Quantization and Parallel Inference for Large Language Models on Edge Devices

[1]Dr. N Priya, [2]Dr. V Keerthika, [3]Ana Dorthy L, [4]Mounikha K V,

[1]Assistant Professor, [2]Assistant Professor, [3]Student, [4]Student,
Department of Computing - Software Systems,
Coimbatore Institute of Technology, Coimbatore, India
npriya@cit.edu.in, vkeerthika@cit.edu.in, 71762131007@cit.edu.in , 71762131030@cit.edu.in

Abstract— **Large language models (LLMs) require high computational power and memory, which has limited their execution to cloud-based environments. This relies on remote infrastructure; however, it introduces challenges such as latency, privacy risks, and dependence on stable connectivity. This work examines the feasibility of performing LLM inference directly on consumer-grade edge devices by combining model compression through quantization with parallel inference on available hardware accelerators. Using the TinyLlama-1.1B model in Q4_K_M format, performance can be evaluated on two representative platforms: a MacBook Air M2 with GPU acceleration via Metal, and an Android smartphone utilizing CPU NEON vectorization. Results show that quantization substantially reduces memory requirements, while parallel execution enables interactive throughput of approximately 96 tokens per second on laptop and 46 tokens per second on smartphone. The analysis further indicates that the autoregressive decoding stage continues to be the important feature that influences the performance. In contrast to earlier studies that explored quantization and parallelism as separate strategies, this study offers empirical evidence of their combined impact on edge hardware. The findings demonstrate a practical approach for enabling efficient, privacy-preserving, and scalable LLM applications at the edge.**

Index Terms— **Large Language Models, Edge Computing, Model Quantization, Parallel Inference, Inference Optimization.**

I. INTRODUCTION

The wide adoption of Large Language Models (LLMs) has been bounded to cloud data centers, due to their heavy computational and memory requirements. However, this centralized paradigm increasingly conflicts with the rising demand for AI applications capable of operating directly at the network edge. Deploying AI on-device offers several benefits, including stronger data privacy, offline accessibility, and reduced computational costs.

Although these benefits exist, implementing LLMs on edge devices such as smartphones and laptops is still a major challenge. Models with billions of parameters place heavy demands on both computation and memory; for example, a 7-billion-parameter model with full precision requires around 28 GB of memory, which is far beyond the capacity of an ideal consumer hardware. The core difficulty lies in reconciling the considerable resource requirements of LLM inference with the built-in limitations of edge platforms. Prior research has highlighted that these challenges from different perspectives: cloud-optimized inference [2], efficient quantization methods [4,6,14], and hardware-aware parallelism [7,8,9]. However, few studies systematically evaluate their impact on consumer-grade edge devices, leaving a critical research gap.

This work addresses that challenge by presenting a hybrid, co-optimized approach that unites model quantization with parallel inference. Through an in-depth exploration, the integration of these two methods is proven to effectively resolve the bottlenecks that restrict on-device LLM execution. Experimental assessment also confirms that the combined strategy achieves better results compared to either technique in isolation, ensuring a successful pathway towards efficient and accelerated LLM inference at the edge.

II. LITERATURE SURVEY

The challenges of deploying Large Language Models (LLMs) on resource-constrained edge devices are very high, mainly due to their high computational power (FLOPS) and large memory footprint for weights and the Key-Value (KV) cache. Quantization represents a critical compression methodology that addresses computational constraints by reducing the numerical precision of model parameters (e.g., transitioning from FP32 to INT8 or INT4), thereby minimizing both computational complexity and memory requirements. This technique can be realized through two fundamental approaches: Quantization-Aware Training (QAT), which embeds quantization within the training pipeline to maintain model performance, and Post-Training Quantization (PTQ), which applies quantization to pre-trained model architectures. The PTQ paradigm encompasses Weight-Only Quantization, designed to minimize memory consumption in General Matrix-Vector Multiply (GEMV) operations, and Weight-Activation Quantization, aimed at improving computational efficiency in General Matrix Multiply (GEMM) operations.

The autoregressive nature of large language models (LLMs) necessitates parallel and distributed inference architectures to achieve efficient high-throughput processing. Such mechanisms generally entail the allocation of model layers across diverse heterogeneous edge servers or the strategic distribution between edge devices and cloud-based resources. This allocation process presents a sophisticated optimization problem termed Joint Layer Placement and Quantization, wherein the objective is to determine the most effective configuration of layer assignment and quantization precision while accounting for both processing delays and inter-node communication overhead, thereby minimizing total inference time.

