ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

EVALUATING SMALL QUANTIZED LANGUAGE MODELS ON APPLE SILICON

Oleh Chaplia, Halyna Klym

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
Authors' e-mails: *oleh.y.chaplia@lpnu.ua, halyna.i.klym@lpnu.ua*

https://doi.org/10.23939/acps2025.01.034

Submitted on 15.03.2025

***Abstract*: This study examines the capabilities and limitations of small, 4-bit quantized language models that run locally on Apple Silicon. Four models have been benchmarked on a dataset of natural language prompts, with runtime metrics including inference time, memory usage, and token throughput, as well as output behavior. The study provides an empirical assessment of the feasibility of deploying language models on resource-constrained devices. The results highlight trade-offs of small language models and underscore the importance of model size, quantization, and prompt tuning in balancing performance, efficiency, and usability. Building on these insights, future work will extend evaluations to multi-turn agentic dialogues, analyze the semantic quality of output, and pursue further optimizations to enhance local inference performance.**

***Index terms*: cloud computing, microservices, software architecture, small language models, Apple Silicon, MLX, MLX-LM.**

I. INTRODUCTION

Large Language Models (LLMs) have rapidly transformed modern AI systems and tools. They can automate complex knowledge tasks, provide expert-level insights, assist decision-making processes, and enhance human productivity at scale. Their ability to process and generate highly accurate natural language content makes them essential for next-generation applications, from intelligent virtual assistants to automated document analysis and domain-specific systems [1].

Their adaptability across domains has made them indispensable tools in the healthcare, finance, law, software development, and other industries [2]. However, the growing reliance on LLMs also introduces significant risks, particularly around data privacy, confidentiality, and security.

Most LLMs are traditionally deployed through cloud services, where sensitive user data, prompts, and model outputs can be exposed to third parties. This external dependency creates potential vulnerabilities: unauthorized access, data breaches, compliance violations, and intellectual property leakage are real concerns when handling proprietary or regulated information [3]. Even minor cloud security or model inference handling lapses could have serious financial and reputational consequences.

Therefore, balancing LLMs' transformative potential with the need for privacy is critical. Moving toward local execution and privacy-preserving architectures ensures organizations can fully harness LLMs' power while safeguarding sensitive data.

Running large language models (LLMs) on consumer devices is appealing for privacy and security [4]. However, the user's device limits model size and performance. Recent open models and quantization techniques have made it feasible to deploy LLMs with hundreds of millions of parameters on devices like Apple's M1 chip (16GB RAM) [5]. Smaller LLMs often struggle with complex tasks that larger models easily handle. Enabling a model to use tools (such as calling external APIs) is challenging for small LLMs [6]. These models may lack the capacity for robust reasoning and often hallucinate facts instead of invoking a tool for up-to-date information. Prior work has shown that allowing LLMs to call tools (e. g., calculators, search engines, or weather services) can significantly expand their capabilities. On the other hand, smaller LLMs tend to be weak tool learners, exhibiting performance limitations unless specialized approaches are used.

Model quantization, such as 4-bit weight compression, offers a promising strategy for efficiently deploying smaller models on constrained hardware [7]. However, it may introduce additional challenges by slightly degrading precision and tool-calling reliability. Therefore, careful calibration and instruction tuning are essential to preserve functional accuracy after quantization.

Specialized frameworks like Apple MLX, optimized for Apple Silicon hardware, provide a solution by enabling efficient, low-latency LLM execution directly on local devices [8]. MLX leverages native neural processing units, reducing dependency on external servers while delivering high inference performance and energy efficiency [9].

This work presents a detailed empirical evaluation of small, 4-bit, quantized instruction-tuned language models running natively on Apple M1 hardware using the MLX-LM framework. Our study benchmarks four representative models (*Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Mistral-7B-Instruct-v0.3,* and *Qwen2.5-1.5B-Instruct*) across the dataset with prompts. Key runtime metrics are measured, such as inference latency, peak memory usage, token throughput, and execution stability under real-world conditions. Furthermore, we investigate how model size,

instruction tuning, and prompt complexity impact local resource demands and response generation behavior. Visualized in plots, results highlight how model scale, instruction tuning, and quantization impact output behavior in constrained, on-device environments. This work offers a broader perspective on LLM deployment for local and secure systems.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

While LLMs offer advanced capabilities across many fields, their deployment in privacy- and security-critical environments presents substantial challenges that must be addressed with technical rigor.

