Efficient Deployment of Lightweight LLMs on Edge Devices: Kernel-Level Profiling and Cross-Platform Insights

Xinyao Liu*, Xingzhou Zhang*†, and Weisong Shi‡
*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, 100190
†University of Chinese Academy of Sciences, Beijing, China, 100190
‡Department of Computer and Information Science, University of Delaware, Newark, USA., 19716
lxyaaao0404@gmail.com, zhangxingzhou@ict.ac.cn, weisong@udel.edu

***Abstract*—As Large Language Models (LLMs) are increasingly adopted in real-time edge scenarios, their efficient deployment on heterogeneous, resource-constrained devices becomes a critical challenge. However, the diversity of model architectures, hardware platforms, and inference frameworks complicates performance prediction and optimization. This paper presents a systematic evaluation of seven lightweight LLMs across four representative edge platforms, including the MacBook Pro, Jetson Orin NX, Jetson AGX Orin, and an edge server with a desktop GPU. The evaluation focuses on key metrics such as model load time, token-level inference latency (TPOT and TTFT), memory usage, power dissipation, and semantic output relevance. Moreover, this paper explores hardware-specific bottlenecks such as memory-bound operations and inefficient kernel execution by using fine-grained profiling tools. The analysis reveals significant performance variation across model-hardware combinations, highlighting the importance of model selection and optimization techniques, including operator fusion and cross-platform insights. These findings offer practical guidance for deploying LLMs efficiently in real-world edge environments.**

***Index Terms*—Edge Computing, Large Language Models, Kernel-Level Profiling, Cross-Platform Evaluation, Model Deployment Optimization**

I. Introduction

The surge of edge computing [1] [2] has reshaped the landscape of Artificial Intelligence deployment [41]. By moving computation closer to data sources, such as sensors, mobile devices, vehicles, and embedded systems, edge computing enables a new class of intelligent applications that are capable of operating under bandwidth constraints or in privacy-sensitive scenarios. Meanwhile, Large Language Models (LLMs) [3] have revolutionized tasks like text generation and coding. As LLMs are increasingly integrated into mobile and embedded systems, the demand for their efficient deployment on resource-limited devices is rising sharply.

However, LLMs were originally designed to be deployed on server-class hardware, benefiting from large memory capacity, high-throughput computation, and dedicated acceleration resources such as high-end NVIDIA GPUs [4]. This design paradigm conflicts with resource-constrained edge environments, creating significant challenges when shifting deployment from cloud data centers to edge devices.

The shift to edge AI introduces significant challenges, primarily inference latency and memory usage [6] [7] [39]. Edge environments are highly sensitive to delays. However, unlike batched cloud processing, edge devices handle interactive, single-stream tasks, often resulting in low hardware utilization. Furthermore, limited memory on devices like the Jetson Orin [36] complicates model hosting, necessitating aggressive quantization [5] [42], or operator-level optimizations [8] to prevent suffering from severe performance degradation due to excessive swapping. Power consumption and hardware heterogeneity further complicate deployment. Unlike the predictable patterns of classic DNNs, LLMs are sequential and dynamic, with attention mechanisms relying on large matrix operations that create specific bottlenecks, further widening the gap between the computational demands of LLMs and the limited capabilities of edge hardware.

To address current gaps, we conduct a comprehensive empirical study of lightweight LLMs on diverse platforms, reflecting real-world deployment conditions from low-power ARM devices to high-throughput workstations. Our evaluation framework is primarily structured around several key aspects, which are comprehensively detailed in Fig. 1.

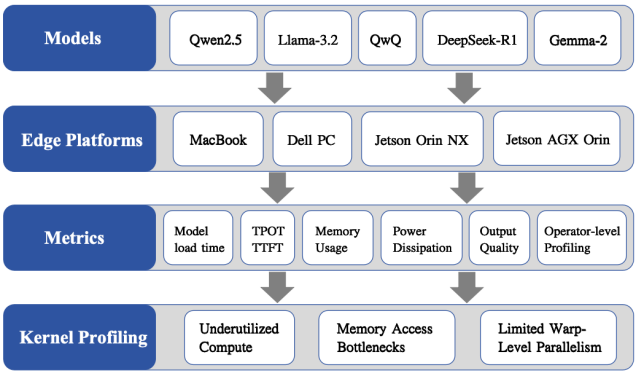

Fig. 1: Overview of Study Methodology and Key Aspects

Importantly, beyond common runtime metrics, our study incorporates fine-grained operator-level profiling using NVIDIA

Nsight Systems [34] and Nsight Compute [35]. This systematic approach allows us to bridge the gap between macro-level throughput and micro-architectural execution, uncovering a bottleneck-shifting phenomenon across heterogeneous platforms that is often overlooked by black-box benchmarking.

Our key contributions are:

- **Comprehensive Experimental Results.** We provide standardized benchmark data for lightweight LLMs, aiding informed model selection for edge deployments.
- **Multi-level Performance Insights.**
  - Model Performance on Diverse Platforms. GPU-based inference is faster than CPU-based, but GPU-based loading is slower due to overhead. Furthermore, architectural differences matter: models like Qwen are more efficient than Gemma at similar sizes, proving parameter count isn't the only performance factor.
  - Power Dissipation. Architectural design, not just model size, is also a primary driver of power consumption. This necessitates selecting an efficiently designed model to minimize power usage.
  - Operator-Level Bottlenecks. GEMM kernels dominate runtime, limited by memory access and underutilized compute capacity. On-device inefficiency isn't just about a lack of compute, but a fundamental mismatch between Transformer kernel configurations and edge memory hierarchies.
- **Foundation for Future Research.** By offering a deeper, hardware-aware understanding of LLM behavior on edge devices, our insights pave the way for future efforts in efficient, scalable, and adaptive on-device LLM inference.

The remainder of this paper is organized as follows: Section 2 provides background and reviews related work. Section 3 details our methodology. Section 4 presents the benchmark results and analysis of various LLMs across edge platforms. Section 5 dives into the operator-level profiling, revealing fine-grained performance bottlenecks. Finally, Section 6 offers a discussion and implications for future research.

II. Background and Related Work

This section provides the necessary background of our study and situates our work within the existing literature.

A. LLM Inference

LLMs such as GPT [9], LLaMA [10], Deepseek [11], and Qwen [12] have achieved remarkable progress in natural language processing. These models are based on the transformer architecture [3] and typically contain hundreds of millions to billions of parameters. To manage their substantial computational and memory demands, particularly during inference, model parallelism strategies are crucial. These approaches, broadly categorized as intra-operator and inter-operator parallelisms [30] [38], facilitate the deployment of larger models but with varying performance implications. During inference, LLMs generate text autoregressively, predicting one token at a time based on the previous context.

Inference involves multiple compute-intensive operations [17], including attention mechanisms and feed-forward layers. This process is particularly sensitive to latency and hardware efficiency, especially outside of the data center environment.

B. Edge Devices

Edge devices [1] refer to computing platforms that operate close to the data sources or end-users. Examples include smartphones, vehicles, and embedded systems such as the NVIDIA Jetson series. Edge computing enables real-time interaction and enhances data privacy [16] by processing information locally instead of relying on cloud servers. This reduces the latency caused by round-trip communication to the cloud, and minimizes the risk of data exposure.

However, edge platforms are significantly constrained compared to server-class hardware [41]. They offer less compute power, limited memory capacity, and are subject to strict power and thermal limits. While some advanced edge devices offer GPU acceleration [36], these resources are often limited in scale and vary widely across vendors and platforms. Thus, making efficient use of edge devices has become a concern.

