Avaliando a Estratificação de Memória na Inferência de LLMs somente em CPU

Luísa Leona Cattai[1], **Alexandro Baldassin**[1], **Allberson Dantas**[2]

[1]IGCE - Universidade Estadual Paulista (UNESP)

[2]Universidade da Integração Internacional da Lusofonia Afro-Brasileira (UNILAB)
Redenção – CE – Brasil

{luisa.leona, alexandro.baldassin}@unesp.br, allberson@unilab.edu.br

Resumo. *A capacidade de DRAM representa um fator limitante de custo e viabilidade para a execução da inferência com Grandes Modelos de Linguagem (LLMs) em ambientes sem aceleradores. Este trabalho investiga o impacto da alocação seletiva dos três principais componentes internos de um LLM (pesos de atenção, pesos da FFN e KV-Cache) entre DRAM local e uma camada de memória de maior capacidade e maior latência, em uma plataforma exclusivamente baseada em CPU. Utilizando o modelo LLaMA 2 7B quantizado em* `Q4_K_M`*, avaliam-se oito estratégias de posicionamento em cinco cenários de uso que variam o tamanho do contexto e a proporção entre* prefill *e* decode*. Os resultados demonstram que não existe um componente universalmente mais tolerante à estratificação: a hierarquia de custos rotaciona ao longo do gradiente de contexto, sendo a KV-Cache o componente mais seguro sob contexto efetivo curto (*$-3{,}9\%$ *de degradação de* decode*) e o mais custoso sob contexto efetivo longo (até* $-46{,}8\%$*). A decisão de alocação é formalizada como um problema bi-objetivo (DRAM liberada* vs. *vazão) resolvido por meio da fronteira de Pareto com um piso de aceitabilidade de vazão, do qual derivam-se diretrizes práticas de alocação sensíveis ao contexto.*

1. Introdução

Grandes Modelos de Linguagem (*Large Language Models* – LLMs), fundamentados na arquitetura Transformer [Vaswani et al. 2017], revolucionaram o processamento de linguagem natural. Seu ciclo de vida compreende duas fases: o treinamento, que ajusta os parâmetros a partir de grandes volumes de dados, e a inferência, na qual o modelo gera respostas a partir de novas entradas. É na inferência, sobretudo, que a pressão sobre a memória se manifesta de maneira persistente [Miao et al. 2024]. A memória consumida durante a inferência distribui-se entre três componentes. Os dois primeiros são estáticos: os pesos das camadas de atenção e os pesos das camadas *Feed-Forward* (FFN), que tipicamente concentram a maior parcela dos parâmetros. Já o terceiro é dinâmico, sendo composto pela KV-Cache (*Key-Value Cache*), que armazena os estados de atenção já computados, evitando seu recálculo durante a geração. Enquanto os pesos permanecem fixos, a KV-Cache cresce linearmente com o comprimento do contexto, podendo atingir volumes comparáveis aos dos próprios pesos em cenários de contextos longos [Luo et al. 2025].

O desafio central da inferência reside na sua natureza autorregressiva, na qual o processador precisa ler da memória a totalidade dos pesos do modelo a cada token

gerado [1]. Essa mecânica torna a geração (*decode*) severamente limitada não pela capacidade aritmética do processador, mas pela largura de banda da memória [Yuan et al. 2024]. Embora aceleradores como GPUs com memória HBM dominem cenários de alto desempenho, sua dependência apresenta barreiras de custo e infraestrutura [Jiang et al. 2025]. Cenários como computação de borda e servidores de menor custo exigem soluções baseadas exclusivamente em CPU [Na et al. 2025], nas quais a DRAM do sistema é o único recurso volátil rápido e precisa acomodar simultaneamente o sistema operacional, os pesos do modelo e a KV-Cache.

Para mitigar esse gargalo, a estratificação de memória (*memory tiering*) organiza a memória em múltiplas camadas com capacidades, latências e custos distintos [Lee et al. 2023]. Um exemplo concreto, adotado neste trabalho, é um sistema composto por DRAM e módulos Intel Optane DC [Yang et al. 2020], cujas características (endereçamento por byte, latência superior à da DRAM porém inferior à de SSDs, maior densidade) são compartilhadas por tecnologias emergentes baseadas no padrão Compute Express Link (CXL) [Das Sharma et al. 2024]. Neste cenário, o problema central consiste em decidir como distribuir os três componentes do modelo entre as camadas de memória disponíveis, equilibrando capacidade, custo e desempenho. A dificuldade reside no fato de que pesos de atenção, pesos da FFN e KV-Cache possuem padrões de acesso e sensibilidades distintas à latência, de modo que deslocar cada um deles para uma camada mais lenta impõe custos de desempenho de natureza e magnitude diferentes. Este trabalho investiga empiricamente como essa alocação seletiva afeta o compromisso entre desempenho de inferência e uso de DRAM em ambientes sem aceleradores, concentrando-se no LLaMA 2 7B [Touvron et al. 2023], que emprega *Multi-Head Attention* (MHA) clássica, e avaliando oito estratégias de posicionamento ao longo de cinco cenários de uso. Diferentemente dos trabalhos existentes, que partem da memória do acelerador e tratam quase exclusivamente a KV-Cache, neste trabalho não se considera nenhum tipo de acelerador. Além disso, tanto os pesos quanto a KV-Cache são objetos de decisão independentes, podendo ser alocados em uma camada rápida em DRAM ou em uma mais lenta ainda endereçável por byte (Optane DC).

As principais contribuições deste artigo são:

- a caracterização empírica granular do impacto de cada componente interno em um sistema com memória estratificada, isolando a degradação de vazão causada pelo deslocamento individual e combinado da KV-Cache, dos pesos de atenção e da FFN;
- a identificação de que o gargalo da estratificação é sensível ao contexto e governado por um único mecanismo denominado *contexto efetivo*, caracterizado pelo volume de bytes relidos por passo de geração;
- a formalização da alocação como problema bi-objetivo, resolvido por meio da fronteira de Pareto combinada a um piso de aceitabilidade de vazão, do qual emergem diretrizes práticas de decisão.