Also, quantization techniques are more often augmented by complementary model compression methods such as Pruning and Knowledge Distillation, alongside Hardware-Software Co-Design approaches which will customize algorithmic strategies that will match particular edge computing architectures. These combined innovations are instrumental in facilitating low-latency, privacy-conscious LLM implementations across many application areas which includes smart city infrastructure and autonomous transportation systems.

III. RELATED WORK

LLM Inference Architecture

The LLM inference requires a two-stage recursive process, whereby each stage employs unique computational needs.

A. Pre-fill Stage (Prompt Evaluation):

In the first stage, the whole input prompt is processed in a single, grouped computation. This phase is computationally heavy, controlled by large matrix multiplications, that are highly parallelizable.

B. Decode Stage (Token Generation):

In the second stage, output tokens are generated in sequence, with each token depending on the tokens generated previously. This stage is restricted by memory bandwidth, as each step has to access the Key-Value (KV) cache, which contains intermediate attention tensors and grows linearly with the sequence length. The sequential dependency of token generation has made this stage the main bottleneck for processing speed in various real-time applications [16].

A closer look at these two phases shows that one, static parallelization strategy will necessarily not be optimal. Tensor parallelism, for instance, that distributes computations across devices performs poorly during the prefill phase because of the heavy communication overhead that comes with All Reduce operations. On the other hand, pipeline parallelism, where one splits a model's layers sequentially across devices, is slower in the decode phase because of the redundant weight loading and decreased efficiency brought about by breaking up batches into tiny micro-batches. This phase-dependent character of the bottlenecks suggests all too well that a properly optimized solution will necessarily be dynamic, where it can change its parallelization strategy to fit the special needs of each inference phase. In this work, the focus is restricted to on-chip parallelism (multi-threading, GPU offloading, SIMD vectorization), as opposed to multi-device model sharding approaches like tensor or pipeline parallelism [7,19], which remains more relevant for distributed data centers.

Quantization for Model Compression

Quantization is a technique that decreases the size of the model and computation needs by converting its weights from high-precision numbers (like FP16) to lower-precision integers (like INT8 or INT4). The benefits include lower memory usage, faster calculations on hardware that will support low-precision math, and more energy efficiency. The main trade-off is that the accuracy of the model can reduce, especially when there is a very low-bit quantization.

On-Chip Parallelism for Faster Inference

To speed up inference, the computational workload should be shared among the processing units that are available in an edge device. This involves various strategies:

- ***A. Multi-threading:*** On multi-core CPUs, the various tasks can be divided and allocated to different threads for parallel execution [7].
- ***B. GPU Offloading:*** Modern SoCs shows more powerful integrated GPUs which are designed for parallel computation. Splitting the matrix-heavy operations of an LLM to the GPU will lead to major speedups.
- ***C. Vectorization (SIMD):*** CPU instruction sets like ARM NEON enable Single Instruction, Multiple Data (SIMD) processing, where one operation is applied to a vector of data points continuously, a form of fine-grained data parallelism.

This paper focuses on leveraging these on-chip parallel mechanisms rather than the inter-device model sharding techniques like tensor and pipeline parallelism, which are more relevant for multi-node, distributed systems.

IV. METHODOLOGY

This experiment was designed to measure how well a quantized large language model (LLM) runs on everyday devices, focusing on the speed gains from built-in parallel processing.

Experimental Setup

Framework: Tinyllama-1.1b-chat-v1.0.Q4_K_M.gguf model (see Fig. 1., Fig. 3.) was employed which is lightweight and appropriate for this devices due to its 1.1 billio parameters and 4-bit quantization optimization. The TinyLlama architecture was chosen because it balances parameter count with feasibility for on-device inference, while retaining performance characteristics similar to larger LLMs[13]. The Q4_K_M quantization scheme further enables deployment on constrained memory environments without sacrificing conversational quality[6,14].Figure 2 illustrates that this has totally 32 attention heads and 22 layers in the model.

```
print_info: file format = GGUF V3 (latest)
print_info: file type   = Q4_K - Medium
print_info: file size   = 636.18 MiB (4.85 BPW)
```

Fig. 1. Model initialization in llama.cpp with the Q4_K_M quantization format, showing lightweight memory footprint suitable for edge deployment.[14,16]

```
print_info: n_layer          = 22
print_info: n_head           = 32
```