C. Deployment of LLMs on the Edge

Deploying LLMs on edge devices presents unique challenges due to the mismatch between model demands and hardware capabilities [24] [25]. One of the most pressing issues is inference latency. Unlike those in the cloud, edge deployments often serve a single user at a time and cannot rely on batch processing to improve efficiency. Each token must be generated quickly and sequentially, making per-token latency a key concern in user-facing applications.

Memory is another major constraint [26] [27]. Edge devices typically come with less than 16GB of memory, which must accommodate both the model weights and intermediate activations. This makes it difficult to deploy large models without optimizations such as quantization, pruning, or operator fusion [31] [44]. Even with compression, managing memory efficiently is crucial for enabling smooth runtime performance on single, constrained GPUs [37].

While LLMs have seen many evaluations in cloud environments [20] [21], and other neural networks have been well-benchmarked on the edge [6] [7] [15], the in-depth analysis of LLMs specifically on the edge remains a nascent area. This gap is significant, given the increasing demand for on-device AI. Table I summarizes current research efforts in evaluating LLMs on edge platforms.

Among them, MELTing Point [14] provides an automated pipeline for evaluating LLMs on mobile and edge platforms. However, it lacks fine-grained operator or kernel-level analysis. Furthermore, its reliance solely on automated deployment frameworks means it lacks the implications of nuanced deployment strategy choices, such as specific back-end selections.

Compared with previous works, our work differentiates through the depth and breadth. While we do use some standard metrics from prior works like inference latency, our work is one of the first to analyze the challenges by going deep into the

TABLE I: Overview of Evaluating LLMs on the Edge

Researches	Year	Affiliation	Devices	Models	Metrics
[14]	2024	Brave Software	Raspberry Pi 4, iPhone SE, Jetson Orin AGX, etc.	TinyLlama, MistralAI-7B, Gemma, Llama-2, etc.	Runtime, Energy efficiency, Thermal behavior, Quality of Experience
[18]	2024	Chinese Academy of Sciences	Jetson Nano, Raspberry Pi 4B, Dell, Lenovo, etc.	Large Qwen2.5, MiniCPM, Llama3, etc.	Load Time, TTFT, TPOT
[23]	2024	Indian Institute of Science	Nvidia Jetson Xavier AGX	Llama-2, Llama-3.1	Throughput, Accuracy
[22]	2025	University of Leeds	Raspberry Pi 5 cluster	InternLM, Phi, Llama2, Zephy, Gemma, etc.	Average Latency, Decode Throughput, Memory Usage, CPU Utilization, Accuracy
This Work	2026	Chinese Academy of Sciences	MacBook, Dell, Jetson Orin NX, Jetson AGX Orin	Qwen2.5, Llama-3.2, QWQ, DeepSeek-R1, Gemma-2	Loading time, Throughput, Memory Usage, Power, Accuracy, **Operator-level Profiling**

kernel level, providing evidence of the underlying bottlenecks and insights that are not available in prior work. Our toolset URL is https://github.com/lxyaaao/toolset.

III. Methodology

This section details our experimental design, covering model and platform selection, the toolchain for measurement, and the metrics used for performance evaluation. We also describe our two-stage profiling methodology to identify performance bottlenecks on heterogeneous edge hardware.

A. Model and Platform Selection Strategy

To evaluate the deployment feasibility and runtime performance of LLMs in edge scenarios, we conduct experiments using a diverse set of models and hardware platforms.

Model Selection. The models used in our evaluation are included in Table II.

TABLE II: Large Language Models

Models	Parameter	Architecture
Qwen2.5	0.5B	Transformers with RoPE, SwiGLU, RMSNorm, Attention QKV bias and tied word embeddings.
Qwen2.5	1.5B	
Qwen2.5	3B	
Llama-3.2	1B	An optimized transformer architecture using SFT and RLHF.
QwQ	1.5B	Transformers with RoPE, SwiGLU, RMSNorm, and Attention QKV bias.
DeepSeek-R1	1.5B	Transformers with Multi-head Latent Attention, Mixture-of-Experts, and flash attention 2, GQA.
Gemma-2	2B	Transformers with RoPE, GeGLU, RMSNorm, multi-query attention and RMSNorm.

This selection emphasizes models around the 1.5B parameter scale to represent a balance between performance and resource consumption. The inclusion of multiple variants from the Qwen family enables intra-architecture comparison under consistent architecture, while Llama, Gemma [13], QwQ, and DeepSeek introduce architectural diversity.

To accommodate hardware constraints and toolchain compatibility, Qwen, Llama, and Gemma series are quantized to 8-bit integer precision (Q8), while DeepSeek and QwQ variants are run using BF16 where supported. While this introduces some variance in numerical precision, we aim to reflect realistic deployment conditions, where quantization formats are often chosen based on platform and model compatibility. Our analysis in Section 4 accounts for these precision differences when interpreting the power and latency trade-offs.

Hardware Platforms. Our evaluation spans a diverse range of hardware platforms chosen to reflect realistic deployment scenarios and allow for a comprehensive assessment of LLM behavior. We deploy and evaluate the models on the following devices, with their detailed parameters provided in Table III.

TABLE III: Hardware Parameters of Edge Devices

Devices	Processor	Memory	Computing Power
MacBook Pro	CPU: Apple M4	16 GB	10-core
	GPU: -	-	-
Jetson Orin NX	CPU: ARM Cortex-A78AE	16 GB	8-core@2GHz
	GPU: Ampere GPU	16 GB	1024-core@0.918GHz
Jetson AGX Orin	CPU: ARM Cortex-A78AE	64 GB	12-core@2.2GHz
	GPU: Ampere GPU	64 GB	2048-core@1.3GHz
Edge Server	CPU: Intel i7-13700	64 GB	16-core@2.1GHz
	GPU: NVIDIA RTX 3090	24 GB	10496-core@1.70GHz

This diverse platform set captures a wide range of compute capabilities, from mobile and embedded systems to workstation-class environments, thereby allowing us to evaluate LLM behavior across edge-to-cloud boundaries and ob-

serve the effect of hardware resources on inference performance.

B. Toolchain and Measurement

To ensure consistency across diverse platforms, we utilize a standardized set of tools for model deployment, inference, and profiling:

llama.cpp [32] llama.cpp is a lightweight, portable C++ inference engine designed for efficient LLM execution across diverse hardware, including both NVIDIA GPUs (CUDA) and CPU-only modes. Its support for quantization and hybrid CPU-GPU execution significantly reduces memory usage and boosts inference speed. We use llama.cpp as our baseline for consistent latency and throughput metrics due to its low overhead and broad cross-platform compatibility.

PyTorch. [33] PyTorch is an open-source machine learning framework. We utilize PyTorch as the inference framework for models needing deeper integration with specific profiling tools. It provides greater flexibility for layer inspection, operator tuning, and integration with GPU profiling tools. This helps us pinpoint framework-level bottlenecks, such as redundant kernel invocations and suboptimal memory access patterns.

In our experiment, llama.cpp is used for inference on all platforms where GGUF-formatted models are available. It offers optimized low-level execution of Transformer-based models, particularly suited for constrained environments. Meanwhile, Transformers + PyTorch is employed on all platforms, using the same models in standard PyTorch format. This enables precise comparison with traditional inference pipelines.

To minimize framework-induced bias, we ensure that both llama.cpp and PyTorch-based executions use synchronized weight quantization scales and identical hyper-parameters. While llama.cpp is prioritized for end-to-end performance benchmarking due to its high optimization for edge devices, PyTorch is exclusively used as a diagnostic vehicle for the fine-grained profiling.

We also adopt platform-specific back-end configurations:

- On the MacBook Pro, the CPU back-end is used due to the lack of NVIDIA GPU support.
- On the edge server, GPU acceleration with CUDA is enabled to maximize performance.
- On the Jetson Orin, both CPU and CUDA back-ends are tested: CPU runs are conducted alongside GPU runs to provide comparative insights into computational bottlenecks and performance differences across back-ends.
- On the Jetson AGX Orin, we evaluate both the default low-power configuration and the MAXN performance mode. This allows us to investigate the trade-off between energy efficiency and inference performance on edge devices.

