

Caracterização da Inferência de LLMs Limitada por Memória em Hierarquias de DRAM e Memória Persistente

Luísa Leona Cattai¹, Alexandro Baldassin¹

¹IGCE - Universidade Estadual Paulista (UNESP)

{luisa.leona, alexandro.baldassin}@unesp.br

Abstract. *Inference of Large Language Models (LLMs) in CPU-only environments is fundamentally limited by the capacity and bandwidth of the main memory (DRAM). This work investigates memory tiering, which distributes the three internal components of the Transformer (attention weights, Feed-Forward Network weights, and the KV-Cache) between local DRAM and a higher-capacity, higher-latency tier (Intel Optane DC). Across two 8B LLaMA models with distinct attention architectures (MHA and GQA), we find that the component that dominates throughput degradation depends on the context regime: the FFN dominates under short contexts, whereas the KV-Cache dominates under long contexts. Based on these findings, we formulate five practical allocation guidelines.*

Resumo. *A inferência de Modelos de Linguagem de Grande Escala (LLMs) em ambientes exclusivamente baseados em CPU é fundamentalmente limitada pela capacidade e pela largura de banda da memória principal (DRAM). Este trabalho investiga a estratificação de memória (memory tiering) que distribui os três componentes internos do Transformer (pesos de atenção, pesos da rede Feed-Forward e KV-Cache) entre DRAM local e uma camada de maior capacidade e maior latência (Intel Optane DC). Sobre dois modelos LLaMA de 8B com arquiteturas de atenção distintas (MHA e GQA), constatamos que o componente dominante na degradação de vazão depende do regime de contexto: a FFN domina sob contextos curtos e o KV-Cache sob contextos longos. A partir desses achados, formulam-se cinco diretrizes práticas de alocação.*

1. Introdução

Os LLMs, fundamentados na arquitetura Transformer [Vaswani et al. 2017], revolucionaram o processamento de linguagem natural. Embora o treinamento demande recursos massivos, é na **inferência** que a pressão sobre a memória se manifesta de forma persistente [Miao et al. 2024]. Essa memória distribui-se entre três componentes: os pesos das camadas de **atenção**, os pesos das camadas **Feed-Forward (FFN/MLP)** e o **KV-Cache**, que cresce linearmente com o comprimento do contexto, podendo rivalizar com o tamanho dos próprios pesos [Luo et al. 2025]. Em particular, a natureza autorregressiva da fase de *decode* exige a leitura da totalidade dos pesos a cada token gerado, tornando a geração limitada pela largura de banda da memória (*memory-bound*) [Yuan et al. 2024].

Em ambientes baseados exclusivamente em CPU, recursos do sistema operacional, pesos do modelo e KV-Cache competem pela mesma DRAM. A adição de uma segunda camada de memória endereçável a byte, como módulos Intel Optane

DC [Yang et al. 2020] ou, prospectivamente, dispositivos CXL [Das Sharma et al. 2024], viabiliza cenários antes impossíveis em termos de viabilidade, custo (TCO) e escalabilidade. Neste trabalho, busca-se investigar o comportamento da inferência de LLMs quando dois níveis de memória endereçada a byte, DRAM e Intel Optane DC, estão disponíveis. Assim, pretende-se verificar como a alocação seletiva de componentes internos de um LLM entre DRAM local e memória Optane DC afeta o desempenho e a viabilidade da inferência em ambientes sem aceleradores. As principais **contribuições** são: (i) caracterização empírica granular do impacto de cada componente; (ii) identificação de que o gargalo dominante depende do contexto (FFN em contexto curto, KV-Cache em contexto longo); (iii) formulação de cinco diretrizes práticas de alocação; e (iv) quantificação do efeito da arquitetura GQA, comparando LLaMA 2 (MHA) e LLaMA 3 (GQA).