O resto do artigo é organizado da seguinte forma. Na Seção 2 apresentam-se os principais conceitos e trabalhos relacionados. A Seção 3 descreve a metodologia experimental utilizada para realização dos experimentos. A Seção 4 apresenta os principais

[1] Há técnicas recentes que geram múltiplos tokens por passo, como a decodificação especulativa [Leviathan et al. 2023], mas aqui considera-se apenas a geração token a token.

resultados conseguidos nos experimentos, descreve as melhores escolhas de alocação que otimizam a quantidade de memória e desempenho por meio da fronteira de Pareto, formulando diretrizes práticas de uso. Por último, as considerações finais são apresentadas na Seção 5.

2. Conceitos e Trabalhos Relacionados

Esta seção começa descrevendo os principais conceitos usados no restante do artigo (Seções 2.1 e 2.2), seguido pelos trabalhos relacionados (Seção 2.3).

2.1. Inferência de LLMs e ocupação de memória

No núcleo da arquitetura Transformer, cada camada combina um mecanismo de Atenção, tradicionalmente o *Multi-Head Attention* (MHA) [Vaswani et al. 2017], e um bloco *Feed-Forward* (FFN), responsável por armazenar a maior parte do conhecimento factual do modelo [Geva et al. 2021]. A distribuição de parâmetros entre esses blocos é estrutural e pode ser estimada analiticamente. Sendo d a dimensão do modelo e d_{ff} a dimensão oculta da FFN, o bloco de Atenção *Multi-Head* soma $P_{attn} \approx 4d^2$, enquanto a FFN contribui com $P_{ffn} \approx 2\,d\,d_{ff}$. Adotando a razão canônica $d_{ff} = 4d$, obtém-se:

$$\frac{P_{ffn}}{P_{attn} + P_{ffn}} \approx \frac{8d^2}{4d^2 + 8d^2} = \frac{2}{3} \approx 67\%. \tag{1}$$

Aproximadamente dois terços dos parâmetros residem, portanto, nas camadas *Feed-Forward*, contra um terço na Atenção. A inferência divide-se em duas fases com perfis de hardware opostos. A fase de *Prefill* processa o *prompt* em paralelo, sendo limitada pela capacidade computacional (*compute-bound*). A fase de *Decode* gera um token por vez, exigindo a leitura de todas as matrizes de pesos a cada token gerado, tornando-se severamente limitada pela largura de banda da memória (*memory-bound*). Como a FFN concentra a maioria dos parâmetros, ela domina o tráfego de memória nessa fase. Além dos pesos estáticos, a inferência autorregressiva depende da KV-Cache, que cresce linearmente com o comprimento da sequência. Para reduzir a pegada de memória, aplicam-se técnicas de quantização [Frantar et al. 2023], como os formatos suportados pelo `llama.cpp`/GGML [Gerganov et al. 2023] (*e.g.*, `Q4_K_M`), que comprimem o modelo com degradação marginal de qualidade [Lin et al. 2024].

2.2. Memórias heterogêneas e estratificação

O limite de escalabilidade e o alto custo das memórias DRAM têm feito com que tecnologias alternativas sejam exploradas [Maruf et al. 2023, Zhong et al. 2024]. Dentre essas tecnologias recentes encontram-se a *High Bandwidth Memory* (HBM) e as memórias persistentes (*Persistent Memory – PM*) [Lee et al. 2016, Baldassin et al. 2021]. Entre as PMs, a Intel Optane DC é praticamente seu único principal representante comercial, conectando-se aos canais de memória via *slots* DIMM e permitindo assim endereçamento por byte com alta densidade. Como referência, a Tabela 1 mostra uma comparação entre DRAM e a Intel Optane DC, na qual observam-se maior latência e menor largura de banda para a Optane, principalmente na escrita (note o comportamento assimétrico). Embora a linha Optane tenha sido descontinuada, seus conceitos arquiteturais constituem a base do padrão CXL [Das Sharma et al. 2024], cujos dispositivos de expansão de memória

Tabela 1. Comparativo de desempenho entre DRAM DDR4 e Intel Optane DC PMem em modo *App Direct*, por módulo DIMM [Yang et al. 2020].

Métrica	DRAM DDR4	Optane PMem
Latência de leitura (aleatória)	≈80 ns	≈346 ns (4,3×)
Banda de leitura	≈25 GB/s	≈6,6 GB/s (3,8× menor)
Banda de escrita	≈25 GB/s	≈2,3 GB/s (10,9× menor)
Endereçamento	Por byte	Por byte
Volatilidade	Volátil	Persistente

(Tipo 3) são mapeados como nós NUMA sem CPU local, com perfil de desempenho análogo ao da Optane em modo *App Direct*. Assim, apesar de os experimentos deste artigo terem usado o dispositivo da Intel, acredita-se que haja equivalência funcional com os emergentes dispositivos baseados no CXL, e que os resultados obtidos nos experimentos reportados aqui permaneçam válidos.

Os diversos tipos de memória presentes em sistemas atuais criam um ambiente heterogêneo no qual a memória fica estratificada. Decidir em qual camada de memória um determinado dado deve residir torna-se um aspecto fundamental para determinar o desempenho do sistema [Lee et al. 2023]. Algoritmos que atuam em um ambiente com memória estratificada decidem quais dados residem em qual nível físico, oportunisticamente posicionando dados quentes nos níveis rápidos e dados frios nos de maior capacidade. As abordagens dividem-se em *transparente*, em que o *kernel* migra páginas reativamente, e *explícita*, em que a aplicação atribui cada estrutura ao nível desejado. A inferência de LLMs é particularmente adequada à abordagem explícita: seus padrões de acesso são estruturalmente previsíveis (a distribuição de parâmetros é determinística e a KV-Cache cresce monotonicamente), permitindo decisões de alocação *a priori* que superam a migração reativa em eficiência e previsibilidade de latência. Este trabalho adota, portanto, a abordagem explícita para investigar diferentes formas de posicionamento dos pesos de uma LLM.

2.3. Trabalhos Relacionados

A inferência de LLMs é fundamentalmente limitada por memória: modelos com bilhões de parâmetros exigem capacidade e largura de banda que frequentemente excedem os recursos disponíveis, tanto em GPUs quanto em servidores baseados apenas em CPU. A literatura ataca esse problema por três frentes complementares: reduzir o volume de dados residente em memória, mover dados entre dispositivos (*offloading*) e expandir a memória com tecnologias heterogêneas.