By unifying model format and toolchain while varying back-ends and platforms, this setup provides a fair ground for comparing runtime efficiency and deployment viability.

C. Benchmark Tasks and Metrics

To evaluate the performance of LLMs across different edge platforms, we adopt a standardized benchmark protocol encompassing the following key metrics:

Load Time (ms). Measures the time needed to load a model and initialize the inference engine. It reflects startup latency, which is especially important in edge scenarios where models may need to be loaded on-demand and where user experience depends heavily on startup latency.

TPOT (ms/token) and TTFT (ms). Time Per Output Token (TPOT) indicates the average latency to generate a token, which directly reflects runtime efficiency and is critical in real-time applications deployed on resource-constrained devices [19] [20]. Time To First Token (TTFT) measures the latency from the submission of the user's prompt until the first token of the response is generated. It is a crucial, user-centric metric as it determines the perceived interactivity of the application.

CPU Memory Usage (GB). Records the peak system RAM usage during inference with CPU back-ends. Since many edge devices are limited to 16 GB RAM, understanding the memory footprint is essential for ensuring successful deployment.

GPU Memory Usage (GB). Captures peak VRAM consumption during GPU inference. On devices like Jetson AGX Orin with limited GPU memory, this metric helps assess whether a model fits within hardware constraints [28].

Power Dissipation (W). Measures the device's instantaneous power consumption during inference. On battery-powered edge devices like Jetson Orin series, this metric helps assess battery life, and ensure sustained performance [46].

Relevance Score. Serves as a symbol of generation quality. It is computed using semantic similarity between model outputs and reference answers in a Chain-of-Thought (CoT) dataset [40]. Unlike simple output matching, this approach captures the logical consistency and structural coherence required for multi-step reasoning.

All evaluations use fixed prompt sets comprising 50 representative samples selected from the CoT dataset. To ensure consistency, we employ a standard zero-shot-CoT template by appending the trigger phrase "Let's think step by step" to each query, which effectively activates the reasoning capabilities of small-scale LLMs. Consistent batch sizes of one are maintained throughout all tests to simulate real-time, single-user edge inference scenarios.

Each experiment is repeated 15 times to assess stability. We report the median value across these 15 runs for all performance metrics to provide a robust assessment that is less sensitive to potential outliers caused by transient system spikes. Both CPU and GPU back-ends are tested where applicable, allowing us to compare inference behavior under different compute modes.

D. Fine-Grained Analysis on Edge Devices

To gain deeper insights into LLM runtime behavior on constrained hardware, we conduct a detailed profiling study across multiple platforms: Jetson Orin NX, Jetson AGX Orin, and a

desktop-class edge server with a discrete GPU, representing diverse compute and memory bandwidth capabilities.

Our profiling process is twofold.

NVIDIA Nsight Systems. Nsys [34] captures kernel-level timelines of the inference process, providing a high-level view of GPU and CPU resource utilization. This allows us to identify time-consuming stages (e.g., token embedding, attention) and inefficiencies, such as idle periods or excessive memory transfer overhead. All selected models are analyzed to provide a comparative view of runtime behavior.

NVIDIA Nsight Compute. Ncu [35] performs micro-architectural profiling on identified kernel bottlenecks. This tool provides detailed metrics on low-level hardware characteristics, allowing us to analyze warp occupancy and shared memory utilization as well as global memory access patterns, and cache utilization. This detailed profiling helps pinpoint inefficiencies, such as suboptimal memory access, and assess hardware impact on overall inference performance.

For this deep-dive analysis, we select Qwen-1.5B and Gemma-2B as representative models due to their different architectural traits and performance profiles. Together, these tools and methods enable a deep understanding of how LLMs behave on different edge devices and inform optimization strategies for future deployment.

IV. Benchmark Results and Analysis

This section presents and analyzes the experimental results obtained from evaluating various small-scale Transformer-based LLMs across different hardware platforms. We organize our findings around the key metrics: model load time, token-level latency (TPOT and TTFT), memory usage, power dissipation, and output quality (relevance score).

A. Load Time Comparison

We first report the model loading times across platforms and inference frameworks as shown in Fig. 2.

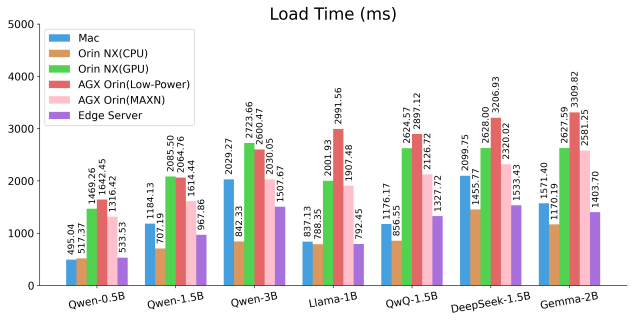

Fig. 2: Average Model Load Time (ms) across Models and Platforms

We observe four primary trends regarding model initialization across platforms.

1. GPU vs. CPU Backend Overhead. On dual-option devices, GPU backend model loading generally takes longer than on CPU. For instance, Qwen-1.5B load time on Orin NX increased by over 195% (from 707 ms on CPU to 2086 ms on GPU), and Gemma-2B by about 124% (from 1170 ms on CPU to 2628 ms on GPU). This delay is likely due to the additional overhead from initializing GPU contexts, memory allocation, and kernel preparation.

2. Architectural and Quantization Influence. Specifically, BF16 precision models load more slowly than Q8 models with similar structure, owing to increased weight size and higher memory bandwidth requirements. For example, on Orin NX (CPU), QwQ-1.5B loads in 857 ms longer than Qwen-1.5B at 707 ms. Additionally, architectural complexity further amplifies latency. This is evident as Gemma-2B (1170 ms on Orin NX CPU) generally exhibits longer load times compared to the Qwen series (e.g., Qwen-3B at 842 ms on Orin NX CPU), likely due to their more resource-intensive designs.

3. Power Mode Impact on Initialization. On the Jetson AGX Orin, low-power mode (30W) loading is substantially slower than MAXN mode, sometimes exceeding the load time of the smaller Orin NX board. For instance, Gemma-2B loads in 3207 ms in low-power mode vs. 2320 ms in MAXN mode (1.4 times slower). Similarly, Llama-1B's default low-power load time (2992 ms) exceeds the Orin NX GPU time (2002 ms), highlighting the critical role of power mode selection for time-sensitive edge deployments.

B. Token-Level Inference Latency

Fig. 3 shows the average time-per-output-token (TPOT) across models and platforms.

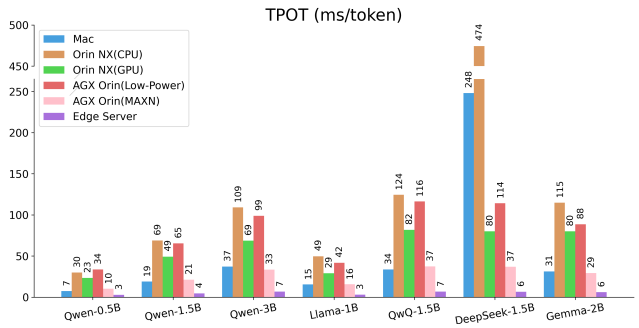

Fig. 3: TPOT (ms/token) across Models and Platforms

Key observations from our findings include:

1. GPU Acceleration for Token Generation. Within the same device, using a GPU backend significantly reduces TPOT compared to CPU-only execution. This effect becomes more pronounced as the model size increases: Qwen-3B sees its TPOT drop from 109 ms/token on CPU to 69 ms/token on GPU, representing a 36% improvement. It shows larger models benefit more from parallelized tensor operations and hardware acceleration, which are not efficiently supported by CPUs.