2. Fundamentação Teórica e Trabalhos Relacionados

Cada bloco do Transformer combina um mecanismo de atenção e um bloco *Feed-Forward*, cujos pesos costumam concentrar cerca de dois terços dos parâmetros do modelo. A inferência ocorre em duas fases com perfis de uso de recursos distintos. Na fase de *prefill*, todos os tokens do *prompt* são processados de uma só vez: as operações são densas e paralelizáveis, tornando essa fase limitada pela capacidade de computação (*compute-bound*), enquanto o KV-Cache é construído. Já na fase de *decode*, os tokens são gerados um a um de forma autorregressiva, e cada novo token exige a leitura da totalidade dos pesos do modelo e do KV-Cache acumulado, passando a ser limitada pela largura de banda da memória (*memory-bound*). Se o modelo for muito grande, a memória DRAM pode não comportar todos os dados necessários à inferência. Nesse caso, o uso de camadas adicionais de memória endereçáveis a byte ajuda a expandir a capacidade disponível, ainda que ao custo de maior latência e menor largura de banda. Na literatura encontramos alguns trabalhos que buscam tratar o problema de memória. A **redução de pegada** via quantização (GPTQ [Frantar et al. 2023]) e compressão de KV-Cache (GQA [Ainslie et al. 2023]) atenua, mas não elimina, o limite de capacidade. A **inferência CPU-only** tem no llama.cpp [Gerganov et al. 2023] sua referência prática, e o LLM in a Flash [Alizadeh et al. 2024] evidencia ganhos ao fazer prefetching dos pesos a partir de SSD. No entanto, nenhum trabalho explora a estratificação *explícita* de componentes do LLM em memória principal heterogênea, em cenário CPU-only, considerando os diversos componentes internos da LLM.

3. Experimentos e Resultados

Os experimentos foram conduzidos em plataforma de duas camadas, DRAM local (256GB DDR3) e Intel Optane DC (1TB), kernel Linux 6.8 de 64 bits, processador Intel(R) Xeon(R) Gold 5317, com 12 cores operando a 3.00GHz. A inferência usou a llama.cpp/GGML estendida para **alocação explícita** de cada componente em DRAM ou Optane. Avaliaram-se dois modelos LLaMA de 8B e quantização equivalente, com arquiteturas distintas: **LLaMA 2 (MHA)** e **LLaMA 3 (GQA)**. As diferentes formas de alocação avaliadas para os componentes da LLM são mostradas na Tabela 1. A métrica principal foi a vazão de *decode* (tokens/s), avaliada nos cinco cenários da Tabela 2, que variam a razão entre tamanho de *prompt*, geração e contexto, cobrindo desde controle (C5) até o pior caso realista (C4). Cada experimento é repetido 10x e a média aritmética é reportada.

Tabela 1. Estratégias de posicionamento usadas nos experimentos.

Config	KV-Cache	Pesos Atenção	Pesos MLP	Racional
Baseline	DRAM	DRAM	DRAM	Referência (tudo na DRAM)
KV	Optane	DRAM	DRAM	KV-Cache tolera Optane?
Attn	DRAM	Optane	DRAM	Atenção tolera Optane?
FFN	DRAM	DRAM	Optane	MLP tolera Optane?
Attn+FFN	DRAM	Optane	Optane	Só KV-Cache na DRAM
KV+Attn	Optane	Optane	DRAM	Só MLP na DRAM
KV+FFN	Optane	DRAM	Optane	Só atenção na DRAM
Tudo	Optane	Optane	Optane	Tudo no Optane (pior caso)

Tabela 2. Cenários de uso para avaliação da estratificação de memória.

Cenário	Caso de uso	Foco do estresse	O que revela sobre estratificação	n_prompt	n_gen	n_ctx
C1	Sumarização, RAG	Leitura de cache grande e estável	Custo fixo por token com cache grande no Optane	4096	128	8192
C2	Geração de texto, código	Cache crescendo continuamente	Degradação progressiva (curva de latência)	128	4096	8192
C3	Chatbot, Q&A	Caso balanceado	Viabilidade prática para uso geral	1024	1024	4096
C4	Multi-turno longo, RAG extenso	Máxima vazão de leitura	Pior caso realista / limite da estratificação	7000	1000	8192
C5	Controle experimental	Nenhum (cache desprezível)	Gargalo fixo do Optane	64	64	512

A Figura 1 sintetiza o **invariante experimental**: o componente de maior impacto não é fixo. Sob **contexto curto** (C2, C3, C5), a **FFN** domina a degradação (coerente com seus $\sim 67\%$ dos parâmetros); sob **contexto longo** (C1, C4), o **KV-Cache** torna-se crítico, à medida que seu crescimento linear passa a dominar o tráfego de memória, comportamento consistente entre MHA e GQA. **Nenhuma combinação** de componentes supera, em vazão, o seu melhor componente isolado, e a combinação de menor degradação muda com o contexto. Por fim, o **GQA reduz substancialmente o custo** de deslocar os pesos de atenção para a camada lenta (linha intermediária do mapa, com degradação baixa e estável), ampliando a margem para configurações que os incluam na Optane.

A partir dos resultados, podemos sintetizar 5 diretrizes principais: **D1**: manter os **pesos da FFN em DRAM**, principal vetor de degradação no *decode*. **D2**: em **contexto curto**, deslocar preferencialmente o **KV-Cache** para a Optane, pois exerce menor pressão que a FFN. **D3**: em **contexto longo**, priorizar a permanência do **KV-Cache em DRAM**, deslocando antes componentes estáticos menos críticos. **D4**: em modelos **GQA**, considerar deslocar os **pesos de atenção** para a camada lenta, aproveitando o baixo custo relativo. **D5**: **evitar combinações** que dispersem múltiplos componentes na camada lenta, optando pelo deslocamento do componente isolado de menor impacto para o regime vigente.