A primeira frente reduz o volume de dados. A quantização diminui a precisão dos parâmetros: GPTQ [Frantar et al. 2023] opera em INT4 com informação de segunda ordem e AWQ [Lin et al. 2024] protege canais *salient* em INT3–INT4; o formato GGUF do `llama.cpp` [Gerganov et al. 2023] viabiliza a execução em hardware convencional. No eixo da KV-Cache, GQA [Ainslie et al. 2023] e MQA [Shazeer 2019] reduzem seu tamanho ao compartilhar projeções, e técnicas de despejo como H_2O [Zhang et al. 2023] e StreamingLLM [Xiao et al. 2024] descartam tokens menos relevantes. Tais técnicas são *ortogonais* à estratificação: mesmo quantizados, modelos de grande porte demandam dezenas de gigabytes e ainda se beneficiam de posicionamento inteligente.

A segunda frente, mais explorada, usa DRAM e armazenamento como extensão da memória da GPU. FlexGen [Sheng et al. 2023] tratou o *offloading* de forma holística, movendo pesos, KV-Cache e ativações entre GPU, CPU e disco via escalonamento por programação linear. Trabalhos posteriores refinaram o mecanismo com foco na KV-Cache, que domina a movimentação em contextos longos: KV-Drive [Lin et al. 2025a] distribui blocos entre HBM, DRAM e SSD por frequência de acesso; Ganjihal [Ganjihal 2025] propõe hierarquia de seis camadas com predição Bayesiana; e Kareto [Zheng et al. 2025] formula a estratificação da KV-Cache como otimização multi-objetivo por fronteira de Pareto. FlexInfer [Na et al. 2025] explora aceleradores AMX da CPU, e HybridGen [Lin et al. 2025b] estende a atenção para execução colaborativa CPU–GPU sobre memória CXL. Todos, porém, são *centrados na GPU*. O posicionamento é formulado do ponto de vista do escalonamento para a GPU e não se aplica diretamente a cenários exclusivamente em CPU, cuja hierarquia e padrões de acesso são distintos.

Um corpo crescente de trabalhos aborda a inferência sem GPU, visando implantação em servidores de baixo custo, dispositivos de borda e *smartphones*. O `llama.cpp` [Gerganov et al. 2023] é a implementação de referência, suportando quantização GGUF e utilizando `mmap` para delegar ao sistema operacional a paginação implícita entre DRAM e disco. Especialmente relevante para este trabalho é o *LLM in a Flash* [Alizadeh et al. 2024], que trata o cenário em que o modelo excede a DRAM disponível, armazenando pesos em memória flash e carregando-os sob demanda. Na mesma linha, FlexInfer [Du et al. 2025], também construído sobre o `llama.cpp`, combina *prefetching* assíncrono por camada, particionamento de tensores e preservação seletiva em DRAM para escalar o desempenho com o *budget* de memória disponível. Já o CENT [Gu et al. 2025] dispensa a GPU combinando expansão via CXL com unidades de processamento em memória (PIM); nesse caso, porém, a heterogeneidade serve como veículo para o PIM, e não como mecanismo de posicionamento diferenciado de estruturas de dados entre camadas. Esses trabalhos demonstram a viabilidade e o interesse crescente na inferência apenas em CPU, e o *LLM in a Flash* é particularmente influente ao mostrar que modelar as características do meio de armazenamento pode gerar ganhos expressivos, raciocínio que este trabalho estende para a memória principal heterogênea. Entretanto, todos tratam a memória principal como *homogênea* (DRAM pura) e posicionam dados apenas entre DRAM e armazenamento secundário (SSD ou flash), com latências na ordem de microssegundos a milissegundos.

Os trabalhos mais similares a este investigam o posicionamento de dados em memória heterogênea. Fang et al. [Fang et al. 2025] oferecem a primeira formalização analítica do posicionamento dinâmico da KV-Cache entre HBM e DRAM, derivando um limite superior teórico de desempenho. Beluga [Yang et al. 2025], da Alibaba Cloud, e o trabalho de Tang et al. [Tang et al. 2024], da ByteDance, empregam *pools* de memória via CXL para armazenar a KV-Cache além da capacidade da DRAM do hospedeiro, enquanto o PAM [Liu et al. 2025] coordena dispositivos PIM em uma arquitetura hierárquica. Esses trabalhos, todavia, compartilham três limitações em relação ao escopo aqui proposto: são centrados na GPU (a hierarquia parte da HBM), focam exclusivamente na KV-Cache, e dependem de hardware especializado (CXL-PIM, interconexões NVLink) nem sempre disponível em servidores.

Desta forma, de acordo com a literatura consultada, não se encontra nenhum trabalho existente que combina simultaneamente: (i) inferência *apenas em CPU*, sem dependência de GPU; (ii) posicionamento diferenciado tanto de *pesos* quanto de *KV-Cache*; e (iii) uso de *memória principal heterogênea* (DRAM combinada com PM ou CXL), sem hardware customizado como PIM. Este artigo, portanto, é pioneiro ao considerar um ambiente com memória estratificada operando apenas com CPU, abrangendo tanto os pesos da LLM quanto a KV-Cache, desta forma caracterizando empiricamente como a sensibilidade de cada componente à estratificação varia com o regime operacional.

3. Metodologia

Esta seção descreve os principais pontos da metodologia empregada nos experimentos de estratificação dos componentes de uma LLM (pesos de atenção, pesos da FFN e KV-Cache), cujos resultados são apresentados na próxima seção. Todos os experimentos foram realizados em um ambiente com Linux 64 bits (kernel 6.8), equipado com dois processadores Intel Xeon Gold 5317 (12 *cores* cada, 3,00 GHz), 256 GB de DRAM e 8 módulos de memória Intel Optane DC (série 200), totalizando 1 TB. Para mitigar ruídos sistêmicos, apenas um processador foi utilizado (máximo de 12 *cores*), evitando efeitos NUMA; os modos *Turbo Boost* e *hyperthreading* foram desabilitados; e uma execução de *warm-up* precedeu cada bateria de experimentos para preenchimento das *caches* e TLB. Os dados foram coletados ao longo de $N = 10$ repetições independentes por configuração. As métricas são reportadas como média μ, acompanhada do desvio-padrão amostral σ e de intervalos de confiança (IC) de 95%. Como critério de distinguibilidade, adota-se a não-sobreposição dos ICs de 95%, deliberadamente mais conservador que um teste-t pareado, dada a baixa variância das medições de *decode*. Distingue-se, ao longo da análise, entre significância estatística e relevância operacional (magnitude suficiente para alterar decisões de alocação). Os resultados da próxima seção são reportados como vazões de *prefill* e *decode*, em tokens por segundo (TPS).

