

Avaliação Experimental de LLMs Quantizados sob Carga Concorrente em um Dispositivo de Borda

Renan Silva¹, Bruno Nogueira², Rodrigo Paes², Ermeson Andrade¹

¹Universidade Federal Rural de Pernambuco (UFRPE)

²Universidade Federal de Alagoas (UFAL)

{renan.beserra,ermeson.andrade}@ufrpe.br

{bruno,rodrigo}@ic.ufal.br

Resumo. A inferência local de modelos de linguagem de grande porte (LLMs) quantizados em dispositivos de borda é limitada pela capacidade de processamento, e seu comportamento sob carga concorrente não está suficientemente caracterizado. Este trabalho avalia o desempenho e a confiabilidade de três modelos quantizados em 8 bits: Llama-3.2-1B, Qwen-2.5-1.5B e TinyLlama-1.1B, executados em um Raspberry Pi 4B sob requisições concorrentes com atendimento serializado. Um projeto fatorial completo combinou três modelos, três tamanhos de contexto e três níveis de concorrência, totalizando 27 condições, 630 requisições e 16.366 amostras de hardware. Os resultados mostram taxas de conclusão de 100% para o Llama, 97,96% para o Qwen e 76,52% para o TinyLlama. Embora seja o menor modelo, o TinyLlama produziu as respostas mais longas e concentrou 31 das 36 falhas por timeout. Nas condições com 100% de conclusão, as taxas sistêmicas de produção de texto variaram de 8,1 a 14,5 caracteres/s. Respostas mais longas estiveram associadas a maior latência e menor capacidade de atendimento. A utilização média da CPU variou de 78,8% a 98,9%, enquanto a memória ocupou no máximo 21,2% da capacidade. Esses achados podem orientar outros pesquisadores na seleção e avaliação de LLMs em dispositivos de borda sob carga concorrente.

Abstract. Local inference of quantized large language models (LLMs) on edge devices is limited by processing capacity, and their behavior under concurrent load remains insufficiently characterized. This paper evaluates the performance and reliability of three 8-bit quantized models: Llama-3.2-1B, Qwen-2.5-1.5B, and TinyLlama-1.1B, running on a Raspberry Pi 4B under concurrent requests with serialized inference. A full factorial design combined three models, three context sizes, and three concurrency levels, totaling 27 conditions, 630 requests, and 16,366 hardware-monitoring samples. The results show completion rates of 100% for Llama, 97.96% for Qwen, and 76.52% for TinyLlama. Despite being the smallest model, TinyLlama generated the longest responses and accounted for 31 of the 36 timeouts. Under conditions with 100% completion, system-level text production rates ranged from 8.1 to 14.5 characters/s. Longer responses were associated with higher latency and lower request-handling capacity. Mean CPU utilization ranged from 78.8% to 98.9%, while memory usage peaked at 21.2% of capacity. These findings can guide other researchers in selecting and evaluating LLMs for edge devices under concurrent load.

1. Introdução

A computação de borda consolidou-se como um paradigma fundamental para o processamento de dados próximo às fontes geradoras, mitigando a latência de comunicação, reduzindo o consumo de largura de banda e preservando a privacidade das informações [Dong and Zhang 2024]. Simultaneamente, as LLMs revolucionaram as interações baseadas em inteligência artificial, possibilitando assistentes conversacionais, geração de texto e sistemas de perguntas e respostas. Contudo, a implantação tradicional de LLMs depende fortemente de infraestruturas em nuvem com vastos recursos computacionais, o que contrasta com a realidade dos ambientes de borda [Ray and Pradhan 2025].

A execução local de LLMs é particularmente relevante em aplicações como robôs móveis, assistentes residenciais inteligentes e sistemas industriais, que podem exigir respostas com baixa latência e operação mesmo quando a conexão com a Internet está instável ou indisponível [Dong and Zhang 2024, Zheng et al. 2025]. Entretanto, executar esses modelos em dispositivos restritos, como computadores de placa única (*Single-Board Computers* – SBCs), impõe limitações de memória e capacidade de processamento. Para viabilizar essa execução, empregam-se técnicas de compressão, como a quantização de pesos pós-treinamento [Huang et al. 2024, Huang et al. 2025]. Embora reduza o espaço ocupado pelos pesos, a quantização não elimina o custo computacional da geração autorregressiva. Sob requisições concorrentes, a contenção pela CPU e o acúmulo de requisições em fila podem elevar a latência e comprometer a confiabilidade do serviço [Ray and Pradhan 2025].

Em serviços de inferência na borda, a confiabilidade não se limita à ausência de interrupções. Sob carga concorrente e recursos restritos, a saturação do processamento pode ampliar as filas, provocar *timeouts* e produzir latências incompatíveis com os requisitos da aplicação, caracterizando falhas de desempenho [Wu et al. 2024]. Em sistemas de tempo real ou missão crítica, como monitoramento médico e controle industrial, esses atrasos podem impedir a reação oportuna a eventos críticos e comprometer a segurança. Embora estudos anteriores caracterizem o uso de recursos e o desempenho da inferência na borda [Dhar et al. 2024, Durán et al. 2025, Suárez and Blanco 2025], ainda não está claro como diferentes LLMs quantizados se comportam sob carga concorrente, especialmente quanto à saturação de recursos, à latência, à conclusão das requisições e à influência da contagem de parâmetros.

