Minstrel: Application-Aware SLM Inference Optimization on Edge Devices

Bakshree Mishra
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
bmishra3@illinois.edu

***Abstract*—Large language models (LLMs) have permeated different fields of computing, including agentic systems and controllers. Recent literature has introduced smaller language models (SLMs) capable of running on edge hardware, unlocking opportunities to significantly impact human and computer interaction. Following trends of LLM inference optimization on data centers, optimization of SLM inference on edge devices focuses on independently accelerating the prefill or decode phases. However, we expect the tasks targeted for SLM inference to not follow the same input and output length distributions as remote LLM inference, necessitating a reevaluation of options for hardware and software optimizations. Further, previous work does not study the impact of their optimizations in context of different downstream applications, and the benefits seen in their isolated evaluations are not generalizable.**

In this work, we present Minstrel, an application-aware optimization framework for SLM inference on edge hardware. Minstrel introduces a hybrid empirical and analytical model to predict the inference latency for an application given an SLM and hardware. Using Minstrel, we divide the application space into a prefill-dominated P-Zone and a decode-dominated D-Zone. Leveraging the two zones, we make the observation that, for a certain range of applications, optimizing prefill phase is ineffective.

***Index Terms*—Edge inference, application-aware optimizations, small language models, SLM, Optimization zones**

I. Introduction

Large language models (LLMs) and foundation models have become ubiquitous and entered the public consciousness with popular applications such as Chat-GPT [35] and Gemini [18], [46]. The generalizability of LLMs has influenced and reshaped how applications are developed in different fields; LLMs now are proposed for integration in operating systems [26], [36], as agents and controllers for sub-tasks in workloads [4], [14], [24], [31], [41], [50], [52]–[54], [59], [64], [70], and as a service (LLMaaS) [62], [63].

While LLM inference has typically required cloud resources such as large GPU clusters to run these large models, recent literature has introduced small language models (SLM) of parameter sizes 3B or lower [1], [25], [29], [45]–[47], [58], [68] that can be executed locally on edge/client devices such as smartphones and desktops with reasonable downstream accuracy. These show an encouraging trend in the ability to compress and distill information learned by the large language models [25], [47], [68] and can be a viable alternative for edge-inference tasks [19].

Prior work in accelerating LLM inference in data centers has leveraged the observation that different phases of LLM inference are memory and compute bound [67] to partition the phases into heterogeneous accelerators [9], [9], [37], [43], [58], [58], [60], [69]. However, the distribution of tasks for traditional LLMs and the new SLMs are different. First, a user is more likely to send larger texts (documents, images, etc.) as input to the larger models for complex tasks, and thus inference tasks can benefit immensely from accelerated prompt processing. SLMs are better suited for shorter prompts and for simple agentic tasks [20]. Further, SLMs with agentic use-cases can reuse shared prompts and context history [8], [15], [54], further reducing the prompt sizes to be processed. Other optimizations at data center level focus on improving inference throughput with batching and scheduling techniques [37], [65], [69]. It is unlikely that multiple inference requests may arise from the same user to leverage batching techniques on the edge. There is a need to systematically study the impact of different inference optimization techniques for SLM-oriented tasks with varying input and output lengths.

In this work, we propose Minstrel, a cross-stack framework that enables application-aware evaluation of hardware and/or software optimizations. Minstrel introduces a hybrid empirical and analytical performance model that predicts performance of an SLM on a hardware configuration. Minstrel learns the empirical model parameters for a hardware with linear regression over extensive offline profiling. Minstrel predicts whether a hardware and SLM can satisfy latency constraints for applications with different input and output lengths. Minstrel's performance model further analytically predicts the overall speedup achievable for applications with a wide range of prompt and output lengths, and identifies the inference phase to be prioritized for any hardware or software optimization. With Minstrel, we can clearly divide the inference application space, with varying prompt and decoded output lengths, into a P-Zone suitable for prefill acceleration and a D-Zone sensitive to decode optimization.

II. Background and Related Work

LLMs are deep neural networks that have had considerable success over natural language processing tasks as well as other modalities, primarily based on transformer [51] architecture.

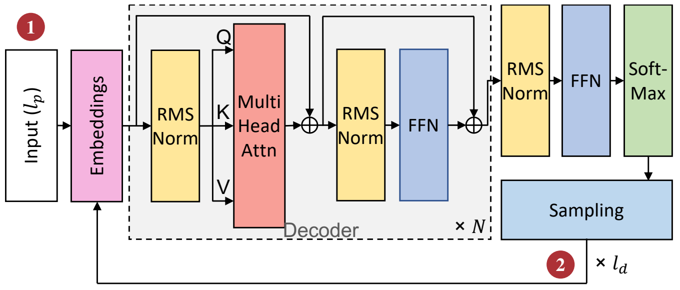

Fig. 1. Block diagram of a decoder-only transformer network. The decoder block consists of the multi-head attention and feed-forward network layers, and is replicated N times in the network. l_p : Prompt length, l_d : Output length.