LLMs are inherently prone to data leakage. Models trained on large corpora may inadvertently memorize rare or sensitive information from their training data [2]. Carefully crafted prompts can trigger training data extraction attacks at inference time, where adversaries recover private information (e. g., user credentials, confidential documents) from the model outputs. Even without direct leakage, model inversion attacks can reconstruct sensitive input queries based on output patterns. Prompt injection attacks can also manipulate LLM behavior, causing models to generate unintended responses that may reveal system internals or breach access control policies.

Cloud-based LLM deployments exacerbate these vulnerabilities. When using external services, sensitive input data must be transmitted over public networks, where it becomes susceptible to interception, man-in-the-middle attacks, or side-channel analysis unless end-to-end encryption is flawlessly implemented [10]. Even assuming secure transport, once the data reaches the cloud provider, it may be logged, cached, or subjected to operational data monitoring practices beyond the organization's control. Attack surfaces increase due to multi-tenancy risks (i. e., different users sharing the same hardware), hypervisor vulnerabilities, and possible insider threats at the provider level. In jurisdictions with aggressive surveillance laws, cloud-hosted data may be subject to access requests without user notification. Additionally, cloud services often rely on complex orchestration layers (e. g., Kubernetes, serverless runtimes) that introduce further complexity and potential misconfiguration risks.

Adopting a microservice architecture can help isolate critical components and minimize the impact of potential breaches. Achieving meaningful security benefits requires careful architectural design and strict boundary enforcement [11]. Without a thoughtfully structured architecture, even local or containerized deployments may remain vulnerable to lateral movement attacks and data leakage within the system.

Governance and compliance become difficult in external environments [12]. Regulatory frameworks such as GDPR, HIPAA, or PCI-DSS require strict guarantees about data locality, access auditing, data deletion, and consent management – guarantees that cloud providers may not fully support without extensive, costly contractual customizations. Maintaining audit trails, ensuring provable deletion of data after processing, and meeting "right-to-be-forgotten" obligations are significantly more onerous when data is processed remotely.

Local and on-premises LLM deployment is a technically superior solution for addressing these challenges [13]. Running inference pipelines within trusted hardware boundaries ensures user data never leaves the controlled environment. In local deployments, embeddings, retrieval contexts (for RAG pipelines), and inference results can be stored entirely on encrypted local disks and processed using trusted runtime environments without external exposure. Further, running quantized models (e. g., 4-bit precision) with frameworks such as MLX on Apple Silicon hardware leverages hardware-accelerated encryption, trusted boot environments, and memory protection features such as pointer authentication (PAC) and data-independent timing, mitigating side-channel leakage risks.

Moreover, local execution eliminates the need for data egress monitoring, inter-region compliance validation, or cloud-specific key management. Developers can implement fine-grained access control at the file system, API layer, or memory segmentation level, significantly reducing attack surfaces compared to internet-exposed endpoints. Updates to models and retrieval databases can be tightly audited and version-controlled, ensuring complete transparency and rollback capability without external dependencies.

Local, privacy-preserving LLM microservices offer the strongest foundation for secure, compliant, and reliable AI, reducing technical vulnerabilities and governance overhead compared to traditional cloud-based approaches.

III. SCOPE OF WORK AND OBJECTIVES

The primary goal of this study is to investigate the capabilities and limitations of small, quantized language models deployed on resource-constrained consumer hardware. Specifically, we aim to evaluate these models' behavior, performance, and robustness when executed locally on an Apple M1 device using the MLX-LM framework. As the demand for on-device AI assistants grows, assessing whether small models, typically under 2–5 GB in size, can deliver both responsiveness and accuracy under real-world usage conditions becomes critical.

A central objective of this work is to benchmark and compare the performance of several representative 4-bit quantized models: *Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Mistral-7B-Instruct-v0.3, and Qwen2.5-1.5B-Instruct*. All models are deployed within the same local environment and evaluated extensively across synthetic task templates and natural prompt datasets. Key metrics such as input / output token distribution, inference latency, and execution reliability are recorded. Visualizations generated from the combined results highlight how mode scale, instruction tuning, and quantization affect system behavior in constrained on-device settings, providing a broader perspective on privacy-preserving LLM deployment for secure, local microservices.