Este trabalho avalia empiricamente o desempenho e a confiabilidade de LLMs quantizados sob carga concorrente na borda. Para isso, foi implementado um serviço cliente-servidor em um Raspberry Pi 4B com três modelos quantizados em Q8_0: Llama-3.2-1B-Instruct, Qwen-2.5-1.5B-Instruct e TinyLlama-1.1B-Chat. O projeto experimental variou o modelo, o tamanho do contexto de entrada e o nível de concorrência para identificar gargalos de recursos e seus efeitos sobre o serviço. Os resultados mostram que a CPU constitui o principal gargalo, com picos próximos de 100%, enquanto a memória RAM permanece com capacidade disponível. Sob maior concorrência, o acúmulo de requisições amplia os tempos de resposta e provoca *timeouts*, sobretudo no TinyLlama. Embora seja o menor modelo avaliado, ele apresentou a menor taxa de conclusão, evidenciando que a contagem de parâmetros, isoladamente, não prediz o desempenho do serviço. Esses resultados fornecem subsídios para a seleção de modelos e o dimensionamento de serviços de inferência na borda.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os

conceitos de desempenho e confiabilidade aplicados à inferência de LLMs na borda; a Seção 3 discute os trabalhos relacionados; a Seção 4 descreve a metodologia experimental; a Seção 5 apresenta e discute os resultados; e, por fim, a Seção 6 reúne as conclusões e as direções para trabalhos futuros.

2. Desempenho e Confiabilidade da Inferência de LLMs na Borda

A avaliação de serviços de inferência em dispositivos de borda requer a análise conjunta de métricas de serviço e de utilização de recursos. A latência representa o tempo decorrido entre o envio de uma requisição e a obtenção da resposta, enquanto a vazão (*throughput*) pode ser representada pelo número de requisições concluídas em uma janela de tempo. A taxa de conclusão e a ocorrência de *timeouts* indicam a capacidade do serviço de atender à demanda recebida. Métricas de CPU, memória e *swap*, por sua vez, auxiliam na identificação do recurso limitante. Sob carga concorrente, médias de latência podem ocultar degradações na cauda da distribuição, tornando necessária sua análise em conjunto com a mediana, a dispersão e percentis elevados [Jain 1991, Dong and Zhang 2024].

Na inferência de LLMs, o tempo de atendimento de cada requisição depende tanto do tamanho do contexto de entrada quanto da quantidade de *tokens* gerados. Contextos maiores ampliam o processamento do *prompt* e o uso de memória, enquanto respostas mais longas prolongam a etapa de decodificação autorregressiva [Huang et al. 2024, Cheng et al. 2024]. Sob carga concorrente e atendimento serializado, respostas mais longas mantêm o servidor ocupado por mais tempo. Quando a demanda supera a capacidade de atendimento do dispositivo, as requisições acumulam-se em fila, ampliando a latência, provocando *timeouts* e reduzindo a taxa de conclusão do serviço [Wu et al. 2024].

3. Trabalhos Relacionados

A viabilidade de implementar LLMs em ecossistemas de borda tem atraído crescente atenção da comunidade científica, com esforços concentrados predominantemente na quantificação de métricas de infraestrutura e na otimização da inferência em arquiteturas móveis. No âmbito da caracterização e análise de recursos, [Dhar et al. 2024] conduziram um estudo empírico detalhado sobre o consumo de recursos ao executar LLMs em dispositivos de borda, estabelecendo linhas de base para memória, energia e latência. Complementarmente, [Durán et al. 2025] analisaram o impacto de diferentes motores de execução e provedores de tempo de execução na utilização de recursos de LLMs, evidenciando como configurações de software influenciam o consumo energético e o tempo de inferência.

De forma abrangente, [Suárez and Blanco 2025] exploraram a eficiência energética e o desempenho de modelos quantizados em múltiplas arquiteturas de hardware, incluindo RISC-V (Reduced Instruction Set Computer - Five), ARM (Advanced RISC Machine) e x86, revelando *trade-offs* críticos entre precisão preditiva e eficiência em cenários práticos de borda. Paralelamente, abordagens focadas em otimização de arquitetura têm buscado extrair o máximo de desempenho dos processadores. [Zhang et al. 2025] propuseram estratégias de co-otimização entre software e hardware especificamente direcionadas para CPUs baseadas na arquitetura ARM, comumente encontradas em dispositivos IoT (Internet of Things) e SBCs. Por sua vez, [Wei et al. 2025] abordaram o gargalo computacional da inferência de LLMs em baixa precisão em CPUs. Os autores apresentaram o T-MAC, um método baseado em tabelas de busca que elimina a necessidade de dequantização durante a multiplicação de matrizes

de precisão mista, comprovando que ineficiências de software exacerbam as limitações do hardware de borda.

Embora os trabalhos anteriores apresentem avanços relevantes na redução do custo de inferência e na caracterização do consumo de recursos, a literatura permanece predominantemente centrada em inferências isoladas ou em otimizações de baixo nível, oferecendo evidências limitadas sobre o comportamento do serviço sob carga concorrente. Essa lacuna é particularmente relevante em dispositivos com recursos restritos, nos quais a saturação pode ampliar as filas, elevar a latência e provocar *timeouts*. Em contraste, este trabalho avalia três modelos quantizados sob a mesma plataforma e condições de carga, analisando conjuntamente a distribuição da latência, a taxa de conclusão, a utilização de CPU e memória e o tempo de esvaziamento da fila. Essa comparação permite examinar como a escolha do modelo e o comprimento das respostas afetam a ocupação do servidor, caracterizando a degradação da capacidade de atendimento durante e após a janela de submissão.

