A Component-Level Characterization of Edge LLM Latency

Jinendra Malekar Jiarong Xu Aishneet Juneja Mohammadreza Mohammadi Ramtin Zand
University of South Carolina
{jmalekar, jiarongx, junejaa, mohammm}@email.sc.edu, ramtin@cse.sc.edu

***Abstract*—This paper presents a comprehensive analysis of four widely used large language model (LLM) families, Phi, LLaMA, Gemma, and OPT, highlighting the diverse architectural design choices that have emerged in recent years and are often not systematically compared. To bridge this gap, we deploy thirteen representative models from these families on the NVIDIA Jetson AGX Orin edge AI platform, and conduct a rigorous empirical evaluation of inference latency across varying sequence lengths. Our study provides a fine-grained breakdown of latency contributions from key LLM operations, including linear projections, attention matrix multiplications, and nonlinear components such as softmax, GELU, and normalization layers. The results reveal nuanced trade-offs between architectural design choices and runtime behavior, highlighting performance bottlenecks that are often overlooked in high-level evaluations. By grounding our study in real-world edge deployment, this work offers actionable insights and a practical roadmap for optimizing LLM performance in resource-constrained environments.**

I. Introduction

The deployment of transformer-based language models [1], [2] has expanded far beyond the data center. The same architectural family that powers cloud-hosted large language models is increasingly being adapted for inference on resource-constrained edge platforms such as the NVIDIA Jetson Nano, Raspberry Pi, and similar single-board computers [3]. This hardware diversity creates a fundamental tension: a configuration that is optimal at one sequence length on one class of device may be entirely wrong for another. Yet most profiling studies treat the transformer as a monolithic unit, reporting aggregate throughput numbers that obscure precisely the component-level information a practitioner needs to make good deployment decisions.

This paper addresses that gap. We present a fine-grained latency breakdown of transformer inference across thirteen distinct computational components, measured at five sequence lengths ranging from 256 to 8,192 tokens. The central observation motivating our work is that the dominant bottleneck in a transformer is not fixed it is a function of sequence length, and it shifts in a predictable and dramatic way. At short contexts (256-512 tokens), the model is bottlenecked by dense matrix multiplications: QKV projection and the feed-forward network (FFN) together account for over 60% of runtime because their cost grows only linearly with sequence length. As context grows beyond roughly 1,024 tokens, the quadratic attention operations including $QK^\top$ matrix multiplication, Softmax normalization, causal masking, and the attention weighted sum begin to dominate, and at 8,192 tokens they collectively consume more than 80% of wall-clock time.

The right optimization strategy is entirely different depending on where the bottleneck actually lies, and the bottleneck depends jointly on the sequence length and the compute profile of the target hardware.

On server-class accelerators such as the NVIDIA A100 or H100, the arithmetic units are powerful enough that even the quadratic attention terms remain manageable up to moderate sequence lengths, and the binding constraint is more often memory bandwidth specifically, the cost of reading the KV cache [4] from high bandwidth memory (HBM) [5], [6]. In this regime, architectural choices such as Multi-Query Attention (MQA) [6] or Grouped-Query Attention (GQA) [7], which reduce the KV cache [4] footprint, yield real throughput improvements during the decode phase, while the prefill pass remains dominated by matrix arithmetic. Hardware-aware kernels such as FlashAttention [5], which reorganize memory access patterns rather than reduce floating-point count, are correspondingly high-value at this tier.

On edge platforms, the constraints are categorically different. The NVIDIA Jetson Nano provides 128 CUDA cores and 4GB of shared LPDDR4 memory with approximately 25GB/s of memory bandwidth roughly 160× less compute and 30× less bandwidth than an A100. The Raspberry Pi4 is CPU-only, operating at roughly 4GFLOPS of sustained FP32 throughput. On these platforms, even the linear-scaling projection layers are heavy: a single QKV projection for a 768-dimensional model at sequence length 256 involves a $768 \times 2{,}304$ matrix multiply that strains these devices' arithmetic capacity. This means the crossover point from projection-dominated to attention-dominated runtime profiles occurs at much longer sequences on edge hardware than on server GPUs indeed, for many practical edge deployments it may never occur within the sequence lengths the application uses [8].