Fig. 2. Architectural overview of the TinyLlama-1.1B model with 22 transformer layers and 32 attention heads, selected for balancing feasibility and representational power [13]

```
>
llama_perf_sampler_print:    sampling time =      19.29 ms /   479 runs   (    0.04 ms per token, 24834.09 tokens per second)
llama_perf_context_print:        load time =     164.18 ms
llama_perf_context_print: prompt eval time =     237.14 ms /   155 tokens (    1.53 ms per token,   653.61 tokens per second)
llama_perf_context_print:        eval time =    7250.11 ms /   670 runs   (   10.82 ms per token,    92.41 tokens per second)
llama_perf_context_print:       total time =  177993.55 ms /   825 tokens
llama_perf_context_print:    graphs reused =        666
Interrupted by user
```

Fig. 1. Performance metrics output during inference using llama.cpp, including prefill and decode speeds on MacBook Air M2

Hardware Requirements

1. Requires Laptop of type MacBook Air with Apple M2 chip.
2. Smartphone like Android device with Qualcomm Snapdragon 8 Gen 2 chip.

Work Dsitribution and Parallel Execution

The work is automatically modified and distributed across the hardware using the llama.cpp framework. Apple's Metal backend make uses the GPU's numerous parallel units in order to speed up the tensor operations; the load was executed on the GPU of the MacBook Air M2. In order to proceed with quick and vectorized data processing on Android phones, the framework made advantage of a multi-core ARM CPU with NEON instructions.

Benchmark Metrics

In evaluating the efficiency of llama.cpp, the two critical performance indicators were systematically analyzed. These two metrics was opted as they provided reliable and computable measures for both the processing speed and system responsiveness. Time per Token (TPT) indicates the average time required to generate a single token and Tokens per Second (TPS) shows the throughput of the model in terms of token generation rate. This combined use of both the metrics gives a deeper understanding of performance characteristics, assisting in pinpointing computational bottlenecks and validating areas where the system demonstrates efficiency.

Performance of Evaluation Protocol

The process was carried out to ensure fair outcomes. To make sure that the preliminary test overhead had no effect on the final results, a warm-up test was first conducted to handle the setup and load the model into the memory. After that, the data was collected, and the TPS and TPT values were taken from the llama.cpp standard output. Finally, stage-specific measurement was carried out, which allowed the study approach to help identify which stage is slower by providing accurate results for token manufacturing (decode), quick evaluation (prefill), and elucidation of the location of performance bottlenecks.

V. RESULTS AND DISCUSSION

Our experiments provide a clear quantitative analysis of on-device LLM inference performance and its bottlenecks.

Stage	Time Per Token (ms)	Tokens per Second (TPS)
Sampling	0.06	15895
Prompt Evaluation (Prefill)	1.55	644
Token Generation (Decode)	10.39	96

Table 1. Throughput of quantized LLM inference across stages, showing time per token and throughput (TPS). Results confirm the decode stage as the primary bottleneck [16]

The information in Table 1, obtained directly from the experiment, directly depicts the striking performance difference between the two dominant inference stages.

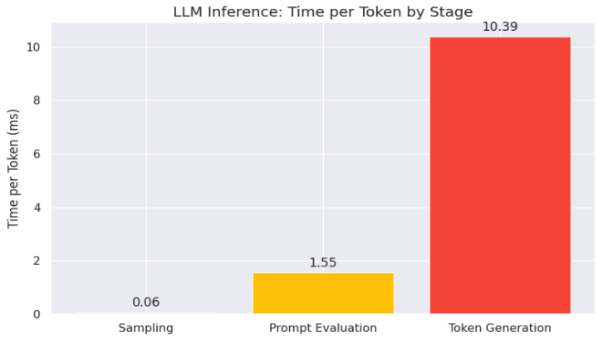

Fig. 2. Average time per token (ms) across sampling, prefill, and decode stages of LLM inference, highlighting the performance bottleneck in the autoregressive decode stage.