2. Model Scaling on CPU-Only Platforms. On CPU-only platforms such as MacBook, larger models suffer from longer TPOT. For example, Qwen-3B exhibits a TPOT of 37 ms/token, which is over 5 times longer than Qwen-0.5B (7 ms/token). This increase is primarily due to large models involving more layers and parameters and CPUs lacking

the massive parallelism required to process large transformer blocks efficiently.

3. Architectural, Quantization and Power Mode Impacts. Consistent with the observations in model load time, differences in model structure and numerical precision also lead to significant performance variations in TPOT. For instance, Gemma-2B's 80 ms/token on Orin NX (GPU) is about 1.6 times higher than Qwen-1.5B's 49 ms/token. This can be attributed to architectural choices such as deeper layers or wider attention mechanisms, which introduce more computation per token.

Additionally, models running in BF16 precision tend to have longer TPOT than Q8 models with similar structure. For example, on Orin NX (GPU), QwQ-1.5B (82 ms/token) demonstrates a 67% improvement over Qwen-1.5B (49 ms/token). This is due to increased memory bandwidth demands and reduced compute efficiency on some edge hardware.

Meanwhile, the AGX Orin in low-power mode (30W) shows noticeably worse TPOT than in MAXN mode. For instance, DeepSeek-1.5B experiences a TPOT of 114 ms/token in low-power mode compared to 37 ms/token in MAXN mode, making it over 3 times slower. For some models like Qwen-0.5B (34 ms/token), the TPOT in AGX Orin's low-power mode is indeed slower than on the smaller Orin NX board (e.g., 23 ms/token for Qwen-0.5B on Orin NX GPU), highlighting the impact of power mode on inference speed.

As for another important token-level inference latency, Fig. 4 shows the average Time To First Token (TTFT).

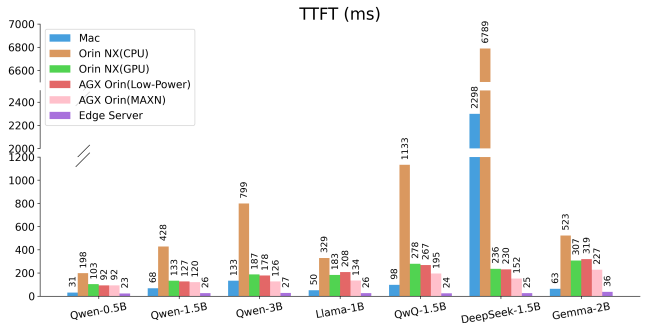

Fig. 4: TTFT (ms) across Models and Platforms

TTFT trends reveal conclusions similar to TPOT, such as platforms with GPU and MAXN power mode offer faster response times, but the extreme latency on CPU platforms highlights a distinct performance bottleneck.

1. Latency on CPU-Constrained Platforms. The TTFT for larger models on CPU-constrained platforms reveals the severe bottleneck. For instance, on the Orin NX (CPU) platform, DeepSeek-1.5B's TTFT reaches 6789 ms, which is 28 times larger than the time on Orin NX (GPU). This extreme latency is primarily caused by the necessity of moving large model weights through system RAM and the CPU, which lacks the necessary memory parallelism for efficient prompt evaluation.

Architectural choice is also critical in this CPU bottleneck. With a 428 ms TTFT on Orin NX (CPU), Qwen-1.5B is over 15 times faster than DeepSeek-1.5B on the same CPU-limited device. This sharp contrast suggests Qwen's architecture is better optimized for resource-constrained host CPUs. This generalized CPU constraint is also seen on the MacBook, where DeepSeek-1.5B requires 2298 ms for the first token.

C. Memory Usage

Memory consumption on both CPU and GPU was monitored during inference as illustrated in Fig. 5 and Fig. 6. Since the MacBook lacks a discrete GPU, GPU memory usage is not monitored on this device.

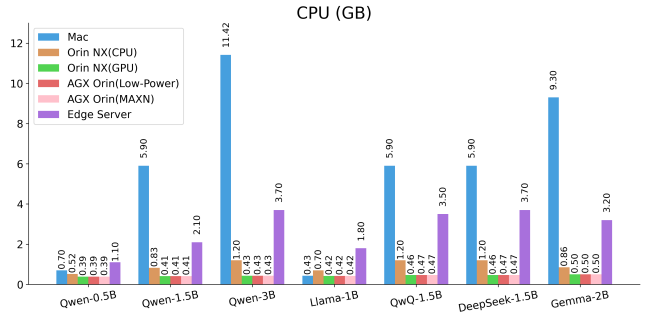

Fig. 5: CPU Memory Usage (GB) across Models and Platforms

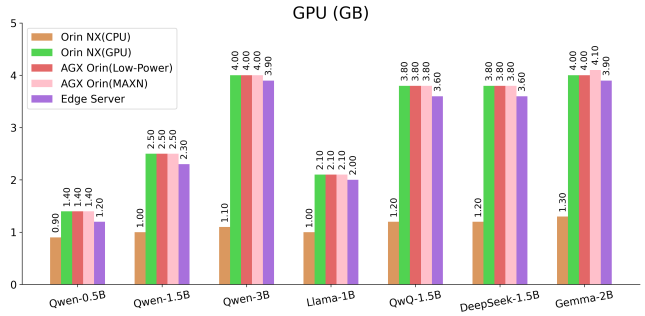

Fig. 6: GPU Memory Usage (GB) across Models and Platforms

We summarize the following key observations:

1. Memory Bottlenecks on CPU-Only Platforms. On the MacBook, the increase in CPU memory usage is particularly pronounced as model size scales up. For instance, running Qwen-3B requires 11.42 GB RAM, a substantial load compared to Qwen-0.5B (0.70 GB) or Qwen-1.5B (5.90 GB). This excessive memory load likely contributes to the poor performance observed in both load time and token processing time on the MacBook.

2. Backend-Specific Memory Allocation and Cross-Platform Differences. For dual-backend devices, we observe that CPU back-end usage results in higher CPU memory usage, while GPU memory remains largely unused. Conversely, using the GPU back-end shifts most of the memory load to the GPU, reducing CPU memory usage. For Qwen-3B on Orin NX, CPU memory drops 64% (from 1.20 GB CPU backend to

0.43 GB GPU backend), while GPU memory rises from 1.10 GB to 4.00 GB.

Interestingly, despite similar overall GPU usage, the edge server exhibits higher CPU memory utilization during GPU inference compared to Jetson series devices. For instance, Qwen-3B on the Edge Server consumes 3.70 GB of CPU memory during GPU inference, roughly 8.6 times more than on the AGX Orin at 0.43 GB. This suggests that the edge system may offload additional pre-processing tasks to the CPU, potentially leveraging a more optimized memory pipeline.

3. Power Mode and Memory Consumption. Comparing AGX Orin in low-power (30W) and MAXN modes, we find that overall memory usage remains similar. For example, Qwen-3B consumes 0.43 GB of CPU memory and 4.00 GB of GPU memory in both modes. However, under MAXN mode, Gemma models tend to use slightly more GPU memory (4.10 GB in MAXN vs 4.00 GB in Low-Power Mode), possibly due to higher kernel-level parallelism and deeper memory pipelines activated in the high-performance setting.

D. Power Dissipation

Fig. 7 presents the average power dissipation on NVIDIA Jetson platforms. Our power analysis specifically focuses on Jetson because they are widely adopted development platforms for edge AI applications, where power efficiency is a paramount concern directly impacting long-term deployment sustainability in low-power operational environments.