The implication is that edge-targeted optimization should prioritize the projection layers first, using methods like quantizing the weight matrices [9]–[11], structured pruning and dimension reduction [12], [13], and batching strategies that amortize the fixed overhead of each linear layer call. Applying FlashAttention-style memory optimizations to these devices, while intellectually appealing, yields negligible benefit if QKV projection still accounts for 40% of total runtime.

A further subtlety arises from KV caching during autoregressive generation [10]. When a model decodes one token at

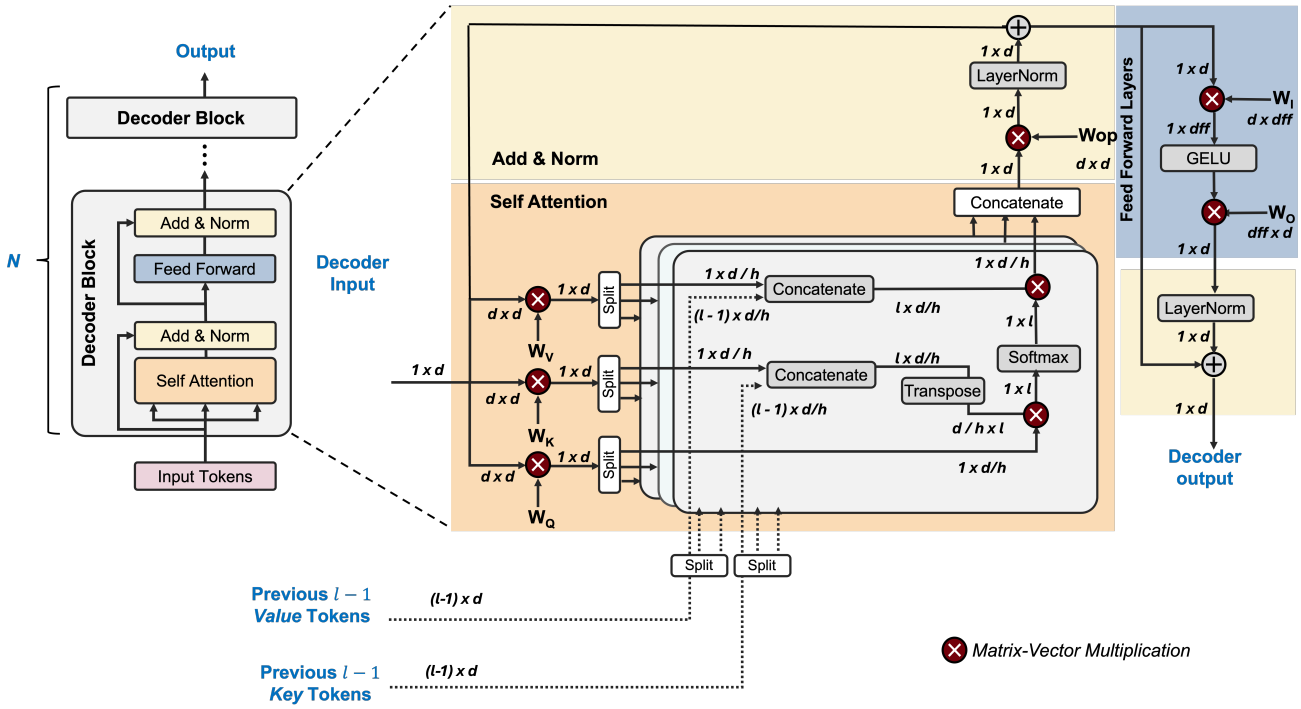


Fig. 1: Underlying operations in typical decoder-only LLMs.

a time, the prompt is processed in a single *prefill* pass where all sequence positions are active simultaneously and attention is expensive and each subsequent *decode* step processes only a single new token against the cached past keys and values, where the attention cost collapses but per-token projection overhead remains fixed. A system that appears efficient on aggregate throughput may be hiding a slow prefill that destroys perceived responsiveness [14], or a slow per-token decode step that causes streaming generation to stutter. Our instrumentation framework captures these aspects distinctly, enabling practitioners to diagnose which half of the inference pipeline is the binding constraint.

