JITK (JURNAL ILMU PENGETAHUAN DAN TEKNOLOGI KOMPUTER)

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435

LLM INFERENCE PERFORMANCE ANALYSIS ACROSS WINDOWS, WINDOWS SUBSYSTEM FOR LINUX 2 AND NATIVE LINUX

Muhammad Anshori[1]; Maulidiansyah[1*]; Moh Sukron[1]

Teknologi Informasi[1]
Universitas Nurul Jadid Paiton, Probolinggo, Indonesia[1]
unuja.ac.id[1]
muhanshori09@gmail.com, maulid@unuja.ac.id*, moh.sukron12012752@gmail.com

(*) Corresponding Author
(Responsible for the Quality of Paper Content)

Abstract— *The increasing use of Large Language Models (LLMs) on consumer devices raises practical concerns regarding inference latency and resource efficiency, which can vary depending on the operating system environment. However, it is still unknown which operating system is most optimal and efficient for executing local LLM workloads on hardware with limited resources. Therefore, this study aims to compare and evaluate the inference performance of LLMs across three operating system environments: native Windows, Windows Subsystem for Linux 2 (WSL2), and native Linux on a low-spec laptop equipped with a 12th Gen Intel Core i3 processor and 8 GB of RAM. The Ollama framework is used to run several lightweight LLM models, including Llama3.2:3B, Qwen2.5:3B, and Phi3.5:3.8B. An experimental benchmarking approach is conducted using three levels of prompt complexity (light, medium, and heavy), with 10 trials for each prompt scenario to ensure consistency of results. Evaluation metrics include inference latency, CPU utilization, and memory consumption during model execution. Results show that the operating system environment significantly impacts the inference performance of LLMs. For example, using the Llama3.2:3B model under heavy prompting, the average latency reaches 71.24 seconds on Windows, 75.96 seconds on WSL2, and 70.80 seconds on native Linux, while under light prompting the average latency is 11.76 seconds, 12.73 seconds, and 10.83 seconds, respectively. These findings indicate that native Linux generally provides more efficient LLM inference on resource-constrained hardware. This study contributes an empirical and repeatable comparison across operating system environments, providing practical insights for selecting deployment platforms for LLM applications.*

Keywords: *Benchmarking, LLM Inference, Linux, Windows, WSL2.*

Intisari— *Meningkatnya penggunaan Large Language Models (LLMs) pada perangkat konsumen menimbulkan perhatian praktis terkait latensi inferensi dan efisiensi sumber daya, yang dapat bervariasi bergantung pada lingkungan sistem operasi. Namun, masih belum diketahui sistem operasi mana yang paling optimal dan efisien untuk menjalankan beban kerja LLM secara lokal pada perangkat keras dengan sumber daya terbatas. Oleh karena itu, penelitian ini bertujuan untuk membandingkan dan mengevaluasi kinerja inferensi LLM pada tiga lingkungan sistem operasi, yaitu Windows native, Windows Subsystem for Linux 2 (WSL2), dan Linux native, menggunakan laptop dengan spesifikasi rendah yang dilengkapi prosesor Intel Core i3 Generasi ke-12 dan RAM 8 GB. Framework Ollama digunakan untuk menjalankan beberapa model LLM ringan, yaitu Llama3.2:3B, Qwen2.5:3B, dan Phi3.5:3.8B. Pendekatan benchmarking eksperimental dilakukan menggunakan tiga tingkat kompleksitas prompt, yaitu ringan, sedang, dan berat, dengan 10 kali pengujian untuk setiap skenario prompt guna memastikan konsistensi hasil. Metrik evaluasi meliputi latensi inferensi, penggunaan CPU, dan konsumsi memori selama eksekusi model. Hasil menunjukkan bahwa lingkungan sistem operasi berpengaruh signifikan terhadap kinerja inferensi LLM. Sebagai contoh, pada model Llama3.2:3B dengan prompt berat, rata-rata latensi mencapai 71,24 detik pada Windows, 75,96 detik pada WSL2, dan 70,80 detik pada Linux native, sedangkan pada prompt ringan masing-masing mencapai 11,76 detik, 12,73*

detik, dan 10,83 detik. Temuan ini menunjukkan bahwa Linux native secara umum memberikan inferensi LLM yang lebih efisien pada perangkat keras dengan sumber daya terbatas. Penelitian ini berkontribusi melalui perbandingan empiris dan dapat diulang pada berbagai lingkungan sistem operasi, serta memberikan wawasan praktis dalam memilih platform deployment untuk aplikasi berbasis LLM.

Kata Kunci: *Benchmarking, Inferensi LLM, Linux, Windows, WSL2.*

INTRODUCTION

The rapid advancement of artificial intelligence has significantly accelerated the development of Natural Language Processing (NLP) technologies in recent years. One of the most significant breakthroughs in this field is the emergence of Large Language Models (LLMs), which have demonstrated remarkable capabilities in performing various language-related tasks such as text generation, machine translation, question answering, and conversational systems. Recent studies highlight that modern LLMs are built upon transformer-based architectures and continue to evolve with increasing scale and capability, enabling significant improvements across diverse NLP applications [1], [2].

The increasing development of LLM technology has encouraged its adoption in various practical domains including intelligent assistants, automated documentation systems, research support tools, and software development environments. Traditionally, LLM services are deployed using cloud computing infrastructures that provide high computational power and large-scale storage resources. However, recent developments in lightweight models and optimized inference frameworks have made it possible to execute LLM inference locally on personal computing devices such as laptops and desktop computers [3], [4], [5]. Recent studies further demonstrate that production-grade local LLM inference can be efficiently deployed using optimized inference engines such as Ollama, MLX, llama.cpp, and PyTorch MPS. Moreover, advances in CPU-oriented inference optimization have significantly improved the feasibility of executing LLMs on consumer-grade hardware with limited computational resources [6], [7]. Running LLMs locally offers several advantages including improved privacy, reduced network latency, and lower operational costs compared to cloud-based deployments.