Beyond measuring performance, an important goal is to explore the effects of linguistic distribution on model behavior. To this end, models are evaluated on two input types: narrowly structured, synthetically generated queries and naturally occurring prompts from real-world datasets. This comparison enables a deeper understanding of each model's robustness to input variation and reveals how training data distribution impacts tool-use behavior and generalization.

Another critical objective is to assess the practicality of 4-bit quantization as an enabler for efficient, real-time LLM inference on consumer-grade hardware. We aim to determine whether small, quantized models can operate within acceptable latency thresholds while maintaining tool-use accuracy and contextual fidelity – requirements for deploying responsive, reliable AI assistants.

Ultimately, this work seeks to provide actionable insights for engineers and researchers focused on compact LLM deployment. By identifying the strengths and limitations of current small-model approaches and shedding light on key trade-offs in architecture design, quantization strategies, and instruction tuning, we aim to inform future efforts toward more efficient, privacy-centric, and robust tool-using AI systems.

IV. MLX-LM COMPATIBILITY AND MODEL SELECTION

Apple Silicon processors, starting with the M1 generation, have redefined the architecture of consumer hardware by tightly integrating CPU, GPU, and Neural Engine components into a unified system-on-chip (SoC) design. Built on an energy-efficient instruction set, Apple Silicon chips offer high memory bandwidth, low latency interconnects, and powerful matrix multiplication capabilities essential for modern AI workloads. The dedicated Neural Engine, optimized for machine learning tasks, provides substantial acceleration for operations such as matrix multiplications and tensor processing. At the same time, the unified memory architecture allows rapid data sharing between processing units. This architecture makes Apple devices well-suited for on-device AI inference, supporting high-throughput and low-latency computations without relying on external servers.

To exploit these hardware capabilities, Apple introduced the MLX framework – a lightweight machine learning library designed specifically for Apple Silicon. MLX offers a NumPy-like API that seamlessly integrates tensor operations with hardware-accelerated execution paths, abstracting away the complexity of directly managing GPU or Neural Engine resources. On top of MLX, the MLX-LM package provides specialized tools for loading, quantizing, and running Large Language Models locally. MLX-LM supports efficient quantization, allowing LLMs to fit within the memory constraints of consumer devices while maintaining high inference speed. It also facilitates rapid model loading, dynamic memory management, and optimized matrix operations tailored for Apple hardware. By operating entirely within the local system, MLX-LM ensures that all user queries, model states, and intermediate computations remain private and confined to the device, eliminating data exposure risks associated with cloud-based inference services. These features make MLX and MLX-LM an ideal foundation for building private, responsive AI assistants on modern macOS and iOS platforms.

In selecting models for this study, several key criteria were considered: model size, instruction-tuning capabilities, compatibility with 4-bit quantization, and general availability for local deployment. *Llama-3.2-1B-Instruct* and *Llama-3.2-3B-Instruct* were selected to balance small parameter counts and strong instruction-following behavior, which is critical for responsive AI applications on resource-constrained hardware. *Mistral-7B-Instruct-v0.3* represents a slightly larger model focusing on high-quality instruction tuning and multilingual capabilities, offering a comparison point for assessing how scaling model size impacts latency and memory use in local inference scenarios. *Qwen2.5-1.5B-Instruct* was selected as a mid-range model optimized for efficient reasoning and conversation tasks, providing insights into trade-offs between output quality and hardware constraints. All models were tested in their 4-bit quantized variants, ensuring the memory footprint remained compatible with Apple Silicon devices equipped with 16 GB of unified memory.

Llama-3.2-1B-Instruct-4bit uses about 1.42 GB on disk, while its 3B-parameter sibling takes 1.82 GB. *Mistral-7B-Instruct-v0.3-4bit* has the most significant footprint at 4.08 GB, and *Qwen2.5-1.5B-Instruct-4bit* is the smallest on disk at 880 MB. Note that, during testing, each model stays resident in memory between prompts, so load times are only incurred once at startup. In practice, if one model isn't fully unloaded before loading the next, you may see overlapping memory usage from the previous model while the new one executes.

Together, these models offer a diverse and representative sample for evaluating the feasibility and performance of local LLM deployment under realistic conditions.

V. DATASET

For this study, a dataset was prepared to evaluate the behavior and performance of the selected quantized models. Each record in the dataset captures a whole model interaction, including the input prompt, the generated output, and detailed inference statistics. Specifically, the dataset includes the following fields: the input prompt provided to the model, the output generated by the model, the name of the model variant used, the number of tokens in the input, the input token processing speed (TPS), the number of tokens in the output, the output TPS, the peak memory usage during inference measured in gigabytes, the total execution time in seconds, and the final execution status (success or failure).

