Análise de desempenho de LLMs usando técnicas de RAG em cenários de hardware com recursos limitados

Gabriela Malveira
Instituto de Computação
Universidade Federal do Amazonas
Manaus-AM, Brasil
gabriela.malveira@icomp.ufam.edu.br

Kaike Maciel
Instituto de Computação
Universidade Federal do Amazonas
Manaus-AM, Brasil
kaike.maciel@icomp.ufam.edu.br

João Alfredo Bessa
Instituto de Computação
Universidade Federal do Amazonas
Manaus-AM, Brasil
joao.bessa@icomp.ufam.edu.br

Ricardo Miranda Filho
Instituto de Computação
Universidade Federal do Amazonas
Manaus-AM, Brasil
ricardo.filho@icomp.ufam.edu.br

Rosiane de Freitas
Instituto de Computação
Universidade Federal do Amazonas
Manaus-AM, Brasil
rosiane@icomp.ufam.edu.br

***Abstract*—This paper presents an analysis of the performance of embedded Large Language Models (LLMs) combined with Retrieval-Augmented Generation (RAG) techniques in hardware-constrained scenarios. Metrics such as response time, memory usage, and token throughput were evaluated on devices with limited resources. The experiments indicate that smaller, quantized models offer the best balance between latency and throughput, while RAG implementations require optimization, such as pre-indexing, to be effective in edge computing. Practical limitations such as token limits and memory bottlenecks are explored, automating the RAG pipeline with a proprietary platform, which increases the scalability and reproducibility of tests on limited hardware. The results demonstrate the feasibility of using LLMs and RAG on restricted devices, contributing to the still scarce literature on the subject.**

***Resumo*— Neste trabalho é apresentada uma análise do desempenho de modelos de Linguagem de Grande Escala (LLMs) embarcados combinados com técnicas de Geração Aumentada por Recuperação (RAG), em cenários com restrição de hardware. Foram avaliadas métricas como tempo de resposta, uso de memória, latência e taxa de transferência (throughput) de tokens em dispositivos com recursos limitados. Os experimentos indicam que modelos menores e quantizados oferecem o melhor equilíbrio entre latência e throughput, enquanto as implementações de RAG necessitam de otimização, como pré-indexação, para serem eficazes em computação de borda (edge computing). É explorada limitações práticas como o limite de tokens e gargalos de memória, automatizando o pipeline RAG com uma plataforma própria, o que amplia a escalabilidade e reprodução dos testes em hardwares limitados. Os resultados evidenciam a viabilidade, embora limitada, do uso de LLMs e RAG em dispositivos restritos, contribuindo para a literatura ainda escassa no tema.**

***Index Terms*—Large Language Models, RAG, Hardware Limitado.**

Este trabalho foi parcialmente financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES-PROEX) - Código de Financiamento 001, pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pela Fundação Amazonas de Apoio à Pesquisa - FAPEAM - através do POSGRAD 2024-2025.

I. Introdução

A crescente integração de Modelos de Linguagem de Larga Escala (LLMs) em aplicações práticas demanda soluções que aliem desempenho à viabilidade técnica. Essa necessidade é particularmente crítica em cenários com hardware limitado, como dispositivos móveis, embarcados e aplicações em regiões com infraestrutura restrita. A descentralização da inferência — com modelos operando fora da nuvem — não apenas reduz latência e consumo energético, como também promove a soberania digital e a privacidade do usuário final. Nesse cenário, é essencial compreender como otimizações como quantização e Geração Aumentada via Recuperação(RAG) se comportam em ambientes reais para ampliar o acesso a tecnologias avançadas de IA em escala global.

Para superar tais limitações, a comunidade científica tem proposto diversas técnicas de otimização. Entre as mais proeminentes estão a quantização de modelos e a Retrieval-Augmented Generation (RAG). A quantização visa reduzir a pegada de memória (memory footprint) e acelerar a inferência através da diminuição da precisão numérica dos pesos do modelo, como de 32-bit para 8-bit, com técnicas como GPTQ que buscam minimizar a perda de eficácia [1]. Em paralelo, a abordagem RAG permite o uso de modelos de linguagem menores e mais eficientes, que são complementados por um mecanismo de recuperação de informação. Este mecanismo consulta uma base de dados externa para obter conhecimento contextual, reduzindo a necessidade de que o modelo armazene toda a informação em seus próprios parâmetros [2].