Despite these advantages, executing LLM inference on consumer-grade hardware remains a challenging task. Large language models require significant computational resources due to the intensive matrix operations and memory access patterns involved in generating output tokens. These requirements often result in high inference latency and substantial memory consumption when deployed on devices with limited hardware capabilities. As a result, optimizing LLM inference performance on resource-constrained environments has become an important research topic in recent years [8], [9]. Recent research has shown that processor efficiency, resource allocation, and CPU-induced execution overhead substantially influence inference latency and throughput, particularly when LLMs are executed on devices with limited computational capabilities [7], [10], [11].

Several studies have explored various strategies to improve the efficiency of LLM inference on limited computing resources. Research on edge-based LLM deployment shows that optimization techniques such as model compression, quantization, and efficient memory management can significantly enhance inference performance on low-resource devices [12], [13], [14]. Recent evaluations on resource-constrained platforms further highlight that hardware characteristics and deployment environments significantly affect latency, throughput, and overall inference efficiency [15], [11]. Other studies emphasize that inference performance is influenced by multiple factors including model architecture, prompt complexity, hardware configuration, runtime environments, inference-serving frameworks, and benchmarking methodologies [16], [17], [18], [19]. These findings indicate that evaluating the interaction between system environment and model execution is essential for understanding real-world LLM deployment performance.

In addition to hardware configuration and model optimization techniques, the operating system environment also plays a critical role in determining system performance. Operating systems are responsible for managing fundamental computing resources including CPU scheduling, memory allocation, process management, and input-output operations. Differences in operating system architectures and resource management mechanisms can lead to variations in system performance when executing computational workloads. Therefore, the choice of operating system environment may significantly influence the efficiency of LLM inference execution [20], [21].

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435

Modern computing platforms provide several operating system environments that can be used to execute machine learning workloads. Native operating systems such as Windows and Linux provide direct access to hardware resources, enabling optimized performance for many computational tasks. On the other hand, virtualization-based technologies such as Windows Subsystem for Linux 2 (WSL2) enable Linux environments to run within Windows systems through lightweight virtualization mechanisms. While this approach offers flexibility, it may introduce additional overhead that affects performance under certain workloads.

Although previous studies have extensively investigated local LLM inference, benchmarking methodologies, CPU optimization, and on-device deployment across various hardware platforms [6], [7], [11], [15], [18]-[20], most of these studies primarily focus on improving inference engines, hardware acceleration, or deployment efficiency rather than evaluating the influence of different operating system environments. Consequently, there is still limited empirical evidence regarding how different operating system environments, particularly native Windows, Windows Subsystem for Linux 2 (WSL2), and native Linux, affect LLM inference performance when deployed on commonly used consumer hardware such as laptops [14], [21]. Furthermore, while prior research has addressed system-level optimizations such as memory-efficient inference mechanisms [22], limited attention has been given to the comparative impact of operating system environments on lightweight LLM inference performance under resource-constrained conditions.

Therefore, this study aims to analyze and compare LLM inference performance across three operating system environments, namely native Windows, Windows Subsystem for Linux 2 (WSL2), and native Linux. The experiments were conducted on a low-specification laptop using lightweight LLM models deployed through the Ollama framework. By evaluating inference latency, CPU utilization, and memory consumption across different environments, this research aims to provide practical insights for selecting the most efficient platform for running LLM-based applications on resource-constrained devices. The primary contribution of this study is a systematic and reproducible benchmarking analysis of lightweight LLM inference performance using Ollama across Windows, WSL2, and Linux environments on resource-constrained hardware.

This empirical evaluation explicitly fills the gap in literature by shifting the research focus from high-end GPU clusters to accessible consumer-grade hardware, thereby providing actionable reference data for developers and researchers aiming to optimize local AI deployments under strict hardware limitations.

MATERIALS AND METHODS

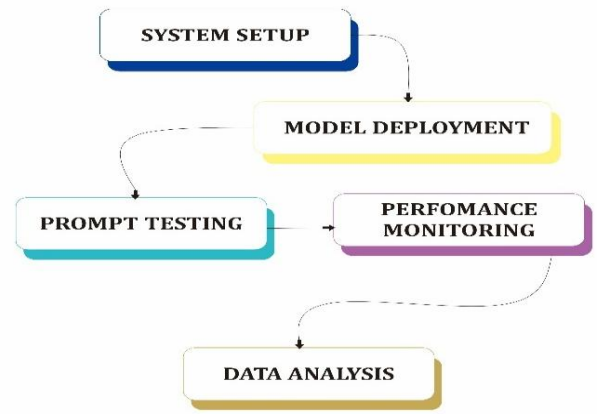
This study applies an experimental benchmarking approach to evaluate the performance of Large Language Model (LLM) inference across various operating system environments. The experimental design and performance metrics adopted in this study were developed by considering recent benchmarking methodologies for LLM inference evaluation proposed in previous studies [18]–[20].

The primary objective of this study is to analyze how the operating system environment affects LLM inference performance when running on low-spec consumer devices. The experimental design compares three operating system environments: native Windows, Windows Subsystem for Linux 2 (WSL2), and native Linux. All experiments were conducted using identical hardware specifications and model configurations to ensure fair performance comparisons between the evaluated environments.

To maintain strict experimental consistency and eliminate confounding external variables, several system-level control mechanisms were rigorously implemented during the benchmarking process. First, all operating system environments utilized identical hardware configurations and model versions, with all inferences executed exclusively in CPU-only mode to establish a uniform hardware baseline. Second, prior to each measurement, all non-essential background applications and system services were terminated to minimize CPU and memory overhead, while the system power plan was configured to high-performance mode.