3.1. Modelo e software de inferência

Este trabalho emprega o LLaMA 2 7B [Touvron et al. 2023] no formato de quantização `Q4_K_M`. A escolha justifica-se pela disponibilidade de pesos abertos e inspecionáveis, condição indispensável para a manipulação manual de tensores, pelo suporte de primeira classe no ecossistema `llama.cpp`/GGML, e por empregar o mecanismo de Atenção *Multi-Head* (MHA) clássico, que acentua o papel dos pesos de atenção na análise. O LLaMA 2 7B possui janela de contexto de 4.096 *tokens* e vocabulário de 32.000 *tokens*, adotando as otimizações RMSNorm, ativação SwiGLU e codificação posicional rotativa (RoPE). O software de inferência foi o `llama.cpp` (*commit* 47a268e), escolhido por sua maturidade na execução de modelos quantizados em CPU. O código original foi estendido neste trabalho para permitir a alocação seletiva dos três blocos (pesos de atenção, pesos da FFN e KV-Cache) entre DRAM e Optane, por meio de novos argumentos de linha de comando. O posicionamento dos pesos estáticos é especificado por intervalos de camadas, com controle independente dos subcomponentes de atenção (`.attn`) e FFN (`.ffn`); o posicionamento da KV-Cache é controlado por opção dedicada. A residência em Optane é obtida mapeando a memória em um arquivo sobre sistema de arquivos com suporte a DAX via `mmap`, contornando o *page cache* do sistema operacional, de modo que a inferência ocorre integralmente em CPU, sem cópias intermediárias. Para garantir reprodutibilidade, a amostragem operou em modo *Greedy Decoding* (temperatura 0,0,

Tabela 2. Estratégias de posicionamento avaliadas.

Config.	KV-Cache	Pesos Atenção	Pesos FFN
Baseline	DRAM	DRAM	DRAM
KV	Optane	DRAM	DRAM
Attn	DRAM	Optane	DRAM
FFN	DRAM	DRAM	Optane
KV+Attn	Optane	Optane	DRAM
KV+FFN	Optane	DRAM	Optane
Attn+FFN	DRAM	Optane	Optane
Tudo	Optane	Optane	Optane

seed 0), com `--ignore-eos` para forçar a geração da meta exata de *tokens* sobre um *prompt* sintético padronizado.

3.2. Estratégias de posicionamento

Consideram-se três elementos alocáveis (pesos da FFN, pesos de atenção e KV-Cache), cada um em DRAM ou Optane, totalizando oito configurações (Tabela 2). A configuração `Baseline` mantém tudo em DRAM (referência de melhor desempenho); as configurações isoladas (`KV`, `Attn`, `FFN`) deslocam um único elemento para a Optane; as combinações de dois elementos mantêm apenas um em DRAM; e a configuração `Tudo` representa o pior caso.

3.3. Cenários de uso

Cinco cenários (Tabela 3) cobrem uma ampla gama de uso, determinados pelo tamanho do *prompt* (`n_prompt`) e da resposta gerada (`n_gen`). Eles variam a proporção entre *prefill* e *decode* e o tamanho da KV-Cache acessada. Note que, como o LLaMA 2 possui janela nativa de 4.096 *tokens*, o cenário C4 é efetivamente truncado a $\sim$4.096 *tokens* neste modelo. Para ordenar os cenários por intensidade de tráfego de estado, define-se o *contexto efetivo* como a quantidade média de KV-Cache lida por passo de *decode*. Para um *prompt* de p *tokens*, geração de g *tokens* e janela nativa W (aqui $W = 4096$), tem-se

$$\overline{\mathrm{ctx}} \;=\; \frac{1}{g}\sum_{i=1}^{g} \min\bigl(p+i,\; W\bigr). \tag{2}$$

Diferentemente do contexto máximo ($\min(p+g, W)$), o contexto efetivo captura a distribuição do estado ao longo da geração. Assim, sob *prompt* longo a cache já parte saturada ($\overline{\mathrm{ctx}} \approx W$), ao passo que sob *decode* longo ela cresce a partir de valor baixo, resultando em média inferior. É o contexto efetivo, e não o máximo, a variável que ordena os cenários nas análises da Seção 4. Entre estes cenários, C1 força um *prefill* pesado com KV-Cache grande e estável, isolando o custo de ler uma cache grande. C2 revela a degradação progressiva conforme a cache cresce durante o *decode*. C3 é o caso balanceado de referência prática. C4 é o pior caso realista, com a KV-Cache quase máxima durante toda a geração. C5 é o controle mínimo, com KV-Cache desprezível, isolando o *overhead* fixo da Optane.

Tabela 3. Cenários de uso para avaliação da estratificação.

Cenário	Caso de uso	n_prompt	n_gen
C1	Sumarização, RAG	4096	128
C2	Geração de texto, código	128	4096
C3	Chatbot, Q&A (balanceado)	1024	1024
C4	Multi-turno longo, RAG extenso	7000	1000
C5	Controle experimental	64	64

Tabela 4. Vazão média (tokens/s) por cenário e configuração para o modelo LLaMA 2. Valores em média $\pm$ desvio-padrão.