A. Transformer Architecture

The baseline transformer architecture [51] consists of encoder and decoder blocks, and subsequent models derive from this architecture. Recent models such as GPT-3 [5] and Llama [48], [49] are decoder-only transformer models. Figure 1 shows a block diagram of a decoder-only network, consisting of components such as attention computation and feed forward networks. Inference with transformer architectures is autoregressive, where the generated embedding token is used to generate subsequent output token embeddings.
LLM inference can be decomposed into two phases. During the *prefill* phase (1 in Figure 1), the model computes over the entire input prompt sequence (of length l_p). Inference performance with smaller sequence lengths is dominated by the feed-forward network in the transformer block, which scales linearly with sequence length; for longer sequence lengths, attention computation dominates which scales quadratically [32]. The KV (Key-Value) cache stores intermediate key and value projections during the prefill phase to avoid recomputation during decoding. Subsequently, the prefill phase generates the first output token. The *decode* phase (2 in Figure 1) is autoregressive. Each iteration of the *decode* phase produces one token, which serves as an input for the next iteration to generate the next token. The *decode* phase continues depending on the length of output tokens (l_g). Thus, LLMs are stateful, where previously generated tokens provide context for subsequent output tokens.

B. Heterogeneous Inference

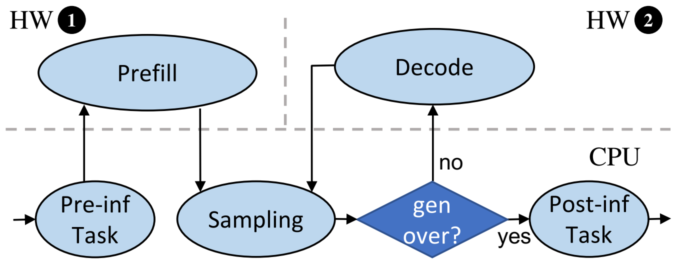

Fig. 2. Flow of executing LLM inference on heterogeneous hardware.

Figure 2 shows a flow chart for an LLM inference task using heterogeneous hardware (CPU, HW 1 , and HW 2). The CPU executes the upstream and downstream tasks. The LLM inference task primarily consists of prefill and decode phases, often offloaded to (same or different) hardware. In general, inference hardware can be CPU-only, or any combination of CPU, GPU, Neural Processing Unit (NPU), or other accelerators (e.g., [12]).

There has been extensive work in leveraging LLM model optimization techniques such as quantization [7], [55], FlashAttention [10], [11], KV caching [2], [40], inference under long context [56] and paged attention [22] to improve inference latency. Previous work [67] has further observed that prefill phase is compute intensive, while the decode phase is more memory intensive. Thus, LLMCompass [67] suggests that prefill phase (HW 1) be mapped to hardware with more computation units, while the decode phase be mapped to hardware with less compute and more memory bandwidth (HW 2). Previous research in data centers apply this strategy to map prompt prefill phase to larger accelerators [37], and further focus on scheduling and batching [37], [42], [44], [65] to improve inference throughput. Table I lists some of such recent work on heterogeneous LLM inference. Other inference scheduling research in data centers [34], [38], [61] optimize for delays and communication costs. In case of edge devices, recent work has attempted to apply similar partitioning of prefill and decode stages onto heterogeneous hardware available on these devices, with most leaning into optimizing prompt prefill.

TABLE I
LLM INFERENCE ON HETEROGENEOUS HARDWARE.

Work	HW 1	HW 2	Target
Splitwise [60]	GPU (large)	GPU (small)	Datacenter
llm.npu [57]	NPU	CPU	Edge
HeteroLLM [9]	NPU	GPU	Edge

C. Application Characteristics of SLMs

Unlike LLM inference on cloud platforms, which support a variety of requests from multiple users, SLM inference on edge devices can be predictable depending on the application and use-case. One category of applications can be agentic use-cases [20] involving short incremental prompts and formatted outputs [4], [14], [24], [31], [41], [50], [52]–[54], [59], [64], [70], where the formatted outputs can be studied to predict the output/decode length. Edge inference of SLMs also provides the capability of caching and reusing context, usually a challenge for LLM inference in data centers [44]. Agentic applications, with multi-turn inference or common recurring prompt prefixes, can utilize this opportunity to reuse the KV-cache and reduce the effective prompt length for inference. However, current literature on optimizing SLM inference on edge devices propose solutions that do not consider these unique opportunities of SLM inference. For example, work on llm.npu (Table I) evaluates heterogeneous inference on benchmarks with long prompts and very short output lengths of 1–2 tokens, but does not study the performance with output lengths representative of actual tasks [39], nor does it consider the impact of reusing the KV-cache from previous inferences that would effectively shorten prompt lengths. Further, other literature predicting LLM inference on hardware provide roofline estimations [21], [66] that map performance of LLM models as singular points, and do not study whether there is impact of any such application characteristics on the measured performance. One option to capture application-specific characteristics is to use the inference prompt and decode lengths as a proxy for complex application behaviors. For example, for an agentic task with common prompt prefixes, we can use the incremental prompt lengths during steady state to predict the actual inference computation. Similarly, we can use the (range of) output lengths observed within a task to predict the overall inference latency. Overall, SLM inference on edge devices provides more visibility to the range of applications and inference workloads. This provides an opportunity for more fine-grained evaluation of different optimizations and their impact per application, which is difficult to achieve with LLM inference on cloud platforms.

III. Minstrel

In this work, we introduce Minstrel, a framework to enable application-aware optimizations for SLM inference on the edge.