Third, to isolate pure token generation performance, each model was preloaded into system memory, thereby excluding model-loading latency from the final evaluation metrics. Furthermore, a mandatory 30-second cooling interval was enforced between successive trials to mitigate thermal throttling and sustain stable processor clock speeds. Finally, the procedure involved a strict warm-up protocol consisting of 2 initial warm-up runs followed by 10 independent steady-state trials for each model across three distinct prompt complexities (light, medium, and heavy).

The final evaluation metrics were computed based on the arithmetic mean of these 10 subsequent repetitions to guarantee statistical stability, eliminate transient measurement anomalies, and isolate pure steady-state computational performance.



Source: (Research Results, 2026)
Figure 1. Research Workflow

The experimental workflow used in this study consists of several stages, including system setup, model deployment, prompt testing, performance monitoring, and data analysis. The overall research process is illustrated in Figure 1.

Experimental Design

The experiments were conducted on a laptop equipped with a 12th Gen Intel Core i3 processor and 8 GB of RAM. This hardware configuration was intentionally chosen to represent low-resource consumer devices commonly used by developers and researchers. The operating systems evaluated in this study consisted of Windows 11 as the native Windows environment, Ubuntu Linux as the native Linux environment, and Windows Subsystem for Linux 2 (WSL2) as a virtualization-based Linux environment running within Windows. Each operating system environment was configured with similar runtime conditions to minimize external factors that could affect the benchmarking results.

This controlled experimental configuration follows the principle of fair benchmarking commonly adopted in recent LLM inference performance studies, where identical hardware, runtime configuration, and model settings are maintained to isolate the influence of the evaluated variable [15], [18], [11].

Hardware and Software Configuration

The hardware and system configurations used in this experiment are summarized in Table 1.

Table 1. Experimental System Specification

Component Name	Specification
Laptop	Acer Aspire Lite 14
Processor	Intel Core i3 12th Gen (i3-1215U)
RAM	8 GB DDR5
OS Tested	Windows 11 Home Single Language 64-bit, WSL2 version 2.6.3.0 (Kernel v6.6.87.2-1), Ubuntu Server 24.04.3 LTS
Framework	Ollama (0.24.0)
Models	Llama3.2:3B, Qwen2.5:3B, Phi3.5:3.8B

Source: (Research Results, 2026)

To ensure strict reproducibility and eliminate software-induced performance variances, all software stacks and operational environments were locked to specific version releases. The benchmarking was conducted utilizing the Ollama framework version 0.24.0 across all three platforms, with WSL2 configured using version 2.6.3.0 (Kernel v6.6.87.2-1) and both virtualized and native Linux environments standardizing on Ubuntu Server 24.04.3 LTS. Under these controlled versions, all environments were configured under similar conditions by running the experiments without GPU acceleration (CPU-only inference), minimizing background applications, setting the system power mode to performance, and using identical model versions as well as the same prompt inputs across all operating systems.

Model Deployment

All Large Language Models (LLMs) evaluated in this study were deployed using the Ollama framework, which provides a unified and efficient interface for running local inference on consumer-grade hardware. Using Ollama allows for consistent model execution across multiple operating system environments, so observed performance variations are more influenced by system characteristics than by differences in deployment methods. Each model was downloaded and configured using the same version to maintain consistency throughout the entire experiment.

Before each test run, the model was preloaded into system memory to avoid initialization overhead that could impact latency measurements. With this approach, the recorded inference times represent the generation process purely, unaffected by model loading times. All experiments were run in CPU-only conditions without GPU acceleration to ensure equitable computing resources across all tested environments.

Furthermore, the Ollama runtime configuration was kept identical, including model

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435

parameters, inference settings, and execution commands used during testing. No additional optimization techniques, such as advanced quantization or dedicated hardware acceleration, were performed beyond the default configuration. This approach aims to ensure that the results are reproducible, objective, and directly reflect the impact of the operating system environment on LLM inference performance.

Prompt Design

To evaluate inference performance under varying workloads, three levels of prompt complexity were defined, namely light, medium, and heavy prompts. The light prompt consists of short input with simple instructions, the medium prompt includes moderate-length input with structured queries, while the heavy prompt contains long input requiring complex reasoning or extended generation. Each prompt category was designed to simulate real-world usage scenarios and ensure consistent token processing behavior.

Experimental Procedure

Each experiment was conducted using a systematic procedure in which the model was first loaded into memory using the Ollama framework. For each combination of model, prompt category, and operating system environment, a strict warm-up protocol of 2 initial runs was executed to stabilize the runtime environment, followed by 10 successive data-collection trials. This process resulted in a total of 90 steady-state evaluation trials (out of 108 total physical executions) performed for each model across the three prompt levels and three operating system environments.

Performance Metrics

The selected evaluation metrics are consistent with recent benchmarking studies that assess LLM inference performance based on latency, throughput, computational resource utilization, and execution efficiency across different deployment environments [18], [9], [11].

The performance evaluation in this study is based on four main metrics. Inference latency is measured in seconds as the total time required by the model to generate output. To rigorously address token-length variability and eliminate it as a confounding factor, token generation throughput—measured as the evaluation rate in tokens per second (tokens/s)—along with the total generated token count, is introduced as a critical core metric. This ensures that any observed latency differences are verified to be a direct consequence of operating system architectural efficiency rather than mere fluctuations in output length.

CPU utilization is recorded as the average percentage of processor usage during inference execution. Memory consumption is measured as the peak RAM usage observed during model execution. These metrics are selected to represent computational efficiency and resource utilization in each operating system environment.

Data Analysis

The collected data were analyzed using descriptive statistical methods, with a primary focus on mean values obtained from repeated experimental trials to ensure measurement stability. Comparative analysis was conducted to evaluate differences in inference latency, CPU utilization, and memory consumption across different operating system environments and prompt complexity levels. This approach enables the identification of performance trends, variability patterns, and system behavior, providing a clear and structured interpretation of how each environment impacts LLM inference performance.