A synthetic dataset was carefully constructed to evaluate the performance of small language models. The prompts were synthetically generated to follow an approximated near-normal distribution based on word

count, ensuring that the dataset would emulate the typical structure and length of real-world queries. The corpus comprises 200 unique prompts (2262 words) that cover a diverse range of computer science and artificial intelligence topics.

Examples of the generated prompts include concise queries around computer science topics, such as "Describe linear regression," "Why is cloud data mining important?" and "When to use a neural network?" More detailed prompts were also included to simulate more complex information requests, such as "Can multithreading improve the performance of reinforcement learning?" "Why is garbage collection critical for runtime memory management?" and "What challenges arise when implementing real-time embedded systems security?" This variety ensured that models were tested on succinct, factual queries and more open-ended, context-driven prompts.

The most frequently used words in the dataset include "How" (79 occurrences), "in" (69), "What" (65), and "for" (40), reflecting the question-driven nature of the prompts.

The question generation process is aimed at centering the word count distribution around 11 words (89 characters) per prompt, reflecting the natural variation observed in human-written sentences. The frequency gradually increases from 1 sentence with 1 word to a peak of 21 sentences, each with 11 words. Counts remain high near the peak – 20 at 12 words, 19 at 10, and 18 at 13 – then steadily decline to just 1 sentence at 21 words.

This design intentionally mirrors real world input length variation, providing a balanced challenge for the evaluated language models. Very short prompts (fewer than eight words) and long (more than 19 words) occur infrequently, ensuring that most evaluations reflect typical user query complexity.

VI. EXPERIMENTAL SETUP

The local environment was created to enable benchmarking of the selected quantized models on Apple Silicon hardware. The first step involved creating a software project with the UV package manager. The next step involved installing the MLX-LM framework and project dependencies. Then, a Python script was developed to automate the download and initialization of the target models. Following the model setup, additional Python scripts were implemented to manage the benchmarking workflow. A test runner script was developed to automate inference calls on the models, submitting synthetic prompts while recording detailed runtime statistics. Data collection scripts were built to capture and store key metrics such as input / output token counts, token processing speeds (TPS), peak memory usage, execution time, and inference status into structured datasets. Datasets with prompts and results are stored in CSV format. Visualization scripts were created to generate plots, including word count histograms and scatter diagrams.

These modular scripts formed a comprehensive testing framework that enabled repeatable, transparent experimentation, significantly streamlining the end-to-end benchmarking process. All evaluations were performed locally, leveraging the MLX-LM backend to ensure data confidentiality was strictly preserved during inference and analysis.

VII. RESULTS

The evaluation of the selected models revealed several important trends in local inference behavior, including differences in execution time, memory consumption, and token generation efficiency. Evaluated models were *Qwen2.5-1.5B-Instruct (Qwen2.5-1.5B), Llama-3.2-1B-Instruct (Llama-3.2-1B), Llama-3.2-3B-Instruct (Llama-3.2-3B), Mistral-7B-Instruct-v0.3 (Mistral-7B).*

The models were evaluated across twelve performance metrics: minimum, maximum, and average values for input tokens, output tokens, peak memory usage, and execution time. Across all metrics, *Llama-3.2-1B-Instruct-4bit* demonstrates the most efficient profile overall, with consistently low memory consumption (average: 0.99 GB) and fast execution (average: 10.33 s) while maintaining competitive token throughput. *Qwen2.5-1.5B-Instruct-4bit* also performs well, particularly in low memory usage (average: 1.07 GB) and faster execution (average: 13.46 s), though with lower output token counts compared to Llama models.

Llama-3.2-3B-Instruct-4bit shows moderately increased memory use (average: 2.05 GB) and execution time (average: 20.74 s), indicating a trade-off for its higher model capacity. In contrast, *Mistral-7B-Instruct-v0.3-4bit*, while offering strong generation capabilities (average output: 444 tokens), requires significantly more memory (average: 4.32 GB) and slower response times (average: 30.55 s), confirming it is best suited for higher-end devices with larger memory capacity.

The peak memory and execution time values indicate that the 7B model (Mistral) exceeds 4 GB of memory and takes over 90 seconds in worst-case scenarios. In contrast, the 1B and 1.5B models remain under 1.5 GB and complete within 15 seconds under average conditions. This highlights the practical feasibility of running 1–3B models even on MacBooks with 16GB of RAM. In comparison, larger 7B models require more capable Apple Silicon hardware.