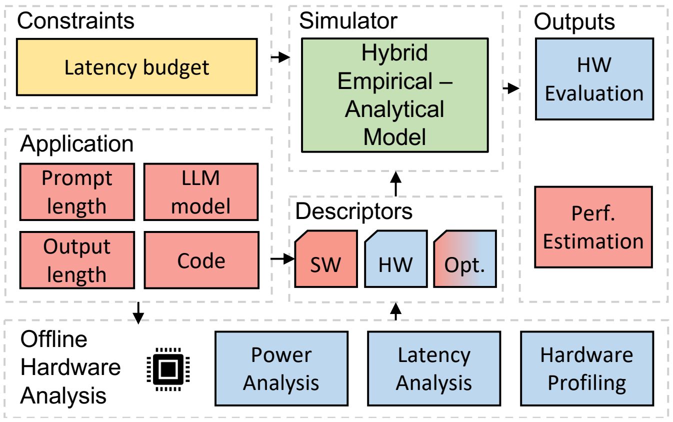

Fig. 3. A block diagram for the Minstrel system. Opt: Optimization.

A. Hybrid Performance Model

Given a language model and hardware, we use a hybrid analytical and empirical performance modeling approach to estimate inference latency $\hat{t}_{e2e}$ for different prompt and decode lengths. SLM inference use-cases include tasks such as natural language commands and agentic tasks [20], [57], where sequence lengths are not very long. As a result, we can reason that the inference would be dominated by linearly-scaling feed-forward networks in the transformer blocks [32]. With this assumption, we define two linear models, $\hat{t}_{pp}$ and $\hat{t}_d$, that accept prompt and decode lengths as inputs and predict prefill latency and decode latency respectively as in Equations 1 and 2:

$$\hat{t}_{pp}(l_p) = \theta_0 l_p + \theta_1, \tag{1}$$
$$\hat{t}_d(l_p, l_d) = \theta_2 l_p + \theta_3 l_d + \theta_4, \tag{2}$$

where

$$\begin{aligned} l_p &: \text{Prompt length}, \\ l_d &: \text{Decode length}, \\ \hat{t}_{pp} &: \text{Predicted prefill latency given } l_p, \\ \hat{t}_d &: \text{Predicted decode latency given } l_p \text{ and } l_d, \\ \theta &: \text{Learnable parameters.} \end{aligned}$$

$\hat{t}_{pp}$ in Equation 1 is only a function of l_p, since prompt prefill phase always terminates with generating one output token and is independent of output length l_d. Minstrel assumes that the prefill and decode phases are sequential, and analytically expresses the end-to-end inference latency as in Equation 3.

$$\hat{t}_{e2e}(l_p, l_d) = \hat{t}_{pp}(l_p) + \hat{t}_d(l_p, l_d). \tag{3}$$

We use empirical data obtained from running inference of SLMs on hardware to train the performance model. The linear parameters $\{\theta_0 \cdots \theta_4\}$ are learned by minimizing MSE.

B. Hardware Evaluation with Minstrel

Minstrel uses Equation 3 to predict inference latency, given a model and hardware, for different applications. Hence, Minstrel can also evaluate whether the model and hardware can satisfy any latency constraint t_{lim} set by the application with l_p input prompt tokens and expecting l_d output tokens. We can define `satisfy` to validate whether the model and hardware (system) can satisfy any latency constraint t_{lim} set by the application with l_p input prompt tokens and expecting l_d output tokens as in Equation 4:

$$\texttt{satisfy}(l_p, l_d) = \begin{cases} 1, & \text{if } \hat{t}_{e2e}(l_p, l_d) \le t_{lim}, \\ 0, & \text{if } \hat{t}_{e2e}(l_p, l_d) > t_{lim}. \end{cases} \tag{4}$$

where

$$\begin{aligned} t_{lim} &: \text{Latency constraint}, \\ \texttt{satisfy} &: \text{System viability given constraint } t_{lim}. \end{aligned}$$

Hardware evaluation with Minstrel can help application designers to preemptively test whether the application, with its prompt and decode lengths, can meet the set latency constraints. This can allow for application-level such as changing the prompt and/or decode lengths, or relaxing constraints, etc.

C. System Optimization with Minstrel

Minstrel can evaluate impact of optimizing prefill and decode phases through hardware or software enhancements. Minstrel expresses the optimized inference latency $\hat{t}_{e2e}^{opt_{pp}, opt_d}$ as the sum of prefill and decode phases, each scaled by their respective optimization factors opt_{pp} and opt_d.

$$\hat{t}_{e2e}^{opt_{pp}, opt_d}(l_p, l_d) = opt_{pp} \cdot \hat{t}_{pp}(l_p) + opt_d \cdot \hat{t}_d(l_p, l_d), \tag{5}$$

$$\texttt{speedup}(l_p, l_d, opt_{pp}, opt_d) = \frac{\hat{t}_{e2e}(l_p, l_d)}{\hat{t}_{e2e}^{opt_{pp}, opt_d}(l_p, l_d)}, \tag{6}$$

where

$opt_{pp} < 1$: Optimization factor for prefill stage,

$opt_d < 1$: Optimization factor for decode stage,

$\hat{t}_{e2e}^{opt_{pp}, opt_d}$: Predicted optimized latency,

$\texttt{speedup}$: Predicted speedup given $l_p, l_d, opt_{pp},$ and opt_d.