Apesar do potencial teórico dessas abordagens, existe uma lacuna na literatura no que tange à análise de desempenho empírica e quantitativa da sua aplicação combinada em cenários de hardware realistas e severamente restritos. Pouco

se sabe sobre a relação de compromisso prática entre latência de inferência, uso de memória e vazão de tokens (throughput) quando diferentes LLMs quantizados são implementados com sistemas RAG em dispositivos com 4 a 8 GB de RAM e sem aceleração por GPU. A sobrecarga computacional introduzida pelo componente de recuperação do RAG, por exemplo, pode anular os ganhos obtidos com o uso de um LLM menor.

O objetivo principal deste trabalho é preencher essa lacuna. Apresentamos um estudo quantitativo que avalia o desempenho de LLMs embarcados e sistemas RAG sob restrições de hardware. A contribuição deste artigo é multifacetada: (i) fornecemos benchmarks (testes de desempenho) para diferentes arquiteturas de LLMs em hardware de baixa capacidade; (ii) analisamos o impacto e a sobrecarga específicos da implementação de RAG nesses cenários; e (iii) identificamos os pontos de equilíbrio cruciais que podem orientar a seleção de modelos e estratégias de otimização para aplicações de IA. O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico, seguida pela metodologia na Seção 3. A Seção 4 detalha a análise dos resultados e, por fim, a Seção 5 sumariza as conclusões do estudo.

Com esse estudo, buscamos não apenas validar técnicas conhecidas, mas também explorar sua aplicabilidade em dispositivos amplamente disponíveis. Acreditamos que entender como tornar LLMs mais acessíveis pode contribuir para democratizar a IA, tornando-a funcional mesmo em contextos de baixa infraestrutura.

II. Referencial Teórico

Para fundamentar nossa análise, esta seção aborda os conceitos tecnológicos centrais do nosso estudo. Discutiremos a arquitetura dos LLMs, as implicações do hardware limitado e o funcionamento da técnica de Retrieval-Augmented Generation (RAG).

Modelos de Linguagem de Larga Escala (LLMs) baseiam-se na arquitetura Transformer, proposta por Vaswani et al. [3], que utiliza mecanismos de atenção multi-cabeça para capturar dependências de longo alcance em texto. Embora esses modelos se destaquem por seu alto desempenho em tarefas de geração e compreensão de linguagem, eles exigem grande capacidade de memória e processamento, o que dificulta sua execução em ambientes com hardware restrito. Para viabilizar LLMs em dispositivos com recursos limitados, duas técnicas ganham destaque: a distilação de conhecimento e a quantização. A distilação permite transferir o ”conhecimento” de um modelo grande para um modelo menor, mantendo boa parte da eficácia original [4]. Por outro lado, a quantização reduz a precisão dos pesos — por exemplo, de 32 bits para 8 bits —, diminuindo de forma expressiva o uso de memória e acelerando a inferência, o que torna os modelos mais eficientes para ambientes de hardware restrito. Estudos recentes, como os de Zheng et al. (2025), têm discutido o potencial estratégico de LLMs operando na borda, enfatizando benefícios como autonomia computacional e privacidade. Além disso, frameworks como TinyLLaMA [5] e iniciativas de benchmarking como BEIR [6] têm acelerado o desenvolvimento de modelos otimizados para dispositivos modestos.

A execução em cenários de **hardware limitado**, onde dispositivos como CPUs sem aceleração CUDA ou sistemas ARM com apenas 4–8 GB de RAM são comuns, exige abordagens específicas. A ausência de GPUs força a execução de todo o fluxo de trabalho em CPU, elevando a latência das operações. Frameworks como o TinyLLaMA oferecem implementações otimizadas para esses ambientes, aproveitando técnicas de quantização e multi-threading para melhorar a eficiência em dispositivos com recursos limitados [5].