4. Metodologia Experimental

Esta seção descreve o ambiente controlado construído para medir desempenho e confiabilidade sob carga concorrente.

4.1. Arquitetura, Modelos e Instrumentação

O servidor de inferência foi hospedado em um Raspberry Pi 4 Model B, selecionado por ser um SBC ARM de baixo custo amplamente empregado em estudos de computação de borda [Dhar et al. 2024, Suárez and Blanco 2025, Zhang et al. 2025]. O dispositivo possui processador Broadcom BCM2711, com quatro núcleos Cortex-A72 a 1,5 GHz, e 4 GB de memória LPDDR4, dos quais 3.795,8 MB estavam disponíveis ao sistema. Todas as inferências foram executadas nos quatro núcleos da CPU por meio do *backend* de CPU do *llama.cpp*. Essa delimitação permitiu avaliar os três modelos sob a mesma configuração de processamento, sem o uso de um acelerador dedicado de inteligência artificial. Como a GPU VideoCore VI não participou da execução dos modelos, sua utilização não foi monitorada. A avaliação de aceleração por GPU permaneceu fora do escopo do estudo.

No Raspberry Pi, o *llama.cpp* foi responsável pelo carregamento e pela execução local dos modelos, enquanto o Flask disponibilizou uma interface HTTP para o recebimento das requisições enviadas pelo cliente. O *llama.cpp* foi escolhido por permitir a execução de modelos quantizados em CPUs ARM [Gerganov and llama.cpp contributors 2023]. Como cada inferência utilizava os quatro núcleos da CPU, um *threading.Lock* restringiu o processamento a uma requisição por vez, evitando que múltiplas inferências disputassem simultaneamente o mesmo processador. As requisições concorrentes permaneciam pendentes até a liberação do bloqueio. Portanto, a concorrência aumentava o tempo de espera em fila, mas não produzia gerações paralelas. Essa mesma disciplina de atendimento foi mantida nas 27 condições experimentais. O cliente, equipado com Intel Core i3-10110U e 16 GB de memória e conectado por Gigabit Ethernet, registrou os instantes de envio e conclusão, o tempo até o primeiro byte, o tempo total, o código de retorno e o texto produzido. No servidor, um processo amostrou, a cada segundo, a utilização da CPU, a memória ativa, *cache*, *buffers*, *swap* e a temperatura do SoC. Os registros foram associados por marca de tempo, totalizando 630 requisições e 16.366 amostras de hardware.

Os modelos foram escolhidos por serem compatíveis com *llama.cpp*, ajustados para instruções ou diálogo, executáveis nos 4 GB disponíveis e próximos em número de parâmetros, permitindo verificar se o tamanho do modelo ordena o desempenho. Todos utilizaram quantização Q8_0 e foram avaliados separadamente, com janela de contexto de 2.048 *tokens*, limite de saída de 256 *tokens* e seus respectivos gabaritos de conversação e marcadores de parada.

Table 1. Modelos de linguagem avaliados

Modelo	Referência	Parâmetros	Arquivo Q8_0
Llama-3.2-1B-Instruct	[Grattafiori et al. 2024]	1,24 B	1,3 GB
Qwen-2.5-1.5B-Instruct	[Qwen Team 2025]	1,54 B	1,6 GB
TinyLlama-1.1B-Chat	[Zhang et al. 2024]	1,10 B	1,2 GB

4.2. Carga de Trabalho, Projeto Experimental e Análise

Adotou-se o *dataset* SQuAD v2.0 [Rajpurkar et al. 2018], um *benchmark* de compreensão de texto com perguntas respondíveis e não respondíveis a partir do contexto fornecido. Um script em Python formatou as amostras como *prompts* de *question answering* no formato *zero-shot*, produzindo três cargas de trabalho com 128, 512 e 1024 *tokens* de entrada.

Adotou-se um projeto fatorial completo com três fatores: modelo, tamanho do contexto e número de usuários simultâneos, cada um com três níveis, totalizando 27 condições. Os contextos de 128, 512 e 1024 *tokens* foram selecionados para representar diferentes magnitudes de entrada, cobrindo uma variação de oito vezes entre os extremos e permitindo avaliar o efeito do crescimento do *prompt*. Os níveis de 5, 10 e 15 usuários foram definidos para submeter o serviço a cargas concorrentes progressivamente maiores. Cada usuário foi representado por uma *thread* no cliente, que enviava uma nova requisição imediatamente após receber a resposta anterior, sem tempo de reflexão. A janela de envio durou cinco minutos por condição. Ao final desse período, o cliente interrompia novos envios, mas aguardava a conclusão das requisições pendentes, conforme discutido na Seção 5.5. Requisições que ultrapassaram 600 segundos foram registradas como falhas por *timeout* e censuradas nesse limite.

A latência foi medida como o tempo entre o envio da requisição e o recebimento do primeiro byte. Como o servidor transmitia a resposta somente após concluir a geração, essa medida correspondia ao tempo de resposta percebido pelo usuário, incluindo a espera em fila e a inferência. A capacidade de atendimento foi representada pelo número de requisições concluídas em cada condição. A taxa de produção de texto foi calculada dividindo-se o total de caracteres gerados pela duração efetiva da condição. Dada a assimetria das distribuições de latência, aplicou-se o teste de Kruskal–Wallis [Kruskal and Wallis 1952] entre os modelos em cada nível de contexto. Quando a hipótese nula foi rejeitada, empregou-se o teste *post-hoc* de Dunn [Dunn 1964], com correção de Bonferroni e nível de significância de 5%. Os procedimentos de medição seguiram as recomendações de [Jain 1991].