Fig. 1 shows a scatter plot of input and output tokens for LLMs. Each point represents a single inference run, with the number of input tokens on the x-axis and output tokens on the y-axis.

The distribution reveals several key patterns. Firstly, output length is generally positively correlated with input length across all models, though the slope and variance differ significantly. The *Mistral-7B* model (triangles) maintains a relatively narrow range of output sizes (~400–800 tokens) despite varying input sizes. In contrast, *Qwen2.5-1.5B* (diamonds) and *Llama-3.2* models (circle and crosses) show more variance in output length, with some examples producing up to 4000 tokens, hitting the configured generation limit (4096 tokens). Notably,

Qwen2.5-1.5B exhibits several high-output cases with relatively modest input sizes, potentially indicating aggressive sampling or differences in prompt structure. Meanwhile, *Llama-3.2-3B* and *1B* models exhibit more proportional growth between input and output.

This visualization illustrates how different models respond to varying prompt lengths under identical conditions, reinforcing the importance of tuning generation parameters (e. g., max tokens, temperature, top_p) per model to achieve consistent output behavior in real-world applications.

Fig. 2 presents a set of scatter plots comparing key performance metrics. The four subplots analyze the relationships between input/output token counts, peak memory usage, and execution time.

Fig. 2, *a* shows input tokens vs. peak memory (GB). Here, models cluster along distinct memory bands corresponding to their size. *Mistral-7B* consistently consumes over 4 GB, while *Qwen-1.5B* and *Llama-3.2-1B* models remain under 1 GB. *Llama-3.2-3B* ranges between 1.8 and 2.1 GB. This clear stratification demonstrates the effectiveness of 4-bit quantization while reflecting the inherent memory demands of each model's parameter count.

Fig. 2, *b* plots input tokens vs. execution time. Execution time increases with input size, but larger models such as *Mistral-7B* exhibit notably higher latencies even for short prompts. In contrast, *Qwen-1.5B* and *Llama-3.2-1B* remain under 20 seconds on average, suggesting their suitability for responsive local inference, especially in resource-constrained or interactive applications.

Fig. 2, *c* examines output tokens vs. peak memory usage. *Mistral-7B* again shows the highest memory footprint regardless of output size, indicating high baseline memory consumption independent of sequence length. Meanwhile, the memory curves for smaller models (*Qwen-1.5B, Llama-3.2-1B*) remain flat across a wide range of output tokens, which is beneficial for predictability in low-memory environments.

Fig. 2, *d* shows output tokens vs. execution time. All models show a roughly linear relationship, though the slope varies with model size. Mistral-7B and Llama-3.2-3B require significantly more time per output token than their smaller counterparts. Notably, Qwen-1.5B and Llama-3.2-1B exhibit tight, linear trends with low variance, further supporting their efficiency for low-latency inference tasks.

These plots highlight the memory and latency trade-offs inherent in model selection for local inference. While larger models like *Mistral-7B* offer richer outputs, they come with high computational costs, making them more suitable for high-performance machines. In contrast, 1B–1.5B models deliver a strong balance of performance and efficiency, confirming their viability for on-device applications where responsiveness and memory constraints are key considerations.

VIII. LIMITATIONS AND FUTURE WORK

While this study provides detailed insights into the performance of small, quantized language models on consumer-grade Apple Silicon hardware, several significant limitations must be acknowledged.