D. Single Phase Optimization

We consider a special case where only one phase can be targeted for optimization. In such a case, the optimization factor for the unchanged phase can be assumed to be 1. Thus, for an application with prompt and decode lengths l_p and l_d, Minstrel evaluates the maximum benefit, max_speedup, that can be obtained through optimizing prefill or decode stages with factors as in Equation 7:

$$\texttt{max\_speedup}(l_p, l_d, opt_{pp}, opt_d) = \max \begin{cases} \texttt{speedup}(l_p, l_d, opt_{pp}, 1), \\ \texttt{speedup}(l_p, l_d, 1, opt_d). \end{cases} \tag{7}$$

Equation 7 evaluates the impact of proposed optimizations with respect to individual applications. For an expected distribution of prompt and decode lengths in a system, Minstrel can evaluate optimization of which phase accelerates the inference more and enable prioritization of such optimizations.

IV. Methodology

We evaluate Minstrel over 5 popular SLMs, Gemma 2 2b [17], Qwen1.5-1.8B-Chat [3], Phi 2 [30], Llama3.2 3B [28], [29], and Llama 3.2 1B Instruct [27], referred to as Gemma, Qwen, Phi, Llama3B, and Llama1B respectively. We use two consumer devices, an Intel i9-10900 CPU with an NVIDIA 3070 GPU [33] (referred to as *desktop*), and an NVIDIA Jetson Orin AGX as a substitute for a mobile device (referred to as *jetson*) [13]. We use the open source Llama.cpp [16] framework to measure inference latency. We profile over prompt lengths 1-1500 and generate outputs of lengths ranging from $\{0 \cdots 256, 512, 1024\}$ from the SLMs to create the performance dataset for the hybrid performance model. We choose this range for prompt lengths to match the range in llm.npu [57]. The Minstrel performance model uses linear regression to learn the learnable parameters $\{\theta_0 \cdots \theta_4\}$.

Generalizing Inference with Synthetic Prompts: While benchmarks with meaningful prompts are required to measure accuracy of a model's inference, the computation involved in an inference depends on the input length for computing prefill and filling the KV-cache, and on the output length for the number of autoregressive iterations. Hence, for an application, the effective prompt length (after considering any reuse of KV-cache from previous inference/common prompt prefixes) and the expected output length can be mapped to synthetic inference with the same prompt and decode lengths.

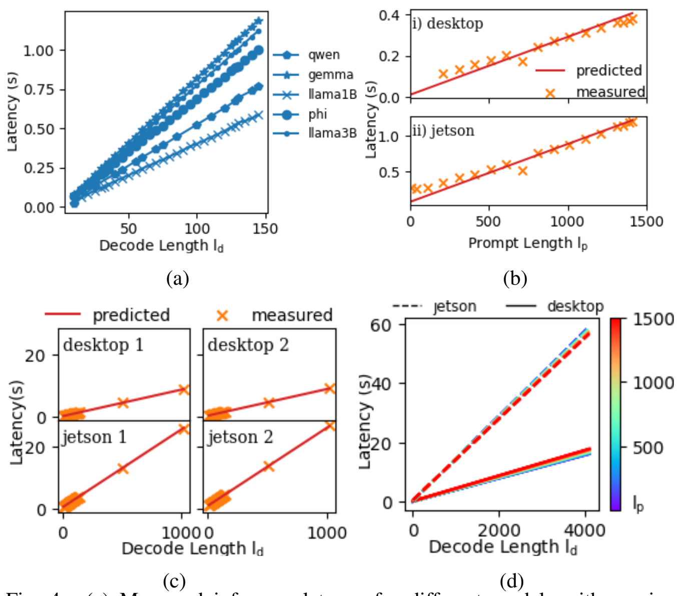

Fig. 4. (a) Measured inference latency for different models with varying decode lengths for fixed prompt length of 32, (b) Linear increase in inference latency with increase in prompt length for Llama3B on *i) desktop* and *ii) jetson*, (c) Predicted and measured inference latencies for Llama3B on *desktop* and *jetson* for 2 prompt sizes, and (d) Predicted inference latency for a sweep of prompt and decode sizes with Llama1B on *desktop* and *jetson*.

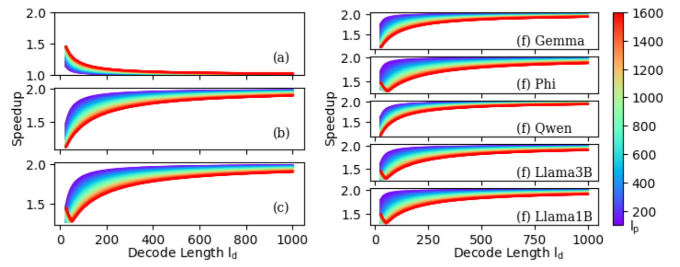

Fig. 5. Y-axis shows predicted speedups of tasks with different decode lengths l_d (on X-axis) and prompt sizes l_p (denoted by the color-bar) by optimizing (a) only prefill phase and (b) only decode phase of Llama1B. (c) shows the max curve for (a) and (b) denoting the trade-off between optimizations. (d-h) show the optimization tradeoffs observed in 5 models.

