Characterizing and Understanding the Performance of Small Language Models on Edge Devices

Md Romyull Islam, Nobel Dhar, Bobin Deng, Tu N. Nguyen, Selena He, Kun Suo
Department of Computer Science, Kennesaw State University, GA, USA
Email: {mislam22, ndhar}@students.kennesaw.edu, {bdeng2, tu.nguyen, she4, ksuo}@kennesaw.edu

***Abstract*—In recent years, significant advancements in computing power, data richness, algorithmic development, and the growing demand for applications have catalyzed the rapid emergence and proliferation of large language models (LLMs) across various scenarios. Concurrently, factors such as computing resource limitations, cost considerations, real-time application requirements, task-specific customization, and privacy concerns have also driven the development and deployment of small language models (SLMs). Unlike extensively researched and widely deployed LLMs in the cloud, the performance of SLM workloads and their resource impact on edge environments remain poorly understood. More detailed studies will have to be carried out to understand the advantages, constraints, performances, and resource consumption in different settings of the edge.**

This paper addresses this gap by comprehensively analyzing representative SLMs on edge platforms. Initially, we provide a summary of contemporary edge hardware and popular SLMs. Subsequently, we quantitatively evaluate several widely used SLMs, including TinyLlama, Phi-3, Llama-3, etc., on popular edge platforms such as Raspberry Pi, Nvidia Jetson Orin, and Mac mini. Our findings reveal that the interaction between different hardware and SLMs can significantly impact edge AI workloads while introducing non-negligible overhead. Our experiments demonstrate that variations in performance and resource usage might constrain the workload capabilities of specific models and their feasibility on edge platforms. Therefore, users must judiciously match appropriate hardware and models based on the requirements and characteristics of the edge environment to avoid performance bottlenecks and optimize the utility of edge computing capabilities.

***Index Terms*—Small Language Models (SLMs), Edge computing, Performance, Resource footprint**

I. Introduction

Nowadays, the exponential growth of artificial intelligence (AI) across various fields has stimulated the demand for advanced models, particularly large language models (LLMs). These models possess robust language understanding and generation capabilities. However, LLMs have notable shortcomings. First, they cannot guarantee the accuracy and quality of their output, often producing biased or misleading information. Second, LLMs can be overly general, leading to inefficiencies in understanding and responding to the nuances of industry-specific terminology, processes, and data. Again, since LLMs operate in the cloud, they present security and privacy concerns. Furthermore, due to their extensive computational and power requirements, deploying large language models such as GPT-4 or Llama 3 on edge devices such as mobile phones or PCs is impractical. Small Language Models (SLMs) offer a compelling solution to this problem [1]. As it is designed to run directly on edge devices, SLMs have recently gained increasing attention from academia and industry.

Unlike traditional LLMs, small language models are much smaller in size and optimized for performance, enabling them to run efficiently within limited computing, memory, and energy-constrained environments. For example, Apple recently launched OpenELM, which has a minimum parameter count of only 270 million and is only 3% of Llama 3. Additionally, using SLMs to process data locally on edge devices can significantly reduce latency and address privacy issues compared to sending data to the cloud for processing. Today, more companies are considering deploying their customized SLMs on edge devices across various sectors, such as healthcare, automotive, industrial IoT, etc. For example, companies like Microsoft, Intel, and AMD have recently proposed enhanced AI personal computing (AI PC) to provide a more personalized and interactive computing experience by integrating smarter and more responsive SLM AI functions into everyday devices. As language models transition from expensive and resource-intensive LLMs to leaner and more efficient SLMs, it becomes exciting and crucial to study different SLM model performances at the edge and their effect on the system, especially from a resource perspective. However, we found a gap in recent research in the edge AI field on resource footprint and performance analysis of various SLMs or applications on different edge computing platforms, which is essential for improving edge platforms and deploying SLMs or related services.

In this study, we performed a detailed analysis of representative small language models on edge, such as TinyLlama, Phi-3, OpenELM, etc., and conducted an empirical analysis of their performance and resource usage. As far as we know, this paper is the first to investigate the efficiency and performance aspects of SLMs on edges. We collected and analyzed a series of indicators through different systems, including inference performance, memory usage, CPU load, swap partition usage, disk activity rate, etc. Through the above assessment, this study aims to identify and understand the bottlenecks that currently hinder the effective deployment of SLM in various edge platforms. We believe our findings can help users and developers choose the right edge platform for their workloads and navigate the optimization of current and future SLMs. The remainder of this article is organized as follows. §-II introduces edge hardware and the most widely used AI edge workloads. §-III proposes our methodology including

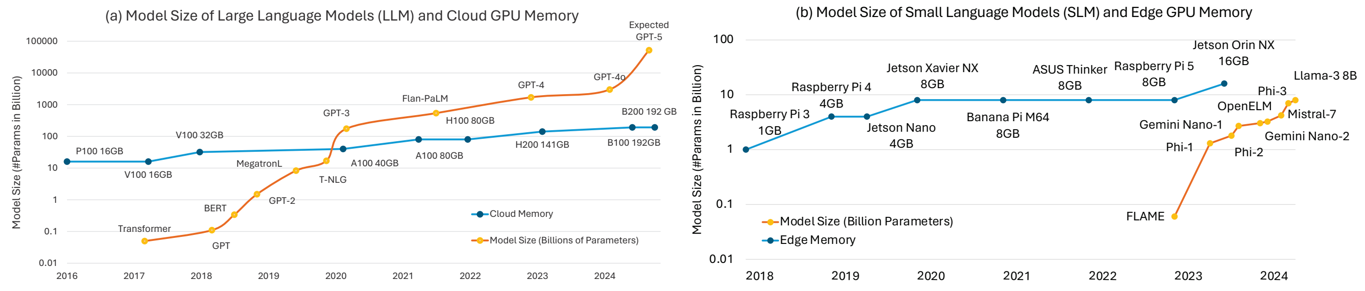

Fig. 1. Growth of Large Language Models/Small Language Models Compared to Cloud/Edge GPU Memory Capacity over the Years.

hardware, software, measurement, and data collection. §-IV presents experimental results and analysis. §-V reviews the related work and §-VI concludes this paper.

II. Background & Motivation

The costs associated with training and maintaining LLMs are substantial. For instance, training a large language model such as GPT-3 can exceed \$4 million, whereas retraining the 2^{nd} version of the BigScience Large Open-science Open-access Multilingual Language Model (BLOOM), which comprises over 176 billion parameters, may incur costs around \$10 million [2]. Figure 1(a) illustrates the growth trajectory of representative LLM sizes in relation to cloud GPU memory capacity. While GPU memory has increased from 16GB (P100) in 2016 to 192GB (B200), which is projected for 2025, LLM sizes have escalated dramatically. Initial models like ”Transformer” and ”GPT” in 2017 featured 500 million and 110 million parameters, respectively, evolving to ”GPT-3” with 175 billion parameters in 2020 and anticipated to reach trillion parameters with ”GPT-4o” by 2024. This expanding disparity between LLM sizes and cloud GPU memory capacity underscores the significant challenges and elevated costs associated with deploying these large-scale models on existing hardware.