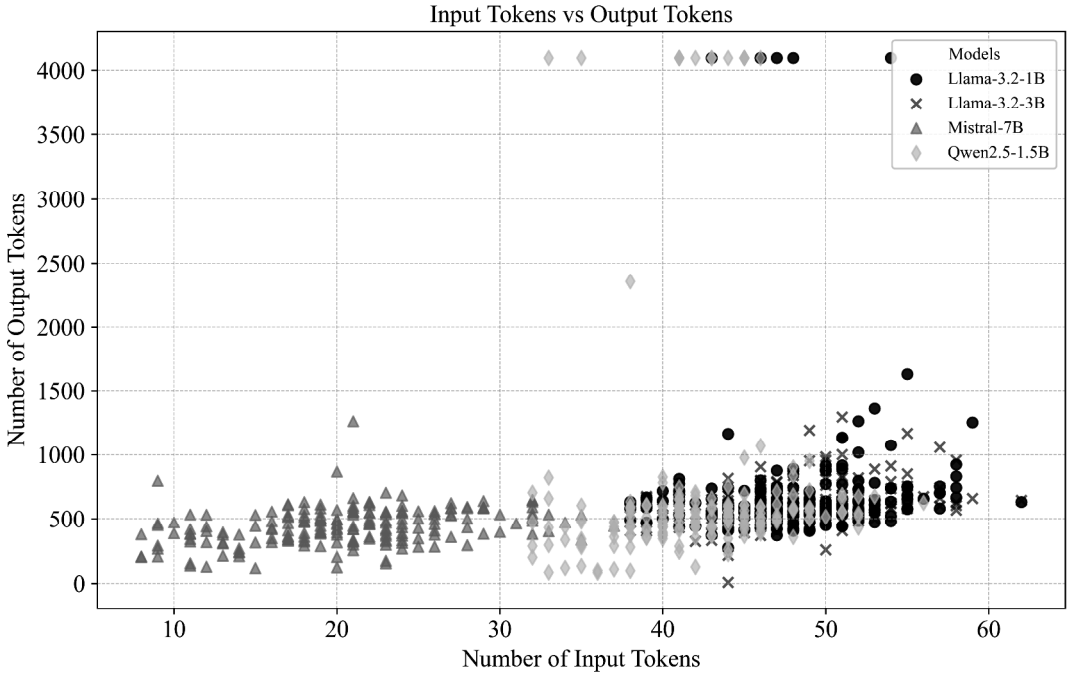

Fig. 1. Input vs output tokens

Firstly, the evaluation focused primarily on runtime performance metrics such as memory consumption, execution time, and token throughput. The semantic quality of the generated outputs, including accuracy, factual correctness, coherence, and relevance, was not systematically assessed or benchmarked against human-labeled ground truth. As such, conclusions about the models' practical effectiveness for real-world tasks like summarization, question answering, or tool use remain open. Additionally, the evaluation was limited to single-turn, stateless, prompt-response interactions. Multi-turn conversational capabilities, context retention, and model behavior over more extended dialogues were not investigated. Only 4-bit quantized models were tested without direct comparison to higher-precision (e. g., 8-bit or 16-bit) variants that might offer different trade-offs between latency and quality. Finally, all experiments were conducted on a single type of Apple Silicon device, and broader hardware diversity was not explored.

Building on these findings, several directions for future work are evident. A critical next step is to evaluate output quality involving automated metrics and human expert annotation. This would allow a more complete assessment of the models' practical utility in knowledge-intensive applications. Expanding the evaluation to multi-turn dialogue settings and tool-augmented reasoning tasks would provide further insight into real-world assistant capabilities. Investigating mixed-precision quantization schemes, adaptive loading strategies, and fine-tuning techniques on-device could improve the balance between model size and output quality. Furthermore, testing on a wider range of Apple Silicon generations and configurations would help generalize the results to broader edge deployment scenarios. Lastly, integrating privacy-preserving features such as secure enclaves, encrypted embedding stores, or trusted execution environments could enhance the security profile of local LLM deployments, addressing even stricter regulatory requirements for sensitive domains.

IX. CONCLUSION

This study evaluated small, quantized language models deployed locally on consumer-grade Apple Silicon hardware using the MLX-LM framework. By benchmarking *Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Mistral-7B-Instruct-v0.3, and Qwen2.5-1.5B-Instruct* models, we assessed key performance metrics, including execution time, peak memory usage, token throughput, and output behavior under real-world prompt distributions. The results demonstrated that smaller quantized models, particularly *Llama-3.2-1B-Instruct* and