No que diz respeito ao uso de **Retrieval-Augmented Generation (RAG)**, essa abordagem combina a recuperação de documentos com a geração condicionada, permitindo que modelos de linguagem acessem informações externas para melhorar a qualidade das respostas [2]. A recuperação de documentos pode ser feita utilizando sistemas como o FAISS [7], que seleciona trechos relevantes de uma base local. Esses trechos são então concatenados ao prompt (instrução) do LLM, aprimorando a factualidade das respostas e reduzindo a probabilidade de alucinações. O uso de RAG em conjunto com plataformas como o `Ollama` simplifica a integração desses sistemas, permitindo a implementação de pipelines eficientes. Adicionalmente, o framework Transformers da Hugging Face disponibiliza componentes como o `RagSequenceForGeneration`, que facilita a integração de sistemas de recuperação avançada com LLMs [6].

III. Análise Preliminar e Contextualização Teórica

Para contextualizar os resultados que serão apresentados posteriormente, esta seção apresenta uma análise preliminar do desempenho dos modelos, integrando nossas observações com achados relevantes da literatura científica. A análise inicial foca nos trade-offs entre tamanho do modelo, latência e a sobrecarga de implementações de Recuperação Aumentada por Geração (RAG).

Os resultados desta mostram que, em cenários de hardware restrito, a escolha do modelo é um fator determinante. Modelos mais leves e quantizados apresentam tempos de resposta e vazão (throughput) significativamente superiores aos de modelos maiores. Por exemplo, nos testes, o modelo `deepseek-r1:1.5b` atinge até 148,94 tokens/s em uma carga de 10.000 tokens, enquanto o `qwen3:8b`, de maior porte, mal ultrapassa 10,61 tokens/s no mesmo cenário. Essa diferença de quase 14 vezes na eficiência corrobora diretamente as conclusões de estudos como o de Chen et al. [8], que demonstram ser a quantização uma técnica indispensável para viabilizar a execução de LLMs em dispositivos com recursos limitados, como aqueles com memória RAM abaixo de 8 GB.

Além do throughput, a análise do tempo absoluto de resposta (latência) reforça a inviabilidade de modelos maiores sem otimização. Nos experimentos, o modelo `qwen3:8b` exibiu um salto de latência de 1,3 segundos para processar 30 tokens para quase 942 segundos (mais de 15 minutos) para 10.000 tokens. Essa escalabilidade limitada em ambientes

sem aceleração por GPU torna tais modelos impraticáveis para qualquer aplicação interativa, evidenciando o "gap" de desempenho que as otimizações buscam resolver.

No que tange ao uso de RAG, o desafio se mostra duplo. Em nossa análise, uma implementação "crua" de recuperação, baseada em TF-IDF com vetorização em tempo real, demonstrou ser um gargalo de performance, adicionando segundos a cada consulta devido à sobrecarga de pré-processamento e cálculo de similaridade. Este desafio de latência é um problema, e trabalhos como o de Feng et al. (2024) buscam solucioná-lo através da integração de RAG com técnicas de Chain of Thought (CoT), otimizando a geração para reduzir o tempo de resposta [9].

Contudo, a sobrecarga de desempenho não é o único obstáculo. Conforme apontado por Jin et al. (2024), a eficácia do RAG também depende da relevância dos documentos recuperados. Um excesso de informações ou a recuperação de dados irrelevantes pode degradar a qualidade da resposta final, um fenômeno que evidencia a necessidade de otimizações não apenas no desempenho, mas também na estratégia de recuperação de informação [10]. Portanto, a viabilidade do RAG depende de uma implementação que seja, ao mesmo tempo, rápida e precisa.

Em resumo, esta análise preliminar, confirma que:

- Modelos quantizados e de menor porte (`1.5B`, `4B`) oferecem o melhor equilíbrio entre latência e throughput, uma observação alinhada com as melhores práticas de otimização de LLMs para a borda [8].
- Modelos maiores (`8B`) não escalam em hardware sem GPU, apresentando tempos de inferência proibitivos que os tornam inadequados para aplicações interativas.
- A implementação de RAG em dispositivos de borda é promissora, mas exige otimizações para superar tanto a sobrecarga de latência quanto os desafios de qualidade na recuperação de informação, como discutido por Feng et al. e Jin et al. [9], [10]. O uso de pré-indexação com sistemas vetoriais eficientes, como FAISS, emerge como um requisito fundamental.