5. Resultados e Discussão

Esta seção analisa a confiabilidade, a latência, a vazão, a utilização de recursos e a persistência da fila, encerrando com a discussão dos resultados e das ameaças à validade.

5.1. Confiabilidade do Serviço

A Figura 1 apresenta a taxa de conclusão em cada condição. O Llama-3.2-1B concluiu as 253 requisições que recebeu sem falha por *timeout*. O Qwen-2.5-1.5B manteve conclusão integral em oito das nove condições e degradou apenas na mais exigente, com 1024 *tokens* e 15 usuários, na qual cinco requisições excederam o limite. O TinyLlama-1.1B começou a falhar em condições menos exigentes. Com 128 *tokens* e 15 usuários, oito requisições não foram concluídas, enquanto, com 1024 *tokens* e 15 usuários, a taxa de sucesso caiu para 47,4%. Na campanha completa, as taxas de conclusão foram de 100%, 97,96% e 76,52%, respectivamente.

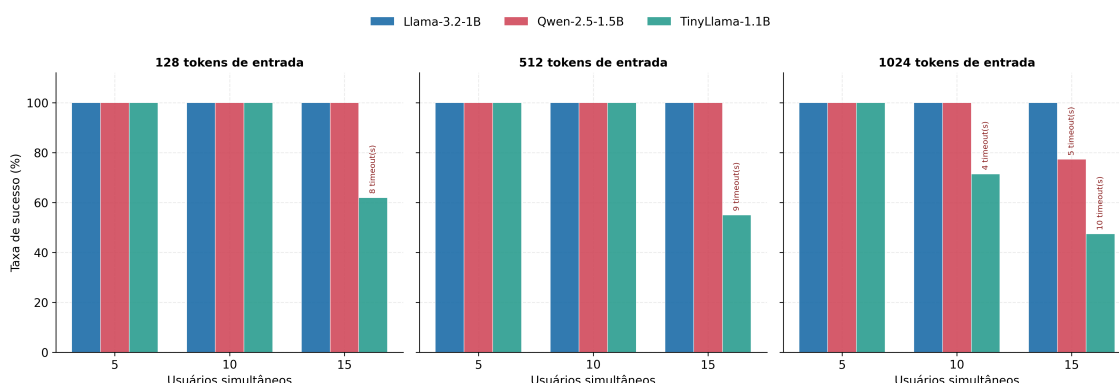


Figure 1. Taxa de conclusão das requisições por modelo, contexto e concorrência. Os rótulos indicam o número de requisições que excederam o limite de 600 segundos.

Esse ordenamento contraria a expectativa de que modelos menores sejam mais adequados a dispositivos restritos. O TinyLlama-1.1B possui a menor contagem de parâmetros entre os três, 1,10 bilhão contra 1,54 bilhão do Qwen-2.5-1.5B, e ainda assim apresentou a pior confiabilidade em todas as condições. A contagem de parâmetros, isolada, não prediz o comportamento do serviço sob concorrência neste ambiente.

5.2. Distribuição da Latência

A Figura 2 apresenta a distribuição do tempo até o primeiro byte. As caixas incluem as requisições que falharam por *timeout*, censuradas em 600 segundos, de modo a não subestimar a cauda superior.

Três padrões se destacam. O Qwen-2.5-1.5B apresenta a menor latência sob contexto de 128 *tokens*, com mediana de 37,5 segundos em 5 usuários, contra 86,7 segundos do Llama-3.2-1B e 191,1 segundos do TinyLlama-1.1B na mesma condição. A vantagem se inverte nos contextos maiores, nos quais o Llama-3.2-1B passa a apresentar as menores medianas e a menor dispersão. O TinyLlama-1.1B mantém as maiores latências e a maior amplitude interquartil em oito das nove condições. A análise da dispersão reforça as diferenças de desempenho entre os modelos. Para o Llama-3.2-1B, o percentil 95 não ultrapassou 281,3 segundos, enquanto, para o TinyLlama-1.1B, atingiu o limite de 600 segundos nos três níveis de contexto. Como as requisições que excederam o tempo máximo foram censuradas nesse valor, esse resultado indica que os *timeouts* afetaram a cauda superior da distribuição de latência do TinyLlama-1.1B.

A Tabela 2 apresenta os testes estatísticos. O teste de Kruskal-Wallis rejeitou a igualdade das distribuições entre os três modelos nos três níveis de contexto, com $p <$

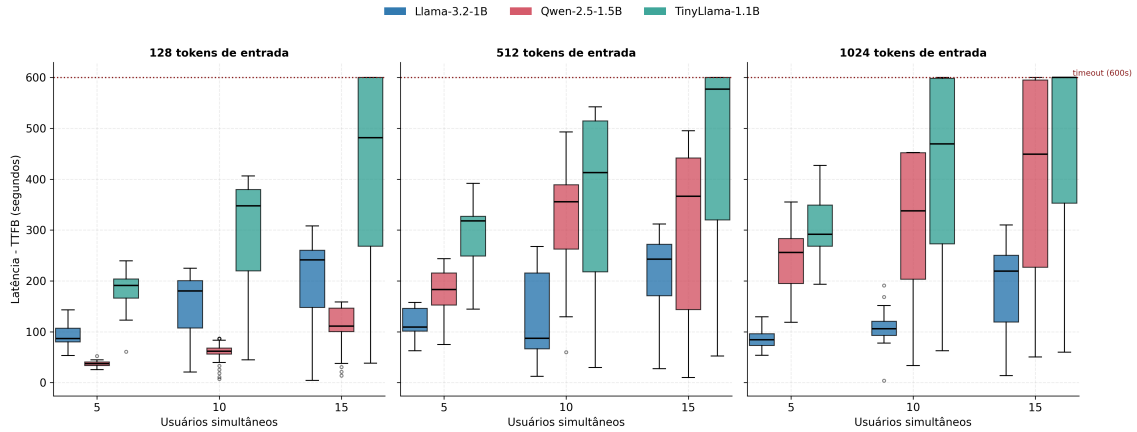


Figure 2. Distribuição do tempo até o primeiro byte por modelo, contexto e concorrência. A linha pontilhada indica o limite de 600 segundos adotado como critério de falha.