Here, we profile thirteen LLM components across six sequence lengths spanning 256 to 8192 tokens on the NVIDIA Jetson Orin AGX, quantifying the crossover point at which quadratic attention operations overtake linear projection costs as the dominant runtime bottleneck. We compare these dynamics across four model families Phi [15], LLaMA [16], Gemma [17], and OPT [18] which differ in projection structure, FFN design (gated versus standard), and auxiliary projection blocks, providing a richer characterization of how architectural choices modulate the crossover point. We further examine MHA and GQA [1], [6], [7] through this component-level lens, and will release an open-source PyTorch instrumentation framework built on CUDA events upon acceptance to facilitate reproduction and extension of our analysis.

II. LLM Components and Model Families

All computations and complexity analyses in this section assume a batch size of 1, i.e., single-sequence inference, which

Block	Op	GPT	LLaMA	OPT	Gemma
Attn Proj.	W_Q $(d \times d)$	✓	✓	✓	fused
	W_K $(d \times d_{kv})$	✓	✓	✓	
	W_V $(d \times d_{kv})$	✓	✓	✓	
	W_O $(d \times d)$	✓	✓	✓	✓
Attn	QK^T $(l \times d/h)$	✓	✓	✓	✓
	SV $(l \times d/h)$	✓	✓	✓	✓
FFN	Up $(d \times d_{ff})$	✓	✓	✓	✓
	Gate $(d \times d_{ff})$	×	✓	×	✓
	Down $(d_{ff} \times d)$	✓	✓	✓	✓
Extra	Proj-In $(d_{in} \times d)$	×	×	✓	×
	Proj-Out $(d \times d_{out})$	×	×	✓	×
Output	LM $(d \times V)$	✓	✓	✓	✓

TABLE I: MatMul structure across model families. $d_{kv} < d$ for GQA (LLaMA, Gemma).

is common in edge-based conversational AI systems. Decoder-only LLMs are composed of N stacked decoder blocks, each containing both projection-based and non-projection operations. The primary projection operations are found in the *Multi-Head Self-Attention (MHSA)* and *Feed-Forward Network (FFN)* modules. Within MHSA, the input representation is linearly transformed into *query* (Q), *key* (K), and *value* (V) vectors using learned weight matrices, followed by an additional projection applied to the attention output:

$$Q = W_Q I, \quad K = W_K I, \quad V = W_V I \tag{1}$$

These vectors are partitioned into h attention heads, each with dimensionality d/h. During autoregressive decoding, previously computed keys and values are retained in a **KV**

cache (see Fig. 1). At decoding step t, the cached tensors have dimensions:

$$K_{\text{cache}}, V_{\text{cache}} \in \mathbb{R}^{(d/h)\times l-1}$$

where l denotes the current sequence length. For each attention head, the computation is given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{2}$$

With batch size equal to 1, the operations simplify to QK^T: $(1 \times d/h)(d/h \times l)$ and the subsequent multiplication with V: $(1 \times l)(l \times d/h)$. As a result, attention during the decode phase reduces to matrix–vector multiplications.

In the FFN module, typically two linear transformations are applied: the first expands the hidden representation to a higher-dimensional space, and the second projects it back to the original model dimension. In contrast, *residual connections* and *layer normalization* do not involve learned projections; instead, they support stable training and facilitate effective information propagation across layers.

$$\text{FFN}(x) = \text{GELU}(xW_I + b_I)W_O + b_O \tag{3}$$

where $W_I \in \mathbb{R}^{d_{FF}\times d}$ and $W_O \in \mathbb{R}^{d\times d_{FF}}$ are trainable weight matrices, and d_{FF} denotes the size of the feed-forward layer. Overall, the computational workload comprises linear projections (matrix multiplications) and nonlinear operations such as Softmax, GELU [19], and LayerNorm. Prior work has studied the computational characteristics of these components [8]; here, we empirically quantify their runtime contributions across sequence lengths.

A. Model Family Differences

Although decoder-only transformers share a common computational backbone, the model families considered in this paper differ in ways that affect both compute and memory behavior. Phi serves as a baseline family with standard multi-head attention and a conventional GELU-based FFN.