RESULTS AND DISCUSSION

LLM Inference Latency Performance

This section presents the empirical findings and comparative analysis of the LLM inference benchmarks across the evaluated operating systems. The experimental data, collected from various prompt granularities, reveal distinct performance trade-offs in latency, CPU utilization, and memory consumption among native Windows, WSL2, and native Linux environments. Rather than merely describing the metrics, the following analysis interprets these variations through the lens of operating system architecture, resource scheduling, and virtualization overhead. To begin with, the comprehensive data for average inference latency across all testing scenarios are summarized in Table 2. Table 2 presents the average inference latency results obtained from the experiments

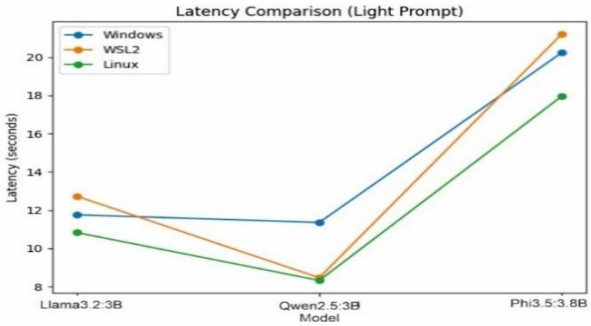
Table 2. Average Latency of LLM Inference Across Prompt Complexity

Model	Prompt	Windows (s)	WSL2 (s)	Linux (s)
Llama3.2:3B	Light	11.76	12.73	10.83
Llama3.2:3B	Medium	40.17	45.40	41.22
Llama3.2:3B	Heavy	71.24	75.96	70.80
Qwen2.5:3B	Light	11.36	8.47	8.33
Qwen2.5:3B	Medium	42.56	48.31	40.39
Qwen2.5:3B	Heavy	76.40	72.91	72.18
Phi3.5:3.8B	Light	20.24	21.21	17.96
Phi3.5:3.8B	Medium	84.05	103.26	82.69

Model	Prompt	Windows (s)	WSL2 (s)	Linux (s)
Phi3.5:3.8B	Heavy	142.39	160.60	141.43

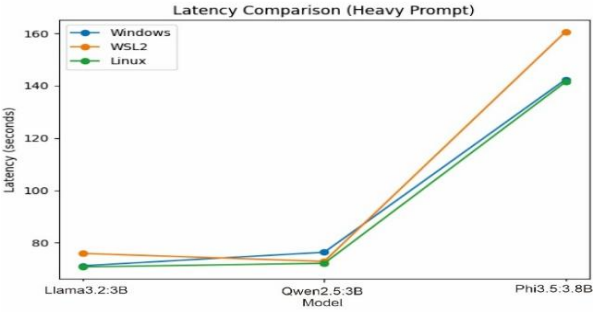
Source: (Research Results, 2026)

As demonstrated by the empirical data in Table 2, the experimental results show that prompt complexity significantly impacts LLM inference latency, with execution times consistently escalating across all evaluated models as prompts scale from light to heavy scenarios. This trend is expected, as larger prompts require more token processing and attention computation in the transformer.



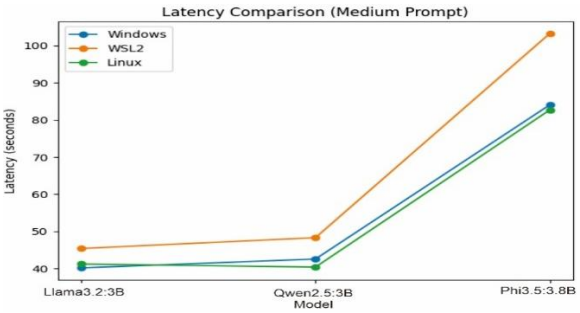
Source: (Research Results, 2026)
Figure 2. Latency Comparison under Light Prompt

As shown in Figure 2, Phi3.5:3.8B consistently exhibits the highest latency among the tested models, particularly under light prompt workloads. The heavy prompt latency comparison across different models and operating systems is presented in Figure 3.



Source: (Research Results, 2026)
Figure 3. Latency Comparison under Heavy Prompt

whereas as demonstrated in Figure 3, the average inference time for Phi3.5:3.8B exceeds 140 seconds on both Windows and Linux. This behavior is primarily due to the model's larger size and computational complexity compared to the other tested models. Conversely, Qwen2.5:3B demonstrates the fastest inference performance in several scenarios, particularly under light prompts, where the average latency is below 9 seconds on Linux. Figure 4 illustrates the latency comparison across different operating systems for the medium prompt scenario.



Source: (Research Results, 2026)
Figure 4. Latency Comparison under Medium Prompt

The data presented in Figure 4 suggests that smaller model architectures can deliver faster response times when implemented on devices with limited computing resources.

Token Throughput Analysis

To rigorously validate that the observed latency variations are driven by operating system efficiency rather than stochastic fluctuations in output text length, an explicit evaluation of token generation throughput was conducted. Table 3 presents the condensed throughput rate measured in tokens per second (tokens/s) alongside the total generated token count enclosed in parentheses.