10^{-11} em todos os casos. O post-hoc de Dunn indicou diferença significativa em oito das nove comparações par a par. A exceção ocorreu entre Qwen-2.5-1.5B e TinyLlama-1.1B sob 1024 *tokens*, com p corrigido de 0,4035. Sob o maior contexto avaliado, não foi detectada diferença estatisticamente significativa entre as distribuições de tempo de resposta desses dois modelos, embora suas taxas de conclusão tenham sido de 89,4% e 65,0%.

Table 2. Teste de Kruskal–Wallis e comparações post-hoc de Dunn sobre a latência. H e p são a estatística e o valor- p de Kruskal–Wallis; z e p_{Bonf} a estatística padronizada de Dunn e o valor- p corrigido por Bonferroni.

Contexto	H	p	Llama vs. Qwen		Llama vs. Tiny		Qwen vs. Tiny	
			z	p_{Bonf}	z	p_{Bonf}	z	p_{Bonf}
128 tok.	128,17	$1,5 \times 10^{-28}$	7,49	<0,001	-3,70	0,0006	-10,34	<0,001
512 tok.	52,03	$5,0 \times 10^{-12}$	-4,42	<0,001	-6,91	<0,001	-2,51	0,0366
1024 tok.	77,84	$1,2 \times 10^{-17}$	-6,45	<0,001	-7,80	<0,001	-1,50	0,4035

5.3. Vazão e Efeito do Comprimento das Respostas

A Figura 3 apresenta o número de requisições concluídas durante a janela de submissão de cinco minutos. Com entradas de 128 *tokens* e cinco usuários, o Qwen-2.5-1.5B concluiu 43 requisições, enquanto o Llama-3.2-1B e o TinyLlama-1.1B concluíram 21 e 11, respectivamente. O Qwen-2.5-1.5B atingiu seu maior valor nessa configuração de entrada, com 56 requisições e dez usuários. Nos contextos de 512 e 1024 *tokens*, o Llama-3.2-1B apresentou a maior vazão. Com 1024 *tokens* e 15 usuários, concluiu 37 requisições, em comparação com 17 do Qwen-2.5-1.5B e 9 do TinyLlama-1.1B.

A interpretação desses resultados deve considerar o comprimento efetivo das respostas. Embora o limite máximo de saída fosse de 256 *tokens*, os modelos podiam encerrar a geração antes por meio de seus marcadores de parada. Com entradas de 128 *tokens*, as respostas do Qwen-2.5-1.5B, Llama-3.2-1B e TinyLlama-1.1B apresentaram, em média, 84, 255 e 621 caracteres, respectivamente. Esses valores mostram diferenças no

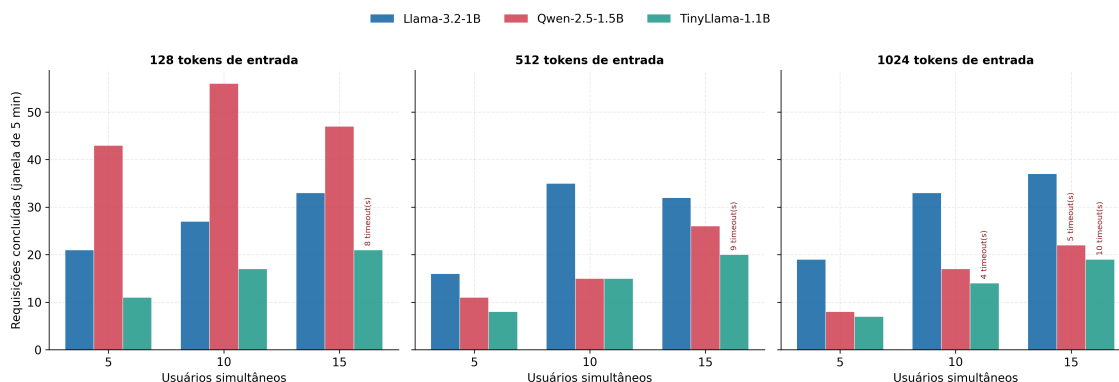


Figure 3. Requisições concluídas durante a janela de submissão de cinco minutos.

comprimento das respostas, que devem ser consideradas na interpretação da quantidade de requisições concluídas por cada modelo.

Para avaliar esse efeito, calculou-se a taxa sistêmica de produção de texto, definida como o total de caracteres gerados dividido pela duração efetiva do teste. Nas condições com 100% de conclusão, essa taxa variou entre 9,0 e 14,2 caracteres por segundo para o Llama-3.2-1B, entre 8,1 e 10,0 para o Qwen-2.5-1.5B e entre 8,6 e 14,5 para o TinyLlama-1.1B. Nessas condições, as faixas de produção de texto apresentaram sobreposição. As respostas mais longas do TinyLlama-1.1B estiveram associadas à menor capacidade de atendimento, enquanto as respostas mais concisas do Qwen-2.5-1.5B acompanharam sua maior vazão com entradas curtas. Entretanto, essa comparação não isola o efeito do comprimento das respostas das demais diferenças entre os modelos e não inclui as condições com timeouts.