V. Results

A. Latency Analysis

Figure 4a shows the measured inference latencies for the 5 SLMs for a fixed prompt length and varying decode lengths l_d on *desktop*. The X-axis denotes prompt length and Y-axis denotes inference latency in seconds. We observe that, for all the 5 models, the inference latency varies linearly with increase in decode length, as captured in Equation 2.

Figures 4b and c show the measured inference latency as well as Minstrel's latency prediction on both hardware configurations, *desktop* and *jetson*, for Llama3B. In Figure 4b, the output length is fixed (to 10) and prompt lengths are varied from 1 to 1500. We confirm linear scaling of latency with increase in prompt length, and that Minstrel is able to predict the expected latencies. In Figure 4c, desktop 1 and jetson 1 provide the measured and predicted inference latencies, on *desktop* and *jetson* respectively, for multiple decode lengths with fixed prompt size of 700; similarly, desktop 2 and jetson 2 provide the measured and predicted inference latencies on *desktop* and *jetson* respectively for fixed prompt size of 1400. Once again, we observe that Minstrel's linear model reasonably captures the inference latency trends.

Finally, Figure 4d shows the predicted latencies from the performance model for Llama1B on *jetson* and *desktop*, for a sweep of prompt and decode lengths. The color-bar provides an indication of prompt lengths, with blue denoting shortest prompts and red denoting the longest. The solid lines show the predicted latencies for *desktop*, while the dashed lines denote the predicted latencies for jetson. We observe that the slope in *jetson* is higher than *desktop*. This indicates that the performance gap between *jetson* and *desktop* is narrow for short decode lengths, and widens with increase in decode length. With Figures 4a–d, Minstrel can test whether *jetson* and *desktop* can `satisfy` (Equation 4) an inference task with t_{lim} set to, e.g., 5 seconds for different tasks. Thus, Minstrel can not only provide a tight latency estimation given an application's characteristics, but also guide the range of prompt and decode lengths feasible for an application targeting different hardware.

B. Case Study: Single Phase Optimization

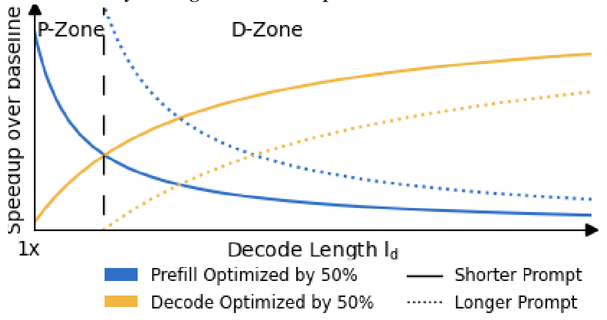

Fig. 6. Speedups trends on optimizing prompt and decode phases for different decode and prompt lengths. X-axis represents variation in decode lengths, solid and dotted lines represent short and long prompt lengths.

We use `speedup` and `max_speedup` from Equations 6 and 7 to predict the impact of independently optimizing prefill and decode phases for Llama1B in Figures 5a–c, where the X-axis denotes decode length l_d, and Y-axis denotes speedup observed over the baseline. For this evaluation, we choose $opt_{pp} = opt_d = 0.5$.

Figure 5a shows the Minstrel prediction for optimizing prefill phase by opt_{pp}. We observe that, for all prompt lengths, optimizing prefill phase sees diminishing returns with increase in decode length. Applications with small output lengths ($<$ 100) and/or long prompt lengths ($>$ 1500) are noted to benefit from the prefill optimization. However, for decode length of $\geq$ 100 tokens, we observe that the benefit of optimizing prefill phase diminishes for all observed prompt lengths.

Figure 5b shows Minstrel's prediction for optimizing decode phase by opt_d. We observe that applications with short prompt lengths ($<$ 1500) benefit from optimizing decode phase even at short decode lengths. On the other hand, applications with longer prompts the benefit from the optimization at longer decode lengths ($\geq$ 100 tokens).

Figure 5c plots the `max_speedup` from Equation 7 for prompt and decode lengths in Figures 5a and b. We observe that Figure 5c first demonstrates a diminishing speedup trend corresponding to the speedup obtained from prefill optimization in Figure 5a, followed by a flattened trend corresponding to the speedup obtained from decode optimization in Figure 5b. We observe that all workloads, irrespective of prompt sizes, demonstrate similar trends of benefiting from prefill optimization for short decode lengths. Prefill optimization, although resulting in diminishing speedup with increase in decode length, remains the dominating optimization till a certain decode length ($\leq$ 100 in Figure 5c). For longer decode lengths ($>$ 100 in Figure 5c), we observe that the applications benefit more from decode optimizations.

Finally, Figures 5d-h summarize the `max_speedup` prediction trends for all 5 models evaluated, Gemma, Phi, Qwen, Llama3B, and Llama1B respectively (Figures 5c and h are identical). We observe that all models in Figures 5d-h demonstrate similar behavior, with a diminishing trend of benefits from prefill optimization for small decode lengths followed by a decode dominated range of decode lengths. Thus, depending on model, prompt, and decode lengths, an application would be better suited for optimization to different phases.