Table 3. Token Generation Throughput (tokens/s) and Total Tokens Across Environments

Model	Prompt	Windows	WSL2	Linux
Llama3.2:3B	Light	12.44 (124)	10.98 (121)	12.71 (125)
	Medium	11.66 (454)	9.34 (456)	11.38 (425)
	Heavy	10.80 (710)	9.76 (713)	10.73 (761)
Qwen2.5:3B	Light	13.57 (149)	11.12 (146)	14.10 (151)
	Medium	12.69 (516)	10.15 (508)	12.94 (498)
	Heavy	12.31 (604)	9.98 (611)	12.55 (622)
Phi3.5:3.8B	Light	4.82 (138)	4.11 (135)	4.95 (140)
	Medium	4.55 (398)	3.88 (402)	4.62 (395)
	Heavy	4.21 (685)	3.62 (689)	4.28 (692)

Source: (Research Results, 2026)

As demonstrated in Table 3, native Linux and Windows Native consistently outperform WSL2 in token generation speed across all testing matrices. Native Linux delivers the highest overall throughput in most scenarios, peaking at 14.10

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435

tokens/s for Qwen2.5:3b under light workloads. Interestingly, Windows Native demonstrates competitive efficiency, slightly leading in Llama3.2:3b medium (11.66 tokens/s) and heavy (10.80 tokens/s) scenarios. Conversely, WSL2 experiences a systematic performance penalty across all models, often dropping below 10 tokens/s in heavier workloads. This empirical evidence validates the virtualization layer overhead during CPU-only local LLM inference, while simultaneously proving that the observed lower latencies are indeed driven by architectural efficiency rather than shorter token responses.

CPU Utilization Analysis

Table 4. Average Total CPU Utilization (mean ± SD, %) across 30 Runs per Configuration

Model	Windows(%)	WSL2 (%)	Linux(%)
Llama3.2:3B	26.20 ± 2.28	51.00 ± 1.19	**24.97 ± 0.52**
Qwen2.5:3B	25.72 ± 2.02	50.55 ± 1.82	**24.85 ± 0.79**
Phi3.5:3.8B	27.11 ± 1.92	51.73 ± 0.70	**25.33 ± 0.43**
Overall	**26.34 ± 2.16**	**51.09 ± 1.40**	**25.05 ± 0.64**

Source: (Research Results, 2026)

Table 4 summarizes the average total CPU utilization (CPU_Total_Avg) for each model and operating system, computed over all 30 runs per configuration. The results reveal a substantial difference between environments. Native Linux recorded the lowest and most stable CPU utilization (overall mean 25.05% ± 0.64%), closely followed by Windows (26.34% ± 2.16%), whereas WSL2 consumed almost twice as much (51.09% ± 1.40%).

Importantly, the higher CPU utilization observed in WSL2 does not translate into faster inference; on the contrary, WSL2 generally produced the highest latency — for example, 160.60 s for Phi3.5:3.8B under heavy prompts, compared with 141.43 s on native Linux. This indicates that a significant portion of the additional CPU activity in WSL2 is consumed by the virtualization layer (the lightweight utility VM, the virtualized Linux kernel, and the associated host-side and I/O virtualization processes) rather than by useful inference computation. In contrast, native Linux sustains the compute-intensive matrix operations of LLM inference with the lowest CPU overhead and the smallest variance, which is consistent with its lowest and most consistent latency. Windows lies between the two: its average CPU utilization is close to that of Linux, but its higher standard deviation reflects less consistent process scheduling under background system activity.

Regarding the workload distribution across CPU cores, it is crucial to clarify that the Ollama framework utilizes a multi-threaded execution backend via llama.cpp. This architectural design inherently parallelizes heavy tensor matrix multiplications across all available physical CPU cores simultaneously rather than saturating a single core. Consequently, the ∼25% average utilization observed in native Linux and Windows signifies well-distributed parallel processing across the 12th Gen Intel hybrid core architecture, completely eliminating single-core bottlenecks. Meanwhile, the elevated ∼51% utilization in WSL2 represents the systemic virtualization overhead required to schedule, mirror, and synchronize these distributed multi-threaded operations across virtualized CPU environments (vCPUs). These empirical findings demonstrate that native operating system environments offer inherently superior resource allocation and lower translation overhead, making direct execution paths far more efficient for compute-intensive local large language model workloads than virtualized runtimes.

Memory Consumption Analysis

Memory usage is another important factor when implementing LLM inference on consumer devices with limited RAM capacity. Experimental results show that each LLM model requires different memory allocations during execution.
For example:

Table 5. Memory Consumption of LLM Models

Model	Memory Usage
Llama3.2:3B	2.5 GB of RAM
Qwen2.5:3B	2.2 – 4.7 GB
Phi3.5:3.8B	5.8 GB of RAM

Source: (Research Results, 2026)

Table 5 shows the average Resident Set Size (RSS) for Ollama across the three operating system environments for each model. Memory consumption is inherently model-dependent, primarily determined by the parameter size and quantization level of the LLM. Relative to the 8 GB of available hardware RAM on the testing device, the three models occupy substantially different proportions of system memory. Llama3.2:3B requires approximately 2.5 GB (about 31% of total RAM), Qwen2.5:3B ranges from 2.2 to 4.7 GB (about 28–59%) depending on context demands, and Phi3.5:3.8B consumes between 3.7 and 5.8 GB (up to 72% of total RAM).

Crucially, a comparative analysis between average RSS and peak RSS reveals important structural insights into how each operating system handles memory allocation during different

inference phases. Average RSS reflects the steady-state memory required to hold the active neural network weights in physical RAM during matrix multiplications. In contrast, peak RSS captures the maximum volatile memory boundary reached during the initial model loading and the context pre-fill phase, where heavy prompt tokens are tokenized and processed into KV caches.