In contrast to LLMs, SLMs are engineered to operate with lower computing resources, making them well-suited for low-cost edge devices with limited processing power and memory. Currently, edge devices and SLMs are becoming increasingly integral to the AI era. As our world becomes more interconnected, the number of smartphones, IoT devices, and edge devices is rapidly growing. For example, the number of consumer devices and enterprise edge IoTs is projected to reach more than 7.7 billion globally by 2030 [3]. SLMs facilitate efficient and seamless integration with these devices, unlocking edge AI capabilities and enabling real-time decision-making. Figure 1(b) illustrates the growth in edge device memory capacity relative to SLM size from 2018 to 2024. Initially, edge devices such as Raspberry Pi 3 featured a modest memory capacity of 1GB, which increased to 4GB in 2019 with devices like Jetson Nano. Concurrently, SLM sizes also expanded, with models like ”FLAME” starting at 0.06 billion parameters and rising to 1.1 billion parameters for ”Phi-1”. After 2020, both edge device memory and SLM size have accelerated significantly. For instance, edge devices such as Jetson Xavier NX and Banana Pi M64 have achieved 8GB, and in 2024, high-end devices like the Jetson Orin NX are expected to reach 16GB. Simultaneously, SLMs are also experiencing exponential growth, with models such as ”Phi-2” and ”Gemini Nano-2” increasing in parameter count, while the smallest of ”Llama-3”, which many open-sourced edge frameworks are built on, is anticipated to reach 8 billion parameters by 2024.

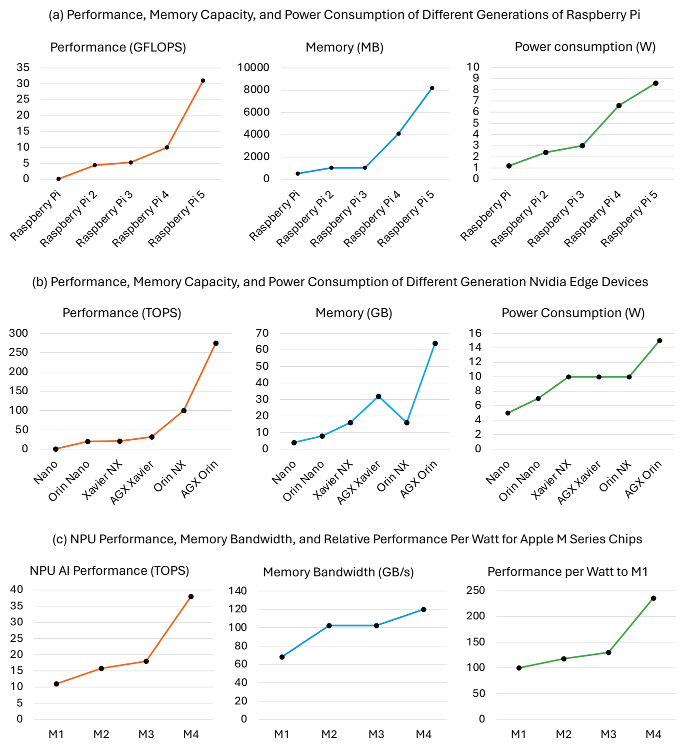

Fig. 2. Performance and Specifications of Representative Edge AI Hardware.

We investigate the recent changes in three representative edge hardware platforms for a wide variety of edge devices. Figure 2 illustrates that edge devices' performance and memory capacity have significantly improved over time. For instance, the Raspberry Pi series has seen substantial improvements in computing power and memory capacity from Generation 1 to Generation 5. Similarly, Nvidia's Jetson series has demonstrated advancements in AI performance and memory capacity from the Nano to the AGX Orin, supporting more demanding computing tasks and edge applications. However, these advancements are also accompanied by higher power consumption, indicating a trade-off between enhanced func-

TABLE I
BASIC SPECIFICATIONS OF USED EDGE DEVICES

Device Name	Memory	CPU Freq.	CPU #	Disk Size
Raspberry Pi 5B	4GB	2.4GHz	4	128GB
Jetson AGX Orin	32GB	2.2GHz	12	64GB
Mac mini	16GB	3.23GHz	8	494.38 GB

TABLE II
USED LANGUAGE MODELS AND THEIR PEREMETERS

Model Name	Model Size	Type	Tokens Trained on
TinyLlama	1.1B	SLM	3T
Phi-3 mini	3.8B	SLM	3.3T
OpenELM	270M	SLM	1.8T
Llama3	8B	LLM	15T

tionality and increased energy requirements. However, gains in performance per watt suggest that these devices are becoming more energy-efficient overall. As depicted in Figure 2(c), the NPU AI performance and memory bandwidth of Apple's M series chips, from M1 to M4, exhibit a significant upward trend, with improvements in performance per watt as well. This trend underscores advanced edge hardware to optimize energy efficiency while enhancing performance. Specifically, the M4 chip shows remarkable progress in AI performance, memory bandwidth, and energy efficiency. Despite the significant growth in SLM sizes, advancements in hardware, such as increased memory capacity and enhanced processing power, enable these devices to perform complex reasoning tasks efficiently. However, as shown by the similarities between Figure 1(a) and (b), the growth in software complexity far outpaces the improvements in hardware capabilities. It is anticipated that in coming years, the resources required for SLMs may catch up with and exceed the capabilities of most edge hardware. Therefore, this study aims to quantify, analyze, and understand the performance and resource footprint of current small language models on representative edge hardware, providing meaningful analysis and shedding light on guidance for future SLM and edge platform development.

III. METHODOLOGY

A. Devices and Hardware Specification

Table I lists the edge hardware specifications used in this study, representing the platforms on which current mainstream edge AI applications run. The Raspberry Pi 5B offers an affordable and powerful computing platform due to its low cost, versatility, small size, and strong community support. It is widely used by individual users, media, small businesses, IoT, robotics, and other fields, making it suitable for general AI tasks. As of May 2021, the Raspberry Pi Foundation announced that total sales of Raspberry Pi had exceeded 37 million units. In contrast, the Nvidia Jetson AGX Orin boasts excellent memory capacity, a multi-core CPU, and a GPU acceleration unit and targets more demanding AI and deep learning applications. Nvidia provides a comprehensive software ecosystem for the Jetson platform, including the JetPack SDK, which supports high-performance computing libraries such as CUDA, cuDNN, and TensorRT. Currently, the Nvidia Jetson series is a highly popular embedded AI platform, especially in scenarios requiring powerful computing power and low power consumption, such as industrial and commercial applications like robots, drones, healthcare, and transportation. Personal computers are also significant edge platforms due to their versatility, computing power, and flexibility. Particularly with the advent of AI PCs, more AI applications are reaching users through personal computer platforms. For this study, we use a Mac mini equipped with an M1 chip and 16 GiB memory as a representative of this type of edge device.