Figure 6 generalizes the trends in Figure 5 of optimizing just prefill or decode phases, shown in blue and yellow respectively. The X-axis denotes decode length l_d, and Y-axis denotes speedup observed over baseline. In general, the application-space with different prompt and decode lengths can be divided into two zones: a prefill dominated P-Zone, and a D-Zone where, primarily, optimization to decode phase impacts speedup. Determining the zone can be crucial to identify the dominating phase for optimization and enable systematic optimization efforts. By identifying these zones, Minstrel is the first step in capturing impact of application characteristics on SLM inference optimization on the edge.

C. Discussion

Minstrel's linear model would not capture the quadratic scaling of attention computation for very large prompt lengths. However, for SLM inference use-cases covered by literature [20], [57], the sequence lengths are not very long. Thus for the purposes of predicting SLM inference latency, Minstrel provides reasonably good accuracy. Further, prior work has also noted similar linear increase in LLM inference latency with increase in prompt lengths [37].
The Minstrel model is limited to capture the impact of change to transformer blocks as an optimization factor. It may not be able to capture optimizations that are outside of the transformer blocks, such as the performance impact of speculative decoding [6], [23] with draft networks which can improve decode throughput. In such a case, however, Minstrel can still be used to provide upper and lower bounds for the benefits of speculation.
Energy is a significant factor for edge inference. Currently, Minstrel does not provide any insights into energy consumed by different configurations. For our future work, we plan to augment Minstrel to predict energy consumption of SLM inference.

VI. Conclusion and Future Work

In this work, we present Minstrel, a framework to determine application-aware predictions of inference latency and the impact of optimizations. Minstrel introduces a hybrid empirical and analytical performance model for SLM inference on edge devices. Minstrel shows that for certain applications, naive acceleration of prefill phase has no impact on performance, and optimizing decode phase would be more valuable. Using Minstrel, we observe that the application space can be divided into P-Zone and D-Zone indicating prefill and decode dominated regions respectively and allow targeted optimization efforts for improving inference latency. Minstrel currently captures empirical hardware characteristics of a desktop and an NVIDIA Jetson device. In the future, we plan on extending our characterization to mobile SoCs. We also plan to validate Minstrel with hardware optimizations to improve prefill and decode phases. We also plan to introduce an energy-performance-offload tradeoff with Minstrel and increasing the solution space.

References

[1] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[2] M. Adnan, A. Arunkumar, G. Jain, P. J. Nair, I. Soloveychik, and P. Kamath, "Keyformer: KV Cache reduction through key tokens selection for Efficient Generative Inference."

[3] Alibaba, "Qwen/Qwen1.5-1.8B-Chat," https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat-GGUF/, [Accessed 29-04-2025].

[4] D. Boiarshinov, J. Guajardo, and G. Lanning, "Providing LLM-generated Point of Interest Description Based on Gaze Tracking," 2024.

[5] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[6] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads," Jan. 2024, arXiv:2401.10774 [cs]. [Online]. Available: http://arxiv.org/abs/2401.10774

[7] Y. Chai, M. Kwen, D. Brooks, and G.-Y. Wei, "FlexQuant: Elastic Quantization Framework for Locally Hosted LLM on Edge Devices," Jan. 2025, arXiv:2501.07139 [cs]. [Online]. Available: http://arxiv.org/abs/2501.07139

[8] C.-M. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, and Z. Liu, "Chateval: Towards better llm-based evaluators through multi-agent debate," 2023. [Online]. Available: https://arxiv.org/abs/2308.07201

[9] L. Chen, D. Feng, E. Feng, Y. Wang, R. Zhao, Y. Xia, H. Chen, and P. Xu, "HeteroLLM: Accelerating Large Language Model Inference on Mobile SoCs with Heterogeneous AI Accelerators," Computer Science and Mathematics, preprint, Jan. 2025. [Online]. Available: https://www.preprints.org/manuscript/202501.0901/v1

[10] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *International Conference on Learning Representations (ICLR)*, 2024.

[11] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[12] N. Dey, G. Gosal, Zhiming, Chen, H. Khachane, W. Marshall, R. Pathria, M. Tom, and J. Hestness, "Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster," 2023. [Online]. Available: https://arxiv.org/abs/2304.03208

[13] M. Ditty, "Nvidia orin system-on-chip," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–17.

[14] X. L. Dong, S. Moon, Y. E. Xu, K. Malik, and Z. Yu, "Towards Next-Generation Intelligent Assistants Leveraging LLM Techniques," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Long Beach CA USA: ACM, Aug. 2023, pp. 5792–5793. [Online]. Available: https://dl.acm.org/doi/10.1145/3580305.3599572

[15] B. Gao, Z. He, P. Sharma, Q. Kang, D. Jevdjic, J. Deng, X. Yang, Z. Yu, and P. Zuo, "Cost-efficient large language model serving for multi-turn conversations with cachedattention," 2024. [Online]. Available: https://arxiv.org/abs/2403.19708

[16] G. Gerganov, "llama.cpp," 2025. [Online]. Available: https://github.com/ggml-org/llama.cpp

[17] Google, "google/gemma-2-2b," https://huggingface.co/google/gemma-2-2b, [Accessed 29-04-2025].