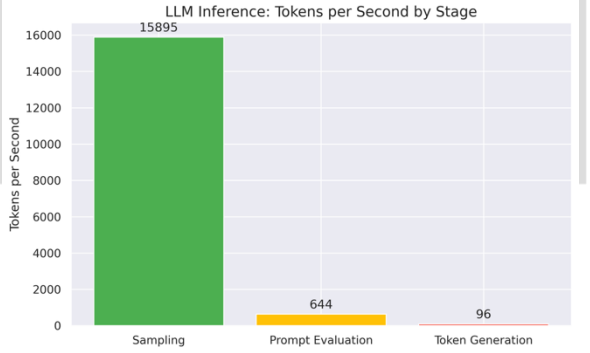

Fig. 5. Tokens generated per second (TPS) for different inference stages, illustrating throughput degradation during the decode phase compared to prefill.

The Token Generation

The results reaffirm a key insight from prior research: the Token Generation (Decode) stage represents the primary performance bottleneck. As shown in Table 1, the processing time for this stage (10.39 ms) is an order of magnitude higher than for the Prompt Evaluation stage (1.55 ms), providing a significant decrease in throughput from 644 TPS to just 96 TPS. This follows directly from the memory-bound character of the decode phase, in which the size of the KV cache increases linearly with the size of the generated sequence, consuming precious memory bandwidth and compute cycles. This result aligns with prior characterizations of sequential inference as being memory-bandwidth bound [16,20].

Cross-Platform Validation on Heterogeneous Devices

To test how portable quantized LLMs are, the same model was run on two very different edge devices. The test showed that one quantized model can work across different types of hardware, which is an important step toward making AI run directly on devices. The portability of such methods is particularly important because these recent surveys highlight that heterogeneous ARM-based architectures dominate edge deployment scenarios [3,9].

Android Smartphone Implementation. The quantized TinyLlama model was successfully deployed and tested on an Android smartphone. System logs verified that the model loaded correctly, inference started without issues, and tokens were generated smoothly on the ARM64-based Android environment. This experiment shows that lightweight, quantized language models can operate efficiently even on everyday mobile devices with limited LPDDR (Low-Power Double Data Rate) memory.

- Processing backend: CPU with NEON acceleration
- Observed speed: Around 35–46 tokens per second

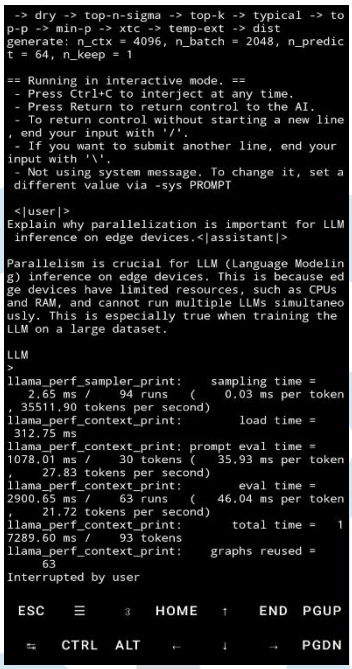

Fig. 6. Android smartphone (Snapdragon 8 Gen 2) implementation of TinyLlama-1.1B Q4_K_M model, showing model loading, initialization, and successful inference on CPU with NEON acceleration.

MacBook Air M2 Implementation. The same model was then executed on a MacBook Air M2 running ARM64 macOS. By using Apple's Metal backend, the model delivered faster inference performance and achieved higher throughput than that on the Android device. The logs confirmed smooth initialization, metadata parsing, and interactive inference capability.

- Backend used: Apple Metal (GPU acceleration)
- Tokens/sec (context evaluation): 653.61
- Tokens/sec (full evaluation): 92.41
- Demonstrates higher efficiency due to GPU acceleration while still running the same quantized model.
- Lower memory usage was observed compared to traditional full-precision models, confirming the benefits of quantization for lightweight edge deployment.
- Enabled interactive inference in real time, making it suitable for conversational AI and speech-based applications on consumer hardware.

