PC, I couldn't make inference work with the 8x7B model. Instead, I used a much smaller MoE (sarashina2.2-3Bx4-moe) which has 3 experts. I ensured that this model (with 3 experts and 32 layers) fits within an 11 GiB GPU memory constraint, with further memory savings achieved via 8-bit quantization.
- **Inference Results:**
  In testing, the modified MoE model demonstrates improved balance across experts without the need for duplicative memory usage. As a sanity check, here's one example output:>

> ```
> Prompt: Write a short story about a space explorer discovering a new planet.
> Response: [{'role': 'user', 'content': 'Write a short story about a space explorer discovering a new planet.'}, {'role': '
> assistant', 'content': "In the vast expanse of the cosmos, an intrepid space explorer named Amelia had always been driven
> by her thirst for discovering new worlds. After her latest missions, she finally received the signal identifying a new pla
> net in her vicinity. It was a world of potential and mystery, just a few light-years away from Earth's home.\n\nAmelia's s
> pacecraft, named the 'Horizon', approached the new planet with great caution. Its surface was a unique  blend of greens an
> d blues, resembling Earth's landscapes but in a cosmic setting. After a series of careful approaches and landings, she des
> cended to a soft-landing on the sallying surface of a vast oasis.\n\nIn the heart of this newfound world, she found a"}]
> ```