Attention Projections. Phi [15], GPT [20], and OPT [18] implement separate projections for Q, K, and V, resulting in three independent matrix multiplications. Gemma [17] uses a *fused projection*, combining these into a single kernel to reduce memory traffic. LLaMA [16] and Gemma further adopt *Grouped Query Attention (GQA)*, where the number of key/value heads h_{kv} is smaller than the number of query heads h, lowering the dimensionality of K and V. This reduces both KV cache size and memory bandwidth during decoding.

Feed-Forward Network. Phi, GPT, and OPT use a standard two-layer FFN, while LLaMA and Gemma employ a *gated FFN* (e.g., SwiGLU), introducing an additional projection:

$$(\sigma(xW_G) \odot xW_I)W_O$$

This increases FLOPs per token but improves model quality, effectively trading compute for expressivity.

Boundary Projections. OPT uniquely includes input/output projection layers (Proj-In, Proj-Out), adding extra matrix multiplications outside the main decoder stack. These layers slightly increase end-to-end latency, particularly during prefill.

These architectural differences also translate into latency variations across model families: Phi provides a useful baseline for comparison, while GQA-based models (LLaMA, Gemma) typically achieve lower decode latency due to reduced KV cache access, and OPT incurs additional latency due to extra projection layers. A detailed latency comparison across Phi, LLaMA, Gemma, and OPT is provided in Section III-C.

III. Experiments and Results

A. Experimental Setup

All experiments are conducted on an NVIDIA Jetson Orin AGX, a representative edge AI platform, with a 12-core Arm Cortex-A78AE CPU, a 2048-core Ampere GPU, and 64 GB of unified LPDDR5 memory shared between CPU and GPU. All four model families (Phi, LLaMA, Gemma, and OPT) share an identical base configuration: `emb_dim = 3072`, `n_heads = 32`, `n_layers = 32`, `vocab_size = 30,257`, and `context_length = 8,192`. This controlled setup isolates the effect of architectural family on latency distribution, holding parameter count and capacity constant. Architectural differences across families are summarized in Table II. Model weights are randomly initialized. We measure structural runtime rather than task performance. Five sequence lengths are evaluated: 512, 1024, 2048, 4096, and 8,192 tokens, all within the shared `context_length` of 8,192.

TABLE II: Architectural variants evaluated. All models share `emb_dim = 3072`, `n_heads = 32`, `n_layers = 32`. h_{kv} denotes the number of key/value heads (equal to h in MHA and smaller than h in GQA).

Family	Attention	h_{kv}	FFN	Extra proj
Phi	MHA	32	GELU	—
LLaMA / Gemma	GQA	8	SwiGLU	—
OPT	MHA	32	GELU	Proj-In/Out