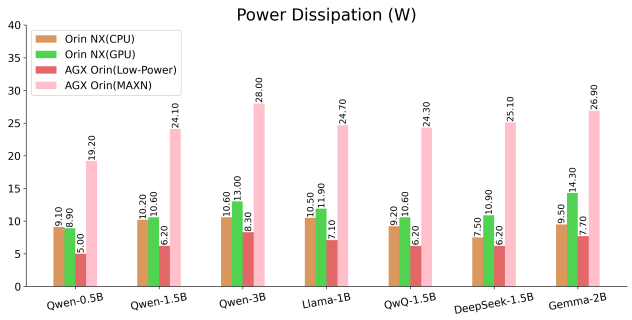

Fig. 7: Power Dissipation (W) across Jetson Platforms

We draw the following conclusions:

1. Model-Specific Power Characteristics. Power dissipation patterns are also influenced by model size and architectural design. Generally, within the same model family, larger models tend to exhibit higher power demands; for example, Qwen-3B at 28.00 W is considerably more than Qwen-0.5B at 19.20 W in the same mode. However, architectural differences cause varied consumption even for similar-sized models. For instance, Llama-1B consumes 24.70 W in AGX Orin MAXN mode, which is slightly higher than Qwen-1.5B's 24.10 W in the same mode.

2. CPU vs. GPU Power Consumption on Orin NX. On the Orin NX platform, GPU-accelerated inference generally results in slightly higher power consumption compared to CPU-only execution for the same model. For Qwen-3B, GPU execution consumes 13.00 W, while CPU-only consumes 10.60 W, an increase of 22.6%. This indicates that while GPU acceleration provides substantial throughput gains, it often comes with a marginal increase in power dissipation on the edge platforms.

3. Impact of Power Modes on AGX Orin. As expected, the AGX Orin's high-performance MAXN mode consistently exhibits the highest power dissipation. For example, DeepSeek-1.5B in MAXN mode consumes 25.10 W, compared to only 6.20 W in low-power mode, representing a 75.3% reduction. This contrast highlights the direct trade-off between energy budget and peak performance on a single device.

E. Output Quality: Relevance Scores

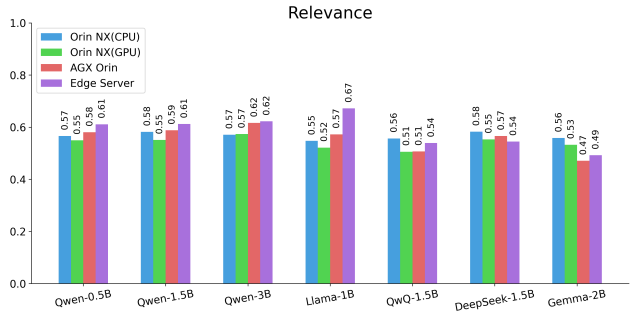

Fig. 8: Relevance Score Comparison across Models and Platforms

Across all platforms, we observe that the relevance scores, measuring the alignment between model output and the CoT (Chain-of-Thought) dataset [40], are relatively stable across different devices for the same model as shown in Fig. 8. For instance, Qwen-0.5B's relevance scores range only from 0.55 (Orin NX GPU) to 0.61 (Edge Server), showing minimal variation. This suggests that the underlying hardware has minimal impact on model reasoning quality.

As model size of the same architecture increases, relevance scores show a mild upward trend, indicating improved reasoning ability and better alignment with human-annotated answers. For example, on AGX Orin, Qwen-0.5B scores 0.58, while Qwen-3B scores 0.62. However, the improvement is not pronounced, with typical gains in relevance scores being only around 0.02 to 0.04 points between model sizes. This result may be attributed to the relatively small parameter scale of the models evaluated (up to 3B), which doesn't fully capture the emergence of complex reasoning capabilities observed in larger language models.

These results show that bigger models perform better in reasoning tasks, but the gains are small at this scale. Further experiments with larger models could help determine whether this trend becomes more significant at higher capacities.

V. Operator-Level Profiling

As shown in Fig. 9, the kernel-level profiling results consistently show that **attention-related operations, particularly matrix-matrix multiplications (GEMM), dominate**

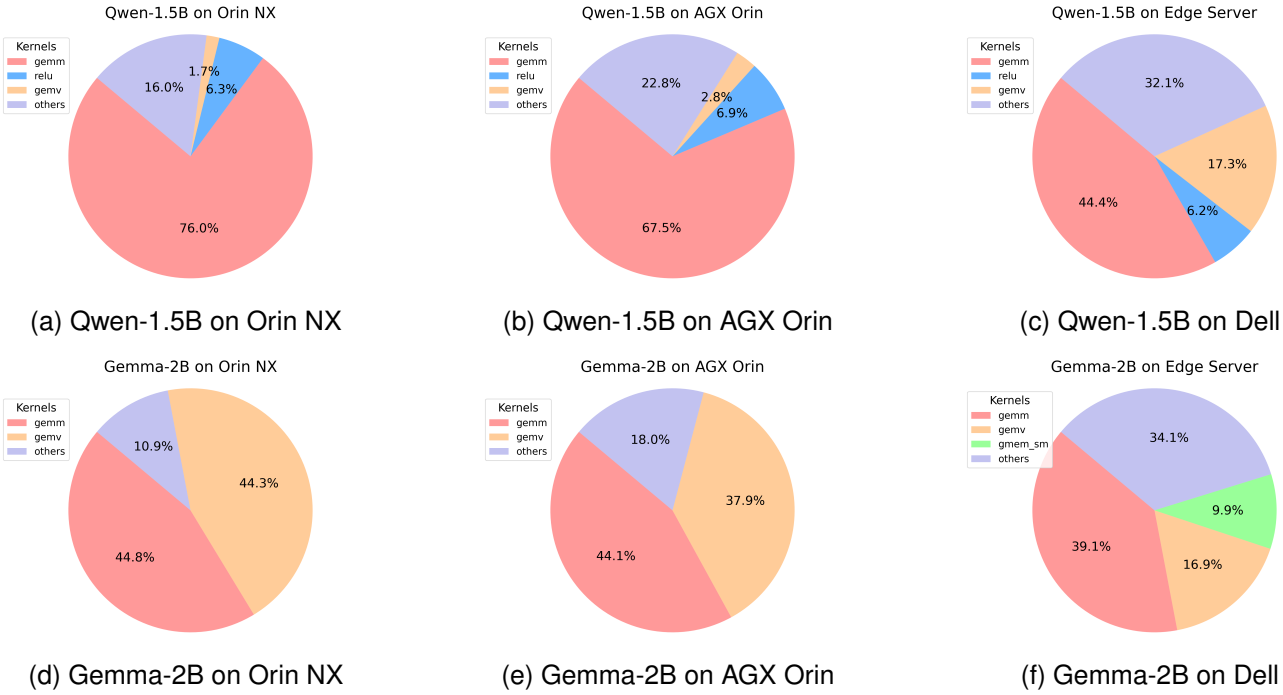

Fig. 9: Operator-Level Kernel Time Distribution Across Devices

execution time across devices and models. For example, GEMM accounts for a significant portion of execution time: 76.0% for Qwen-1.5B on Orin NX, 67.5% on AGX Orin, and for Gemma-2B, GEMM consumes 44.8% on Orin NX. This is expected due to GEMM's central role in Transformer architectures and its scaling with model size. These two models reflect the common trends observed across the entire model set.

Notably, we observe a platform-dependent shift in kernel-level composition, offering crucial cross-platform insights:

1. Bottleneck Shifting on Powerful vs. Less Powerful GPUs. On devices with more powerful GPUs, the proportional runtime spent on computationally intensive GEMM operations decreases. Conversely, the relative cost of other finer-grained kernels like element-wise operations becomes more prominent. For Qwen-1.5B, the GEMM share drops from 76.0% on Orin NX to 44.4% on Edge Server, while the share of small operations rises from 16.0% to 32.1%. This suggests that as compute capacity increases, performance bottlenecks shift from large matrix multiplications to more granular operations including layer normalization, activations, which become the new primary bottlenecks.