5.4. Utilização de Recursos no Servidor

A utilização média de CPU permaneceu entre 78,8% e 98,9% nas 27 condições, com pico igual ou próximo a 100% em todas elas, conforme a Figura 4. Não há diferença estrutural entre os modelos quanto a esse aspecto. Mesmo a condição menos carregada já opera próxima ao limite físico do processador, e o aumento da concorrência de 5 para 15 usuários não produz aumento correspondente na utilização, pois o recurso já se encontra saturado. A CPU é o gargalo comum às três implantações.

A Figura 5 apresenta o comportamento da memória. Em nenhuma das 27 condições o percentual de memória em uso ultrapassou 21,2% da capacidade reportada de 3.795,8 MB, e o pico de memória ativa foi de 711,6 MB. O consumo cresce com o tamanho do contexto, o que reflete a alocação do *cache* de chaves e valores, mas a quantização em 8 bits foi suficiente para acomodar os três modelos com folga. O Llama-3.2-1B não utilizou *swap* em nenhuma condição, enquanto o Qwen-2.5-1.5B atingiu pico de 41,9 MB e o TinyLlama-1.1B atingiu 52,4 MB. O TinyLlama-1.1B foi, ao mesmo tempo, o modelo de menor consumo de RAM, com pico de 14,9%, e o de maior uso de *swap*. Como a memória disponível era abundante, a paginação não decorre de pressão de memória do processo de inferência, e sim da permanência mais longa dos processos em execução. O volume é pequeno em termos absolutos e não configura fator limitante.

O comportamento térmico descarta a limitação térmica como explicação para a

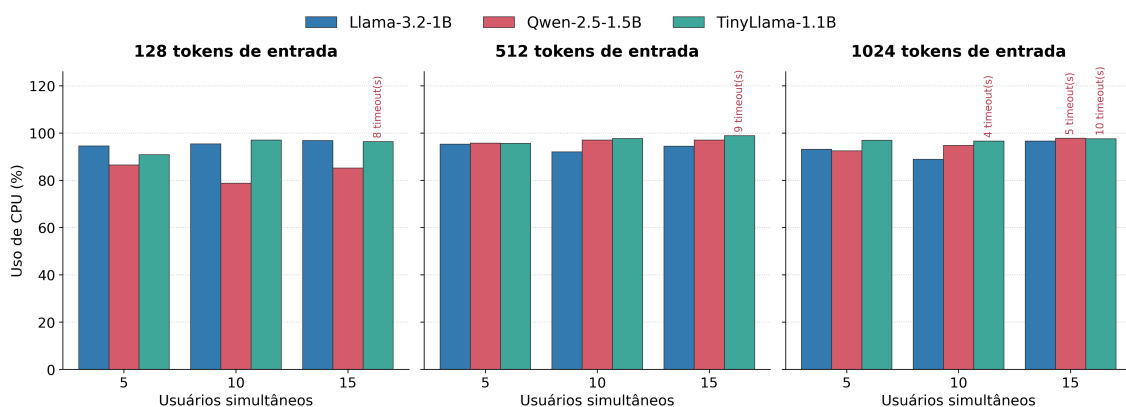


Figure 4. Utilização média de CPU no servidor. Os marcadores indicam o valor de pico observado em cada série.

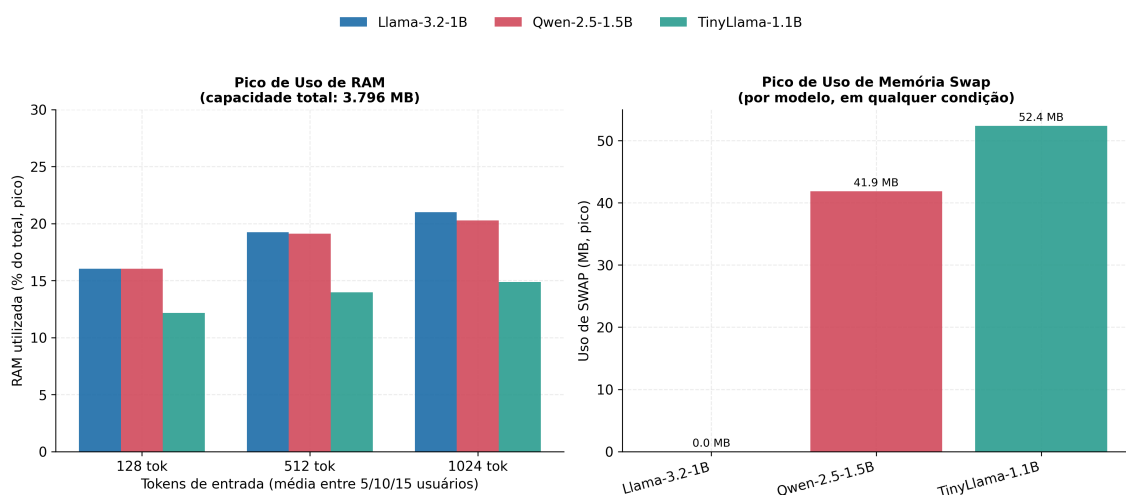


Figure 5. Pico de uso de RAM por modelo e contexto, à esquerda, e pico de uso de memória de troca por modelo, à direita.

degradação observada. As temperaturas médias permaneceram entre 50,1 °C e 56,2 °C, com pico absoluto de 63,7 °C. Como o limiar de redução de frequência por temperatura no Raspberry Pi 4 situa-se entre 80 °C e 85 °C, a margem observada supera 16 °C mesmo nas condições de maior duração, nas quais a CPU permaneceu saturada por mais de 900 segundos consecutivos. A queda de desempenho decorre da capacidade de processamento insuficiente, e não de redução de frequência induzida por temperatura.