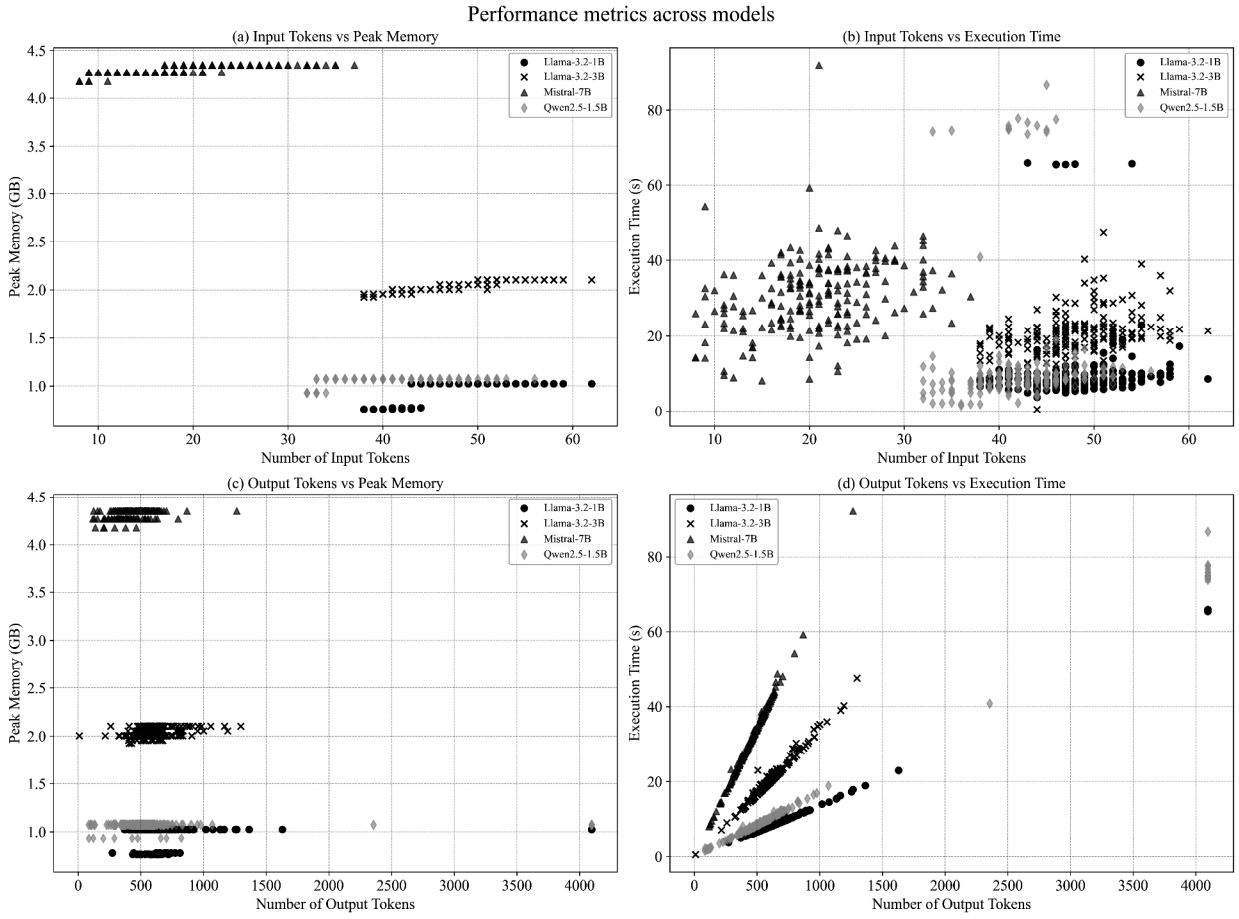

Fig. 2. Performance metrics across four quantized models

Qwen2.5-1.5B-Instruct, strike a favorable balance between responsiveness, memory efficiency, and deployment feasibility, making them highly suitable for privacy-preserving, real-time AI assistants operating directly on local consumer devices.

While larger models such as *Mistral-7B-Instruct* exhibited the ability to generate richer and more verbose outputs, they incurred significantly higher computational costs and memory demands, limiting their practicality for lightweight on-device applications without further optimizations. Our findings emphasized the importance of model size, instruction tuning quality, and quantization strategies in enabling efficient and secure edge AI deployments.

However, several limitations remain. Semantic output quality was not formally assessed, and evaluations were limited to single-turn interactions without a broader conversational context. Future work will evaluate output accuracy, expand to multi-turn dialogue benchmarks, and explore optimizations such as adaptive quantization and model fine-tuning for edge environments.

Overall, this work contributes a foundational analysis toward enabling private, efficient, and scalable deployment of language models on local Apple Silicon hardware. This supports the growing demand for AI systems prioritizing user data sovereignty, low-latency responsiveness, and resource-conscious operation. The insights gained from this study provide a practical reference for engineers, researchers, and organizations building next-generation on-device AI solutions.

References

[1] Patil, R. & V. Gudivada (2024). A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). *Applied Sciences*, 14(5), 2074. DOI: 10.3390/app14052074.