Config.	C1	C2	C3	C4	C5
Decode TPS					
Baseline	$7{,}7 \pm 0{,}04$	$10{,}2 \pm 0{,}03$	$11{,}1 \pm 0{,}02$	$7{,}9 \pm 0{,}03$	$15{,}3 \pm 0{,}02$
KV	$4{,}1 \pm 0{,}01$	$6{,}4 \pm 0{,}01$	$7{,}6 \pm 0{,}02$	$4{,}3 \pm 0{,}02$	$14{,}7 \pm 0{,}02$
Attn	$6{,}1 \pm 0{,}01$	$7{,}4 \pm 0{,}01$	$7{,}9 \pm 0{,}01$	$6{,}2 \pm 0{,}01$	$9{,}5 \pm 0{,}01$
FFN	$4{,}8 \pm 0{,}01$	$5{,}5 \pm 0{,}01$	$5{,}8 \pm 0{,}01$	$4{,}8 \pm 0{,}01$	$6{,}7 \pm 0{,}01$
KV+Attn	$3{,}6 \pm 0{,}01$	$5{,}2 \pm 0{,}01$	$5{,}9 \pm 0{,}01$	$3{,}7 \pm 0{,}00$	$9{,}3 \pm 0{,}01$
KV+FFN	$3{,}1 \pm 0{,}04$	$4{,}2 \pm 0{,}03$	$4{,}6 \pm 0{,}02$	$3{,}2 \pm 0{,}04$	$6{,}6 \pm 0{,}01$
Attn+FFN	$4{,}0 \pm 0{,}01$	$4{,}5 \pm 0{,}01$	$4{,}7 \pm 0{,}00$	$4{,}0 \pm 0{,}01$	$5{,}2 \pm 0{,}01$
Tudo	$2{,}7 \pm 0{,}02$	$3{,}6 \pm 0{,}01$	$3{,}9 \pm 0{,}01$	$2{,}8 \pm 0{,}01$	$5{,}1 \pm 0{,}01$
Prefill TPS					
Baseline	$28{,}6 \pm 0{,}11$	$85{,}2 \pm 0{,}46$	$69{,}9 \pm 0{,}33$	$30{,}9 \pm 0{,}06$	$83{,}7 \pm 1{,}75$
KV	$10{,}4 \pm 0{,}10$	$86{,}2 \pm 0{,}31$	$65{,}2 \pm 1{,}85$	$11{,}6 \pm 0{,}12$	$83{,}3 \pm 0{,}96$
Attn	$31{,}1 \pm 0{,}08$	$86{,}7 \pm 0{,}44$	$74{,}0 \pm 0{,}23$	$33{,}5 \pm 0{,}08$	$82{,}7 \pm 0{,}96$
FFN	$30{,}1 \pm 0{,}05$	$81{,}5 \pm 0{,}18$	$70{,}1 \pm 0{,}49$	$32{,}6 \pm 0{,}04$	$77{,}3 \pm 0{,}57$
KV+Attn	$10{,}8 \pm 0{,}04$	$87{,}0 \pm 0{,}35$	$70{,}1 \pm 1{,}16$	$12{,}0 \pm 0{,}10$	$83{,}1 \pm 0{,}58$
KV+FFN	$10{,}7 \pm 0{,}09$	$81{,}3 \pm 0{,}25$	$65{,}0 \pm 1{,}62$	$11{,}8 \pm 0{,}15$	$77{,}2 \pm 0{,}40$
Attn+FFN	$30{,}4 \pm 0{,}06$	$81{,}1 \pm 0{,}24$	$70{,}6 \pm 0{,}20$	$32{,}8 \pm 0{,}05$	$77{,}4 \pm 0{,}50$
Tudo	$10{,}7 \pm 0{,}03$	$80{,}8 \pm 0{,}16$	$66{,}1 \pm 1{,}60$	$11{,}9 \pm 0{,}09$	$76{,}5 \pm 0{,}95$

4. Análise experimental

A análise experimental examina dois aspectos das diferentes estratégias de posicionamento (Tabela 2): o desempenho (vazão da inferência) e o consumo de memória. O eixo de memória é estrutural (estimável *a priori* a partir dos hiperparâmetros), enquanto o de desempenho é empírico. Para caracterizar o eixo empírico, elencam-se três questões de pesquisa (QP): **QP1** – existe um componente sistematicamente mais tolerante à estratificação? **QP2** – como a sensibilidade varia com o regime de operação? **QP3** – há configurações combinadas que reduzam o uso de DRAM mantendo degradação aceitável? A Tabela 4 oferece a visão panorâmica da vazão coletada para o LLaMA 2, e todas as degradações relativas reportadas adiante tomam por referência a `Baseline` do respectivo cenário. Os cenários são ordenados pelo contexto efetivo (Eq. 2) em todas as tabelas a seguir.

4.1. QP1 – Existência de um componente universalmente mais tolerante

A QP1 investiga se, ao alocar isoladamente cada componente na Optane, existe um componente sistematicamente mais tolerante. A análise concentra-se no *decode*, fase em que a KV-Cache é lida repetidamente e cujo comportamento é *memory-bound*, tornando os

Tabela 5. Degradação relativa do *decode* TPS (%) das configurações isoladas face à `Baseline`, por cenário (ordenados por contexto efetivo crescente), para o LLaMA 2. O componente *mais tolerante* em cada cenário aparece em negrito.

Comp.	C5 ∼96	C3 ∼1536	C2 ∼2112	C1 ∼4096	C4 ∼4096
`KV`	**−3,9**	−31,5	−37,3	−46,8	−45,6
`Attn`	−37,9	**−28,8**	**−27,5**	**−20,8**	**−21,5**
`FFN`	−56,2	−47,7	−46,1	−37,7	−39,2

Tabela 6. Penalidade da configuração `KV` sobre o *decode* TPS, por cenário, para o LLaMA 2. Cenários ordenados por contexto efetivo crescente.

Cenário	Ctx. efetivo	Baseline	Degr.
C5 (Controle)	∼96	15,30	−3,9%
C3 (Conversação)	∼1536	11,14	−31,5%
C2 (Geração)	∼2112	10,18	−37,3%
C1 (Sumarização)	∼4096	7,71	−46,8%
C4 (Estresse)	∼4096	7,86	−45,6%

componentes diretamente comparáveis. O achado central é que *não existe* um componente universalmente mais tolerante. A hierarquia de custos entre `KV`, `Attn` e `FFN` depende do regime de operação (Tabela 5).