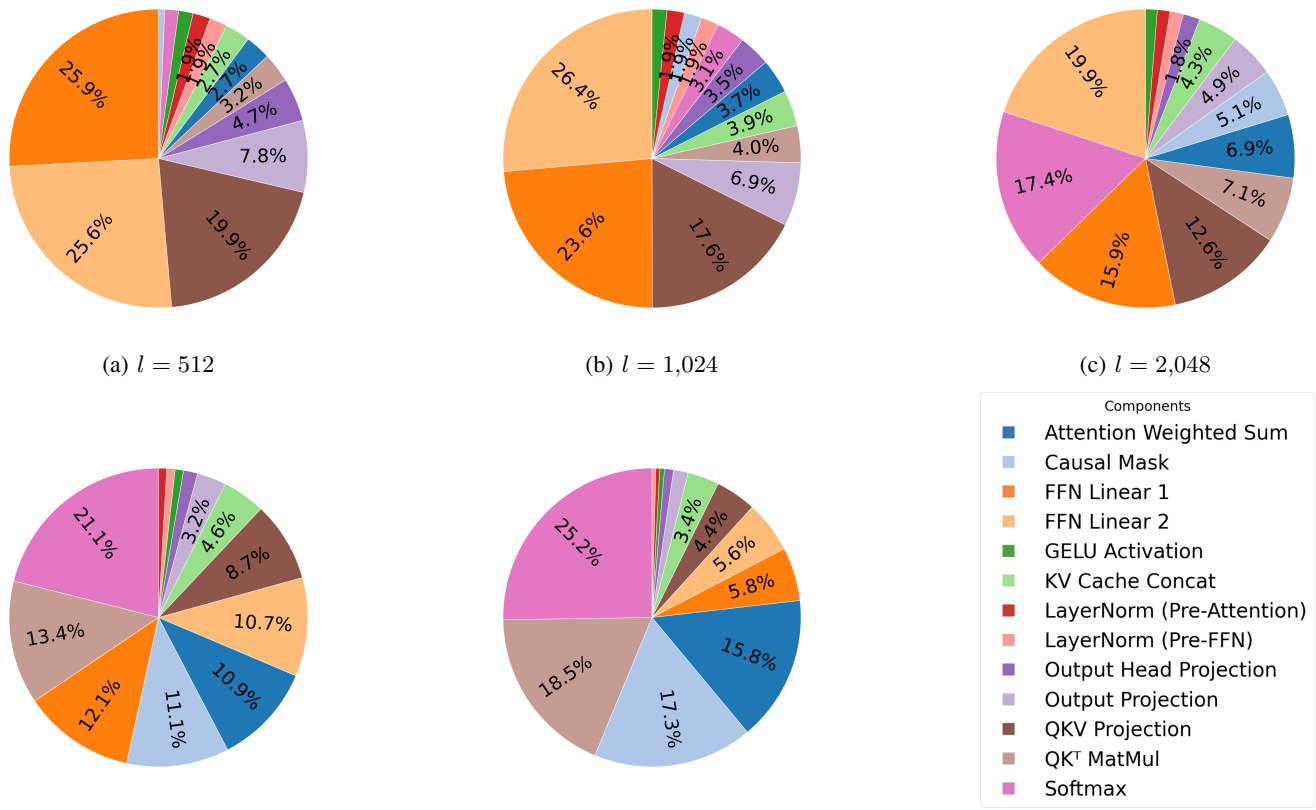
TABLE III: Runtime breakdown (%) of key components; other runtime is attributable to LayerNorm, activations, residual connections, and the LM head.

Component	256	512	1,024	2,048	4,096	8,192
QKV Projection	23.2	19.9	17.6	12.6	8.7	4.4
Output Projection	7.4	7.8	6.9	4.9	3.2	1.6
$QK^\top$	2.2	3.2	4.0	7.1	13.4	18.5
Softmax	1.6	1.5	3.1	17.4	21.1	25.2
Causal Mask	0.3	0.7	1.9	5.1	11.1	17.3
Score.Value (SV)	2.4	2.7	3.7	6.9	10.9	15.8
FFN Linear (1 + 2)	55.1	51.4	50.0	35.8	22.8	11.4

B. Runtime Breakdown Results

Table III summarizes the runtime breakdown of key components across sequence lengths during the decode phase

At the shortest evaluated sequence length ($l = 256$), model behavior is strongly compute-bound dominated by feed-forward layers. The FFN (Linear 1 and 2) alone accounts for



(a) $l = 512$ (b) $l = 1{,}024$ (c) $l = 2{,}048$

(d) $l = 4{,}096$ (e) $l = 8{,}192$

Fig. 2: Runtime distribution of decoder-only transformer components across sequence lengths (l). Each pie chart shows per-component contributions, while the shared legend maps colors to components. At small l, projection layers dominate; as l increases, quadratic attention operations become the primary bottleneck.

55.1% of total runtime, making it the dominant component. QKV projection contributes 23.2%, followed by output projection at 7.4%. Attention-related operations remain relatively small: $QK^\top$ (2.2%), Softmax (1.6%), Causal Mask (0.3%), and Attention Weighted Sum (2.4%). This confirms that at small sequence lengths, runtime is dominated by dense matrix multiplications, particularly within the FFN. At $l = 512$, the distribution remains similar. FFN continues to dominate at 51.4%, while QKV projection contributes 19.9%. Attention components show only modest growth, with $QK^\top$ increasing to 3.2% and Causal Mask to 0.7 while Softmax remains nearly unchanged at 1.5%.