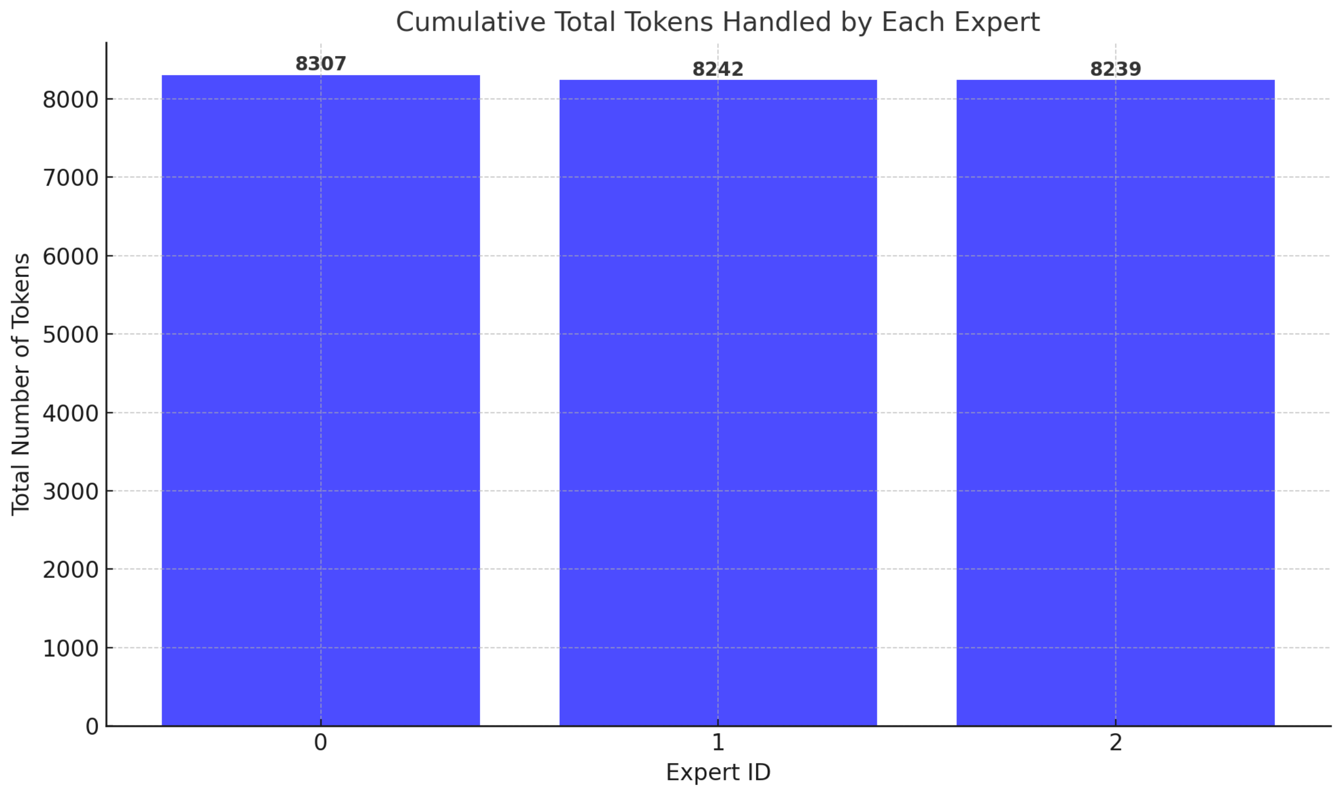
3. Results

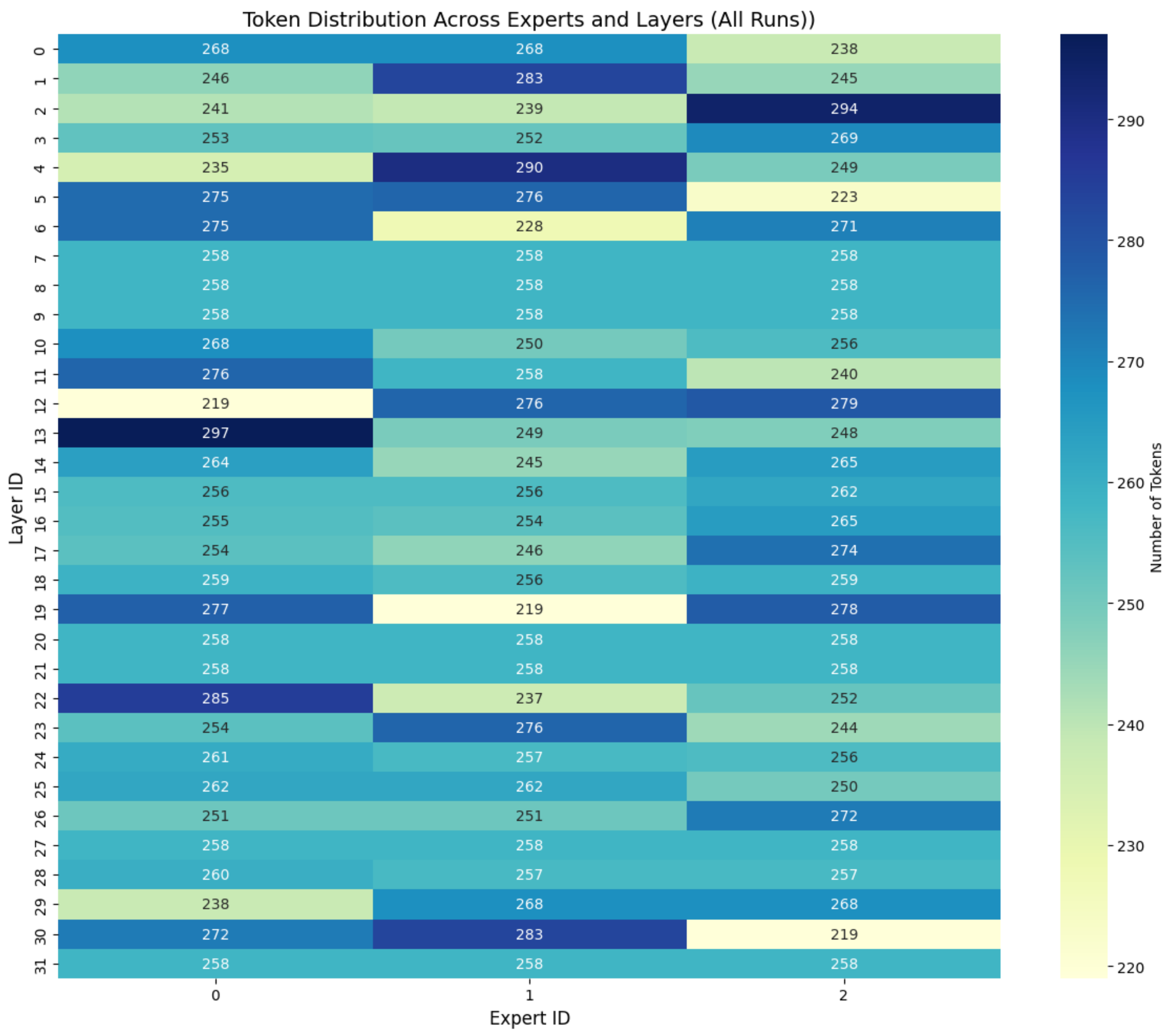
Initially, the frequency of expert activation looked like this:





After the modifications to the transformers module, the overall routing distribution has improved.





To be honest, this doesn't look much impressive yet - this is because I am using a very small MoE that doesn't even have many experts. With only 3 experts, the vanilla token routing is fairly uniform. However, as these models scale and the number of experts increases (up to 64 in larger models like Deepseek V3 or OLMoE-Instruct), we observe a much more uneven distribution. In such cases, the effects of reducing latency while sustaining performance become pronounced.