Além de validar achados prévios, estes experimentos também ampliam o entendimento prático sobre os desafios e oportunidades da execução local de LLMs. Ao replicar e expandir condições realistas de uso, mostramos como adaptações tecnológicas podem ser efetivamente traduzidas para dispositivos comuns. Assim, a aproximação da literatura a soluções concretas, tem potencial de impactar diretamente a adoção de IA em contextos sociais diversos.

IV. Metodologia

Nesta seção, são descritos em detalhes os procedimentos experimentais adotados para avaliar o desempenho de LLMs embarcados, com e sem RAG, em cenários de hardware restrito. Em seguida, os modelos testados, o ambiente de execução, a preparação dos dados para RAG, o protocolo de execução e as métricas coletadas.

A. Configuração

Para avaliar o desempenho de LLMs em cenários de hardware restrito, foi montado um ambiente experimental com escolhas tecnológicas deliberadas, visando simular condições realistas e garantir a reprodutibilidade dos resultados.

O ambiente de hardware foi configurado para emular dispositivos de borda, como notebooks de baixo custo ou sistemas embarcados, utilizando máquinas com 4 a 8 GB de RAM, processamento via CPU (x86-64) e alocação variável de camadas de GPU. Essa configuração é crucial para que os benchmarks de latência e consumo de memória reflitam os desafios de desempenho em cenários de uso real, fora da nuvem.

A orquestração dos experimentos foi realizada em Python 3.9+, escolhido por seu robusto ecossistema de bibliotecas para IA e análise de dados. Para a execução e gerenciamento dos modelos, optou-se pela API do Ollama, uma ferramenta que simplifica a inferência local de múltiplos LLMs e permite focar na análise de desempenho. O monitoramento de recursos, como o pico de uso de RAM, foi feito com a biblioteca `psutil`, selecionada por ser leve e precisa.

A seleção de modelos (Llama 3, Phi 3, Qwen, Mini Orca, DeepSeek) foi intencionalmente diversa, incluindo diferentes arquiteturas e tamanhos para permitir uma análise comparativa ampla dos trade-offs de desempenho. Para os testes de Recuperação Aumentada por Geração (RAG), foi implementado um pipeline de linha de base utilizando ferramentas de código aberto: a biblioteca `PyMuPDF` para a extração de texto e o `TfidfVectorizer` do Scikit-learn para a vetorização. Essa abordagem foi escolhida para medir a sobrecarga de um sistema de recuperação simples e eficaz. Em paralelo, o framework Hugging Face Transformers foi usado para testes com o componente `RAGSequenceForGeneration`, representando uma abordagem mais moderna baseada em embeddings.

A Tabela I serve como um guia de referência rápida para as principais ferramentas utilizadas.

Table I: Resumo das tecnologias do ambiente experimental.

Componente	Especificação
Hardware e Ambiente	
Plataforma	CPU (x86-64), 4–8 GB RAM, GPU (variável)
Software e Orquestração	
Linguagem	Python 3.9+
Orquestrador	Ollama API
Framework IA	Hugging Face Transformers
Monitoramento	`psutil`
Modelos e Dados	
LLMs	Llama 3, Phi 3, Qwen, Mini Orca, DeepSeek
Dados (RAG)	`PyMuPDF` e `TfidfVectorizer`