[18] Google. (2024) Gemini. Google. [Online]. Available: https://gemini.google.com/u/1/app

[19] C. Irugalbandara, A. Mahendra, R. Daynauth, T. K. Arachchige, J. Dantanarayana, K. Flautner, L. Tang, Y. Kang, and J. Mars, "Scaling down to scale up: A cost-benefit analysis of replacing openai's llm with open source slms in production," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2024, pp. 280–291.

[20] C. Irugalbandara, A. Mahendra, R. Daynauth, T. K. Arachchige, J. Dantanarayana, K. Flautner, L. Tang, Y. Kang, and J. Mars, "Scaling Down to Scale Up: A Cost-Benefit Analysis of Replacing OpenAI's LLM with Open Source SLMs in Production," Apr. 2024, arXiv:2312.14972 [cs]. [Online]. Available: http://arxiv.org/abs/2312.14972

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[22] 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," Sep. 2023, arXiv:2309.06180 [cs]. [Online]. Available: http://arxiv.org/abs/2309.06180

[23] Y. Leviathan, M. Kalman, and Y. Matias, "Fast Inference from Transformers via Speculative Decoding," May 2023, arXiv:2211.17192 [cs]. [Online]. Available: http://arxiv.org/abs/2211.17192

[24] Z. Li, C. Gebhardt, Y. Inglin, N. Steck, P. Streli, and C. Holz, "SituationAdapt: Contextual UI Optimization in Mixed Reality with Situation Awareness via LLM Reasoning," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. Pittsburgh PA USA: ACM, Oct. 2024, pp. 1–13. [Online]. Available: https://dl.acm.org/doi/10.1145/3654777.3676470

[25] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi, L. Lai, and V. Chandra, "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases."

[26] K. Mei, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang, "Aios: Llm agent operating system," *arXiv e-prints, pp. arXiv–2403*, 2024.

[27] Meta, "meta-llama/Llama-3.2-1B-Instruct," https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct, [Accessed 29-04-2025].

[28] Meta, "meta-llama/Llama-3.2-3B-Instruct," https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct, [Accessed 29-04-2025].

[29] Meta, "Llama 3.2: Revolutionizing edge ai and vision with open, customizable models," 2024. [Online]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[30] Microsoft, "microsoft/phi-2," https://huggingface.co/microsoft/phi-2, [Accessed 29-04-2025].

[31] J. Mok, M. Kachuee, S. Dai, S. Ray, T. Taghavi, and S. Yoon, "LLM-based Frameworks for API Argument Filling in Task-Oriented

Conversational Systems," Jun. 2024, arXiv:2407.12016 [cs]. [Online]. Available: http://arxiv.org/abs/2407.12016

[32] N. Nayak, X. Wu, T. O. Odemuyiwa, M. Pellauer, J. S. Emer, and C. W. Fletcher, "Fusemax: Leveraging extended einsums to optimize attention accelerator design," *arXiv preprint arXiv:2406.10491*, 2024.

[33] Nvidia, "Geforce rtx 3070 user guide," https://www.nvidia.com/content/geforce-gtx/GEFORCE_RTX_3070_USER_GUIDE_v02.pdf, [Accessed 29-04-2025].

[34] H. Oh, K. Kim, J. Kim, S. Kim, J. Lee, D.-s. Chang, and J. Seo, "ExeGPT: Constraint-Aware Resource Scheduling for LLM Inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla CA USA: ACM, Apr. 2024, pp. 369–384. [Online]. Available: https://dl.acm.org/doi/10.1145/3620665.3640383

[35] OpenAI. (2024) Chatgpt. OpenAI. [Online]. Available: https://chat.openai.com/

[36] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "Memgpt: Towards llms as operating systems," *arXiv preprint arXiv:2310.08560*, 2023.

[37] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[38] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, "T3: Transparent Tracking & Triggering for Fine-grained Overlap of Compute & Collectives," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla CA USA: ACM, Apr. 2024, pp. 1146–1164. [Online]. Available: https://dl.acm.org/doi/10.1145/3620665.3640410

[39] D. F. Perez-Ramirez, D. Kostic, and M. Boman, "Castillo: Characterizing response length distributions of large language models," 2025. [Online]. Available: https://arxiv.org/abs/2505.16881

[40] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently Scaling Transformer Inference," Nov. 2022, arXiv:2211.05102 [cs]. [Online]. Available: http://arxiv.org/abs/2211.05102

[41] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face," 2023.

[42] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Re, I. Stoica, and C. Zhang, "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Jul. 2023, pp. 31 094–31 116, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v202/sheng23a.html

[43] Y. Song, Z. Mi, H. Xie, and H. Chen, "PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU," Dec. 2023, arXiv:2312.12456 [cs]. [Online]. Available: http://arxiv.org/abs/2312.12456

[44] V. Srivatsa, Z. He, R. Abhyankar, D. Li, and Y. Zhang, "Preble: Efficient distributed prompt scheduling for llm serving," *arXiv preprint arXiv:2407.00023*, 2024.