B. Models and Software

This research leverages a powerful open-source software ecosystem to deploy advanced small-language AI models on edge devices. Specifically, through the Hugging Face model repository, we accessed the latest versions of TinyLlama, Phi-3, OpenELM-270M, and Llama3-8B models, renowned for their efficiency in natural language processing tasks. We further optimized these models through quantization to reduce the model size and boost inference time, enabling deployment on memory-constrained devices. Concurrently, we adjusted the models' weights and activations to balance accuracy, model size, and performance, optimizing their parameters for the limited memory capacity of edge devices. The specifications of the models are shown in Table II. To run and utilize these models on edge platforms, we used Llama.cpp [4] for model inference. Llama.cpp is a C++ implementation for running various language models, providing an efficient and flexible way to deploy and execute these models, especially in resource-constrained edge environments. For OpenELM, we utilized the mlx-lm package and the MLX framework to enable the use of language models on Apple chips. MLX is an open-source machine learning framework developed by Apple, designed to leverage the unique capabilities of Apple hardware to enhance machine learning tasks.

C. Measurement and Collection

To evaluate different models running on various edge hardware, we standardized the model as 4-bit and user input to the same token request and constrained output to a fixed number of tokens while maintaining consistent end-to-end evaluation. This approach ensures a uniform and objective measure of each model's ability to generate text from given prompts. We utilized WikiText-2 [5] benchmark to assess model performance, focusing on metrics such as tokens per second and perplexity [6]. Perplexity is a critical metric in natural language processing that measures how well a probabilistic model predicts a sample, with lower scores indicating better performance in predicting word sequences. To expedite experiments with larger models, we downsampled WikiText-2 dataset, reducing evaluation time from 26 hours to 2 hours. For results presented in Figure 8 and 9, we used full WikiText-2 dataset; for Figure 10, we used downsampled version.

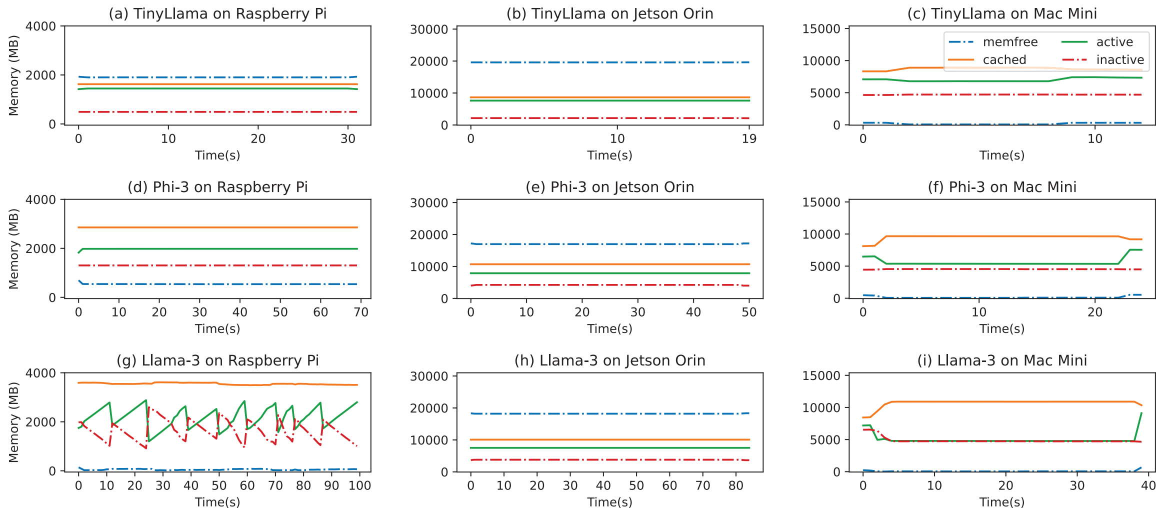

Fig. 3. Memory Utilization of Executing TinyLlama, Phi-3, and Llama 3 on Different Edge Devices

Perplexity, defined as the exponential mean negative log-likelihood of the sequence [7], is crucial for understanding a model's language capabilities. This metric helps quantify the trade-offs between model size, computational efficiency, and language accuracy, particularly important for small language models (SLMs) designed for efficiency and reduced computational load. By focusing on perplexity, we gain insights into the balance between performance and resource constraints, which is essential for deploying models on edge devices [8]. Given a tokenized sequence $X = (x_0, x_1, \ldots, x_t)$, the perplexity of X (denoted as PPL(X)) is defined as:

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_{i}^{t}\log p_\theta(x_i \mid x_{<i})\right\}$$

where $\log p_\theta(x_i \mid x_{<i})$ represents log-likelihood of the ith token, conditioned on the previous token $x_{<i}$ (i.e., all tokens before i). This metric intuitively assesses the model's ability to predict every token in the sequence uniformly and effectively. To compare the performance of various language models, it is important to take into account the tokenization method, as it can have a substantial impact on computed perplexity.

During the model executions, we leveraged the power of Llama.cpp and MLX to ensure accurate quantization and benchmarking of TinyLlama, Phi-3, OpenELM, and Llama 3 models, capturing detailed performance metrics on different edge platforms. Specifically, we used $iostat$, $Nmon$, and $PyNmonAnalyzer$ to collect and analyze system data on Raspberry Pi and Nvidia Jetson devices, including CPU utilization, memory usage, swap space, disk activity, and process switching rates. For Mac platforms, we utilized the PSutil library and Python scripts to gather system data on macOS. We employed the subprocess and time libraries to track the initial input time and the time of the first output token. We tracked the last token print time using the same method to calculate the model's latency across all devices. This comprehensive data collection and analysis allowed us to evaluate the performance of each model in real-world edge scenarios. The source code and raw data in this paper are available at https://github.com/Romyull-Islam/SLM.

IV. Performance Observations and Analysis

A. Memory Utilization

Observations. As illustrated in Figure 3(a), (d), the memory usage data for the Raspberry Pi 5 indicates that the available memory remains above zero when running the TinyLlama and Phi-3 models, suggesting that the Raspberry Pi can adequately support the operation of small language models. Conversely, the frequent exchange between active and inactive memory observed with the Llama-3 model in Figure 3(g) indicates high resource pressure and insufficient memory capacity. For the Nvidia Jetson Orin, the TinyLlama and Phi-3 models show substantial available memory, approximately 19,594 MB and 17,000 MB, respectively. Llama-3 also operates efficiently on the Jetson Orin, demonstrating that the 32GB memory of the Nvidia Jetson Orin can easily accommodate various small language models and some LLMs. The Mac Mini, with a total memory capacity of 16 GB, shows that the TinyLlama model utilizes approximately 8,894 MB of memory, leaving 62 MB available. Similarly, the Phi-3 model uses around 9,659 MB of memory, also leaving 62 MB available. Llama-3's memory usage peaks at 10,882 MB on the Mac Mini, with only 62 MB available at its lowest point. This discrepancy might be attributed to the differing memory management strategies of macOS and Linux, where macOS typically maintains a buffer to handle additional load, acting as a cache to expedite access to recently used data or applications.