Sob contexto curto (C5), a hierarquia é `KV` < `Attn` < `FFN`. Isolar a KV-Cache custa apenas −3,9%, tornando-a o componente mais tolerante, enquanto a FFN é a mais penalizadora (−56,2%). Com contexto mínimo, o estado da KV-Cache é desprezível e quase não gera tráfego para a Optane, ao passo que os pesos de FFN, de tamanho fixo, são relidos integralmente a cada passo. Sob contexto longo (C1, C4), a ordem se inverte para `Attn` < `FFN` < `KV`. A Atenção torna-se a mais tolerante e a KV-Cache, a mais cara. A FFN nunca ocupa a posição de mais tolerante, caracterizando-se como a candidata menos atraente à estratificação isolada em qualquer regime. É importante notar que o comportamento observado não significa que os pesos em `Attn` e `FFN` se tornem mais baratos sob contexto longo. O custo absoluto de relê-los a cada passo é aproximadamente fixo. O que muda é o referencial: a `Baseline` desacelera com o crescimento da KV-Cache (de 15,3 tps em C5 para 7,9 tps em C4), diluindo o peso relativo da penalidade fixa dos pesos, ao passo que o custo da `KV` cresce em termos absolutos por aumentar o volume de estado relido por token. A composição desses dois movimentos inverte a hierarquia ao longo do gradiente de contexto. Assim, qualquer conclusão sobre "o componente mais seguro de deslocar" extraída de um único cenário seria, por construção, enviesada.

4.2. QP2 – Variação da sensibilidade com o regime de operação

A QP2 caracteriza a dependência da penalidade em relação ao regime. A discussão concentra-se na KV-Cache, único componente cuja penalidade é modulada pelo contexto no sentido próprio do termo. A Tabela 6 consolida a penalidade da configuração `KV` sobre o *decode*, organizada pelo tamanho do contexto.

A penalidade associada à KV-Cache não é constante, mas cresce de forma consis-

Tabela 7. Magnitude da degradação relativa do *decode* TPS (%) face à `Baseline` do respectivo cenário, para as combinações de dois ou mais componentes, no LLaMA 2. Cenários ordenados por contexto efetivo crescente. A combinação menos penalizadora em cada cenário aparece em negrito.

Cenário	Ctx. efetivo	`KV+Attn`	`Attn+FFN`	`KV+FFN`	`Tudo`
C5 (Controle)	∼96	**−39,2**	−66,0	−56,9	−66,7
C3 (Conversação)	∼1536	**−46,8**	−57,7	−58,6	−64,9
C2 (Geração)	∼2112	**−49,0**	−55,9	−58,8	−64,7
C1 (Sumarização)	∼4096	−53,2	**−48,1**	−59,7	−64,9
C4 (Estresse)	∼4096	−53,2	**−49,4**	−59,5	−64,6

tente com o contexto efetivo (Eq. 2). Em C5, a degradação é inferior a 4%; à medida que o contexto efetivo cresce (C5 → C3 → C2 → C1), aproxima-se de −46,8%. A única quebra de monotonicidade ocorre entre C1 e C4 (−46,8% e −45,6%), valores estatisticamente indistinguíveis e coerentes com seu contexto efetivo idêntico (∼4.096 *tokens*). Note que em C4 o *prompt* de 7000 *tokens* é truncado à janela nativa, saturando a KV-Cache desde o primeiro passo, tal como em C1.

A comparação entre C1 e C2 isola o efeito da distribuição do contexto. Ambos atingem o mesmo contexto máximo (∼4.096 *tokens*), mas diferem no contexto efetivo. C1 concentra o contexto no *prompt* (4096) com geração curta (128), mantendo $\overline{\text{ctx}} \approx 4096$; C2 inverte, partindo de 128 e atingindo 4096 apenas ao final ($\overline{\text{ctx}} \approx 2112$). Coerentemente, C1 degrada mais que C2 (−46,8% contra −37,3%). Em C1 cada passo de *decode* relê um estado grande durante toda a geração, ao passo que em C2 a cache parte de quase nada. A variável governante, portanto, não é o tamanho final do contexto, mas a leitura média de estado por passo, precisamente o contexto efetivo (Eq. 2).

Quanto ao *prefill*, no LLaMA 2 a fase já degrada acentuadamente sob contexto moderado. Em C1, isolar a KV-Cache reduz o *prefill* de 28,6 para 10,4 tps (−63,6%, Tabela 4), indicando que o modelo, por empregar MHA, opera próximo do limite de memória mesmo em contexto moderado. Confirma-se que a viabilidade da estratificação está condicionada ao regime de contexto, à fase de inferência e à margem de memória do par modelo/dispositivo.

4.3. QP3 – Configurações combinadas

A QP3 investiga se configurações combinadas mantêm degradação aceitável. No *decode*, nenhuma configuração combinada atinge vazão superior à do melhor de seus componentes isolados. Assim, deslocar dois ou mais blocos para a Optane jamais compensa a carga adicional. Quando a economia de DRAM obtida por um único componente já é suficiente, adicionar um segundo só piora a vazão. No *prefill*, as diferenças entre combinações e isolados são majoritariamente indistinguíveis, coerentemente com a natureza *compute-bound* da fase.

A questão prática relevante surge quando o espaço em DRAM *obriga* o deslocamento de dois componentes. Neste cenário, qual combinação minimiza a degradação? Emergem dois padrões invariantes. Primeiro, a configuração `Tudo` é sistematicamente a mais penalizadora, e `KV+FFN` ocupa a segunda faixa de maior degradação (Tabela 7). Qualquer combinação que desloque a KV-Cache junto à FFN concentra as maiores perdas

Tabela 8. Estimativa do tamanho de cada bloco em memória para o LLaMA 2 7B. Pesos em `Q4_K_M` (≈4,5 bits/parâmetro); KV-Cache em FP16 (*batch* = 1).