[45] Y. Tang, F. Liu, Y. Ni, Y. Tian, Z. Bai, Y.-Q. Hu, S. Liu, S. Jui, K. Han, and Y. Wang, "Rethinking Optimization and Architecture for Tiny Language Models," Feb. 2024, arXiv:2402.02791 [cs]. [Online]. Available: http://arxiv.org/abs/2402.02791

[46] G. Team, "Gemini: A Family of Highly Capable Multimodal Models," Dec. 2023, arXiv:2312.11805 [cs]. [Online]. Available: http://arxiv.org/abs/2312.11805

[47] O. Thawakar, A. Vayani, S. Khan, H. Cholakal, R. M. Anwer, M. Felsberg, T. Baldwin, E. P. Xing, and F. S. Khan, "MobiLlama: Towards Accurate and Lightweight Fully Transparent GPT," Feb. 2024, arXiv:2402.16840 [cs]. [Online]. Available: http://arxiv.org/abs/2402.16840

[48] 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.

[49] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[50] H.-R. Tsai, S.-K. Chiu, and B. Wang, "GazeNoter: Co-Piloted AR Note-Taking via Gaze Selection of LLM Suggestions to Match Users' Intentions," Jul. 2024, arXiv:2407.01161 [cs]. [Online]. Available: http://arxiv.org/abs/2407.01161

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] Y. Wei, Z. Wang, Y. Lu, C. Xu, C. Liu, H. Zhao, S. Chen, and Y. Wang, "Editable scene simulation for autonomous driving via collaborative llm-agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024.

[53] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "AutoDroid: LLM-powered Task Automation in Android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 543–557. [Online]. Available: https://doi.org/10.1145/3636534.3649379

[54] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation framework," *arXiv preprint arXiv:2308.08155*, 2023.

[55] 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*. PMLR, Jul. 2023, pp. 38 087–38 099, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v202/xiao23c.html

[56] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han, "Duoattention: Efficient long-context llm inference with retrieval and streaming heads," 2024. [Online]. Available: https://arxiv.org/abs/2410.10819

[57] D. Xu, H. Zhang, L. Yang, R. Liu, G. Huang, M. Xu, and X. Liu, "Fast On-device LLM Inference with NPUs," Dec. 2024, arXiv:2407.05858 [cs]. [Online]. Available: http://arxiv.org/abs/2407.05858

[58] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, and Z. Ling, "On-Device Language Models: A Comprehensive Review," Sep. 2024, arXiv:2409.00088 [cs]. [Online]. Available: http://arxiv.org/abs/2409.00088

[59] S. Xu, Y. Wei, P. Zheng, J. Zhang, and C. Yu, "LLM enabled generative collaborative design in a mixed reality environment," *Journal of Manufacturing Systems*, vol. 74, pp. 703–715, Jun. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0278612524000967

[60] Z. Xue, Y. Song, Z. Mi, L. Chen, Y. Xia, and H. Chen, "PowerInfer-2: Fast Large Language Model Inference on a Smartphone," Jun. 2024, arXiv:2406.06282 [cs]. [Online]. Available: http://arxiv.org/abs/2406.06282

[61] Y. Yao, H. Jin, A. D. Shah, S. Han, Z. Hu, Y. Ran, D. Stripelis, Z. Xu, S. Avestimehr, and C. He, "ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency," Sep. 2024, arXiv:2408.00008 [cs]. [Online]. Available: http://arxiv.org/abs/2408.00008

[62] W. Yin, M. Xu, Y. Li, and X. Liu, "LLM as a System Service on Mobile Devices," Mar. 2024, arXiv:2403.11805. [Online]. Available: http://arxiv.org/abs/2403.11805

[63] W. Yin, R. Yi, D. Xu, G. Huang, M. Xu, and X. Liu, "ELMS: Elasticized Large Language Models On Mobile Devices," Sep. 2024, arXiv:2409.09071 [cs]. [Online]. Available: http://arxiv.org/abs/2409.09071

[64] R. Yousri, Z. Essam, Y. Kareem, Y. Sherief, S. Gamil, and S. Safwat, "IllusionX: An LLM-powered mixed reality personal companion," Feb. 2024, arXiv:2402.07924 [cs]. [Online]. Available: http://arxiv.org/abs/2402.07924

[65] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A Distributed Serving System for {Transformer-Based} Generative Models," 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu

[66] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee, Y. Yan, B. Chen, G. Sun, and K. Keutzer, "LLM Inference Unveiled: Survey and Roofline Model Insights," Feb. 2024, arXiv:2402.16363 null version: 1. [Online]. Available: http://arxiv.org/abs/2402.16363

[67] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1080–1096.

[68] P. Zhang, G. Zeng, T. Wang, and W. Lu, “TinyLlama: An Open-Source Small Language Model,” Jan. 2024, arXiv:2401.02385 [cs]. [Online]. Available: http://arxiv.org/abs/2401.02385

[69] X. Zhao, B. Jia, H. Zhou, Z. Liu, S. Cheng, and Y. You, “HeteGen: Heterogeneous Parallel Inference for Large Language Models on Resource-Constrained Devices,” Mar. 2024, arXiv:2403.01164 [cs]. [Online]. Available: http://arxiv.org/abs/2403.01164

[70] Z. Zhao, S. Lou, R. Tan, and C. Lv, “An AR-assisted Human-Robot Interaction System for Improving LLM-based Robot Control,” in *2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*. Hangzhou, China: IEEE, Aug. 2024, pp. 144–149. [Online]. Available: https://ieeexplore.ieee.org/document/10673005/