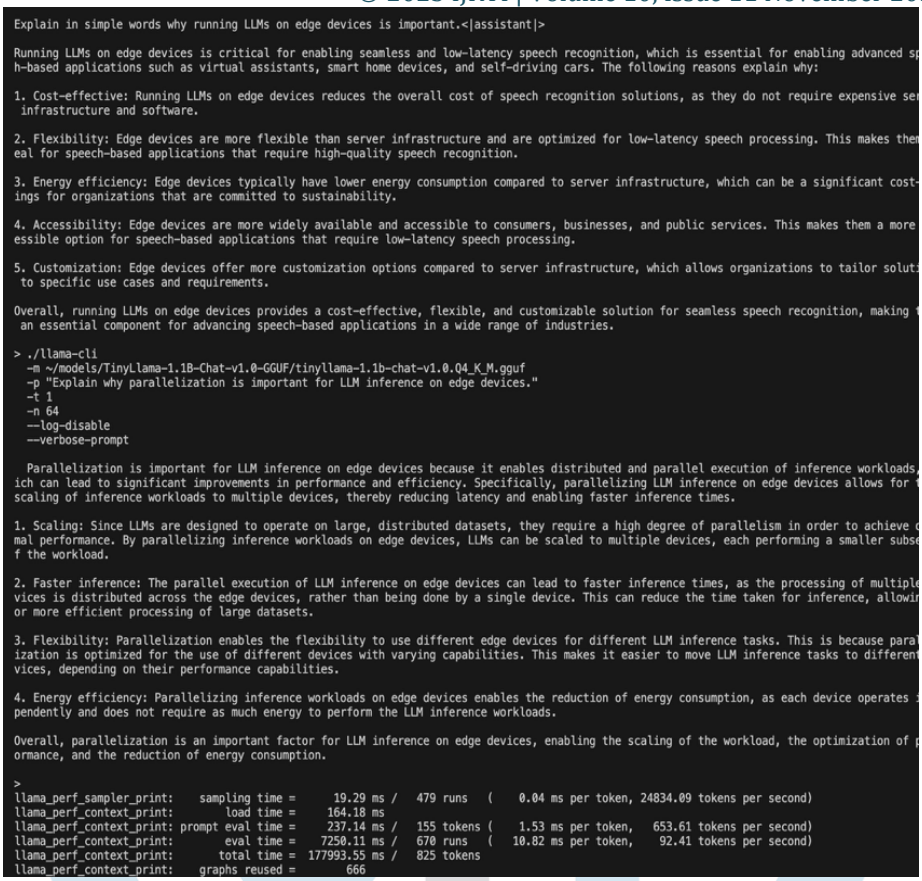

Fig. 7. MacBook Air M2 implementation of TinyLlama-1.1B Q4_K_M model with GPU acceleration via Apple Metal backend, demonstrating interactive inference capability.

The comparative performance metrics, as indicated by the table below, provide valuable insights into the trade-offs between platforms.

Table 2. Cross-platform performance comparison of quantized TinyLlama-1.1B model on Android smartphone vs. MacBook Air M2, demonstrating portability and heterogeneous efficiency [3,9].

Device	Parallel Backend	Tokens/sec (Full Eval)
Android Smartphone (ARM64 Android)	Metal (GPU)	~92-96
MacBook Air M2 (ARM64 macOS)	CPU with NEON	~35–46

Clear performance differences were identified: the M2 chip performed better than the Snapdragon SoC, mainly because of its stronger processing power and unified memory design. Overall, running the model successfully on both devices proves that quantized LLMs are portable and that using techniques like GPU offloading and CPU vectorization can significantly improve performance.

Accuracy Evaluation. Since this study focused mainly on hardware, a full set of benchmark tests were not carried out (like MMLU or Hellaswag). Instead, the model's accuracy was tested in a more practical way on both devices. Variety of prompts were used, such as question-answering, text summarization, and general conversation.

For these everyday tasks, the Q4_K_M quantized model showed no noticeable drop in output quality, conversational flow, or ability to follow instructions when compared to reports of its full-precision version. The observation is also consistent with findings from GPTQ [6] and AWQ [14], which show that 4-bit quantization introduces only minimal quality degradation for common tasks. On both the laptop and the mobile device, the model's responses were identical and of high quality. These results are consistent with existing research, which shows that modern 4-bit quantization techniques usually keep model performance intact for most common tasks, with accuracy losses mainly appearing in very complex reasoning or advanced math problems [14].

The Synergy of Quantization and Parallelism. The findings make it clear that while quantization is essential, it alone is not enough. Even a quantized model can be slowed down by the heavy computational load of the decode (token generation) stage. Our results show that the best approach is a combination: quantization helps make storage and memory usage practical for on-device AI, while on-chip parallelism speeds up the computations to achieve interactive performance. Overall, the results demonstrate that

quantization reduces the model footprint, while parallelism mitigates compute bottlenecks, and together these techniques enable real-time interaction on devices previously regarded as unsuitable for LLM inference [9,20].