Insights. The data collected from Raspberry Pi 5 indicates that both TinyLlama and Phi-3 exhibit significant memory consumption. However, TinyLlama demonstrates a smaller memory resource footprint, suggesting its greater suitability for memory-constrained edge environments. The pronounced memory fluctuations observed in Llama-3 underscore the challenges resource-constrained edge devices face in handling more demanding models, leading to increased memory swapping and consequent performance degradation. Similar

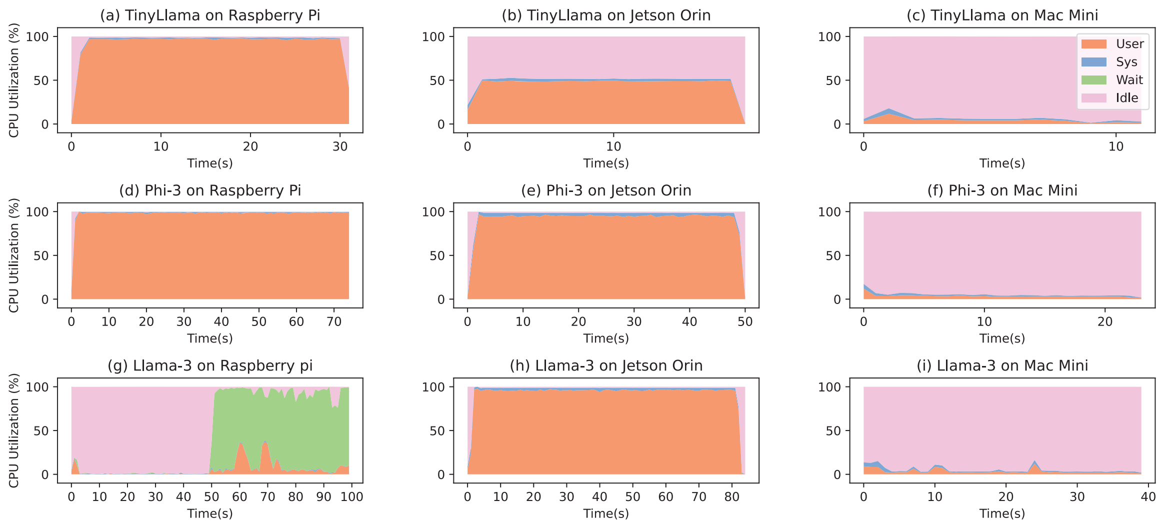

Fig. 4. CPU Utilization of Executing TinyLlama, Phi-3, and Llama 3 on Different Edge Devices

observations were made for small language models in the Nvidia Jetson Orin. However, the Nvidia Jetson Orin can also accommodate LLMs akin to Llama-3. The memory dynamics of the Mac Mini reveal that under moderate to heavy load conditions, the substantial amount of inactive memory serves as a fast-access resource pool. Despite minimal available memory, the Mac Mini's memory management is optimized for versatility and sustained performance under varying conditions, enabling reliable multitasking and efficient resource utilization. As mentioned earlier, all data underscores the necessity for optimization strategies for memory-intensive language models, particularly on resource-constrained platforms such as the Raspberry Pi, to ensure efficient application performance.

B. CPU Utilization

Observations. Figure 4 reflects the CPU utilization of TinyLlama, Phi-3, and Llama-3 during inference on Raspberry Pi, Nvidia Jetson Orin, and Mac Mini. The CPU utilization of TinyLlama and Phi-3 on Raspberry Pi (Figure 4(a), 4(d)) shows that the user CPU is always high, often reaching 96-99%, with almost no idle CPU, indicating that these small language models completely occupy almost all CPU resources during inference. There is significant fluctuation in Llama-3 on Raspberry Pi (Figure 4(g)). Initially, the system shows a high CPU idle rate, but after about 50 seconds, the user CPU increases significantly, and the I/O wait time also increases. Notably, the user CPU peaks at around 59 seconds, indicating a significant increase in CPU utilization. On Nvidia Jetson Orin (Figure 4(b), (h)), TinyLlama has moderate CPU utilization, with idle CPU close to 51%, indicating that Nvidia Jetson Orin can effectively support TinyLlama reasoning. Phi-3 and Llama-3 show high utilization, with user CPU peaking at 99% with little variation. CPU utilization for TinyLlama, Phi-3, and Llama-3 on Mac Mini (Figure 4(c), (f), (i)) is low to moderate, with idle CPU remaining above 80%.

Insights. The high CPU utilization of TinyLlama and Phi-3 on a Raspberry Pi highlights the computational intensity of small language models, which can push resource-constrained edge hardware to its limits and leave less capacity for other processes. This can cause instability or slowdowns in the system. In comparison, the larger model Llama-3 exhibited highly variable CPU usage on the Raspberry Pi. For instance, user CPU utilization spiked significantly at 50 seconds, 59 seconds, and 69 seconds, indicating that the system struggled to handle the workload, resulting in performance fluctuations. Notably, the model took approximately 50 seconds to load and generate the output, suggesting that an inappropriate pairing of models and edge devices can lead to latency and inefficiency. Similarly, the high disk busy rates and full memory utilization observed on the Raspberry Pi indicate heavy resource pressure, likely due to intense read/write operations and insufficient RAM, resulting in frequent swapping. On the Nvidia Jetson Orin, TinyLlama's CPU utilization was moderate, while Phi-3 and Llama-3 were at full CPU utilization. This shows that Jetson Orin can effectively handle smaller models like the 1.1B TinyLlama. The Mac mini's low CPU utilization, especially with Llama-3, demonstrates that desktop platforms can efficiently handle various SLMs and some large ones. When answering the same question, TinyLlama had the shortest running time, followed by Phi-3, while the larger language model Llama-3 took significantly longer. This is due to the larger models' more complex and detailed responses. Additionally, a horizontal comparison reveals that more powerful edge hardware results in shorter reply times for responses from the same model. Therefore, when deploying models, it is essential to consider CPU resource consumption and running time, as these factors may affect the operation of other applications at the edge.

C. Swap Area Utilization

Observations. As illustrated in Figure 5, when running on the Raspberry Pi, TinyLlama consistently maintained approximately 100 MB of available swap space throughout the test. This indicates that the physical RAM was sufficient, with minimal reliance on swap space. In contrast, Phi-3 had