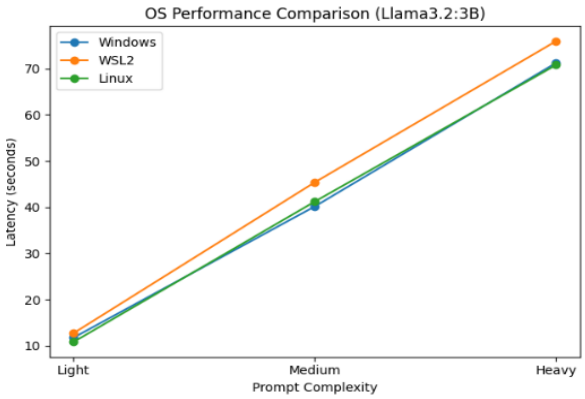
The high memory footprint of models like Phi3.5:3.8B leaves only a small margin for the operating system and background processes, which significantly increases the risk of memory pressure. Under such conditions, the operating system may resort to paging or swapping, which severely degrades inference performance because disk access is far slower than RAM access. These empirical results indicate that, on 8 GB devices, models whose memory footprint approaches or exceeds 70% of total RAM should be used with extreme caution due to the high probability of triggering OS-level memory mitigation.

The variance between average and peak RSS further illuminates how different environments apply distinct memory-management and paging strategies under high memory pressure. Native Linux demonstrates superior architectural efficiency by leveraging native memory mapping (mmap), which allows the Ollama runtime to map the large LLM binary weights directly into the process's virtual address space. This avoids explicit buffer duplication, minimizes kernel-space paging overhead even under high memory constraints, and grants the Completely Fair Scheduler (CFS) direct access to memory pages. Conversely, native Windows manages memory through its Working Set mechanism, which aggressively trims physical memory allocations under background system activity to stave off memory pressure, leading to slightly lower average allocations but introducing subtle latency penalties due to frequent soft page faults when pages must be re-allocated.

WSL2 operates on an intermediate layer, utilizing a virtualized Linux kernel within a lightweight utility VM. Memory management in WSL2 relies on dynamic memory ballooning and page reclamation enforced by the Windows Hyper-V hypervisor. This virtualization layer introduces an additional address translation abstraction (Guest Physical Address to Host Physical Address), adding severe overhead during heavy context expansions when the system approaches its 8 GB physical limit and triggers hypervisor-level memory swapping.

Operating System Performance Comparison

A comparative analysis across different operating system environments revealed several important findings. First, native Linux generally provided the most stable inference performance in terms of latency and system resource utilization. Figure 5 presents a comparison of Llama3.2:3B latency across various operating system environments and prompt complexities.



Source: (Research Results, 2026)

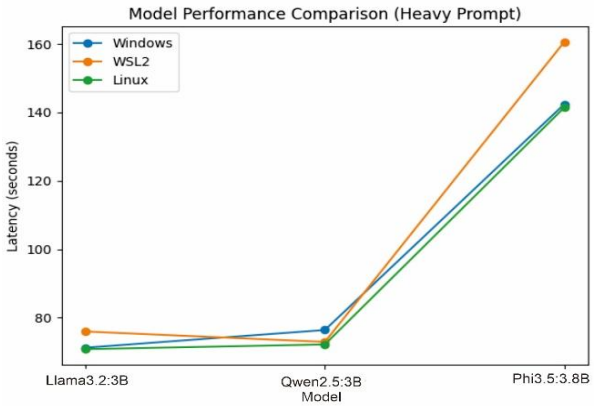
Figure 5. Operating System Performance Comparison

As demonstrated by the execution latency curves in Figure 5, native Linux consistently achieves the shortest processing times across all testing configurations, which can be attributed to Linux's efficient process scheduling mechanism and optimized resource management for computational workloads. This performance advantage occurs because Linux's Completely Fair Scheduler (CFS) maintains strict thread affinity, preventing unnecessary core migrations and maximizing $L1/L2/L3$ CPU cache locality during intense matrix multiplications. Second, the empirical data plotted in Figure 5 reveals that WSL2 introduced additional latency overhead in some scenarios, particularly under demanding prompt workloads. This behavior is directly driven by the virtualization layer used by WSL2 to run the Linux environment within Windows.

Specifically, the host Windows Hyper-V hypervisor imposes a severe double-scheduling penalty. Compute threads must first be managed by the virtualized guest Linux kernel as virtual CPUs (vCPUs) before being re-mapped onto physical CPU cores (pCPUs), which disrupts synchronization and slows down the token generation loop. Finally, while Figure 5 indicates that native Windows environments were still capable of successfully running LLM inference with intermediate latency margins, a comparison of the overall model performance under heavy prompt workload is illustrated in Figure 6.

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435



Source: (Research Results, 2026)

Figure 6. Model Performance Comparison under Heavy Prompt

Experimental results showed that Linux-based environments often provided slightly better performance consistency when running machine learning workloads.

Discussion

The observed performance differences across operating system environments can be explained by a structured interplay between underlying system-level mechanisms (such as process scheduling, memory management, and virtualization layers) and the inherent computational complexity of the language models.

Quantifying Operating System Overhead and Architectural Impacts

The experimental results provide strong numerical evidence regarding the architectural efficiency differences between environments. Native Linux consistently demonstrates the lowest and most stable resource overhead, maintaining an overall mean CPU utilization of only 25.05% ± 0.64%. This efficiency directly translates to optimized execution speeds. For instance, looking at Llama3.2:3B under heavy prompt workloads, the average latency is 70.80 s on native Linux, 71.24 s on Windows, and 75.96 s on WSL2. This indicates that the operating system environment alone accounts for a modest latency increase of approximately 0.6% on Windows and 7.3% on WSL2 relative to native Linux.

To systematically eliminate the potential bias of one-time cold-start initialization overhead and isolate pure steady-state computational performance, our framework utilizes a strict warm-up protocol consisting of 2 initial warm-up runs followed by 10 successive trials per scenario. In virtualized architectures like WSL2, these initial warm-up runs allow for virtual machine instantiation, dynamic memory ballooning stabilization, and page table mapping to complete, and are explicitly excluded from the final latency metrics. The reported averages are computed solely from the 10 subsequent steady-state trials. Because WSL2 consistently maintains a significant 7.3% to 13.6% latency deficit even under these fully stabilized conditions, it provides robust empirical evidence that the observed slowdown is a reflection of sustained runtime virtualization and cross-layer system call overhead during active inference, rather than a transient initialization bottleneck.