Componente	Tamanho
Pesos de Atenção (Q, K, V, O)	≈1,15 GB
Pesos da FFN (gate, up, down)	≈2,32 GB
KV-Cache C5 (∼128 tokens)	≈64 MB
KV-Cache C3 (∼2.048 tokens)	≈1,0 GB
KV-Cache C1/C2/C4 (∼4.096 tokens)	≈2,1 GB

(exceto em C5, onde a KV-Cache é desprezível). Segundo, a disputa pelo ponto ótimo dá-se entre `KV+Attn` e `Attn+FFN`, e o melhor candidato depende do contexto. Sob contexto curto a moderado (C5, C3, C2), `KV+Attn` é consistentemente menos penalizadora, pois deslocar a KV-Cache desprezível custa pouco e manter a FFN em DRAM evita o custo fixo de reler seus pesos volumosos. A inversão consuma-se em C1, na qual a partir de ∼4.096 tokens com *prompt* longo, `Attn+FFN` torna-se preferível porque preservar a KV-Cache em DRAM evita o gargalo que, sob contexto extenso, domina o custo de qualquer combinação que a inclua. Não existe, portanto, combinação universalmente dominante: a regra operacional preserva em DRAM o componente que constitui o gargalo de cada regime.

4.4. Ocupação de memória e síntese bi-objetivo

O eixo de uso de memória é estrutural. A Tabela 8 estima o tamanho de cada bloco para o LLaMA 2 sob `Q4_K_M` (pesos) e FP16 (KV-Cache, *batch* = 1). A FFN é o maior bloco estático (≈2,32 GB, cerca do dobro dos pesos de atenção), sendo o alvo de maior impacto na economia de DRAM, mas também o de maior custo de vazão sob contexto curto. A KV-Cache é desprezível sob *prompt* curto (≈64 MB em C5), mas rivaliza com os pesos quando a janela é preenchida (≈2,1 GB em C4), aproximando-se da FFN sem, contudo, superá-la. A decisão de alocação é intrinsecamente bi-objetivo: deseja-se maximizar simultaneamente a DRAM liberada e a vazão de *decode*. Diz-se que uma configuração A *domina* B quando A não é pior em nenhum objetivo e é estritamente melhor em ao menos um:

$$\begin{aligned} A \succ B \iff & \big(\mathrm{DRAM}_A \geq \mathrm{DRAM}_B \wedge \mathrm{TPS}_A > \mathrm{TPS}_B\big) \\ & \vee \big(\mathrm{DRAM}_A > \mathrm{DRAM}_B \wedge \mathrm{TPS}_A \geq \mathrm{TPS}_B\big). \end{aligned} \tag{3}$$

A *fronteira de Pareto* é o conjunto das configurações não dominadas. Adicionalmente, introduz-se um *limiar de aceitabilidade* de TPS $\geq$ 3,0 tokens/s, abaixo do qual a geração é considerada inaceitavelmente lenta para uso interativo; configurações abaixo desse piso são excluídas por inviabilidade. A Tabela 9 aplica ambos os critérios ao LLaMA 2. Emergem três padrões. Primeiro, os componentes isolados `Attn` e `FFN` são Pareto-eficientes em todos os cenários: a preferência por um único componente na Optane é consequência da dominância, não heurística. Segundo, o corte de aceitabilidade elimina `Tudo` sob contexto longo (C1, C4). Terceiro, sob contexto efetivo longo, deslocar a KV-Cache é Pareto-dominado: combinando os tamanhos da Tabela 8 com as vazões

Tabela 9. Pertinência à fronteira de Pareto por configuração e cenário, para o LLaMA 2. ✓: eficiente; ×: dominada; ∅: inviável (decode TPS $< 3{,}0$). A `Baseline` é omitida por ser trivialmente eficiente (maior TPS, nenhuma DRAM liberada).

Config.	C5	C3	C2	C1	C4
KV	✓	×	✓	×	×
Attn	✓	✓	✓	✓	✓
FFN	✓	✓	✓	✓	✓
KV+Attn	✓	✓	✓	×	×
KV+FFN	✓	×	✓	✓	✓
Attn+FFN	✓	✓	✓	✓	✓
Tudo	✓	✓	✓	∅	∅

da Tabela 4, embora a KV-Cache em C4 alcance ∼2,1 GB, ela libera *menos* DRAM que a FFN (∼2,3 GB) a um custo de vazão estritamente superior; a FFN, portanto, *domina* a KV-Cache, derrubando `KV` e `KV+Attn` da fronteira.

Os resultados convergem para um conjunto enxuto de diretrizes prescritivas:

D1 A política é sensível ao contexto efetivo, não fixa. Preserve em DRAM o componente que constitui o gargalo do regime: a FFN sob contexto efetivo curto, a KV-Cache sob contexto efetivo longo;

D2 Prefira deslocar um único componente. `Attn` e `FFN` isolados são Pareto-eficientes em todos os cenários; combine apenas quando a meta de memória exceder o que um componente isolado oferece;

D3 A KV-Cache tem tolerância assimétrica ao contexto efetivo. Sob contexto curto, deslocá-la custa menos de 4% de vazão; sob contexto longo, torna-se Pareto-dominada pela FFN, que libera mais DRAM a menor custo;

5. Conclusão

Este trabalho investigou a alocação seletiva de componentes internos do LLaMA 2 7B (pesos de atenção, pesos da FFN e KV-Cache) entre DRAM e uma camada de memória de maior capacidade e maior latência (Intel Optane DC) em ambiente de CPU, visando equilibrar desempenho de inferência e uso de DRAM. A análise empírica revelou que o gargalo da estratificação é governado por um único mecanismo: o volume de bytes relidos por passo de geração, que determina qual componente se torna o principal limitador em função do regime de operação. Sob contexto efetivo curto, a maior sensibilidade recai sobre os pesos da FFN; sob contexto efetivo longo, a KV-Cache passa a dominar. Observou-se que não existe um componente universalmente mais tolerante à estratificação: a hierarquia de custos rotaciona ao longo do gradiente de contexto. A sensibilidade da alocação é governada pela leitura média de estado por passo de geração, e não pelo componente ou pelo tamanho final do contexto isoladamente. Além disso, nenhuma combinação de componentes é universalmente dominante: a política de menor custo é condicionada ao contexto, preservando em DRAM o componente que constitui o gargalo de cada regime.

Agradecimentos. Os autores agradecem à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos nº 2024/02372-7 e 2025/02790-6 pelo apoio a este trabalho.