5.5. Persistência da Fila Além da Janela de Submissão

A Figura 6 apresenta o tempo total transcorrido entre o início da submissão e o esvaziamento da fila. A janela nominal foi de 300 segundos em todas as condições, e em nenhuma delas o servidor concluiu o atendimento dentro desse intervalo. O tempo total variou de 371 segundos, para o Llama-3.2-1B com 1024 *tokens* e 5 usuários, a 988 segundos, para o TinyLlama-1.1B com 128 *tokens* e 15 usuários.

O acúmulo residual quantifica o desequilíbrio entre a taxa de chegada e a taxa de atendimento. O excedente sobre a janela nominal variou de 24% a 101% para o Llama-3.2-1B, de 27% a 204% para o Qwen-2.5-1.5B e de 74% a 229% para o TinyLlama-1.1B.

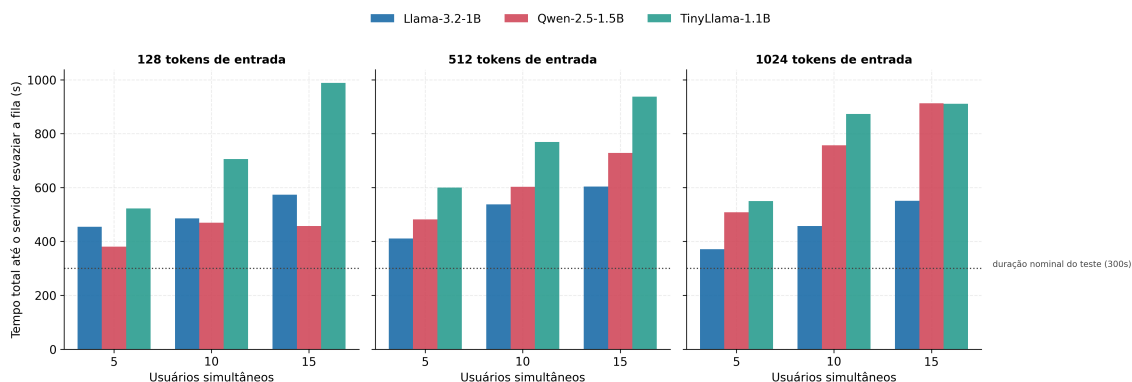


Figure 6. Tempo total até o esvaziamento da fila do servidor. A linha pontilhada indica a duração nominal da janela de submissão.

Um volume concentrado de requisições por apenas cinco minutos deixa o nó de borda ocupado por mais onze minutos após o encerramento da submissão. Requisições que chegarem nesse intervalo enfrentarão fila preexistente, o que amplifica o efeito de picos de demanda. Sem controle de admissão, o sistema não retorna ao regime normal em tempo comparável à duração do evento que o sobrecarregou.

5.6. Discussão e Ameaças à Validade

Os resultados mostram que o número de parâmetros não determina, isoladamente, a adequação de um modelo à execução na borda. Os três modelos avaliados pertencem a linhagens distintas e foram submetidos a diferentes processos de treinamento e alinhamento. O Llama-3.2-1B-Instruct foi desenvolvido como um modelo leve voltado à execução local e ao seguimento de instruções, enquanto o Qwen-2.5-1.5B-Instruct recebeu treinamento posterior direcionado ao aprimoramento dessa capacidade. O TinyLlama-1.1B-Chat, por sua vez, utiliza a arquitetura e o tokenizador do Llama 2 e foi posteriormente ajustado para interação conversacional [Grattafiori et al. 2024, Qwen Team 2025, Zhang et al. 2024]. Portanto, os modelos não representam apenas três tamanhos de uma mesma arquitetura, mas artefatos com diferentes dados, tokenizadores, modelos de conversação e critérios aprendidos para encerrar uma resposta.

Essa heterogeneidade oferece uma interpretação plausível para o comportamento observado. Embora os modelos tenham produzido texto a taxas sistêmicas comparáveis, o Qwen-2.5-1.5B gerou respostas mais concisas, enquanto o TinyLlama-1.1B produziu respostas mais longas e frequentemente se aproximou do limite de saída. Diferenças no ajuste para instruções, no estilo conversacional e no aprendizado dos marcadores de término podem influenciar a quantidade de texto emitida e, conseqüentemente, o tempo de ocupação da CPU por requisição. Os experimentos não permitem atribuir causalmente o resultado a um desses componentes, mas demonstram que o comportamento de geração constitui uma propriedade operacional relevante do modelo implantado. Assim, a seleção para dispositivos de borda deve considerar conjuntamente o comprimento das respostas, a taxa de geração, a utilização de recursos e a capacidade de atendimento, em vez de se basear apenas no número de parâmetros ou no tamanho do arquivo de pesos.