This performance hierarchy is governed mainly by the operating system layer rather than the model, as CPU utilization remains nearly constant across different models within each specific environment (about 25% on Linux, 26% on Windows, and 51% on WSL2). Linux's Completely Fair Scheduler (CFS) achieves this by minimizing thread scheduling jitter and preserving strict core affinity. In sharp contrast, WSL2 introduces a severe virtualization penalty, consuming almost twice the CPU resources (51.09% ± 1.40%). This massive duplication of CPU activity represents systemic overhead caused by the Hyper-V hypervisor layer, which repeatedly executes guest-to-host system calls and manages virtual machine page tables, subtracting valuable clock cycles from the actual inference threads.

Deconvoluting Model Size vs. Environment Bottlenecks

While operating system architecture sets the performance ceiling, the model parameter size dictates the baseline resource demand. In fact, a quantitative comparison reveals that model size exerts a far more dominant effect on latency than the OS environment. Under heavy prompts on native Linux, changing the model from Llama3.2:3B (70.80 s) to Phi3.5:3.8B (141.43 s) requires almost exactly twice the inference time. The critical impact of this scaling is tied directly to the hardware boundaries; Llama3.2:3B requires only a 2.5 GB RAM footprint, whereas Phi3.5:3.8B consumes up to 5.8 GB of RAM, occupying roughly 72% of the total available 8 GB system memory.

When a large model footprint is subjected to heavy prompt weights, the operating system's memory-management strategy becomes the definitive bottleneck. Under heavy prompt complexities, the execution latency for Phi3.5:3.8B spikes to 160.60 seconds on WSL2 compared to 141.43 seconds on native Linux, displaying a pronounced 13.6% latency penalty. On an 8 GB RAM device, the 5.8 GB footprint of Phi3.5 leaves

less than 2.2 GB for the OS and background processes. Under this extreme memory pressure, native Linux minimizes latency by utilizing direct memory mapping (mmap) to bypass buffer duplication. Conversely, WSL2 triggers dynamic memory ballooning and Hyper-V address translations (Guest Physical Address to Host Physical Address), forcing the host Windows OS to resort to aggressive Working Set trimming and background paging, which exponentially degrades performance.

From a practical perspective, these findings suggest that developers targeting low-resource environments should prioritize lightweight models and consider deploying them on native Linux systems to achieve optimal performance. This has important implications for edge computing, local AI deployment, and privacy-preserving applications. Despite these insights, this study has critical limitations regarding its hardware environment that must be technically contextualized. Primarily, the experiments were conducted strictly using CPU-only configurations, which completely fail to capture the massive SIMT (Single Instruction, Multiple Threads) parallelism and specialized tensor core acceleration offered by modern GPUs.

In CPU-only setups, inference is fundamentally bound by sequential instruction pipelines and standard system bus bandwidth, creating a severe CPU compute bottleneck during large-scale matrix multiplications that masks the advanced operating system scheduling behaviors typically found in GPU environments. Furthermore, the reliance on a single hardware configuration consisting of a 12th Gen Intel Core i3 processor with 8 GB of RAM introduces an inherent architectural bias. Hardware-specific signatures—such as the configuration of $L1/L2/L3$ caches and memory bus width—directly dictate how effectively an operating system can maintain thread affinity and minimize cache thrashing, thereby limiting the immediate generalizability of these findings to heterogeneous setups like multi-channel servers or unified-memory architectures (e.g., ARM-based SoCs).

Beyond hardware architectural constraints, the evaluation focused exclusively on lightweight 3B-class models (ranging from 3B to 3.8B parameters), which does not accurately represent the non-linear scaling behavior of larger LLMs (e.g., 7B, 13B, or 70B parameters). For lightweight models on an 8 GB system, memory utilization approaches but mostly remains within manageable operating system paging thresholds. However, scaling to larger models fundamentally shifts the system bottleneck from a compute-bound state to an absolute I/O-bound crisis. Under severe memory constraints, the operating system is forced to engage in aggressive disk swapping (paging) to handle the model footprint, meaning that the magnitude of performance degradation is governed almost entirely by storage read/write bandwidth rather than OS scheduling efficiencies. To address these boundaries, future work should expand upon these limitations by exploring heterogeneous hardware setups, GPU acceleration frameworks (such as CUDA or ROCm), larger model scales, and container-based deployment scenarios to isolate the interaction between containerization layers and host operating system schedulers.

CONCLUSION

This study successfully demonstrates that the choice of operating system environment is a critical determinant for optimizing local LLM inference on resource-constrained hardware. Empirical results directly answer the research objectives by establishing that native Linux consistently delivers the most optimal and stable performance, yielding the lowest latency and lowest CPU utilization (25.05% ± 0.64%) across all testing scenarios. Conversely, WSL2 introduces a consistent performance penalty, resulting in a latency increase of up to 13.6% under heavy prompt workloads. In terms of model scalability, while prompt complexity drives execution times higher across all configurations, Qwen2.5:3B achieves the fastest response times under light workloads, whereas Phi3.5:3.8B represents the baseline hardware ceiling due to its larger memory footprint.

Ultimately, these findings provide a clear roadmap for developers, confirming that lightweight 3B-class LLMs can be effectively deployed locally on consumer-grade CPU hardware without dedicated GPU acceleration, provided that an efficient runtime environment like native Linux is prioritized. Additionally, to mitigate the risks of thermal throttling and heat-induced CPU performance degradation during the 30 steady-state inference trials per operating system configuration on this resource-constrained device, a strict cooling interval was maintained between successive trials. This experimental control is empirically validated by the remarkably low standard deviations observed across all datasets, confirming that progressive thermal degradation did not skew or distort the comparative performance metrics. To expand upon these boundaries, future research will transition from CPU-only limitations to explore the efficiency

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 12. NO. 1 AUGUST 2026
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v12i1.8435

impacts of model quantization techniques, GPU acceleration frameworks, and container-based deployment environments on low-resource edge devices.