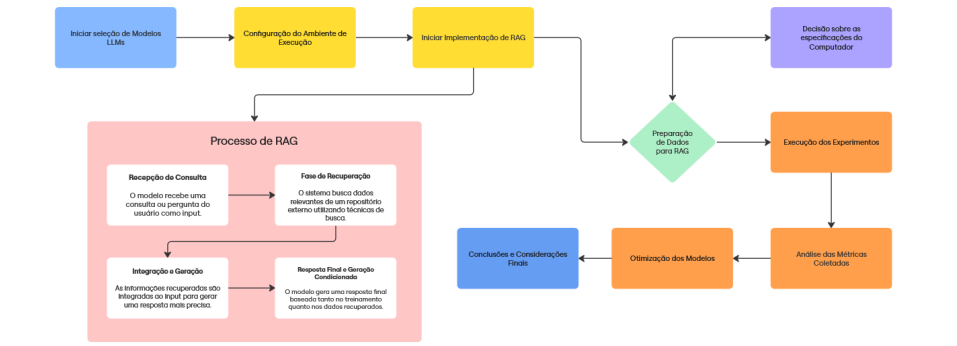

Figure 1: Diagrama ilustrando o processo de Recuperação Aumentada por Geração (RAG). A imagem mostra as etapas principais do processo, desde a seleção de modelos LLMs até as conclusões finais, incluindo a configuração do ambiente de execução, recuperação e integração de dados, e otimização de modelos.

B. Considerações Qualitativas e Decisões de Engenharia

A escolha da arquitetura experimental foi guiada por desafios práticos enfrentados no início do projeto. Inicialmente, foram considerados utilizar frameworks populares como o Hugging Face Transformers para interagir com modelos via API. No entanto, essa abordagem rapidamente impôs barreiras significativas para o cenário de estudo. A principal dificuldade era a dependência de uma conexão de internet estável e a necessidade de gerenciar tokens de autenticação. Para um estudo focado em desempenho sob restrições, depender de uma infraestrutura externa introduziria latência de rede e possíveis falhas de conexão, variáveis que contaminariam as métricas de performance do hardware local. Além disso, o ciclo de iteração para depuração e testes tornava-se lento e custoso.

Essa experiência prática de desenvolvimento levou a uma abordagem local e offline, utilizando a biblioteca Ollama. Essa decisão foi crucial por três motivos:

1) **Controle e Reprodutibilidade:** A execução local permitiu controle total sobre o ambiente, permitindo isolar o desempenho do hardware como a principal variável de análise, eliminando a instabilidade da rede.
2) **Agilidade no Desenvolvimento:** Sem a necessidade de chamadas de rede ou gerenciamento de tokens de API, o processo de testar diferentes modelos e prompts tornou-se significativamente mais eficiente.
3) **Soberania e Privacidade:** Manter todos os dados e modelos localmente garantiu a privacidade e a segurança das informações processadas.

Mesmo no ambiente local, outros desafios de engenharia surgiram. O limite de tokens dos modelos, por exemplo, revelou-se um gargalo prático ao tentar processar documentos extensos via RAG. A composição manual de prompts com trechos relevantes era inviável e não escalável. Para superar essa barreira, foi desenvolvido uma **plataforma própria de automação** que gerencia a indexação de documentos, a fragmentação de texto (chunking) e a geração dinâmica de prompts, adaptando-os aos limites de cada modelo.

Portanto, a metodologia deste trabalho não reflete apenas um benchmark de desempenho, mas também um conjunto de decisões de engenharia tomadas para contornar limitações reais, agregando um olhar de desenvolvedor que valida a viabilidade e a eficiência da IA em hardware limitado.

*C. Arquitetura da Plataforma: **PromptHDScope***

Para conduzir os benchmarks de forma sistemática, foi desenvolvida uma plataforma experimental própria. A plataforma consiste em uma aplicação web construída com a biblioteca Streamlit, que serve como interface para a orquestração dos testes e coleta de métricas. A arquitetura foi deliberadamente projetada para operar localmente, garantindo que as medições de desempenho reflitam exclusivamente o impacto no hardware, sem a interferência de latência de rede ou dependência de APIs externas.

Os principais componentes da plataforma são:

- **Interface de Controle:** A aplicação Streamlit permite ao pesquisador configurar dinamicamente cada ciclo de teste, selecionando o modelo de linguagem (LLM) a ser avaliado a partir de uma lista pré-definida, ajustando os limites de recursos de hardware (RAM, núcleos de CPU, camadas de GPU) e realizando o upload de documentos em formato PDF para os testes de RAG.
- **Orquestração de Modelos Locais:** O núcleo da plataforma utiliza a biblioteca `ollama` para gerenciar e executar inferências em modelos de linguagem locais. Essa escolha foi uma decisão de engenharia fundamental para garantir a execução *offline*, simplificar o processo de troca entre diferentes modelos e eliminar a complexidade associada a tokens de autenticação de APIs de nuvem.
- **Pipeline de RAG Simplificado:** Para avaliar a sobrecarga do RAG em hardware restrito, foi implementado um pipeline de recuperação leve. Este pipeline utiliza a biblioteca `scikit-learn` para extrair texto dos documentos PDF carregados, vetorizá-los usando TF-IDF (`TfidfVectorizer`) e calcular a relevância com base na similaridade de cosseno (`cosine_similarity`) em relação à consulta do usuário. Essa abordagem foi escolhida para medir o impacto de um sistema RAG funcional, porém computacionalmente mais simples, ideal para cenários de borda.