4. Conclusão

Este trabalho apresentou uma caracterização empírica granular do impacto da estratificação explícita de componentes de LLMs em sistemas heterogêneos sem aceleradores, demonstrando que o gargalo de desempenho alterna entre a FFN (contexto curto) e o KV-Cache (contexto longo) e que a arquitetura GQA amplia o espaço de decisão ao baratear o deslocamento dos pesos de atenção. As cinco diretrizes propostas (D1–D5) oferecem orientação prática para maximizar a capacidade efetiva de memória em plataformas de custo moderado, viabilizando a inferência de modelos que, de outra forma, exigiriam hardware significativamente mais dispendioso.

Referências

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the EMNLP*, pages 4895–4901.

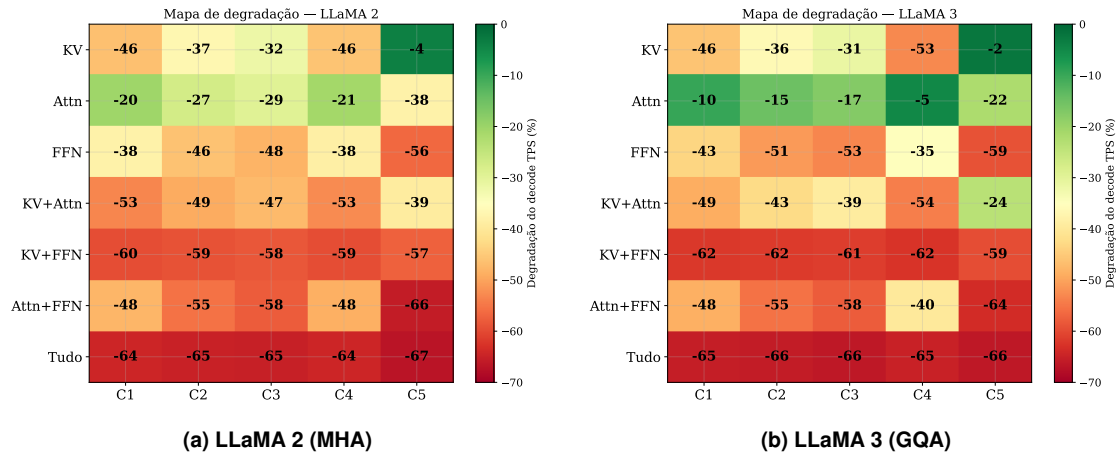


Figura 1. Mapas de calor da degradação relativa do *decode* TPS (%) por configuração (linhas) e cenário (colunas). Tons mais escuros indicam maior degradação. A comparação entre as figuras evidencia o efeito do GQA sobre o espaço de decisão de alocação: no LLaMA 3 (b), as configurações que preservam a KV-Cache em DRAM exibem células sistematicamente mais claras do que no LLaMA 2 (a), sobretudo sob contexto longo (C1, C4).

- Alizadeh, K., Mirzadeh, I., Belenko, D., Khatamifard, S. K., Cho, M., Del Mundo, C. C., Rastegari, M., and Farajtabar, M. (2024). LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd ACL*, pages 12562–12584.
- Das Sharma, D., Blankenship, R., and Berger, D. (2024). An introduction to the compute express link (cxl) interconnect. *ACM Comput. Surv.*, 56(11).
- Frantar, E., Ashkboos, S., Hoeffler, T., and Alistarh, D. (2023). OPTQ: Accurate quantization for generative pre-trained transformers. In *Proceedings of the 11th ICLR*.
- Gerganov, G. et al. (2023). llama.cpp. <https://github.com/ggerganov/llama.cpp>. Accessed: 2025.
- Luo, C., Cai, Z., Sun, H., Xiao, J., Yuan, B., Xiao, W., Hu, J., Zhao, J., Chen, B., and Anandkumar, A. (2025). HeadInfer: Memory-efficient LLM inference by head-wise offloading. *arXiv preprint arXiv:2502.12574*.
- Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Jin, H., Chen, T., and Jia, Z. (2024). Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.
- Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J., and Swanson, S. (2020). An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST '20)*, pages 169–182.
- Yuan, Z., Shang, Y., Zhou, Y., Dong, Z., Zhou, Z., Xue, C., Wu, B., Li, Z., Gu, Q., Lee, Y. J., Yan, Y., Chen, B., Sun, G., and Keutzer, K. (2024). LLM inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*.