Quanto às ameaças à validade, a censura em 600 segundos limita a caracterização da cauda da distribuição de latência do TinyLlama-1.1B nas condições de maior carga. Além disso, a produção de texto foi medida em caracteres por segundo, sem contagem

pelo tokenizador específico de cada modelo. Essa métrica caracteriza a produção textual do serviço, mas não equivale à velocidade de geração em tokens por segundo. As diferenças entre famílias, dados de treinamento, ajustes conversacionais, tokenizadores e marcadores de parada impedem atribuir o comportamento observado a uma característica isolada. Os modelos também não foram avaliados quanto à qualidade das respostas, e os resultados se restringem a um dispositivo, uma estratégia de quantização, um conjunto de dados e uma disciplina de atendimento serializada.

6. Conclusões

Este trabalho avaliou três LLMs quantizados em 8 bits em um Raspberry Pi 4B sob requisições concorrentes com atendimento serializado. O projeto fatorial abrangeu 27 condições, variando modelo, contexto e número de usuários. O Llama concluiu 100% das requisições, seguido pelo Qwen, com 97,96%, e pelo TinyLlama, com 76,52%, indicando que a contagem de parâmetros, isoladamente, não explica o desempenho do serviço. A CPU constituiu o principal gargalo, enquanto a memória permaneceu com capacidade disponível.

Os resultados também evidenciam que respostas mais longas mantêm o servidor ocupado por mais tempo e, sob atendimento serializado, ampliam a espera em fila e reduzem a capacidade de conclusão das requisições. O esvaziamento da fila prolongou a execução em até 229% além da janela de submissão, reforçando a necessidade de considerar o comportamento de geração e a confiabilidade do serviço, além do tamanho do modelo, no dimensionamento de aplicações de borda. Como trabalhos futuros, pretende-se avaliar processamento em lote, múltiplas instâncias, controle de admissão, qualidade das respostas, outros formatos de quantização e diferentes dispositivos de borda.

Agradecimentos

Este trabalho foi parcialmente financiado pelo projeto “ISOP Engineering: Métodos e Ferramentas de Engenharia Inteligente para Sensoriamento Inteligente”, apoiado pelo Centro de Competência EMBRAPPI VIRTUS em Hardware Inteligente para Indústria (VIRTUS-CC), com recursos financeiros do PPI HardwareBR do MCTI, por meio do convênio nº 055/2023 firmado com a EMBRAPPI. Os autores também agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brasil, pelo apoio financeiro por meio do projeto nº 401147/2025-8.

References

- Cheng, F., Zhang, J., Zhang, J., Guo, C., Zhou, C., Liu, X., and Chen, Y. (2024). Ecco: Improving memory bandwidth and capacity for LLMs via entropy-aware cache compression. In *Proceedings of the International Symposium/Conference*.
- Dhar, N., Wu, X., Deng, B., and Zhao, L. (2024). An empirical analysis and resource footprint study of deploying large language models on edge devices. In *2024 ACM Southeast Conference (ACMSE)*. ACM.
- Dong, Z. and Zhang, X. (2024). Evaluating large language models on the edge based on evaluatology. In *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*. Springer.
- Dunn, O. J. (1964). Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252.

- Durán, F., Martínez, M., Lago, P., and Martínez-Fernández, S. (2025). Insights into resource utilization of code small language models serving with runtime engines and execution providers. *Journal of Systems and Software*, 230:112574.
- Gerganov, G. and llama.cpp contributors (2023). llama.cpp: LLM inference in C/C++. <https://github.com/ggml-org/llama.cpp>. Acesso em agosto de 2026.
- Grattafiori, A., Dubey, A., Jauhri, A., et al. (2024). The Llama 3 herd of models.
- Huang, W., Zheng, X., Ma, X., Qin, H., Lv, C., Chen, H., Luo, J., Qi, X., Liu, X., and Magno, M. (2024). An empirical study of LLaMA3 quantization: from LLMs to MLLMs. *Visual Intelligence*, 2(36).
- Huang, Y., Zheng, K., Yu, Z., and Bouganis, C.-S. (2025). ITERA-LLM: Boosting sub-8-bit large language model inference via iterative tensor decomposition. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE.
- Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons.
- Kruskal, W. H. and Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621.
- Qwen Team (2025). Qwen2.5 technical report.
- Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don’t know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.
- Ray, P. P. and Pradhan, M. P. (2025). DLUSEdge: Dynamic load-unload scheduling for localized LLMs on resource-constrained edge. *KI - Künstliche Intelligenz*.
- Suárez, D. and Blanco, V. (2025). Energy-efficient inference on RNN and LLM networks: A quantized evaluation on RISC-V, ARM, and x86 devices. In *The 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY ’25)*.
- Wei, J., Cao, S., Cao, T., Ma, L., Wang, L., Zhang, Y., and Yang, M. (2025). T-MAC: CPU renaissance via table lookup for low-bit LLM deployment on edge. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys ’25)*. Association for Computing Machinery.
- Wu, J., Dai, H., Wu, J., Jin, W., and Wang, Y. (2024). QMLFQ: A quantization-based queue scheduling framework for efficient LLM serving. In *2024 IEEE Smart World Congress (SWC)*. IEEE.
- Zhang, C., Zhu, X., Chen, L., Yang, T., Pan, E., Yu, G., Zhao, Y., Wu, X., Li, B., Mao, W., and Han, G. (2025). Enhancing LLM inference performance on ARM CPUs through software and hardware co-optimization strategies. *CICAS Integrated Circuits and Systems*.
- Zhang, P., Zeng, G., Wang, T., and Lu, W. (2024). TinyLlama: An open-source small language model.
- Zheng, Y., Chen, Y., Qian, B., Shi, X., Shu, Y., and Chen, J. (2025). A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57(8):1–35.