- **Coleta Automatizada de Métricas:** Cada consulta executada através da interface dispara um processo de log automatizado. A função `log_to_file` registra as principais métricas de desempenho — como tempo de resposta, pico de uso de memória (aferido com `psutil`), e throughput de tokens — em um arquivo de texto estruturado (‘response_log.txt‘). Essa automação garante a consistência e a precisão dos dados coletados, que formam a base para a análise de resultados apresentada neste artigo.

A construção desta plataforma não foi apenas um meio para um fim, mas uma parte integral da pesquisa. Ela representa uma solução prática para os desafios de realizar benchmarking de LLMs, validando a abordagem de usar ferramentas leves e um ecossistema local para pesquisa e desenvolvimento em IA.

D. Métricas de Avaliação e Protocolo de Coleta

Para avaliar o desempenho dos modelos em cenários de hardware restrito, definimos três métricas principais: latência de inferência, throughput de tokens e pico de uso de memória RAM. Cada métrica foi registrada por consulta individual, permitindo uma análise comparativa detalhada.

- **Latência de Inferência** (L): Medida em segundos (s), representa o intervalo de tempo total decorrido desde a submissão da requisição ao modelo até o recebimento completo da resposta gerada. É a principal medida da agilidade e do tempo de resposta percebido pelo usuário.
- **Throughput de Tokens** (T): Medido em tokens por segundo (tokens/s), quantifica a velocidade de processamento do modelo. É calculado como a razão entre o número total de tokens processados (entrada + saída) e a latência de inferência, conforme a fórmula:

$$T = \frac{N_{prompt}}{L}$$

  onde N_{prompt} é o número de tokens no prompt de entrada e L é a latência total da inferência.
- **Pico de Uso de Memória RAM** (M_{pico}): Medido em Gigabytes (GB), corresponde ao valor máximo de memória RAM alocada pelo processo de inferência durante sua execução. Esta métrica é fundamental para determinar a viabilidade de um modelo em dispositivos com memória limitada e foi aferida utilizando a biblioteca `psutil` em Python, especificamente com a função `Process.memory_info().rss`.

Para garantir a reprodutibilidade e a análise sistemática, todos os dados de desempenho foram registrados em um arquivo. Cada entrada no arquivo contém um registro completo da execução, incluindo: o identificador do **modelo**, a **quantidade de RAM** do sistema, a **quantidade de núcleos da CPU** e **camadas da GPU** alocadas, o total de **tokens processados**, o **tempo de execução** (Latência L) e o throughput de **tokens_por_segundo** (T). Os dados coletados foram registrados em arquivos estruturados, possibilitando análises estatísticas pós-processadas e visualizações comparativas, que discutimos a seguir.

V. Análise dos Resultados

Nesta seção, apresentamos uma análise aprofundada dos resultados quantitativos, estruturada para demonstrar como o desempenho dos modelos degrada conforme as restrições de hardware aumentam. A análise parte de uma linha de base e avança por cenários de ”estresse” progressivo, revelando os limites práticos de cada arquitetura.

Primeiramente, estabelecemos um ponto de partida em um ambiente com recursos relativamente altos (até 8 GB de RAM) para avaliar a eficiência inerente de cada modelo. A Figura 2 revela uma disparidade fundamental: o modelo `deepseek-r1:1.5b`, por ser mais leve, demonstra uma escalabilidade muito superior, enquanto o `qwen3:8b`, de maior porte, se torna um gargalo de processamento. Este resultado inicial já confirma uma premissa crucial para a computação de borda: o tamanho do modelo é um fator determinante de desempenho, mesmo antes de submetido a restrições severas.