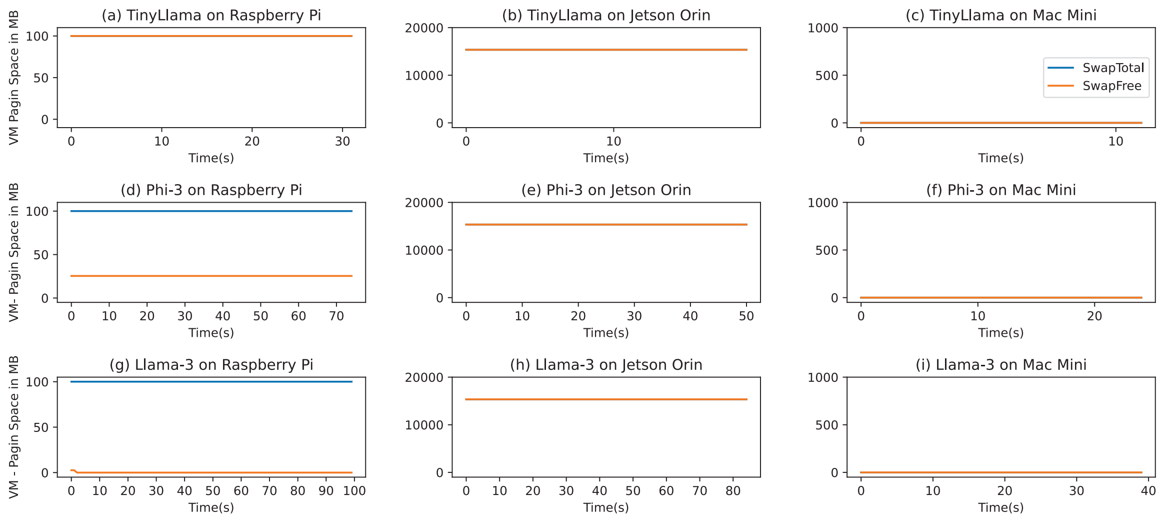

Fig. 5. Swap Area Utilization of Executing TinyLlama, Phi-3, and Llama 3 on Different Edge Devices

25.4 MB of swap memory available, suggesting effective memory management despite the workload exceeding the device's physical memory capacity. However, Llama-3 quickly exhausted its swap space, as corroborated by Figure 3. On both Jetson Orin and Mac Mini, neither TinyLlama nor Phi-3 utilized swap memory. This can be attributed to their larger physical memory, which provided sufficient resources to operate without depending on swap memory. Similarly, Llama-3 did not engage in swap memory on either the Jetson Orin or Mac Mini, highlighting the capability of these devices to handle high memory demands effectively.

Insights. Swapping is essential to maintaining computer performance and system stability, especially for memory-intensive language models deployed on edge hardware. Observations on the Raspberry Pi indicate it is well-suited for smaller, more lightweight language models. On the other hand, the Mac Mini and Jetson Orin do not utilize swaps, indicating their capability to handle the demands of various models effectively. This reflects their suitability for deploying resource-intensive models such as LLMs. These observations highlight the importance of aligning model requirements with appropriate hardware specifications to ensure optimal performance without overextending system memory resources. For users, this insight can guide the optimization of models such as Phi-3 and Llama-3 to better fit the memory constraints of less powerful devices or to target deployments on more capable hardware.

D. Disk Busy Rate

Observations. When the language models execute on edges, notable peaks in total disk activity are observed. As illustrated in Figure 6, the disk busy rate for TinyLlama initially surges to 4%, followed by a smaller peak of 3% at approximately 10 seconds, and then subsequently fluctuates around 2%. For Phi-3, the initial peak reaches approximately 16.6%, after which it fluctuates within 5%. Contrarily, Llama-3 on the Raspberry Pi rapidly increases to 200%, maintaining this elevated level throughout the observation period, indicating continuous disk activity. On the Jetson Orin, TinyLlama displays multiple peaks, with the highest reaching approximately 5% at 15 seconds. Phi-3 shows consistent smaller peaks, with the highest at about 4% initially, and fluctuates between 1-3% most of the time. Llama-3 on the Jetson Orin also exhibits consistent small peaks, predominantly around 2-4%. For TinyLlama on the Mac Mini, disk usage remains within .02% throughout the measurement, rendering very tiny disk activities. Phi-3 and Llama-3 show a significant spike of .05% in the middle.

Insights. After the first data is loaded into memory, the model runs mostly in RAM, which minimizes disk activity. This behavior benefits performance, as excessive disk activity can become a bottleneck, slowing overall processing. Both TinyLlama and Phi-3 were observed to run without sustained high disk activity post-initialization, suggesting that these models are optimized for efficient memory usage over sustained disk access. On the other hand, during initialization, Llama-3 on the Raspberry Pi demonstrated a sharp rise in combined disk busy rate of about 200%, which remained high for the duration of the observation period. This sustained high disk usage indicates Llama-3 exceeds available memory, leading to extensive swapping. Comparatively, the disk busy rate on the Nvidia Jetson Orin and Mac mini remained consistently lower and more stable, with the Mac mini benefiting from faster disk read and write speeds. These findings underscore the importance of disk I/O performance when deploying language models at the edge, particularly in scenarios where significant memory swapping is observed.

E. Latency Evaluation

Observations. In language models deployed at the edge where real-time performance is crucial, latency significantly affects user experience. Here, we measured two types of latency: Time to First Token (TTFT) and full generation time. TTFT refers to the interval between the user input and the first token output by the model. A high TTFT indicates slow initial responsiveness of language models. Full generation time measures the interval between user input and the complete

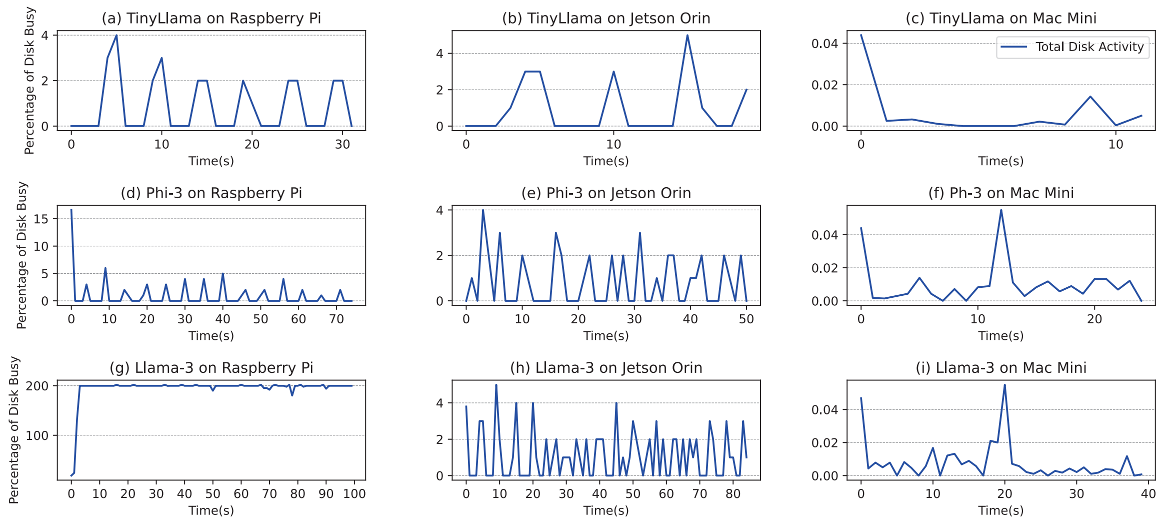

Fig. 6. Disk Busy Rate of Executing TinyLlama, Phi-3, and Llama 3 on Different Edge Devices