2. Limited Absolute Impact of Fine-grained Operations. Despite their rising relative prominence on powerful hardware, the absolute impact of these smaller kernels remains modest across all platforms. Our high-level metrics (e.g., TPOT) show performance plateauing rather than drastic slowdowns. This indicates that element-wise operations, while more visible in profiling, contribute only a small fraction of total latency and are unlikely to become dominant bottlenecks in diverse edge environments.

3. Model-Specific Profiles Exhibiting Overall Cross-Platform Trends. When analyzing individual model profiles across platforms, Gemma-2B exhibits a distinct performance signature compared to Qwen-1.5B, showing higher average latency per kernel, including a larger share of GEMV (matrix-vector multiplication) operations. For instance, on Orin NX, Gemma-2B's GEMM share is 44.8%, while Qwen-1.5B's is 76.0%. This difference likely stems from model architecture or framework implementation. Nevertheless, the overall operator-level trends remain remarkably consistent across all platforms and models, reinforcing the need to holistically optimize both dominant (GEMM) and long-tail (element-wise) kernels for maximum LLM efficiency on diverse edge hardware.

To further understand how platform-specific constraints impact kernel-level performance, we conduct a comparative analysis among Orin NX, AGX Orin and the edge server workstation. While the overall kernel composition across platforms follows similar trends, profiling on Jetson devices reveals several architectural limitations that constrain inference efficiency in resource-constrained edge environments.

Detailed analysis using Nsight Compute (ncu) exposes a range of hardware-level inefficiencies on Jetson devices:

Insight 1. Underutilized Compute and Instruction Pipelines. We consistently observe that GPU compute pipelines are significantly under-utilized, and instruction issuance rates are suboptimal (e.g., some schedulers only issuing an instruction every 5.4 cycles despite being capable of one per cycle). This indicates a fundamental lack of sufficient computational work or inefficient instruction-level parallelism

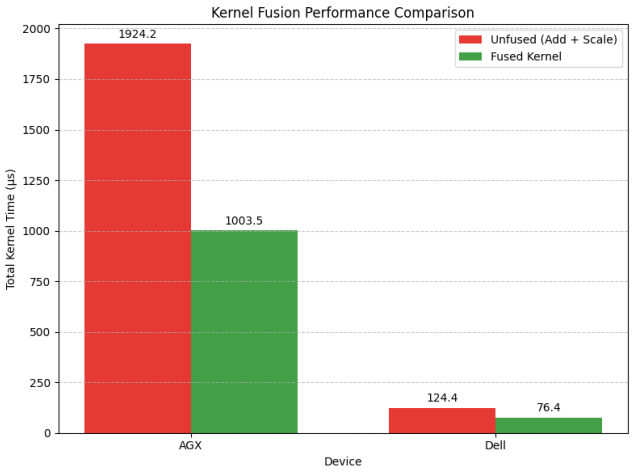

Fig. 10: Kernel Fusion Performance Comparison

and serialization within the kernels, directly limiting overall throughput.

Insight 2. Memory Access Bottlenecks. Memory access patterns for global loads in the L1/TEX cache (the unified L1 cache and texture cache on NVIDIA GPUs, serving as the primary buffer for global memory reads) are suboptimal. This is likely due to uncoalesced or misaligned accesses, which significantly increases the number of memory sectors loaded per access and degrades memory throughput, thus starving the compute units.

Insight 3. Limited Warp-Level Parallelism and Eligibility. Warp-level execution, where warps (groups of 32 parallel threads) are the fundamental execution units on NVIDIA GPUs, is severely constrained by low theoretical occupancy and poor runtime warp eligibility. Specifically, theoretical occupancy is often very low (e.g., 16.7%) due to **excessive shared memory usage**, which restricts the number of concurrent thread blocks. This, combined with an average of only 0.11 eligible warps per cycle (out of 1.95 active warps per scheduler), means the GPU frequently finds few warps ready to issue instructions. This directly contributes to the underutilization of compute pipelines (Insight 1) and hinders latency hiding, resulting in severely limited GPU utilization and overall performance degradation.

These findings highlight the mismatch between certain kernel configurations and Jetson's memory and compute architecture, especially when deploying LLMs originally designed for server-grade GPUs. They underscore **the importance of not only optimizing dominant kernels like GEMM, but also rethinking how small kernels are scheduled and launched on low-power devices.** Addressing these inefficiencies through techniques such as kernel fusion, shared memory reuse, or restructured operator logic could significantly improve the performance and energy efficiency of edge LLM inference.

To validate the efficacy of kernel fusion, we conduct a simple experiment comparing unfused (Add + Scale) operations against a fused kernel on both the Jetson AGX (Edge) and Dell (General-Purpose) platforms.

As shown in Fig. 10, kernel fusion yields significant performance gains across both devices:

- Jetson AGX Orin (Edge): The fused kernel reduces the total execution time from 1,924.2 μs to 1,003.5 μs, achieving a 47.8% reduction in latency.
- Dell (General-Purpose GPU): The execution time of fused kernel drops from 124.4 μs to 76.4 μs, resulting in a 38.6% reduction.

This evidence confirms that fusion is a powerful and portable method for overcoming inefficiencies caused by low compute-to-memory ratios and maximizing the efficiency of underutilized warps on diverse GPU hardware.

In addition to cross-platform analysis, we further compared Gemma-2B and Qwen-1.5B to investigate architectural differences in kernel behavior. On Jetson platforms, we observe that Gemma exhibits the same limitations as the edge server system but with more pronounced imbalances in resource utilization.

Specifically, memory subsystems are more heavily utilized than compute pipelines during Gemma's inference execution. This suggests that **memory throughput, rather than compute capacity, becomes the primary bottleneck.** Nsight Compute reports indicate that global memory traffic dominates kernel execution, with low compute-to-memory ratio and relatively sparse instruction issue rates. Such patterns point to inefficient memory-bound kernel configurations, where limited coalescing, irregular tensor shapes, or excessive activation caching could throttle performance—especially on Jetson's lower-bandwidth memory interfaces.

This finding is consistent with Gemma's higher latency and memory usage observed in high-level metrics, and emphasizes the need for architecture-aware deployment decisions. In contrast, Qwen's kernels demonstrate more balanced compute and memory usage across platforms, indicating better hardware fit and more efficient execution on edge GPUs.

Overall, these kernel-level observations reveal that edge deployment challenges are not solely tied to compute limitations, but also to how well a model's architecture and memory access behavior align with the hardware's execution model. Fine-grained profiling provides critical insights that go beyond aggregate metrics, enabling developers to identify bottlenecks that are otherwise invisible. Such understanding is essential for guiding low-level optimizations, informing model selection, and ultimately achieving robust and efficient LLM inference on diverse edge platforms.

VI. Discussion and Implications for Future Research

A. Discussion

The experimental results and data in section4 and section 5 provide valuable insights into the deployment of small-scale LLMs on edge and desktop platforms, with several key observations emerging from the analysis.

At a high level, GPU-based inference generally introduces longer load times than CPU due to device initialization and

memory allocation overhead. Despite this, GPU inference outperforms CPU in token processing time, especially for larger models, confirming that stronger parallelism accelerates computation. This is most evident on the edge server, which provides low TPOT and TTFT across all models, inherently reflecting the impact of varying numerical precisions.