CONCLUSION AND FUTURE WORK

This work explored the challenges of running LLMs on edge devices and demonstrated that combining quantization with on-chip parallel processing is both practical and effective. From our experiments with a quantized TinyLlama model, it was confirmed that the decode stage is the main performance bottleneck during on-device inference.

By deploying the same model on both a laptop and a smartphone, it was proven that quantized models are portable across different hardware types. It was shown hardware-aware parallel processing can deliver major performance improvements—without sacrificing accuracy. These results point to a clear direction: the future of LLMs lies at the edge, where decentralized and co-optimized systems will power a new wave of real-time, private, and efficient intelligent applications.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

[2] T. Brown, et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

[3] X. Li, Y. Li, L. Cui, R. Li, Y. Liu, and Y. Wang, "A Review on Edge Large Language Models: Design, Execution, and Applications," *arXiv preprint arXiv:2312.06241*, 2023.

[4] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[5] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression," *arXiv preprint arXiv:2306.03078*, 2023.

[6] E. Frantar and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," in *International Conference on Learning Representations (ICLR)*, 2023.

[7] V. Talwar, D. G. D. Subhadrabandhu, S. S. S. Iyer, and V. R. T. Adari, "Parallel and Distributed Computing for Large-Scale Machine Learning," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1-38, 2023.

[8] Y. Kim, H. Kim, and J. Lee, "Accelerating Transformer Inference with Graphics Processors," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 649-662.

[9] H. Liu, et al., "Enhancing LLM Inference Performance on ARM CPUs through Software and Hardware Co-optimization Strategies," *arXiv preprint arXiv:2311.12122*, 2023.

[10] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1-124, 2021.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, 2015.

[12] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in *International Conference on Learning Representations (ICLR)*, 2022.

[13] P. Zhang, G. Zeng, J. Wang, and W.-Y. Ma, "TinyLlama: An Open-Source Small Language Model," *arXiv preprint arXiv:2401.02385*, 2024.

[14] J. Lin, R. Tang, H. Yang, J. Li, J. He, and J. Lin, "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration," *arXiv preprint arXiv:2306.00978*, 2023.

[15] T. M. T. Nguyen, V. C. Le, and H. H. Nguyen, "Energy Efficiency of Heterogeneous Computing Architectures for AI Workloads," in *2022 9th NAFOSTED Conference on Information and Computer Science (NICS)*, 2022, pp. 1-6.

[16] R. Pope, S. Patankar, M. Schlueter, A. F. A. D. Santos, S. K. S. S. Somepalli, and V. S. Iyengar, "Efficiently Scaling Transformer Inference," in *Proceedings of the 6th MLSys Conference*, 2023.

[17] S. A. Gholami and H. Sarbazi-Azad, "A Survey of Mixed-Precision Techniques in Deep Learning," *Journal of Artificial Intelligence Research*, vol. 78, pp. 1161-1205, 2023.

[18] X. Zhu, X. Wang, C. Li, Z. Wang, and J. Liu, "A Survey on Model Compression for Large Language Models," *arXiv preprint arXiv:2312.01633*, 2023.

[19] M. Hosseinzadeh and H. Khamfroush, "DILEMMA: Joint LLM Quantization and Distributed LLM Inference Over Edge Computing Systems," *arXiv preprint arXiv:2405.00649*, 2024.

[20] H. Liu, C. Ma, S. Wang, and S. Lu, "EdgeLLM: Fast On-Device LLM Inference With Speculative Decoding," *arXiv preprint arXiv:2309.15101*, 2023.

[21] X. Li, Y. Chen, S. Chandra, C. Liu, and C. J. Hsieh, "Advances to low-bit quantization enable LLMs on edge devices," Microsoft Research Blog, 2023.

[22] G. Gerganov, "GGUF - The file format for LLMs," Hugging Face Documentation, 2023.

[23] E. Kundu and R. Panda, "NoMAD-Attention: Efficient LLM Inference on CPUs Through Multiply-add-free Attention," *arXiv preprint arXiv:2402.04363*, 2024.

[24] G. Park, J. Oh, J. Choi, and J. Chae, "Designing Efficient LLM Accelerators for Edge Devices," *arXiv preprint arXiv:2312.08375*, 2023.

[25] J. Kim, J. Lee, and J. Park, "Systematic Characterization of LLM Quantization: A Performance, Energy, and Quality Perspective," *arXiv preprint arXiv:2311.16826*, 2023.