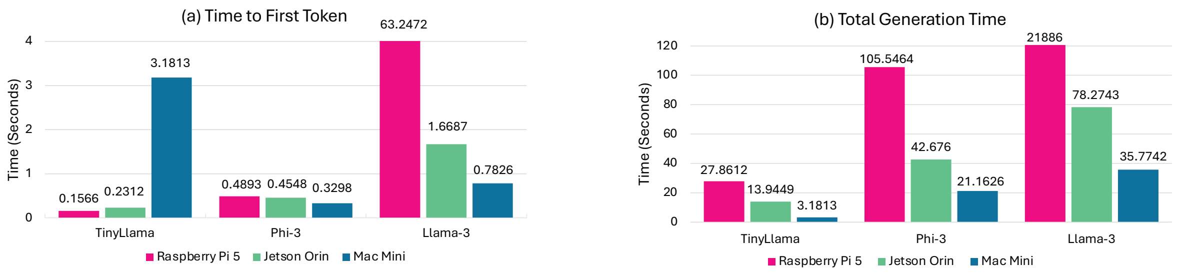

Fig. 7. Latency Comparison among Language Models across Edges

response output by the model. To standardize comparisons, we used the same user input and limited output length to 400 tokens. The latency comparison data for various 4-bit quantization models on different devices is shown in Figure 7, which represents the data in a logarithmic scale for better perception. TinyLlama consistently demonstrates impressively low latency on different edge devices, achieving the shortest TTFT and full generation times. Phi-3, while fast in providing initial responses, takes considerably longer to complete the full generation, particularly on resource-constrained devices like the Raspberry Pi. Phi-3 takes up to four times longer compared to TinyLlama, indicating higher computational demands. In comparison, Llama-3 exhibits a very high TTFT of 63 seconds and an unusually long total generation time of 21,886 seconds on the Raspberry Pi 5, highlighting the severe resource constraints of this device when handling such models. On the more resource-rich edges like Mac Mini, both the small language model and the optimized large language model perform much faster. However, the impact of latency on user experience remains significant and cannot be neglected.

Insights. The performance data underscores the importance of aligning model complexity with suitable hardware, particularly regarding latency, which is closely tied to user experience. The Raspberry Pi 5 demonstrates significant limitations when handling complex language models such as Llama-3, as evidenced by the extremely high TTFT and total generation time. This indicates that most edge devices are only suitable for deploying resource-intensive language models with substantial optimization. In contrast, the efficient performance of a small language model like TinyLlama on the Raspberry Pi illustrates that models optimized for low resource consumption can be effectively utilized in constrained environments. Compared to the Raspberry Pi, the Nvidia Jetson Orin, and the Mac Mini exhibit significant improvements in processing all models, especially Phi-3 and Llama-3. This highlights their capability to handle more complex models with reduced latency. When deploying various language models in different edge scenarios, balancing optimal performance and user experience is crucial by matching the model requirements with the appropriate hardware specifications.

F. *Inference Performance*

Observations. For half-precision floating point format (F16) quantization using the Wikitext dataset, TinyLlama demonstrates superior processing speed with 22.85 tokens per second on the Raspberry Pi 5, which increases to 40.74 tokens per second on the Nvidia Jetson Orin and peaks at 694.69 tokens per second on the Mac Mini. Phi-3 performs well, with 4.77 tokens per second on the Raspberry Pi, 13.31 tokens per second on the Jetson Orin, and 211.88 on the Mac Mini. Introducing 4-bit quantization to TinyLlama shows an improvement in processing speed on the Raspberry Pi 5 from 22.85 to 27.61 and on the Nvidia Jetson Orin 40.74 to 46.52 tokens per second. However, it decreases slightly on Mac Mini from 694.69 to 603.94 tokens per second. This suggests that

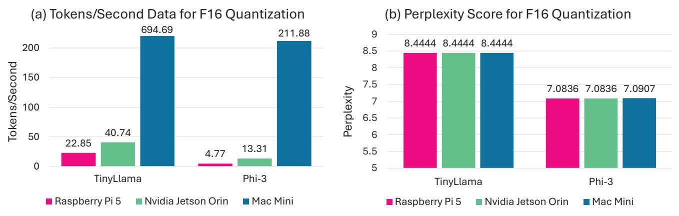

Fig. 8. Performance of TinyLlama, Phi-3 for F16 Quantization

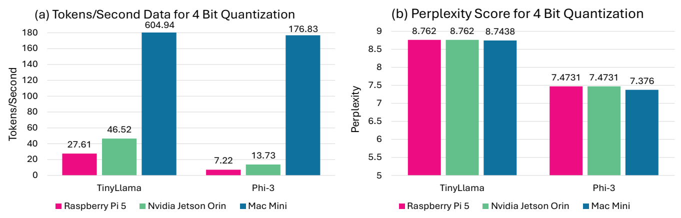

Fig. 9. Performance of TinyLlama, Phi-3 for 4-Bit Quantization

quantization in smaller memory devices effectively reduces computational overhead without substantially reducing model performance. However, quantization may be less beneficial for devices with sufficient computing power for this type of operation, such as the Mac Mini. Phi-3's performance also reduced with 4-bit quantization on Mac Mini, with 176.83 tokens per second on the Mac Mini. But, the Raspberry Pi 5 and Jetson Orin increased to 7.22 from 4.77 and 13.73 from 13.31 tokens per second on the Jetson Orin. Figures 8 and 9 represent performance data for both F16 and 4-bit quantized models in various environments.

Insights. TinyLlama's significant performance leap on the Jetson Orin indicates its ability to effectively leverage advanced hardware, highlighting its scalability and suitability for high-end computational environments. The performance of Phi-3 on the Jetson Orin, while improved, emphasizes a design focus on operational efficiency over raw performance, making it suitable for applications prioritizing consistency and reliability, particularly in resource-constrained settings. The consistent performance of Phi-3 on various devices highlights its optimized efficiency in diverse operational contexts. Despite the differences in computing power, the stable processing speeds across both devices showcase Phi-3's robust design, making it an attractive option for edge-computing applications.

The increase in tokens per second after applying 4-bit quantization to the models emphasizes the benefits of this technique in computational efficiency. Although there is a slight decrease in prediction precision, as indicated by a small increase in perplexity, the trade-off remains favorable. This demonstrates the effectiveness of quantization strategies in enhancing processing speeds, which is particularly beneficial in resource-constrained environments like the Raspberry Pi 5. However, as the M1 chip of Mac Mini is highly optimized for floating-point operations, including F16, there are better options than 4-bit quantization compared to its capacity.

G. Prediction Accuracy

Observations. As illustrated in Figure 8(b), for F16 quantization using the Wikitext dataset, TinyLlama achieves a perplexity score of 8.4444 across all tested devices, which indicates that the model's prediction accuracy is largely independent of the hardware platform. In contrast, Phi-3 demonstrates a perplexity of 7.0836 on the Raspberry Pi and Jetson Orin and 7.0907 on the Mac Mini, highlighting its relatively high prediction accuracy across different edge environments. Figure 9(b) reveals that with the introduction of 4-bit quantization, the perplexity of TinyLlama increases slightly to 8.7620 on the Raspberry Pi, 8.76 on the Jetson Orin, and 8.7438 on the Mac Mini. Similarly, Phi-3, when subjected to 4-bit quantization, exhibits a perplexity of 7.4731 on both the Raspberry Pi and Jetson Orin and 7.3760 on the Mac Mini. These findings suggest that while quantization impacts the perplexity scores, the effect remains relatively consistent across different hardware.

