Training-Free Mixture-of-Agents for CUDA Kernel Translation and Optimization

Arijit Bhattacharjee
Iowa State University
Oak Ridge National Laboratory

Ali Jannesari
Iowa State University

Pedro Valero Lara
William Godoy
Keita Teranishi
Jefferey Vetter
Oak Ridge National Laboratory

I. Poster Summary

1) Motivation: Modern GPU kernel optimization requires balancing performance improvement with the computational cost of optimization itself. While large language models can propose CUDA transformations, applying every possible optimization layer to every kernel can waste tokens, energy, and time. Different kernels often require different optimization depths. Therefore, this poster studies a kernel-aware multi-agent optimization framework that evaluates not only the final optimized CUDA kernel, but also which optimization layers contributed most to the final output and how much each mode costs.

2) Optimization Modes: We compare three optimization modes: *Fixed*, *Adaptive*, and *Collaborative*. In the Fixed mode, each kernel follows a predefined optimization sequence: coalescing, tiling, and atomic-related transformations. This provides a simple and consistent pipeline, but it may apply unnecessary transformations. In the Adaptive mode, a router decides which optimization layers are useful for the current kernel and can stop early when further layers are not needed. In the Collaborative mode, multiple models propose optimizations at each layer, and GPT-OSS aggregates the strongest candidate before moving to the next stage.

3) Kernels Studied: : The evaluation focuses on two CUDA kernels: the Lattice-Boltzmann Method, referred to as LBM or `opt2`, and SYMM, a symmetric matrix-matrix multiplication kernel. These two kernels represent different optimization behaviors. LBM benefits from deeper staged refinement, where later layers continue to shape the final optimized code. SYMM, in contrast, converges earlier: coalescing followed by tiling is sufficient, and an atomic-focused layer is not needed.

4) Performance Progression: The performance results show that optimization depth should depend on the kernel. For LBM, later optimization layers remain important, suggesting that deeper refinement can improve the final generated CUDA kernel. For SYMM, Adaptive optimization achieves the best final performance by stopping after Layer 2. The Adaptive path performs coalescing followed by tiling, reaching a final runtime of 48.75 ms and throughput of 2819.02 GFLOP/s. Since atomic optimization was unnecessary, Adaptive avoids an extra layer without sacrificing performance.

5) Layer-Influence Analysis: To understand which layers shaped the final optimized kernel, we perform a layer-influence analysis. Influence is measured by comparing each intermediate layer output with the final optimized code using similarity-based measures such as token overlap, Jaccard similarity, and LCS-style agreement. Higher influence scores indicate that a layer contributed more strongly to the final kernel structure. The results show that LBM is influenced most by later layers, especially Layer 3 in Fixed and Collaborative modes, while SYMM reaches its dominant final structure earlier, with Layer 2 being the most influential in Adaptive mode.

6) Cost Analysis: The poster also evaluates the cost of optimization using total tokens, power consumption in watt-hours, and optimization time. For LBM, Collaborative optimization incurs higher cost because each layer involves multiple model proposals followed by GPT-OSS aggregation. This additional cost may be useful when deeper multi-agent refinement improves the final kernel. For SYMM, however, Adaptive optimization is the most cost-effective mode, using $21{,}305$ tokens and 8 Wh, compared with $30{,}235$ tokens and 11.5 Wh for Fixed, and $68{,}149$ tokens and 10 Wh for Collaborative.

7) Token-per-Wh Efficiency: Token-per-Wh efficiency measures how much optimization reasoning is generated per unit energy. A higher value indicates better energy-normalized token productivity. Across both LBM and SYMM, GPT-OSS generally achieves the highest token-per-Wh efficiency. However, the efficiency trend depends on both the kernel and the optimization mode. SYMM benefits strongly from Adaptive optimization because unnecessary layers are skipped, while LBM can justify deeper refinement when later layers continue to meaningfully influence the final output.

8) Key Takeaway: The results show that no single optimization strategy is best for every CUDA kernel. LBM benefits from deeper layer refinement, while SYMM converges earlier after coalescing and tiling. This demonstrates the importance of kernel-aware routing and layer-aware stopping. Adaptive optimization is especially promising because it can reduce unnecessary model calls, token usage, energy consumption, and optimization time while still preserving or improving final kernel performance.

9) Conclusion: Overall, this poster shows that practical LLM-driven CUDA optimization should optimize for both performance and cost. By analyzing final runtime, layer influence, token usage, power consumption, and token-per-Wh

efficiency, the framework provides a more complete view of multi-agent GPU optimization. These findings suggest that future GPU optimization systems should dynamically decide which transformations to apply, when to stop, and which models to involve for each kernel.