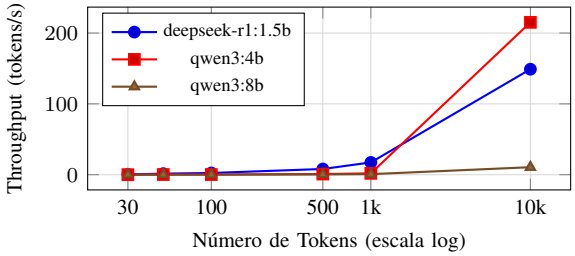

Figure 2: Comparativo de throughput (tokens/s) em diferentes modelos conforme o número de tokens processados em até 8GB de RAM.

A seguir, aprofundamos a investigação submetendo quatro modelos distintos a um teste de estresse com recursos de hardware progressivamente menores. O primeiro cenário, apresentado na Figura 3, representa um ambiente de hardware robusto, com 32 camadas de GPU e até 8 GB de RAM.

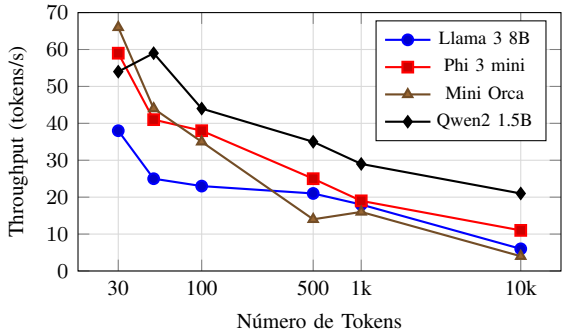

Figure 3: Throughput dos Modelos com 32 Camadas de GPU (4 núcleos, até 8 GB de RAM).

Nesta configuração ideal, todos os modelos operam de forma estável, mas já revelam uma importante relação de compromisso. O **Qwen2-1.5B-Instruct** se destaca pela escalabilidade superior, sugerindo uma arquitetura eficiente para

processar grandes volumes de dados. Em contrapartida, o **Llama 3 8B**, o maior modelo, já apresenta o desempenho mais modesto, indicando que seu tamanho impõe uma sobrecarga de processamento mesmo em condições favoráveis.

O verdadeiro ponto de inflexão ocorre ao restringirmos o ambiente pela metade, como mostra a Figura 4.

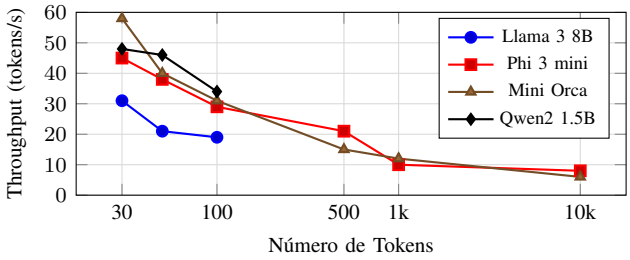

Figure 4: Throughput dos Modelos com 16 Camadas de GPU (2 núcleos, até 4 GB de RAM).

Com 16 camadas de GPU e 4 GB de RAM, o impacto no desempenho é significativo. Os modelos maiores, **Llama 3 8B** e **Qwen2-1.5B**, sofrem uma falha ao processar mais de 500 tokens. A falha não é uma mera lentidão, mas a incapacidade de concluir a tarefa, indicando que o pico de uso de memória (M_{pico}) excedeu o limite disponível. Este é um resultado crucial, pois demonstra um ”muro” de hardware intransponível para esses modelos. Em contrapartida, a resiliência dos modelos mais leves, **Mini Orca** e **Phi 3 mini**, se torna evidente: embora mais lentos, eles permanecem funcionais, provando sua adequação para dispositivos com memória restrita.

O cenário final, ilustrado na Figura 5, simula um dispositivo de borda de baixíssimo custo.