Insights. The consistent perplexity scores of TinyLlama and Phi-3 across various devices show their stable prediction accuracy across different hardware platforms, demonstrating the robustness and adaptability essential for models deployed in diverse computing environments. Notably, Phi-3 exhibits a lower perplexity, indicating superior prediction accuracy and efficiency in processing complex language data. Under 4-bit quantization, the perplexity of both TinyLlama and Phi-3 increases, attributable to the loss of accuracy following quantization, which can lead to an increased likelihood of producing hallucinations. Despite this, our results indicate that the prediction accuracy of small language models such as TinyLlama and Phi-3, even with 4-bit quantization, remains within an acceptable range. For future deployments of models in edge scenarios, it is crucial to consider the trade-off between performance and accuracy.

H. Comparison with Larger Language Models

Observations. In addition to examining resource footprint and performance, we are also interested in exploring the differences between SLMs and LLMs. We compare Phi-3, TinyLlama, and Llama-3 on various devices using a modified Wikitext-2 dataset. TinyLlama's efficiency on the Raspberry Pi 5 is demonstrated by its achievement of 27.22 tokens/s with a perplexity of 8.02, as shown in Figure 10. Phi-3 demonstrates better prediction accuracy than TinyLlama, processing 7.27 tokens/s with a perplexity of 5.9147. Llama-3, constrained by edge resource limitations, shows lower performance, achieving only 3.13 tokens per second with a perplexity of 7.8452. On Nvidia Jetson Orin, Phi-3, TinyLlama, and Llama-3 exhibit similar performance patterns to those observed on Raspberry Pi 5, with perplexity remaining relatively constant but throughput improved. All three language models achieve much higher throughput on the more powerful Mac Mini. For example, Llama-3 performs 26 times better than TinyLlama, processing 620.96 tokens/s. Notably, the perplexity of the models remains largely unchanged across different hardware platforms.

Insights. Our experiments showed a correlation between throughput and prediction accuracy across various models and edge device combinations. TinyLlama continuously displays high tokens per second, underscoring its robust processing

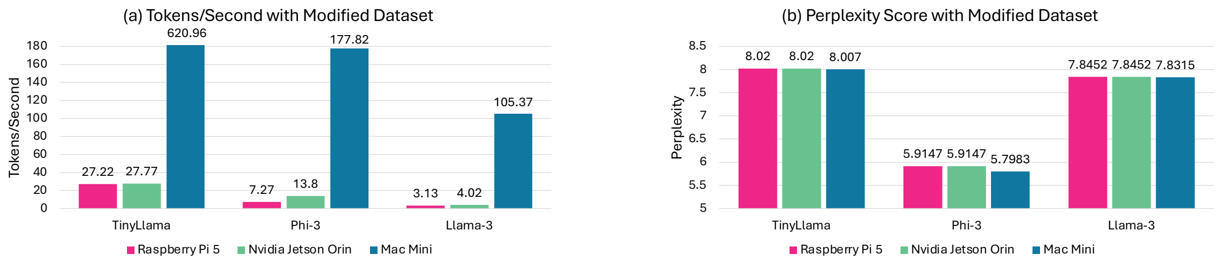

Fig. 10. Throughput and Perplexity Comparison between SLM and LLM on Different Edges

efficiency across all tested edges. This characteristic renders it well-suited for deployment in edge scenarios where resources are limited or rapid response times are necessary. Furthermore, TinyLlama's high tokens per second on the Mac Mini and Nvidia Jetson Orin demonstrate its scalability and suitability for high-end computing environments. On the other hand, Phi-3's lower perplexity scores on all platforms demonstrate its superior prediction accuracy, making it a great choice for applications that require consistent and dependable predictions, especially in settings with limited resources. On the other hand, the large language model Llama-3, despite achieving commendable perplexity scores (relative to TinyLlama), demands higher resources. It is more suited for applications where high language accuracy is paramount, provided that adequate computing resources are available.

I. Performance Observation and Analysis of OpenELM

Observation. In addition to TinyLlama, Phi-3, and Llama-3, we also measured Apple's OpenELM-270M model. Since it cannot be run on other hardware, we measured it exclusively on a Mac Mini. OpenELM-270M was executed using the MLX-LM framework without any quantization. As shown in Figure 11, the CPU usage chart predominantly indicates high idle time, with brief spikes in user and system activities, especially at the start. Memory usage remains stable across all parameters (memfree, used, active, inactive). Swap usage is negligible, with the total and free swap lines nearly zero. Disk usage is consistently low, with no significant fluctuations in total disk activity. These observations suggest that no significant swap activity occurs between the main memory and disk during OpenELM inference. Initial high disk activity due to the execution of third-party activity monitors to capture disk information.

Insights. The high idle time in CPU utilization indicates that the Mac Mini's SoC effectively handles the OpenELM model with 270 million parameters. The observed stable memory usage, consistently low disk activity, and negligible swap usage confirm the sufficiency of physical memory, thereby ensuring responsive system performance. Furthermore, due to macOS's unique memory management and compression techniques, we posit that OpenELM could be well-suited for edge hardware with smaller memory capacities, such as iPhones and iPads. With the recent release of Apple Intelligence, we anticipate further evaluating the performance of OpenELM on other edge hardware in future macOS and iOS systems.

V. Related Work

Resource Utilization on Edges. Deploying applications on edge devices requires careful control over resource allocation and usage. For example, Park et al. [9] dived into the nuances of hardware resource utilization, specifically memory and CPU usage, during distributed training of deep learning models. In recent years, more and more scholars have begun to explore the resource footprint of various new artificial intelligence models on edge devices. For example, Rahman et al. [10] evaluated the model size, latency, and performance trade-offs of the TensorFlowLite MobileBERT model in an edge computing environment. Vucetic et al. [11] proposed how to fine-tune the BERT model to improve the resource efficiency on edge devices. Dhar et al. [12] explore the challenges and potential bottlenecks that currently hinder the effective deployment of large language models on edge devices. Unlike the above work, our work focuses on the deployment and performance analysis of state-of-the-art small language models on different edge platforms.

Optimizing Edge Language Models. There are two main categories of approaches to running language models on resource-constrained edge devices. The first category focuses on reducing model size and complexity. For example, Liu et al. [13] proposed removing less important weights or neurons from the network and focusing on the effectiveness of the pruned architecture itself. Quantization can also reduce the number of bits required to represent model weights and activations, reducing memory usage and computational load while ensuring model accuracy [14], [15]. Knowledge distillation involves training smaller models to replicate the behavior of larger models. For example, Jiao et al. applied this technique to the BERT model, ultimately creating a smaller and more efficient version called TinyBERT [16]. In addition to various model compression techniques, another category of approaches focuses on edge-tailored customization for language models. For example, Bandit-NAS [17] proposed a method to explore and find the best architecture based on hardware constraints, data slicing, and training duration. Liu et al. [18] discussed Differentiable Model Scaling (DMS), which optimizes the network width and depth in a fully differentiable manner to achieve lower perplexity and higher accuracy. Different from these optimizations, we provide a comparative study and analysis of the performance and resource footprint of the latest small language models on edges, and our work is orthogonal to these optimizations.