At $l = 1{,}024$, early signs of redistribution emerge. FFN remains dominant at 50.0%, but attention-related operations begin to increase more noticeably. Softmax rises to 3.1%, Causal Mask to 1.9%, and Attention Weighted Sum to 3.7%. QKV projection continues to decline in relative contribution (17.6%), reflecting the growing cost of sequence-length-dependent operations. At $l = 2{,}048$, a clear transition occurs. FFN drops to 35.8%, while attention components grow significantly. Softmax becomes a major contributor at 17.4%, followed by $QK^\top$ at 7.1%, Attention Weighted Sum at 6.9%, and Causal Mask at 5.1%. Collectively, attention-related operations now rival the combined cost of projection and FFN layers, marking a shift away from purely compute-bound behavior. At $l = 4{,}096$, attention components continue to grow in importance. Softmax reaches 21.1%, $QK^\top$ rises to 13.4%, Attention Weighted Sum to 10.9%, and Causal Mask to 11.1%. Meanwhile, FFN decreases further to 22.8%, and QKV projection drops to 8.7%. Attention-related operations now account for the majority of runtime, although projection and FFN layers still contribute non-trivially. At $l = 8{,}192$, this trend becomes even more pronounced. Softmax (25.2%), $QK^\top$ (18.5%), Attention Weighted Sum (15.8%), and Causal Mask (17.3%) together dominate the runtime profile. In contrast, FFN contributes 11.4% and QKV projection just 4.4%. It is important to note that in the decode phase, attention operations scale linearly with sequence length; their increasing share reflects growing memory access and KV-cache traversal costs rather than true quadratic compute dominance.

Overall, these results reveal three distinct regimes: (i) a compute-bound regime at small sequence lengths dominated by FFN and projection layers, (ii) a transitional regime where attention costs become comparable, and (iii) a long-context regime where attention-related operations dominate runtime due to increasing memory and data movement overheads. This

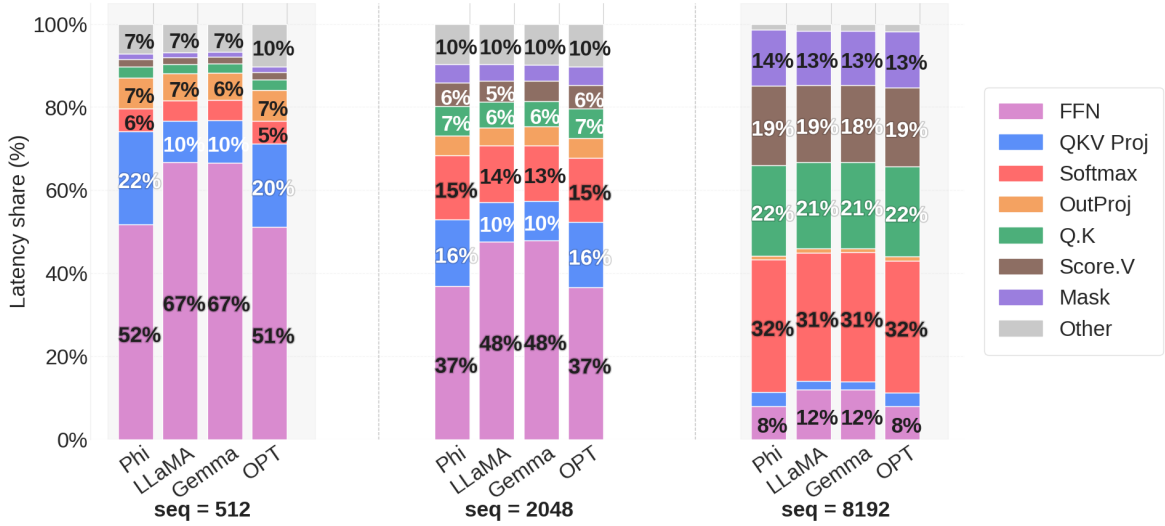


Fig. 3: Latency share (%) by component at sequence lengths 512, 2048, and 8192 for Phi, LLaMA, Gemma, and OPT. Each stacked bar sums to 100% of total inference latency. Components below 6% are unlabelled for clarity.

progression highlights the need for optimization strategies that jointly address compute-intensive projections and memory-bound attention mechanisms during autoregressive inference.

Figure 2 shows detailed runtime distribution across sequence lengths, confirming the three-regime behavior.

C. Comparative Analysis of Different Model Families

All model families are instantiated at a comparable scale (2.5–3B parameters) to enable a controlled analysis of architectural differences. While these configurations are not exact replicas of their original counterparts (e.g., LLaMA, Gemma, OPT), they preserve the key structural characteristics such as GQA, FFN design, and projection layers.