Memory analysis reveals platform-specific trends. On MacBook, CPU memory usage grows disproportionately with model size, likely explaining its degraded performance. On hybrid platforms, using the GPU back-end significantly reduces CPU memory load, suggesting efficient migration of compute to the accelerator. However, the edge server utilizes more CPU during GPU inference than the Jetson series, indicating potential for resource scheduling optimization.

Our analysis also delves into power dissipation on NVIDIA Jetson platforms. Consistent with design intent, the AGX Orin in MAXN mode exhibits the highest power consumption, while its low-power configuration significantly conserves energy. On Orin NX, GPU acceleration results in a marginal power consumption increase compared to CPU-only. This highlights the fundamental trade-off between power budget and peak performance, noting that architectural design often influences power profiles more than parameter count alone.

In the deeper analysis, our operator-level profiling highlights the dominant role of attention kernels like GEMM in runtime cost. The broader value of this analysis is in informing deployment strategies and model-system co-design, revealing the underlying mismatch between model architecture and edge hardware—insights unavailable from high-level metrics alone.

The comparison between Gemma-2B and Qwen-1.5B illustrates how architectural choices affect hardware utilization. On Jetson, Gemma is memory-bound, while Qwen yields more balanced execution, which means It is better suited for edge deployment. This imbalance confirms that edge inefficiency is tied to operator granularity, not just limited compute. Such insights enable platform-aware model selection and motivate low-level optimizations.

Moreover, our cross-platform profiling demonstrates that the bottleneck shifts with hardware capability. On high-end GPUs, time on dominant operations like GEMM is reduced, while smaller, element-wise kernels become more prominent. This requires targeted and adaptive optimization: reducing GEMM latency is critical for entry-level GPUs, but optimizing long-tail operators gains importance as compute capacity increases. Nevertheless, since edge devices typically feature weak or bandwidth-constrained GPUs, special attention must still be paid to optimizing matrix-heavy computations like GEMM, as they remain the dominant latency source in most constrained transformer models.

While our study focuses on specific lightweight LLMs, the core principles and methodologies are applicable to wider transformer architectures. Performance bottlenecks like the CPU/GPU loading trade-off and GEMM dominance are not unique to Qwen and Gemma; they arise from the transformer's fundamental reliance on large matrix multiplications and attention mechanisms. Therefore, our profiling techniques and insights can be extended to other LLM models. Our methodology also serves as a practical guide to quickly diagnose and optimize performance on other LLMs and similar hardware.

Meanwhile, we acknowledge two limitations. First, our study is primarily diagnostic, not prescriptive. We analyze performance characteristics and suggest optimization ideas like GEMM kernel fusion as directions for future work, rather than concrete, implemented solutions. Second, our experiments rely solely on the llama.cpp inference framework. We chose this for consistent environment and hardware-specific bottleneck isolation. However, this means our findings do not reflect the full range of optimizations available in other leading frameworks. We plan to conduct a comparative study with other backends, such as TensorRT, for a more comprehensive future analysis.

B. Implications for Future Research

These findings suggest several directions for future optimization work:

1. Exploration of Model and Operator-Level Optimizations. While this study utilizes 8-bit quantization as a baseline, future work should explore more aggressive low-bit techniques (e.g., 4-bit or 2-bit quantization, AWQ [47], and SmoothQuant [48]) to further alleviate memory bandwidth bottlenecks on edge devices. Beyond traditional dense models, investigating the deployment of Mixture-of-Experts (MoE) architectures on the edge presents a promising yet challenging direction, as MoE can reduce active parameters during inference but imposes higher demands on memory capacity to store the sparse weights.

Further reducing model size via structured pruning or quantization-aware training [42] can significantly improve load and inference times, directly alleviating the memory-pressure bottlenecks we identified for larger models like Gemma-2B. Concurrently, our profiling results suggest that improving core kernel efficiency is vital, particularly for GEMM under low parallelism. Specifically, to address the underutilization and instruction-issue stalls observed on Jetson devices, we propose adopting techniques such as kernel fusion, shared memory tuning, and better tensor alignment, all of which can significantly enhance execution throughput [44].

2. Profiling-Guided and Adaptive Deployment Strategies. Future work could explore automated pipelines that integrate operator-level profiling feedback into model selection, back-end scheduling, and inference graph restructuring to better match edge device constraints. This could include dynamic switching between CPU and GPU back-ends based on runtime conditions to optimize both performance and energy efficiency [43] (e.g., offloading lightweight pre-processing tasks to the CPU while reserving GPU resources for parallel execution of large model components).

3. Expanded Scope: Broader Model and Hardware Coverage. Expanding experiments to include a wider range of models (e.g., larger models or those optimized for mobile) and exploring support for new and evolving hardware platforms (e.g., RISC-V [45], NPUs) would provide more insights. These

platforms may offer improved energy efficiency or specialized instruction sets better suited for LLM inference on edge devices, and studying their interaction with scaling model size is crucial for understanding edge device performance and effective optimizations.

Overall, this work demonstrates the feasibility of running small LLMs on diverse platforms and highlights both the promise and challenges of edge deployment. The insights gained form a foundation for continued exploration of efficient, portable, and high-quality LLM inference on various hardware.

VII. Conclusion

With Large Language Models (LLMs) gaining significant traction across diverse real-world contexts, ensuring their high-performance inference capabilities on edge devices presents a critical challenge. This paper comprehensively evaluates lightweight LLMs on diverse edge and general-purpose platforms like Apple M4 CPUs, CUDA GPUs, and NVIDIA Jetson Orin. Using llama.cpp and PyTorch, we analyzed deployment feasibility, runtime performance, and efficiency through metrics like load time, TPOT, TTFT, memory usage, and power dissipation. Our findings reveal that attention-related operations (e.g., GEMM) dominate runtime, with significant variations in compute utilization and memory efficiency, particularly Gemma-2B's memory-bound behavior on Jetson platforms. These identified bottlenecks underscore the necessity for targeted, profiling-guided, and operator-level optimizations tailored to edge constraints. Overall, our work offers quantitative insights and practical guidelines for LLM deployment on resource-constrained platforms, with future efforts focusing on power profiling and emerging hardware back-ends.

VIII. Acknowledgement

The authors would like to thank all the reviewers for their helpful comments and Mr. Zhikun Dong for his support during the early experimental phase. This research was partly supported by the National Natural Science Foundation of China under Grant Nos. 62402475 and U24A20232. The corresponding author is Xingzhou Zhang (*zhangxingzhou@ict.ac.cn*).

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[5] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale," *arXiv preprint arXiv:2208.07339*, 2022. [Online]. Available: https://arxiv.org/abs/2208.07339

[6] T. Zhao et al., "A Survey of Deep Learning on Mobile Devices: Applications, Optimizations, Challenges, and Research Opportunities," in Proceedings of the IEEE, vol. 110, no. 3, pp. 334-354, March 2022, doi: 10.1109/JPROC.2022.3153408.

[7] M. Zhang, F. Zhang, N. D. Lane, Y. Shu, X. Zeng, B. Fang, *et al.*, "Deep learning in the era of edge computing: Challenges and opportunities," *arXiv preprint arXiv:2010.08861*, 2020. [Online]. Available: https://arxiv.org/abs/2010.08861

[8] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," *arXiv preprint arXiv:2205.14135*, 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[9] OpenAI, "Introducing ChatGPT," OpenAI Website, Nov. 30, 2022. [Online]. Available: https://openai.com/index/chatgpt/.