Declaração de uso de IA. Um modelo de linguagem de grande porte (LLM) foi utilizado como apoio na revisão gramatical e ortográfica da versão final deste texto. Todo o conteúdo científico, a análise dos resultados e as conclusões são de inteira responsabilidade dos autores, que revisaram e validaram todas as sugestões incorporadas.

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 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4895–4901, Singapore. Association for Computational Linguistics.

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 Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 12562–12584. arXiv:2312.11514.

Baldassin, A., Barreto, J. a., Castro, D., and Romano, P. (2021). Persistent memory: A survey of programming support and implementations. *ACM Comput. Surv.*, 54(7).

Das Sharma, D., Blankenship, R., and Berger, D. (2024). An introduction to the compute express link (cxl) interconnect. *ACM Comput. Surv.*, 56(11).

Du, H., Wu, S., Kharlamova, A., Guan, N., and Xue, C. J. (2025). FlexInfer: Breaking memory constraint via flexible and efficient offloading for on-device LLM inference. In *Proceedings of the 5th Workshop on Machine Learning and Systems (EuroMLSys)*, pages 56–65. ACM. arXiv:2503.03777.

Fang, Y., Xie, R., Haq, A. U., Ma, L., El Maghraoui, K., Wang, N., Wang, M., Liu, L., and Zhang, T. (2025). Accelerating LLM inference via dynamic KV cache placement in heterogeneous memory system. *arXiv preprint arXiv:2508.13231*.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2023). OPTQ: Accurate quantization for generative pre-trained transformers. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*. Originally titled GPTQ on arXiv (arXiv:2210.17323). Renamed to OPTQ at camera-ready.

Ganjihal, S. R. (2025). Predictive multi-tier memory management for KV cache in large-scale GPU inference. *arXiv preprint arXiv:2604.26968*.

Gerganov, G. et al. (2023). llama.cpp. `https://github.com/ggerganov/llama.cpp`. Accessed: 2025.

Geva, M., Schuster, R., Berant, J., and Levy, O. (2021). Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Gu, Y., Khadem, A., Umesh, S., Liang, N., Servot, X., Mutlu, O., Iyer, R., and Das, R. (2025). PIM is all you need: A CXL-enabled GPU-free system for large language model inference. In *Proceedings of the 30th ACM International Conference on Ar-*

chitectural Support for Programming Languages and Operating Systems (ASPLOS). arXiv:2502.07578.

Jiang, X., Zhou, Y., Cao, S., Stoica, I., and Yu, M. (2025). NEO: Saving GPU memory crisis with CPU offloading for online LLM inference. In *Proceedings of Machine Learning and Systems (MLSys)*.

Lee, D., Kim, Y., Seshadri, V., Liu, J., Subramanian, L., and Mutlu, O. (2016). Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *ACM Transactions on Architecture and Code Optimization*, 12(4):49:1–49:29.

Lee, T., Monga, S. K., Min, C., and Eom, Y. I. (2023). Memtis: Efficient memory tiering with dynamic page classification and page size determination. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*. ACM.

Leviathan, Y., Kalman, M., and Matias, Y. (2023). Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286.

Lin, J., Mi, J., Hong, Z., Wang, H., Liu, Q., Zhang, H., Li, P., and Guo, S. (2025a). KVDrive: A holistic multi-tier KV cache management system for long-context LLM inference. *arXiv preprint arXiv:2605.18071*.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. (2024). AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the 7th Conference on Machine Learning and Systems (MLSys)*. Best Paper Award. arXiv:2306.00978.

Lin, M., Wang, X., Cox, G., Li, D., and Jeon, H. (2025b). HybridGen: Efficient LLM generative inference via CPU–GPU hybrid computing. *arXiv preprint arXiv:2604.18529*.

Liu, L., Zhao, S., Zhou, Y., He, Y., Wang, M., Han, Y., and Wang, Y. (2025). PAM: Processing across memory hierarchy for efficient KV-centric LLM serving system. *arXiv preprint arXiv:2602.11521*.

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*.

Maruf, H. A., Wang, H., Dhanotia, A., Weiner, J., Agarwal, N., Bhattacharya, P., Petersen, C., Chowdhury, M., Kanaujia, S., and Chauhan, P. (2023). Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, page 742–755.

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*.

Na, S., Jeong, G., Ahn, B. H., Jezghani, A., Young, J., Hughes, C. J., Krishna, T., and Kim, H. (2025). FlexInfer: Flexible LLM inference with CPU computations. In *Proceedings of the 8th Conference on Machine Learning and Systems (MLSys)*.

Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. (2023). FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *PMLR*, pages 31094–31116.

Tang, Y., Cheng, R., Zhou, P., Liu, T., Liu, F., Tang, W., Bae, K., Chen, J., Xiang, W., and Shi, R. (2024). Exploring CXL-based KV cache storage for LLM serving. In *NeurIPS 2024 Workshop on Machine Learning for Systems*.

Touvron, H., Martin, L., Stone, K., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2024). Efficient streaming language models with attention sinks. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. arXiv:2309.17453.

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, Santa Clara, CA. USENIX Association.

Yang, X., Hu, Q., Li, J., Li, F., Zhu, Y., Zhou, Y., Lin, Q., Dai, J., Kong, Y., Zhang, J., Xu, G., and Liu, Q. (2025). Beluga: A CXL-based memory architecture for scalable and efficient LLM KVCache management. *arXiv preprint arXiv:2511.20172*.

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*.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. (2023). H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Zheng, X., Wang, Z., Ma, R., Wang, R., Wang, X., Chen, R., Zhang, P., Pan, S., Huang, Z., Wu, C., Zhang, Y., Cai, B., Liu, K., Ma, T., Du, Y., Deng, D., Wu, S., Zhu, G., Zhang, W., and Li, F. (2025). Adaptive multi-objective tiered storage configuration for KV cache in LLM service. *arXiv preprint arXiv:2603.08739*.

Zhong, Y., Berger, D. S., Waldspurger, C., Wee, R., Agarwal, I., Agarwal, R., Hady, F., Kumar, K., Hill, M. D., Chowdhury, M., and Cidon, A. (2024). Managing memory tiers with CXL in virtualized environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 37–56, Santa Clara, CA. USENIX Association.