[2] Yao, Y., J. Duan, K. Xu, Y. Cai, Z. Sun, & Y. Zhang (2024). A survey on large language model (LLM) security and privacy: *The Good, The Bad, and The Ugly. High-Confidence Computing*, 4(2), 100211. DOI: 10.1016/j.hcc.2024.100211.

[3] Gallagher, S., B. Gelman, S. Taoufiq, T. Vörös, Y. Lee, A. Kyadige, & S. Bergeron (2024). Phishing and Social Engineering in the Age of LLMs. Large Language Models in Cybersecurity: Threats, *Exposure and Mitigation*, 81–86. DOI: 10.1007/978-3-031-54827-7_8.

[4] Piccialli, F., D. Chiaro, P. Qi, V. Bellandi, & E. Damiani (2025). Federated and edge learning for large language models. *Information Fusion,* 117, 102840. DOI: 10.1016/j.inffus.2024.102840.

[5] Lamaakal, I., Y. Maleh, K. El Makkaoui, I. Ouahbi, P. Pławiak, O. Alfarraj, M. Almousa, & A. A. Abd El-Latif (2025). Tiny Language Models for Automation and Control: Overview, Potential Applications, and Future Research Directions. *Sensors,* 25(5), 1318. DOI: 10.3390/s25051318.

[6] Jay, R. (2024). Building Different Types of Agents. Generative AI Apps with LangChain and Python: A Project-Based Approach to Building Real-World LLM Apps, 345–414. DOI: 10.1007/979-8-8688-0882-1_9.

[7] Chitty-Venkata, K. T., S. Mittal, M. Emani, V. Vishwanath, & A. K. Somani (2023). A survey of techniques for optimizing transformer inference. Journal of Systems Architecture, 144, 102990. DOI: 10.1016/j.sysarc.2023.102990.

[8] Kassinos, S. & A. Alexiadis (2025). Beyond language: Applying MLX transformers to engineering physics. *Results in Engineering*, 26, 104871. DOI: 10.1016/j.rineng.2025.104871.

[9] Kasperek, D., P. Antonowicz, M. Baranowski, M. Sokolowska, & M. Podpora. (2023). Comparison of the Usability of Apple M2 and M1 Processors for Various Machine Learning Tasks. *Sensors,* 23(12), 5424. DOI: 10.3390/s23125424.

[10] Berardi, D., S. Giallorenzo, A. Melis, M. Prandini, J. Mauro, & F. Montesi (2022). Microservice security: a systematic literature review. *PeerJ Computer Science,* 7. DOI: 10.7717/PEERJ-CS.779.

[11] Chaplia, O. & H. Klym (2023). Node.js project architecture with shared dependencies for microservices. *Measuring Equipment and Metrology,* 84(3), 53–58. DOI: 10.23939/istcmtm2023.03.053.

[12] Bližnák, K., M. Munk, & A. Pilková (2024). A Systematic Review of Recent Literature on Data Governance (2017–2023). *IEEE Access,* 12, 149875–149888. DOI: 10.1109/ACCESS.2024.3476373.

[13] Josa, A. D. & M. Bleda-Bejar (2024). Local LLMs: Safeguarding Data Privacy in the Age of Generative AI. A Case Study at the University of Andorra. I*CERI2024 Proceedings,* 7879–7888. DOI: 10.21125/iceri.2024.1931.

Oleh Chaplia was born in Lviv, Ukraine. He is a PhD student in the Specialized Computer Systems Department at Lviv Polytechnic National University. Since earning his master's degree in 2015, he has been working in the software engineering field. He has extensive commercial experience in technical leadership, cloud architecture, software design, research, and development of modern software and cloud solutions, including those with artificial intelligence. His academic research focuses on emerging technologies, cloud computing, and artificial intelligence. Oleh's experience also includes authoring articles participating in international conferences, working as a lecturer, mentoring, and delivering guest lectures at universities.

Halyna Klym – doctor of technical sciences, professor, professor of the department of Specialized Computer Systems at Lviv Polytechnic National University. In 2008, she received the degree of Doctor of Philosophy in the specialty: physical and mathematical sciences at Ivan Franko Lviv National University. In 2016, she received a Doctor of Science degree in Physical and Mathematical Sciences at Lviv Polytechnic National University. She conducts lecture courses on the design of ultra-large integrated circuits and methods and means of automated design of computer systems. She is an author of more than 170 scientific articles in international publications.