[10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[11] DeepSeek-AI, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," *arXiv preprint arXiv:2501.12948*, 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[12] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024. [Online]. Available: https://arxiv.org/abs/2407.10671

[13] Gemma Team, "Gemma," Kaggle, DOI: 10.34740/KAGGLE/M/3301, 2024. [Online]. Available: https://www.kaggle.com/m/3301

[14] S. Laskaridis, K. Katevas, L. Minto, and H. Haddadi, "MELTing point: Mobile evaluation of language transformers," *arXiv preprint arXiv:2403.12844*, 2024. [Online]. Available: https://arxiv.org/abs/2403.12844

[15] A. Saini, O. B. Shende, M. K. Pandit, R. Sen, and G. Ananthanarayanan, "Bang for the buck: Evaluating the cost-effectiveness of heterogeneous edge platforms for neural network workloads," in *2023 IEEE/ACM Symposium on Edge Computing (SEC)*, Wilmington, DE, USA, 2023, pp. 94–107, doi: 10.1145/3583740.3628437.

[16] A. Atrey, R. Sinha, S. Mitra, and P. Shenoy, "SODA: Protecting proprietary information in on-device machine learning models," in *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing*, Dec. 2023, pp. 121–132, doi: 10.1145/3583740.3626617.

[17] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *arXiv preprint arXiv:2009.06732*, 2020. [Online]. Available: https://arxiv.org/abs/2009.06732

[18] Z. Dong and X. Zhang, "Evaluating large language models on the edge based on evaluatology," in BenchCouncil International Symposium on Intelligent Computers, Algorithms, and Applications, pp. 199–212, Springer, 2024.

[19] A. Agrawal, A. Agarwal, N. Kedia, J. Mohan, S. Kundu, N. Kwatra, *et al.*, "Etalon: Holistic performance evaluation framework for LLM inference systems," *arXiv preprint arXiv:2407.07000*, 2024. [Online]. Available: https://arxiv.org/abs/2407.07000

[20] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, *et al.*, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," *arXiv preprint arXiv:2401.09670*, 2024. [Online]. Available: https://arxiv.org/abs/2401.09670

[21] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024. [Online]. Available: https://arxiv.org/abs/2404.14294

[22] Z. Nezami, M. Hafeez, K. Djemame, S. A. R. Zaidi and J. Xu, "Descriptor: Benchmark Dataset for Generative AI on Edge Devices (BeDGED)," in IEEE Data Descriptions, vol. 2, pp. 50-55, 2025, doi: 10.1109/IEEEDATA.2025.3552083.

[23] M. Arya and Y. Simmhan, "A Preliminary Performance Analysis of LLM Inference on Edge Accelerators," 2024 IEEE 31st International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Bangalore, India, 2024, pp. 183-184, doi: 10.1109/HiPCW63042.2024.00069.

[24] R. Qin, D. Liu, C. Xu, Z. Yan, Z. Tan, Z. Jia, *et al.*, "Empirical guidelines for deploying LLMs onto resource-constrained edge devices," *arXiv preprint arXiv:2406.03777*, 2024. [Online]. Available: https://arxiv.org/abs/2406.03777

[25] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, "A review on edge large language models: Design, execution, and applications," *arXiv*

preprint arXiv:2410.11845, 2025. [Online]. Available: https://arxiv.org/abs/2410.11845

[26] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, *et al.*, "LLM in a flash: Efficient large language model inference with limited memory," *arXiv preprint arXiv:2312.11514*, 2023. [Online]. Available: https://arxiv.org/abs/2312.11514

[27] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, *et al.*, "AI benchmark: Running deep neural networks on Android smartphones," *arXiv preprint arXiv:1810.01109*, 2018. [Online]. Available: https://arxiv.org/abs/1810.01109

[28] W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. Ragan Kelly, "Reducing transformer key-value cache size with cross-layer attention," *arXiv preprint arXiv:2405.12981*, 2024. [Online]. Available: https://arxiv.org/abs/2405.12981

[29] P. Vellaisamy, T. Labonte, S. Chakraborty, M. Turner, S. Sury, and J. P. Shen, "Characterizing and optimizing LLM inference workloads on CPU-GPU coupled architectures," *arXiv preprint arXiv:2504.11750*, 2025. [Online]. Available: https://arxiv.org/abs/2504.11750

[30] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, *et al.*, "AlpaServe: Statistical multiplexing with model parallelism for deep learning serving," *arXiv preprint arXiv:2302.11665*, 2023. [Online]. Available: https://arxiv.org/abs/2302.11665

[31] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, *et al.*, "Efficient memory management for large language model serving with PagedAttention," *arXiv preprint arXiv:2309.06180*, 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[32] ggml-org, "llama.cpp," [Online]. Available: https://github.com/ggml-org/llama.cpp.

[33] PyTorch, "PyTorch," [Online]. Available: https://pytorch.org/.

[34] NVIDIA, "Nsight Systems," [Online]. Available: https://developer.nvidia.com/nsight-systems.

[35] NVIDIA, "Nsight Compute," [Online]. Available: https://developer.nvidia.com/nsight-compute.

[36] NVIDIA, "Jetson Orin," [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.

[37] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, *et al.*, "FlexGen: High-throughput generative inference of large language models with a single GPU," *arXiv preprint arXiv:2303.06865*, 2023. [Online]. Available: https://arxiv.org/abs/2303.06865

[38] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, *et al.*, "Alpa: Automating inter- and intra-operator parallelism for distributed deep learning," in *16th USENIX Symp. Oper. Syst. Design Implement. (OSDI '22)*, Carlsbad, CA, USA, Jul. 2022, pp. 1097–1115, doi: 10.5555/3572887.3572996.

[39] X. Zhang, Y. Wang, S. Lu, L. Liu, W. Shi, *et al.*, "OpenEI: An open framework for edge intelligence," in *2019 IEEE 39th Int. Conf. Distributed Computing Systems (ICDCS)*, Dallas, TX, USA, Jul. 2019, pp. 782–792, doi: 10.1109/ICDCS.2019.00078.

[40] YorickHe, "CoT," ModelScope, [Dataset]. Available: https://www.modelscope.cn/datasets/YorickHe/CoT.git. [Accessed: Jun. 18, 2025].

[41] V. Shankar, "Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges," 2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET), Ghaziabad, India, 2024, pp. 1-6, doi: 10.1109/ACET61898.2024.10730112.

[42] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, *et al.*, "Q-BERT: Hessian based ultra low precision quantization of BERT," *arXiv preprint arXiv:1909.05840*, 2019. [Online]. Available: https://arxiv.org/abs/1909.05840

[43] A. Adeyemo, T. Sandefur, T. A. Odetola, and S. R. Hasan, "Towards enabling dynamic convolution neural network inference for edge intelligence," *arXiv preprint arXiv:2202.09461*, 2022. [Online]. Available: https://arxiv.org/abs/2202.09461

[44] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," *arXiv preprint arXiv:1802.04799*, 2018. [Online]. Available: https://arxiv.org/abs/1802.04799

[45] I. Mukhin, Y. Rodimkov, E. Vasiliev, V. Volokitin, A. Sidorova, E. Kozinov, *et al.*, "Benchmarking deep learning inference on RISC-V CPUs," in *Supercomputing (RuSCDays 2024)*, Lecture Notes in Computer Science, vol. 15406. Cham, Switzerland: Springer, 2025, pp. 331–346, doi: 10.1007/978-3-031-76634-1_23.

[46] E. Kwon, S. Han, Y. Park, J. Yoon and S. Kang, "Reinforcement Learning-Based Power Management Policy for Mobile Device Systems," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 10, pp. 4156-4169, Oct. 2021, doi: 10.1109/TCSI.2021.3103503.

[47] J. Lin, J. Tang, H. Tang, S. Yang, W. M. Chen, W. C. Wang, G. Xiao, X. Dang, C. Gan and S. Han, "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration," in Proceedings of Machine Learning and Systems (MLSys), vol. 6, 2024.

[48] 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 Proceedings of the 40th International Conference on Machine Learning (ICML), vol. 202, pp. 38087-38099, 2023.