Figure 3 presents the per-component latency breakdown across three sequence lengths for four model families, where three trends emerge. At $l = 512$, the FFN accounts for 51–67% of total latency across all models, reflecting the compute-bound nature of large linear projections when the sequence dimension is small. Phi and OPT exhibit an FFN share of 52% and 51% respectively, while LLaMA and Gemma reach 67%, attributable to their SwiGLU FFN which uses three weight matrices instead of two. As sequence length increases to 2048 and 8192, the FFN share falls progressively to 37-48% at $l = 2048$ and 8-12% at $l = 8192$ while Softmax rises from <10% to 32%, and Score·V (weighted sum) grows from negligible to 19-22%. This shift is consistent with the $\mathcal{O}(l^2)$ scaling of self-attention versus the $\mathcal{O}(l)$ scaling of FFN layers.

LLaMA and Gemma use GQA with 8 KV heads, which reduces the key and value projection cost relative to full Multi-Head Attention. At $l = 512$, their QKV Proj share is 10%, compared to 22% and 20% for Phi and OPT respectively. This advantage persists at longer sequences (16% vs. 10% at $l = 2048$), confirming that GQA provides a consistent reduction in projection overhead. Notably, LLaMA and Gemma produce nearly identical latency profiles across all sequence lengths, indicating that GQA and SwiGLU are the dominant architectural factors rather than any other model-specific difference. OPT includes additional input and output projection layers absent in the other families. This is visible as a consistently elevated "Other" bucket of 10% at $l = 512$).

IV. Conclusion

We presented a fine-grained latency breakdown of LLM inference across five architectural families and five sequence lengths (l) on the NVIDIA Jetson Orin AGX. Results reveal three regimes: a compute-bound phase at short contexts dominated by FFN and projection layers, a transitional region near $l \approx 2{,}048$, and a memory-bound phase at long contexts where attention operations prevail. Architectural differences including GQA in LLaMA and Gemma, the gated SwiGLU FFN, and OPT's fixed boundary projections modulate the magnitude of each component's contribution but do not alter this qualitative progression. These findings directly inform optimization strategy: projection-targeted techniques (quantization, kernel fusion) yield the greatest returns at short contexts, while attention memory optimizations (FlashAttention, KV-cache compression) become dominant beyond $l \approx 4{,}096$.

V. Limitations and Societal Impact

Limitations. Our evaluation is limited to one edge platform and a predefined set of configurations. While this setup enables a clean comparison of architectural effects, the results may not fully capture behavior on other hardware, larger production models, or task-specific workloads. Therefore, the findings should be viewed as a structural analysis of inference latency rather than a universal benchmark of model performance.

Societal Impact. Efficient edge deployment reduces cloud dependence and energy consumption, improving accessibility in resource constrained environments, though easier on-device deployment may also lower barriers to misuse.

VI. Code availability

The following GitHub repository contains the code necessary to reproduce the results found in this paper. https://github.com/iCAS-Lab/LLM-profiler

Acknowledgments

This work is supported by the National Science Foundation (NSF) under grant numbers 2340249 and 2409697.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] B. C. Reidy, M. Mohammadi, M. E. Elbtity, and R. Zand, "Efficient deployment of transformer models on edge tpu accelerators: A real system evaluation," in *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.

[3] M. Ardakani, J. Malekar, and R. Zand, "Llmpi: optimizing llms for high-throughput on raspberry pi," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025, pp. 6378–6387.

[4] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[5] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[6] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019. [Online]. Available: https://arxiv.org/abs/1911.02150

[7] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," 2023. [Online]. Available: https://arxiv.org/abs/2305.13245

[8] J. Malekar and R. Zand, "Amdahl's law for LLMs: A throughput-centric analysis of extreme LLM quantization," *Transactions on Machine Learning Research*, 2025. [Online]. Available: https://openreview.net/forum?id=JtrQJJQYpP

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

[10] 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 llm compression and acceleration," 2024. [Online]. Available: https://arxiv.org/abs/2306.00978

[11] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023. [Online]. Available: https://arxiv.org/abs/2210.17323