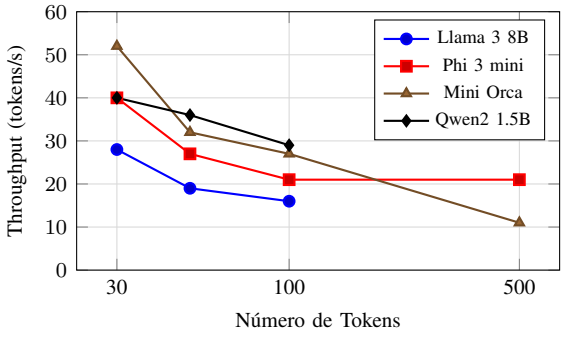

Figure 5: Throughput dos Modelos com 8 Camadas de GPU (1 núcleo, até 2 GB de RAM).

Com apenas 8 camadas de GPU e 2 GB de RAM, a maioria dos modelos se torna inviável. Neste ambiente extremo, a métrica mais importante deixa de ser a velocidade para ser a viabilidade funcional. A capacidade de “sobrevivência” dos modelos **Phi 3 mini** e **Mini Orca**, mesmo que para tarefas mais curtas, os estabelece como as únicas opções práticas para essa classe de hardware.

Em suma, os resultados demonstram de forma inequívoca uma correlação inversa entre o tamanho de um modelo e sua resiliência em hardware limitado. A demonstração de desempenho não é linear, mas manifesta-se como uma falha quando os limites de memória são atingidos. Esta evidência empírica valida nossa hipótese central: para aplicações de borda, a seleção do modelo não é apenas uma questão de otimização de desempenho, mas de garantir a própria execução da tarefa.

VI. Considerações Finais

Este estudo demonstrou que a escolha de LLMs para hardware restrito é uma questão de viabilidade funcional, não apenas de otimização. Nossos resultados revelam que modelos maiores não apenas perdem desempenho, mas atingem um ponto de ruptura e falham completamente quando os limites de memória são excedidos. Consequentemente, modelos compactos e eficientes são, em muitos casos, as únicas opções capazes de operar em dispositivos de borda, uma distinção crucial para o desenvolvimento de aplicações no mundo real.

Como próximo passo, uma análise qualitativa se faz necessária para avaliar se o desempenho quantitativo dos modelos resilientes se traduz em respostas com qualidade e coerência satisfatórias. Este desafio é agravado em sistemas RAG, cuja sobrecarga computacional também exige futuras otimizações. Em suma, este trabalho preenche uma lacuna empírica ao mapear os ”muros” de hardware para diferentes LLMs, fornecendo um guia prático para que pesquisadores e engenheiros desenvolvam aplicações de IA mais robustas e acessíveis.

Disponibilidade de Material e dados - A ferramenta está disponível em: https://github.com/kaikermaciel/teia-ufam

References

[1] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *arXiv preprint arXiv:2210.17323*, 2022.

[2] P. Lewis, E. Perez *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *NeurIPS*, 2020.

[3] A. Vaswani, N. Shazeer, and N. e. a. Parmar, “Attention is all you need,” *arXiv preprint arXiv:1706.03762*, 2017.

[4] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[5] H. Zhao, J. Liu, K. Nguyen *et al.*, “Tinyllama: Efficient transformer models for edge deployment,” *arXiv preprint arXiv:2305.16420*, 2023.

[6] N. Thakur, J. Lin *et al.*, “Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models,” in *arXiv preprint arXiv:2104.08663*, 2021.

[7] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus,” *arXiv preprint arXiv:1702.08734*, 2019.

[8] Y. Chen, C. Wu, R. Sui, and J. Zhang, “Feasibility study of edge computing empowered by artificial intelligence—a quantitative analysis based on large models,” *Big Data and Cognitive Computing*, vol. 8, no. 8, 2024. [Online]. Available: https://www.mdpi.com/2504-2289/8/8/94

[9] K. Feng, L. Luo, Y. Xia, B. Luo, X. He, K. Li, Z. Zha, B. Xu, and K. Peng, “Optimizing microservice deployment in edge computing with large language models: Integrating retrieval augmented generation and chain of thought techniques,” *Symmetry*, vol. 16, no. 11, 2024. [Online]. Available: https://www.mdpi.com/2073-8994/16/11/1470

[10] B. Jin, J. Yoon, J. Han, and S. O. Arik, “Long-context llms meet rag: Overcoming challenges for long inputs in rag,” *arXiv preprint arXiv:2410.05983*, 2024.