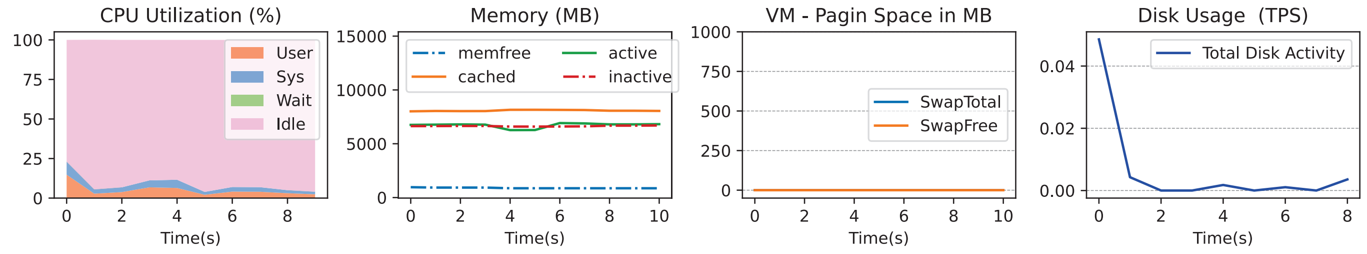

Fig. 11. Performance Statistics and Analysis of OpenELM on Mac Mini

Performance Analysis in Custom Edge-AI Systems. With an increasing number of edge devices running various custom AI systems, understanding the performance of machine learning or artificial intelligence models on these IoT and edge devices is critical to their effective deployment. Fanariotis et al. [19] proposed creating energy-efficient and low-latency models on microcontroller-based systems to reduce resource overhead in IoT applications. Using a large language model, Shen et al. [20] proposed an autonomous edge AI system to accommodate various user needs in a physically connected and highly networked environment. Another study evaluated the performance of MobileBERT models transformed with FlatBuffer on edge devices [10]. These models were fine-tuned for reputation analysis of English tweets and achieved significantly smaller footprints with minimal accuracy loss. In addition, a Transformer-based speech recognition system [21] study evaluated the inference performance of different edge devices. These studies highlight the effectiveness of model transformation and quantization techniques in optimizing performance on resource-constrained devices. Unlike the above works, we focus more on the performance of SLM on representative edge devices, such as Raspberry Pi and Jetson AGX Orin. At the same time, we also explore a wider range of computing constraints and resource footprints to gain a more comprehensive understanding of how to deploy SLM effectively in different edge computing environments.

VI. Conclusion

In this paper, we present a thorough analysis of representative SLM workloads across different edge devices. In particular, we conduct a comprehensive empirical analysis of different state-of-the-art SLMs, including TinyLlama, Phi, OpenELM, etc., on edge devices ranging from resource-constrained Raspberry Pi units to powerful Nvidia Jetson Orin systems, as well as desktop edge platforms such as Mac mini. We quantitatively evaluate the resource consumption and footprint of SLMs on these diverse edge hardware. Furthermore, we also compare the performance variations of different SLMs on edges and provide corresponding analysis and insights. As far as we are concerned, this is the first study to investigate the performance and resource consumption of small language models on edge platforms. Our analysis and findings aim to assist users in deploying appropriate models and workloads in their specific scenarios and provide guidance for optimizing the next generation of AI workloads in the most recent edge environments.

Acknowledgement

We thank the anonymous reviewers for their suggestions and feedback. This research was in part supported by US NSF under Grants: CPS-2103459, SHF-2210744, CNS-2103405, CNS-2244450, and AMPS-2229073.

References

[1] *Everything You Need to Know about Small Language Models (SLM) and its Applications*, available: https://tinyurl.com/29ye83ap/.
[2] *ChatGPT and Generative AI are Booming, but the Costs can be Extraordinary*, available: https://tinyurl.com/5e8r299b.
[3] *Number of Edge Enabled Internet of Things (IoT) Devices Worldwide from 2020 to 2030, by Market*, available: https://www.statista.com/statistics/1259878/edge-enabled-iot-device-market-worldwide/.
[4] *Llama.cpp*, available: https://github.com/ggerganov/llama.cpp.
[5] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.
[6] *Two Minutes NLP — Perplexity Explained with Simple Probabilities*, available: https://medium.com/nlplanet/two-minutes-nlp-perplexity-explained-with-simple-probabilities-6cdc46884584.
[7] *Perplexity of Fixed-length Models*, https://huggingface.co/docs/transformers/en/perplexity.
[8] *How LLM Quantization Impacts Model Quality*, available: https://deci.ai/blog/how-llm-quantization-impacts-model-quality/.
[9] S. Park, J. Lee, and H. Kim, "Hardware resource analysis in distributed training with edge devices," *Electronics*, 2019.
[10] M. W. U. Rahman, M. M. Abrar, H. G. Copening, S. Hariri, S. Shao, P. Satam, and S. Salehi, "Quantized transformer language model implementations on edge devices," *arXiv preprint arXiv:2310.03971*, 2023.
[11] D. Vucetic, M. Tayaranian, M. Ziaeefard, J. J. Clark, B. H. Meyer, and W. J. Gross, "Efficient fine-tuning of bert models on the edge," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022.
[12] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, and K. Suo, "An empirical analysis and resource footprint study of deploying large language models on edge devices," in *Proceedings of ACM Southeast Conference*, 2024.
[13] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.
[14] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *ICML*, 2023.
[15] T. Huang, T. Luo, M. Yan, J. T. Zhou, and R. S. M. Goh, "RCT: resource constrained training for edge AI," *CoRR*, 2021.
[16] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling BERT for natural language understanding," *CoRR*, 2019.
[17] Y. Lin, Y. Endo, J. Lee, and S. Kamijo, "Bandit-nas: Bandit sampling and training method for neural architecture search," *Neurocomputing*, 2024.
[18] K. Liu, R. Wang, J. Gao, and K. Chen, "Differentiable model scaling using differentiable topk," *arXiv preprint arXiv:2405.07194*, 2024.
[19] A. Fanariotis, T. Orphanoudakis, K. Kotrotsios, V. Fotopoulos, G. Keramidas, and P. Karkazis, "Power efficient machine learning models deployment on edge iot devices," *Sensors*, 2023.
[20] Y. Shen, J. Shao, X. Zhang, Z. Lin, H. Pan, D. Li, J. Zhang, and K. B. Letaief, "Large language models empowered autonomous edge ai for connected intelligence," *IEEE Communications Magazine*, 2024.
[21] S. Gondi and V. Pratap, "Performance evaluation of offline speech recognition on edge devices," *Electronics*, 2021.