[12] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," 2023. [Online]. Available: https://arxiv.org/abs/2301.00774

[13] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" 2019. [Online]. Available: https://arxiv.org/abs/1905.10650

[14] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," 2023. [Online]. Available: https://arxiv.org/abs/2211.17192

[15] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl, A. Benhaim, M. Bilenko, J. Bjorck, S. Bubeck, M. Cai, Q. Cai, V. Chaudhary, D. Chen, D. Chen, W. Chen, Y.-C. Chen, Y.-L. Chen, H. Cheng, P. Chopra, X. Dai, M. Dixon, R. Eldan, V. Fragoso, J. Gao, M. Gao, M. Gao, A. Garg, A. D. Giorno, A. Goswami, S. Gunasekar, E. Haider, J. Hao, R. J. Hewett, W. Hu, J. Huynh, D. Iter, S. A. Jacobs, M. Javaheripi, X. Jin, N. Karampatziakis, P. Kauffmann, M. Khademi, D. Kim, Y. J. Kim, L. Kurilenko, J. R. Lee, Y. T. Lee, Y. Li, Y. Li, C. Liang, L. Liden, X. Lin, Z. Lin, C. Liu, L. Liu, M. Liu, W. Liu, X. Liu, C. Luo, P. Madan, A. Mahmoudzadeh, D. Majercak, M. Mazzola, C. C. T. Mendes, A. Mitra, H. Modi, A. Nguyen, B. Norick, B. Patra, D. Perez-Becker, T. Portet, R. Pryzant, H. Qin, M. Radmilac, L. Ren, G. de Rosa, C. Rosset, S. Roy, O. Ruwase, O. Saarikivi, A. Saied, A. Salim, M. Santacroce, S. Shah, N. Shang, H. Sharma, Y. Shen, S. Shukla, X. Song, M. Tanaka, A. Tupini, P. Vaddamanu, C. Wang, G. Wang, L. Wang, S. Wang, X. Wang, Y. Wang, R. Ward, W. Wen, P. Witte, H. Wu, X. Wu, M. Wyatt, B. Xiao, C. Xu, J. Xu, W. Xu, J. Xue, S. Yadav, F. Yang, J. Yang, Y. Yang, Z. Yang, D. Yu, L. Yuan, C. Zhang, C. Zhang, J. Zhang, L. L. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, and X. Zhou, "Phi-3 technical report: A highly capable language model locally on your phone," 2024. [Online]. Available: https://arxiv.org/abs/2404.14219

[16] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[17] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret, P. Liu, P. Tafti, A. Friesen, M. Casbon, S. Ramos, R. Kumar, C. L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J.-B. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C. A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozińska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucińska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J. P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. yeong Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. Lago, L. McNealus, L. B. Soares, L. Kilpatrick, L. Dixon, L. Martins, M. Reid, M. Singh, M. Iverson, M. Görner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. Bardoliwalla, N. Devanathan, N. Dumai, N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. Jin, P. Georgiev, P. Culliton, P. Kuppala, R. Comanescu, R. Merhej, R. Jana, R. A. Rokni, R. Agarwal, R. Mullins, S. Saadat, S. M. Carthy, S. Cogan, S. Perrin, S. M. R. Arnold, S. Krause, S. Dai, S. Garg, S. Sheth, S. Ronstrom, S. Chan, T. Jordan, T. Yu, T. Eccles, T. Hennigan, T. Kocisky, T. Doshi, V. Jain, V. Yadav, V. Meshram, V. Dharmadhikari, W. Barkley, W. Wei, W. Ye, W. Han, W. Kwon, X. Xu, Z. Shen, Z. Gong, Z. Wei, V. Cotruta, P. Kirk, A. Rao, M. Giang, L. Peran, T. Warkentin, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, D. Sculley, J. Banks, A. Dragan, S. Petrov, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, S. Borgeaud, N. Fiedel, A. Joulin, K. Kenealy, R. Dadashi, and A. Andreev, "Gemma 2: Improving open language models at a practical size," 2024. [Online]. Available: https://arxiv.org/abs/2408.00118

[18] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://arxiv.org/abs/2205.01068

[19] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023. [Online]. Available: https://arxiv.org/abs/1606.08415

[20] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165