REFERENCE

[1] K. Alizadeh *et al.*, "LLM in a flash: Efficient Large Language Model Inference with Limited Memory," *Proc. Annu. Meet. Assoc. Comput. Linguist.*, vol. 1, pp. 12562–12584, 2024, doi: 10.18653/v1/2024.acl-long.678.

[2] W. X. Zhao, K. Zhou, and J. Li, "A Survey of Large Language Models," *Front. Comput. Sci.*, vol. 19, no. 7, pp. 1–144, 2025, doi: 10.1007/s11704-024-40663-9.

[3] J. Xiao, Q. Huang, X. Chen, and C. Tian, "Understanding Large Language Models in Your Pockets: Performance Study on COTS Mobile Devices," vol. 14, no. 8, pp. 1–18, 2024, [Online]. Available: http://arxiv.org/abs/2410.03613

[4] Y. Shen, Y. Zhang, and D. Yuan, "A Hybrid Online and Offline Requests Inference Serving System for LLM in Private Computer Environment," *IEEE Trans. Comput.*, vol. 75, no. 3, pp. 888–900, 2025, doi: 10.1109/TC.2025.3643296.

[5] K. T. Chitty-Venkata *et al.*, "LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators," *Proc. SC 2024-W Work. Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1362–1379, 2024, doi: 10.1109/SCW63240.2024.00178.

[6] V. Rajesh, O. Jodhpurkar, P. Anbuselvan, M. Singh, and A. Jallepali, "Production-Grade Local LLM Inference on Apple Silicon: A Comparative Study of MLX, MLC-LLM, Ollama, llama.cpp, and PyTorch MPS," *arXiv Prepr.*, no. Llm, 2026.

[7] H. Shen, H. Chang, B. Dong, Y. Luo, and H. Meng, "Efficient LLM Inference on CPUs," no. NeurIPS, pp. 33–46, 2025, doi: 10.1007/978-3-031-85747-8_3.

[8] S. Samsi *et al.*, "From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference," *2023 IEEE High Perform. Extrem. Comput. Conf. HPEC 2023*, 2023, doi: 10.1109/HPEC58863.2023.10363447.

[9] D. Huang and Z. Wang, "LLMs at the Edge: Performance and Efficiency Evaluation with Ollama on Diverse Hardware," *Proc. Int. Jt. Conf. Neural Networks*, 2025, doi: 10.1109/IJCNN64981.2025.11228317.

[10] E. Chung, Y. Jia, A. Jezghani, and H. Kim, "Characterizing CPU-Induced Slowdowns in Multi-GPU LLM Inference," 2026, [Online]. Available: http://arxiv.org/abs/2603.22774

[11] Q. Song *et al.*, *A Systematic Evaluation of On-Device LLMs: Quantization, Performance, and Resources*, vol. 1, no. 1. arXiv, 2026. [Online]. Available: http://arxiv.org/abs/2505.15030

[12] E. Dinçer and Z. H. Kilimci, "Real-Time and Offline Large Language Models on Edge Devices: A Systematic Review," pp. 0–17, 2025, doi: 10.20944/preprints202512.2383.v1.

[13] D. Park and B. Egger, "Improving Throughput-Oriented LLM Inference with CPU Computations," *Parallel Archit. Compil. Tech. - Conf. Proceedings, PACT*, no. March, pp. 233–245, 2024, doi: 10.1145/3656019.3676949.

[14] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile Edge Intelligence for Large Language Models: A Contemporary Survey," *IEEE Commun. Surv. Tutorials*, vol. 27, no. 6, pp. 3820–3860, 2025, doi: 10.1109/COMST.2025.3527641.

[15] T. Nguyen and T. Nguyen, "An Evaluation of LLMs Inference on Popular Single-board Computers," *arXiv Prepr.*, 2025, [Online]. Available: http://arxiv.org/abs/2511.07425

[16] S. Na *et al.*, "FlexInfer: Flexible LLM Inference with CPU Computations," *MLSys*, 2025, [Online]. Available: https://mlsys.org/virtual/2025/poster/3234%0Ahttps://openreview.net/forum?id=sFNRNTduKO

[17] F. Liu, Z. Kang, and X. Han, "Optimizing RAG Techniques for Automotive Industry PDF Chatbots: A Case Study with Locally Deployed Ollama Models," *Proc. 2024 3rd Int. Conf. Artif. Intell. Intell. Inf. Process. AIIIP 2024*, no. March, pp. 152–159, 2025, doi: 10.1145/3707292.3707358.

[18] M. Lazuka, A. Anghel, and T. Parnell, "LLM-Pilot: Characterize and Optimize Performance of your LLM Inference Services," *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC*, 2024, doi: 10.1109/SC41406.2024.00022.

[19] H. Chen, C. Tian, Z. He, B. Yu, Y. Liu, and J. Cao, "Inference performance evaluation for LLMs on edge devices with a novel benchmarking framework and metric," no. 2, 2025, [Online]. Available:

http://arxiv.org/abs/2508.11269
[20] L. Stuhlmann, M. F. Argerich, and J. Fürst, "Bench360: Benchmarking Local LLM Inference from 360 Degrees," 2026, [Online]. Available: http://arxiv.org/abs/2511.16682
[21] Z. Yuan *et al.*, "LLM Inference Unveiled: Survey and Roofline Model Insights," 2024, [Online]. Available: http://arxiv.org/abs/2402.16363
[22] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," *SOSP 2023 - Proc. 29th ACM Symp. Oper. Syst. Princ.*, pp. 611–626, 2023, doi: 10.1145/3600006.3613165.

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri