MobileLLM-Flash: Latency-Guided On-Device LLM Design for Industry Scale Deployment

Hanxian Huang[1,*], Igor Fedorov[1,*], Andrey Gromov[1], Bernard Beckerman[1], Naveen Suda[1], David Eriksson[1], Maximilian Balandat[1], Rylan Conway[1], Patrick Huber[1], Chinnadhurai Sankar[1], Ayushi Dalmia[1], Zechun Liu[1], Lemeng Wu[1], Tarek Elgamal[1], Adithya Sagar[1], Vikas Chandra[1], Raghuraman Krishnamoorthi[1]

[1]Meta AI

Correspondence: hanxianhuang@meta.com, ifedorov@meta.com

Abstract

Real-time AI experiences call for on-device large language models (OD-LLMs) optimized for efficient deployment on resource-constrained hardware. The most useful OD-LLMs produce near-real-time responses and exhibit broad hardware compatibility, maximizing user reach. We present a methodology for designing such models using hardware-in-the-loop architecture search under mobile latency constraints. This system is amenable to industry-scale deployment: it generates models deployable without custom kernels and compatible with standard mobile runtimes like Executorch. Our methodology avoids specialized attention mechanisms and instead uses attention skipping for long-context acceleration.

Our approach jointly optimizes model architecture (layers, dimensions) and attention pattern. To efficiently evaluate candidates, we treat each as a pruned version of a pretrained backbone with inherited weights, thereby achieving high accuracy with minimal continued pretraining. We leverage the low cost of latency evaluation in a staged process: learning an accurate latency model first, then searching for the Pareto-frontier across latency and quality.

This yields MobileLLM-Flash, a family of foundation models (350M, 650M, 1.4B) for efficient on-device use with strong capabilities, supporting up to 8k context length. MobileLLM-Flash delivers up to **1.8**× and **1.6**× faster prefill and decode on mobile CPUs with comparable or superior quality. Our analysis of Pareto-frontier design choices offers actionable principles for OD-LLM design.

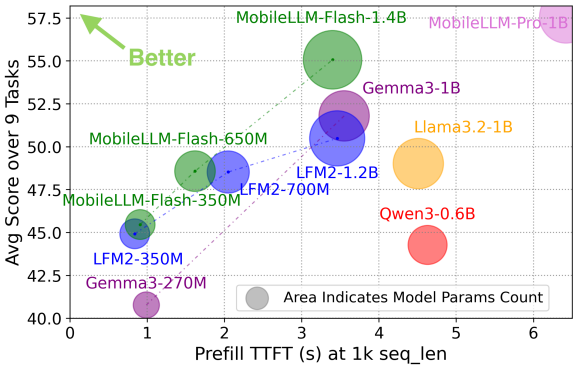

Figure 1: Comparison between MobileLLM-Flash and state-of-the-art OD-LLMs. MobileLLM-Flash achieves up to **1.8** × /**1.6**× faster prefill / decode on mobile CPUs with superior accuracy than LFM2 (Amini et al., 2025). Evaluation / benchmarking details are in Sec. 4.2.

1 Introduction

Deployment of efficient on-device large language models (OD-LLMs) on resource-constrained devices, e.g., mobile phones and smart glasses, is critical for enabling real-time AI experiences. The two distinguishing features of real-world on-device AI assistants serving large scale industry-grade traffic are: 1) They must operate **near-real-time**, with a specific emphasis on time-to-first-token (TTFT, i.e., time from receiving the request to outputting the first generated token) since the decode rate can often be hidden by streaming the model output back to the user; 2) They must be close to **general-purpose** from a runtime perspective, avoiding complex building blocks that require specialized kernels. These requirements ensure models operate across diverse software and hardware stacks (Android, iPhone, wearables).

The real-time requirement places an upper bound on TTFT, and therefore, on the number of prefill tokens. While a 4s TTFT can still yield a reasonable user experience, a 10s TTFT does not (Nielsen, 1994; Kim et al., 2026). As such, models should be optimized for the most practical use cases. We find that $\sim$ 2k tokens is a practical sweet spot where OD-LLMs can both perform useful tasks and achieve TTFT $\leq$ 4s (Tab. 6).

Many existing efficient LLM designs often op-

* Co-first authors

timize proxy metrics, such as parameter count or floating-point operations per second (FLOPs) rather than measured on-device latency, or focus on server-side optimizations (Fu et al., 2025; Gu et al., 2025; Yang et al., 2025b). However, these methods do not adequately capture the practical constraints and performance bottlenecks encountered in real-world on-device deployment, leaving a gap for methodologies grounded in empirical latency and hardware-in-the-loop evaluation. Furthermore, some designs relying on specialized kernels for efficient attention (Gu et al., 2025; Yang et al., 2025c; Dao and Gu, 2024) face barriers to large-scale adoption due to limited portability.

Our main contributions are:

- To our knowledge, the first joint architecture and attention pattern NAS for OD-LLMs, directly optimizing mobile CPU latency via a novel two-stage Bayesian optimization flow.
- A pruning-based search that inherits pretrained weights, using only 35% of the training tokens required by from-scratch training, making our hardware-in-the-loop LLM NAS practical.
- MobileLLM-Flash (350M, 650M, 1.4B) (Fig. 1), a family of deployment-ready OD-LLMs with up to 1.8× prefill and 1.6× decode speedup on mobile CPUs, runnable out of the box without specialized kernels.
- We present an analysis of the latency-accuracy Pareto frontier, deriving actionable design principles to guide future OD-LLM development. Our search reveals that interleaved skip-attention consistently outperforms sliding-window attention on mobile CPUs.

2 Related Work

OD-LLM design and architecture search. While prior works (Liu et al., 2024; Huber et al., 2025) achieve parameter efficiency, their deep-and-thin structures often fail to improve on-device latency. Recent studies (Acun et al., 2025; Gu et al., 2025; Fu et al., 2025; Yang et al., 2025b; Cowsik et al., 2025) use neural architecture search (NAS) to discover OD-LLMs, but focus on either parameter efficiency or server-side GPU optimization. LFM2 (Amini et al., 2025) introduces an edge-optimized model family based on a new block gated short convolution attention, but the underlying `conv1d` operator can perform poorly at small batch sizes or sequence lengths (Heinecke et al., 2017). Latency-aware NAS has been explored for server-scale 50–70B LLMs (Puzzle (Bercovich et al., 2025)) and diffusion models (NanoSD (Sanyal et al., 2026)), but neither searches attention patterns nor targets mobile CPU latency for on-device scale LLMs, which is the focus of our work.

Table 1: Comparison of methodologies.

Feature	MobileLLM-Flash	LFM2 (Amini et al., 2025)	Jet-Nemotron, (Gu et al., 2025) Nemotron-Flash (Fu et al., 2025)
Directly optimizes mobile CPU latency	✓	✓	✗
Unified architecture (shape+attention) search	✓	✗	✗
Efficient pruning-based search	✓	✗	✗
Compatible with any hardware / runtime	✓	✓	✗

Efficient attention alternatives and hybrid models. Sub-quadratic attention modules like Mamba2 (Dao and Gu, 2024), Gated DeltaNet (Yang et al., 2025c) and JetBlock (Gu et al., 2025), built for server-side GPUs, reduce compute and memory costs but lack on-device runtime support (e.g., Executorch). Hybrid models (Lenz et al., 2025; Glorioso et al., 2024; Ren et al., 2025; Pilault et al., 2023; De et al., 2024; Yang et al., 2025c) combines linear and quadratic attentions to improve recall and reasoning, but they typically rely on tedious manual design. Our work automates the selection of efficient attention patterns in hybrid models, utilizing runtime-supported mechanisms such as skip attention and sliding-window attention (SWA), enabling more scalable development and optimal design choices. Automatic search with sub-quadratic modules is performed in Amini et al. (2025); however, it does not adopt such hybrids, possibly due to the absence of highly optimized kernels they automatically lose by latency. A comparison of our paper with the most relevant related works is shown in Tab. 1.

Bayesian optimization (BO) is a sample-efficient framework for optimizing expensive, noisy black-box functions $f : \mathcal{S} \to \mathbb{R}$ by leveraging a surrogate model and an acquisition function to balance exploration and exploitation (Frazier, 2018). Multi-objective Bayesian Optimization (MOBO) extends BO to optimize multiple objectives $f_1, \ldots, f_M$, aiming to efficiently approximate the Pareto frontier with respect to a user-specified reference point r. MOBO leverages specialized acquisition functions, such as Noisy Expected Hypervolume Improvement (NEHVI), to guide the search towards solutions that maximize the expected improvement

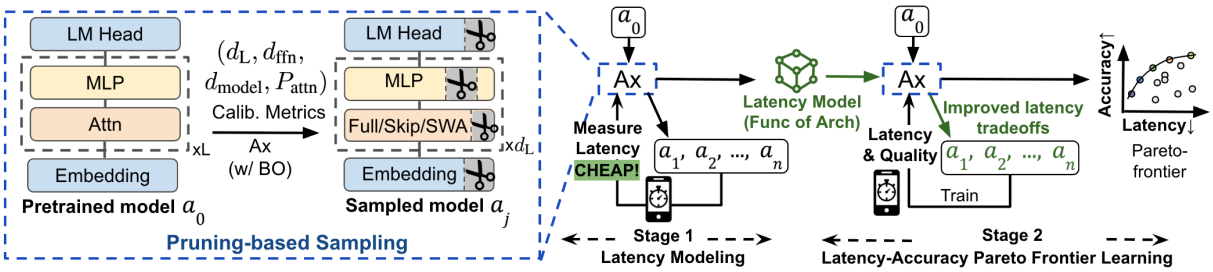

Figure 2: Overview of our two-stage OD-LLM design. We jointly search the architecture and attention pattern by pruning a pretrained model a_0 (Sec. 3.3) using Bayesian Optimization (BO) with Ax. In Stage 1, we sample pruned architectures and measure their latency on phone to cheaply learn a latency model. In Stage 2, leveraging the latency model, we efficiently search the space to generate the Accuracy-Latency Pareto-frontier (Sec. 3.4).

in hypervolume (Daulton et al., 2021; Eriksson et al., 2021).

3 Latency-Aware OD-LLM Design

Our methodology addresses the challenge of designing OD-LLMs for resource-constrained devices. We target the most common hardware platform and *maximize model quality while minimizing TTFT*. Fig. 2 provides an overview.

3.1 Latency-Aware Optimization

We search candidate architectures $a \in \mathcal{A}$ for a configuration that maximizes model quality $Q(a)$ while meeting a constraint on our primary latency metric, TTFT, $T_{\text{prefill}}(a) < \tau_{\text{prefill}}$. We use loss as a proxy for $Q(a)$, based on the established findings that pretraining loss reliably predicts downstream quality (Kaplan et al., 2020; Hoffmann et al., 2022). The threshold τ_{prefill} is product-dependent and may change as the use-case evolves; because it is not known *a priori*, we optimize both Q and T_{prefill} jointly. Because improvements in latency often reduce quality, and vice versa, we target the Pareto frontier where no solution can improve one objective without worsening another. This frontier represents the set of optimal tradeoffs between the objectives as shown in Fig. 3. An optimal point can be selected from this frontier given any τ_{prefill}.

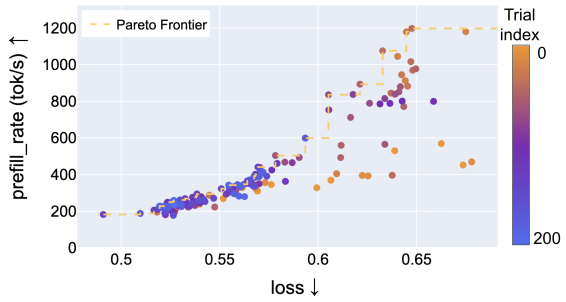

Figure 3: A latency-loss Pareto-frontier

3.2 Evaluating Model Configurations

To reduce the cost of the search relative to our target full training budget of 500B tokens, we train each candidate architecture (obtained by pruning the base model) for only 2.6B tokens. Empirically, we find that by $\sim$ 2.6B tokens the relative ordering of architectures is already predictive of their ordering after the full 500B-token run (Fig. 4). We refer to this early-but-predictive ordering as a *stable ranking*. Due to resource constraints, we do *not* sweep the optimizer hyperparameters.

We emphasize that our focus is the on-device latency, and it is motivated by practical considerations. While total parameter counts and total FLOPs per token are important metrics, they do not fully determine the on-device latency. To be more concrete, in Tab. 2 we present the Kendall tau correlation coefficient between these quantities *for our search space* across 100 architectures exported with Executorch on a Samsung Galaxy S25 for 2k sequence length. See Appendix A for visual representation of these correlations.

Table 2: Correlation coefficients between real measured prefill / decode speed vs. parameter count / FLOPs.

Kendall tau correlation ((Kendall, 1938))*	prefill	decode
#params vs. latency per tok	0.40	0.40
FLOPs vs. latency per tok	0.46	0.55

* The closer to 1, the higher correlation.

Key Insight-1 Model parameter count and FLOPs are suboptimal proxies for latency; hardware-in-the-loop optimization is necessary for accurate on-device latency improvements.

3.3 Hybrid Architecture Search Space

Structured Pruning-based Search: Instead of training candidates from scratch, we obtain candidates by pruning a larger pretrained model and

inheriting its weights. Our approach is similar to weight-sharing approaches (Fedorov et al., 2022; Cai et al., 2019; Kusupati et al., 2022) as the cost of training is amortized. However, we train candidates in isolation and are therefore not subject to the inductive bias of weight-sharing approaches, which assume that all candidates can be simultaneously trained in a nested fashion. We can think of a pruned pretrained model as a more data-efficient initialization for the architecture at hand. We observe empirically that the rank ordering of models trained from scratch aligns with that of pruned models (with the same architecture and inherited weights) under continued pretraining (CPT), with a Kendall tau correlation of 0.74 across 20 candidates. Despite this alignment, CPT achieves a significantly lower loss and approaches final convergence values much faster than random initialization. Furthermore, we need fewer tokens to get to a stable ranking of models; as shown in Fig. 4, the candidate ranking established at 10k steps matches the ranking at 120k steps.

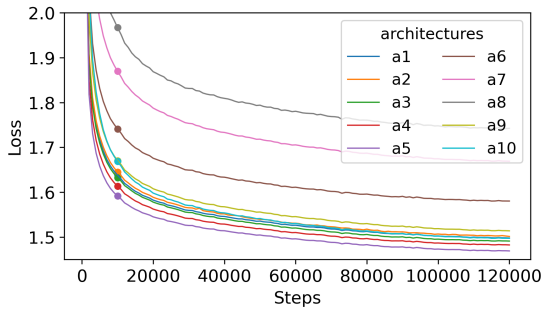

Figure 4: Pruned model loss evolution.

Pruning can also be viewed as a natural method to efficiently explore the architecture search-space $\mathcal{S}$, consisting of the following parameters: number of transformer layers d_{L}, feed-forward network (FFN) hidden dimension d_{ffn}, residual stream dimension d_{model}, and efficient attention pattern $\mathbf{P}_{\text{attn}}$ (i.e., attention type for each transformer layer).

We employ activation energy-based metrics, measured on a small calibration dataset to decide what to prune (similar to (Fedorov et al., 2024)). Specifically, the hidden size and MLP metrics quantify the average activation L2 norm magnitude (energy) across batch and sequence dimensions: FFNMetric $= \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|$ and ModelDimMetric $= \frac{1}{N}\sum_{i=1}^{N}\|\text{LN}(\mathbf{x}_i)\|$, while the layer metric captures the functional transformation energy via cosine similarity between input and output activations (as in Gromov et al. (2024)): LayerMetric $= 1 - \frac{1}{N}\sum_{i=1}^{N}\frac{\mathbf{x}_i \cdot \mathbf{x}_{i+1}}{\|\mathbf{x}_i\|\|\mathbf{x}_{i+1}\|}$, where the $\mathbf{x}_i$ is the input vector at position i, N is the total number of positions (batch size $\times$ sequence length), and LN is LayerNorm (Ba et al., 2016). Guided by the the FFNMetric and ModelDimMetric, we structurally prune the FFN hidden and residual stream dimensions for all decoder layers to the same target size d_{ffn} and d_{model} in block units (e.g., 128). The setup of block size makes the resulting architecture compatible with group-wise quantization in post-training. We rank all decoder layers by LayerMetric and remove those with the lowest contribution until d_{L} layers remain. After pruning, zeroed blocks and layers are removed, and the remaining architecture is concatenated into a compact and dense checkpoint.

Key Insight-2 Pruning offers a more data-efficient approach to architecture search than training candidates from scratch.

Efficient Attention Pattern: Although many efficient attention mechanisms exist (Sec. 2), most prior work targets server-side GPU optimization. Our focus is on real-world, on-device deployment, requiring each candidate to be compiled and benchmarked within production-grade deployment stacks such as Executorch. This ensures our evaluation reflects practical constraints and operational realities of mobile and edge devices. Consequently, we restrict our search to efficient attention operations that are natively supported in Executorch, specifically (1) skip attention, which allows certain layers to bypass attention computation; (2) global attention and (3) sliding window attention (SWA) (Child et al., 2019). Both global attention and SWA use grouped query and QK-Norm.

Formally, the architecture space $\mathcal{A}$ consists of the model architecture and attention pattern $\mathbf{P}_{\text{attn}}$, where $\mathbf{P}_{\text{attn}} =< p_1, p_2, \ldots, p_{\text{L}} >$, $i \in \text{L}$, p_i specifies the attention type for the transformer layer i in a model with L layers. Using the importance metrics, we define a formal operator $\text{Prune}(\cdot) : \mathcal{S} \rightarrow \mathcal{A}$. This operator takes $s = (d_{\text{L}}, d_{\text{ffn}}, d_{\text{model}}, \mathbf{P}_{\text{attn}})$ and calibration metrics (FFNMetric, ModelDimMetric, LayerMetric) as input, and produces a unique architecture $a \in \mathcal{A}$ that optimizes the activation metrics. $\mathcal{A} = \{a \mid a = \text{Prune}(\text{Calib. Metrics}, s)\}$. Our search space (with $\sim 70B$ possible options) is shown in Tab. 3.

Table 3: Search space $\mathcal{S}$.

Parameters	Parameter Choices
d_{L}	$\{10, 11, 12, 13, 14, 15, 16\}$
d_{ffn}	$\{2048, 2304, 2560, \ldots 8192\}$
d_{model}	$\{1024, 1152, 1280 \ldots 2048\}$
p_i	{full_attn, SWA, skip_attn}

3.4 Optimization Strategy

We leverage Bayesian optimization (BO) with the Ax platform (Olson et al., 2025) to optimize for the latency–quality Pareto frontier by searching over $\mathcal{S}$. We use a *two-stage BO* approach to leverage the fact that latency is nearly instantaneous while model quality requires substantial training resources. In the first stage, we use Sobol quasi-random sampling (Sobol', 1967) to densely cover the latency landscape of $\mathcal{S}$, followed by BO. Since latency evaluation is significantly cheaper than quality evaluation, we collect ~800 on-device latency measurements to build a high-quality Gaussian Process surrogate, achieving a cross-validation $R^2 = 0.97$. This allows the second stage to rely on predicted latency, focusing expensive quality evaluations on architectures in latency-favorable regions. The second stage optimizes both objectives, accelerating the multi-objective search by focusing expensive model-quality evaluations on regions more likely to exhibit favorable latency tradeoffs. While BO methods such as HVKG (Daulton et al., 2023) support objectives with different evaluation costs, our setting assumes latency is essentially free to evaluate. We leverage the NEHVI acquisition function and set the reference point r to a loss of 0.6 and a TTFT of 4 seconds, focusing on configurations that outperform a hand-tuned baseline model. The loss threshold of 0.6 is a heuristic calibrated by training a small number of candidates and validating against downstream tasks, and based on our own rank-ordering stability analysis (Sec 3.3).

3.5 OD-LLM Tuning Principles

By analyzing the optimal candidates along the Pareto-frontier, we distill the following efficiency principles for latency-aware OD-LLM design:
(1) **Model architecture**: We find that (at a fixed parameter count) deeper models generally have lower loss and higher latency, while shallow-and-wide models have higher loss and lower latency. Yet, at sufficiently low latency it is preferable to switch to shallow models. As shown in Fig. 5, on the Pareto-frontier curve, the 30-layer models (yellow dots) achieve the best model quality with the slowest prefill speed, while the shallower architectures (dark blue dots) offer a better balance on accuracy–latency trade-off. This aligns with observations in prior work (Fu et al., 2025).

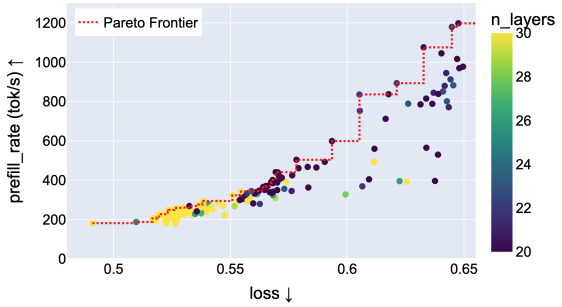

Figure 5: A Pareto curve with different model depths.

Efficiency Principle-1 For on-device deployment, shallow-and-wide models better balance accuracy and latency than deeper models.

(2) **Attention pattern**: We find, surprisingly, that our latency-guided search consistently favors skip attention over SWA, indicating that skipping a module provides a better latency-quality trade-off than SWA. We provide a high-level explanation for the inefficiency of SWA in our settings in Appendix B. Moreover, we observe that skipping too many attention layers consecutively (empirically, more than 3) degrades model quality under a fixed latency constraint and can even impair performance on harder generative tasks, as shown in Tab. 4. Consequently, we find the optimal attention pattern interleaves skip attention and global attention blocks. Therefore, we add a constraint in the final search: don't include 3 or more consecutive efficient attention types (either SWA or skip attention).

Table 4: Candidates with identical latency / identical architecture and skip attention counts can differ in performance due to the consecutive attention patterns.

	TQA	NQ	WinoG
With >3 consecutive skip attention	8.8%	2.5%	59.7%
Without >3 consecutive skip attention	33.2%	10.0%	61.2%

Efficiency Principle-2 Skip attention is more efficient than SWA; optimal patterns interleave skip attention and global attention to balance long-range modeling and latency.

4 MobileLLM-Flash: A New Family of Fast On-Device LLMs

4.1 Implementation

Table 5: MobileLLM-Pro variants and MobileLLM-Flash model architectures.

Model	Layers	d_{model}	d_{FFN}	H/KV/H_{size}	#Attn blocks
MobileLLM-Pro-1B*	30	1280	6144	20/4/64	16
MobileLLM-Pro-Shallow-1.8B	16	2048	8192	32/8/64	16
MobileLLM-Flash-350M*	12	1024	4096	32/8/64	7†
		full_attn_idxs: [0,1,3,6,7,9,11]†			
MobileLLM-Flash-650M*	13	1280	6144	32/8/64	8†
		full_attn_idxs: [0,2,3,5,7,8,9,10]†			
MobileLLM-Flash-1.4B*	16	2048	8192	32/8/64	16

† The remaining layers skip attention.
* Shared weights between input embeddings and output projection.

Our pruning-based architecture search is implemented as follows. **(1) Starting checkpoint**: As Sec. 3.5 highlights the benefits of shallow architectures, we select MobileLLM-Pro-Shallow-1.8B (Tab. 5) as the starting point – a shallow variant of MobileLLM-Pro (Huber et al., 2025) trained with the same recipe and data. We use the same pretraining data, instruction fine-tuning (IFT) data and tokenizer (202k vocabulary size) as MobileLLM-Pro (Huber et al., 2025) throughout our experiment. **(2) Calibration**: We calibrate the activation-based importance metrics (Sec. 3.3) on a 600M-token ($\sim$0.1% of the pretraining tokens) calibration set, following common practice for structured pruning (Fedorov et al., 2024; Gromov et al., 2024). **(3) Sampling and pruning**: Ax samples 8 candidates per iteration, determined by available GPU resources for parallel evaluation within each BO round. We prune the initial model based on these configurations, applying efficient attention patterns and inheriting weights to create hybrid small, dense models. **(4) Candidate evaluation**: Candidate architectures are trained for 2.6B tokens to ensure stable rank-ordering of architectures (as discussed in Sec. 3.3) and the loss is used to measure model quality. We then export models with ExecuTorch (v1.1.0) and measure TTFT on a Samsung Galaxy S25 at a 2k sequence length. **(5) Candidate selection and final training**: We run 200 trials to maximize coverage of the latency–quality tradeoff space within our compute budget. After 200 trials, we select Pareto-optimal candidates that satisfy our specific latency constraints. Then we CPT the best candidates for 500B tokens using knowledge distillation with MobileLLM-Pro-Shallow-1.8B as teacher.

Our approach only requires lightweight CPT rather than training from scratch: we use $\sim 35\%$ of the tokens used to pretrain MobileLLM-Pro. Finally, the models are IFTed for 800B tokens to prepare them for downstream tasks. During CPT, we set the sequence length to 2048, and the SWA window size to 256. During IFT, we set the sequence length to 8192, enabling the model to generalize to longer-sequence downstream tasks. Note that the total search cost is $200 \times 2.6\text{B} = 520\text{B}$ tokens across all candidates. Only 3 final Pareto-optimal candidates undergo full training: 500B tokens of CPT and 800B tokens of IFT each. This makes our method practical for OD-LLM development, compared to MobileLLM-Pro (1.6T) (Huber et al., 2025) and LFM2 (10-12T) (Amini et al., 2025).

4.2 Experimental Results

4.2.1 Model Quality

We evaluate our pre-trained models across reasoning, retrieval, and knowledge-intensive tasks: HellaSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2019), SIQA (Sap et al., 2019), *Wino*Grande (Sakaguchi et al., 2021), *ARC E*asy (Clark et al., 2018) in 0-shot settings, *T*rivia*QA* (Joshi et al., 2017) and *N*at*Q* (Kwiatkowski et al., 2019) with 5 shots and *ARC C*hallenge with 25 shots. Using lm-eval-harness (Gao et al., 2023), we report exact-match rate for TriviaQA/NQ and character-level accuracy for other tasks. We compare model accuracy and efficiency with OD-LLM baselines LFM2-350M/700M/1.2B (Amini et al., 2025), Nemotron-FLash-1B (Fu et al., 2025), Qwen3 0.6B (Yang et al., 2025a), Llama3.2-1B (Grattafiori et al., 2024) and Gemma3-270M/1B (Team et al., 2025), as shown in Tab. 6. MobileLLM-Flash achieves the highest average scores across all size regimes, establishing it as the most capable on-device model. For reference, Tab. 6 includes the unpruned MobileLLM-Pro-Shallow-1.8B base model. MobileLLM-Flash-1.4B trades a modest 1.1% average accuracy for up to $1.2\times/1.3\times$ faster prefill/decode, illustrating the favorable latency-quality tradeoff identified by our search.

We evaluate our IFTed models across knowledge (MMLU), coding (MBPP, HumanEval), rewriting (OpenRewrite), and summarization (TLDR9+) tasks (Tab. 7). All tasks are formatted as user-assistant conversations and evaluated on the final response. The results demonstrate that

Table 6: Comparison of model quality and efficiency performance across various on-device scale models

Model (Parameter Count)	HellaSwag	BoolQ	PIQA	SocialIQA	TQA	NQ	ARC-c	ARC-e	WinoG	Avg ↑	Prefill TTFT(s) ↓			Decode rate (tok/s) ↑		
											1k	2k	4k	1k	2k	4k
Gemma3 270M	41.38	58.17	68.34	39.71	15.4	4.04	29.0	57.32	53.59	40.77	0.99	3.13	5.64	123.71	82.85	52.40
LFM2 350M	49.00	**64.37**	69.48	35.01	14.97	4.96	**44.54**	**66.04**	55.96	44.92	**0.84**	**2.18**	**4.52**	147.54	96.90	63.60
MobileLLM-Flash 350M	**49.16**	62.39	**70.08**	**43.50**	**19.84**	**5.90**	38.89	63.26	**56.09**	**45.46**	0.91	2.78	5.33	**165.56**	**112.58**	**95.55**
Qwen3 0.6B	53.80	69.39	69.86	43.14	2.41	4.32	38.57	58.08	58.88	44.27	4.63	11.59	25.40	44.56	28.67	18.48
LFM2 700M	45.01	**71.38**	71.11	37.41	22.50	6.90	**49.40**	**74.60**	58.40	48.52	2.05	6.01	**8.24**	86.30	53.57	44.36
MobileLLM-Flash 650M	**54.60**	64.77	**71.82**	**45.45**	**24.49**	**7.56**	42.58	66.55	**59.35**	**48.57**	**1.62**	**3.34**	8.48	**96.64**	**85.35**	**60.74**
Nemotron-Flash 1B	45.80	–	75.41	–	–	–	41.47	74.83	59.67	–	–	–	–	–	–	–
Gemma3 1B	62.30	63.20	73.80	**48.90**	**39.80**	9.48	38.40	73.00	58.20	51.79	3.55	9.29	18.86	58.58	**43.11**	**36.29**
Llama3.2 1B	65.69	62.51	75.14	45.60	23.81	5.48	38.28	63.47	61.09	49.01	4.51	15.09	26.95	39.01	18.13	14.34
LFM2 1.2B	45.26	66.09	74.27	37.72	33.50	8.60	**52.20**	**77.90**	58.80	50.48	3.46	**8.41**	16.88	**61.14**	42.15	29.14
MobileLLM-Flash 1.4B	**66.87**	**71.07**	**75.52**	47.34	36.06	**11.83**	50.56	72.26	**64.01**	**55.06**	**3.40**	9.08	**16.50**	60.52	42.65	34.01
Base model (before pruning)																
MobileLLM-Pro-Shallow-1.8B	67.74	72.63	76.82	47.34	39.00	12.41	51.33	74.12	64.48	56.20	3.82	9.20	19.93	46.52	35.88	25.69

Table 7: Comparison of downstream task results

	MMLU	MBPP	HumanEval	Open Rewrite	TLDR9+
Gemma3-1B	29.90	35.20	41.50	–	–
Llama3.2-1B	**49.30**	**39.60**	37.80	41.60	16.80
MobileLLM-Flash 650M	35.37	33.00	**45.12**	**46.84**	14.93
MobileLLM-Flash 1.4B	47.89	35.60	**46.34**	40.10	**16.89**

MobileLLM-Flash achieves superior or comparable performance, making it as the leading on-device instruction-tuned model for assistant applications.

4.2.2 Model Efficiency

For latency evaluation, we export models using ExecuTorch (v1.1.0) and optimize them for mobile CPUs via the XNNPACK backend. All models are quantized to 4-bit weights (group size 32) and 8-bit dynamic activations, with a quantized KV cache. Nemotron-Flash-1B is excluded because its custom JetBlock operators are not supported by ExecuTorch. We conduct latency benchmarking on the Samsung Galaxy S25, a flagship Android phone powered by the Snapdragon 8 Elite chipset with an octa-core CPU and 12 GB of memory. We evaluated models at context lengths of 1k, 2k, and 4k using 4 CPU threads. Reported metrics are averaged over three runs following a warmup. The results demonstrate that MobileLLM-Flash yields $\mathbf{1.8} \times / \mathbf{1.6} \times$ faster prefill / decode speeds than LFM2, establishing MobileLLM-Flash as the fastest OD-LLM at this scale.

Our models use a 202k vocabulary size. While this increases the embedding parameters, it enhances information density per token. By representing common words more efficiently, the model requires fewer tokens to encode the same semantic content compared to models with smaller vocabulary sizes (e.g., 32k), leading to cheaper inference. Besides, this vocabulary is shared with Llama4-Scout (Meta AI, 2025), enabling effective knowledge distillation that benefits both our shallow and deep backbones (Huber et al., 2025).

MobileLLM-Flash uses only standard operators natively supported by ExecuTorch and thus deploys on any compatible backend — including Apple devices (CoreML dispatches to CPU, GPU, Apple Neural Engine) — without specialized kernels. Retargeting the search to a new platform requires only substituting the benchmark device; the methodology itself is hardware-agnostic. We note that Pareto-optimal architectures for one accelerator class (e.g., CPU) are not necessarily optimal for another (e.g., Apple Neural Engine) due to differing execution characteristics. Nevertheless, we observe that latency rankings tend to transfer well across mobile CPUs: architectures searched on a Samsung Galaxy S25 preserve their relative advantage on an iPhone 17 (Tab. 8).

Table 8: MobileLLM-Flash maintains competitive or superior TTFT across devices.

TTFT 1k (s) ↓	S25	iPhone 17
LFM2 350M	**0.84**	**1.47**
MobileLLM-Flash 350M	0.91	1.55
LFM2 700M	2.05	3.40
MobileLLM-Flash 650M	**1.62**	**2.81**
LFM2 1.2B	3.46	5.51
MobileLLM-Flash 1.4B	**3.40**	**4.96**

5 Conclusion

We introduce MobileLLM-Flash, a model family optimized for mobile latency through a novel 2-stage hardware-in-the-loop architecture search. By prioritizing shallow-and-wide structures and interleaved skip-attention patterns, we achieve state-of-the-art quality with significant speedups over strong baselines. Our methodology is compatible with Executorch and establishes a practical, scalable framework for delivering efficient, real-time AI on edge devices without specialized kernels.

Limitations

Due to the high computational cost of training candidate architectures, our Bayesian Optimization search focused exclusively on architectural parameters (depth, width, attention patterns). We did not perform a co-optimization of training hyperparameters (e.g., learning rate schedules, optimizer settings) via Ax. It is possible that specific architectural candidates could achieve higher quality with bespoke hyperparameter tuning, which was outside the scope of this study.

To ensure immediate industry-scale deployability and compatibility with standard runtimes like ExecuTorch, in this paper we did not explore novel sub-quadratic attention mechanisms (such as SSMs or linear attention variants mentioned in Sec. 2) that currently lack mature runtime support. Extending the search space to include these emerging architectures remains a direction for future work as their software support matures.

Ethical Considerations

Our work contributes to "Green AI" by focusing on efficiency. By optimizing for lower latency and smaller model sizes, MobileLLM-Flash reduces the computational energy required for inference. Furthermore, our pruning-based search method is data-efficient, requiring significantly fewer (only 35%) training tokens (and thus less GPU energy) to discover optimal architectures compared to training candidates from scratch.

References

Bilge Acun, Prasoon Sinha, Newsha Ardalani, Sangmin Bae, Alicia Golden, Chien-Yu Lin, Meghana Madhyastha, Fei Sun, Neeraja J. Yadwadkar, and Carole-Jean Wu. 2025. Composer: A search framework for hybrid neural architecture design. *Preprint*, arXiv:2510.00379.

Alexander Amini, Anna Banaszak, Harold Benoit, Arthur Böök, Tarek Dakhran, Song Duong, Alfred Eng, Fernando Fernandes, Marc Härkönen, Anne Harrington, Ramin Hasani, Saniya Karwa, Yuri Khrustalev, Maxime Labonne, Mathias Lechner, Valentine Lechner, Simon Lee, Zetian Li, Noel Loo, and 14 others. 2025. Lfm2 technical report. *Preprint*, arXiv:2511.23404.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Akhiad Bercovich, Tomer Ronen, Talor Abramovich, Nir Ailon, Nave Assaf, Mohammed Dabbah, Ido Galil, Amnon Geifman, Yonatan Geifman, Izhak Golan, Netanel Haber, Ehud Dov Karpas, Roi Koren, Itay Levy, Pavlo Molchanov, Shahar Mor, Zach Moshe, Najeeb Nabwani, Omri Puny, and 7 others. 2025. Puzzle: Distillation-based NAS for inference-optimized LLMs. In *Forty-second International Conference on Machine Learning*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Aditya Cowsik, Tianyu He, and Andrey Gromov. 2025. Towards distributed neural architectures. *arXiv preprint arXiv:2506.22389*.

Tri Dao and Albert Gu. 2024. Transformers are ssms: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Sam Daulton, Maximilian Balandat, and Eytan Bakshy. 2023. Hypervolume knowledge gradient: A lookahead approach for multi-objective Bayesian optimization with partial information. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7167–7204. PMLR.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2021. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. *CoRR*, abs/2105.08195.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud

Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. 2024. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *Preprint*, arXiv:2402.19427.

David Eriksson, Pierce I-Jen Chuang, Samuel Daulton, Peng Xia, Akshat Shrivastava, Arun Babu, Shicong Zhao, Ahmed A Aly, Ganesh Venkatesh, and Maximilian Balandat. 2021. Latency-aware neural architecture search with multi-objective bayesian optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

Igor Fedorov, Ramon Matas, Hokchhay Tann, Chuteng Zhou, Matthew Mattina, and Paul Whatmough. 2022. Udc: Unified dnas for compressible tinyml models for neural processing units. *Advances in Neural Information Processing Systems*, 35:18456–18471.

Igor Fedorov, Kate Plawiak, Lemeng Wu, Tarek Elgamal, Naveen Suda, Eric Smith, Hongyuan Zhan, Jianfeng Chi, Yuriy Hulovatyy, Kimish Patel, Zechun Liu, Changsheng Zhao, Yangyang Shi, Tijmen Blankevoort, Mahesh Pasupuleti, Bilge Soran, Zacharie Delpierre Coudert, Rachad Alao, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024. Llama guard 3-1b-int4: Compact and efficient safeguard for human-ai conversations. *Preprint*, arXiv:2411.17713.

Peter I Frazier. 2018. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Yonggan Fu, Xin Dong, Shizhe Diao, Matthijs Van keirsbilck, Hanrong Ye, Wonmin Byeon, Yashaswi Karnati, Lucas Liebenwein, Maksim Khadkevich, Alexander Keller, Jan Kautz, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Nemotron-flash: Towards latency-optimal hybrid small language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. 2024. Zamba: A compact 7b ssm hybrid model. *Preprint*, arXiv:2405.16712.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Yuxian Gu, Qinghao Hu, Haocheng Xi, Junyu Chen, Shang Yang, Song Han, and Han Cai. 2025. Jet-nemotron: Efficient language model with post neural architecture search. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Alexander Heinecke, Evangelos Georganas, Kunal Banerjee, Dhiraj Kalamkar, Narayanan Sundaram, Anand Venkat, Greg Henry, and Hans Pabst. 2017. Understanding the performance of small convolution operations for cnn on intel architecture. In *Poster in the International Conference for High Performance Computing, Networking, Storage, and Analysis*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Patrick Huber, Ernie Chang, Wei Wen, Igor Fedorov, Tarek Elgamal, Hanxian Huang, Naveen Suda, Chinnadhurai Sankar, Vish Vogeti, Yanghan Wang, Alex Gladkov, Kai Sheng Tai, Abdelrahman Elogeel, Tarek Hefny, Vikas Chandra, Ahmed Aly, Anuj Kumar, Raghuraman Krishnamoorthi, and Adithya Sagar. 2025. Mobilellm-pro technical report. *Preprint*, arXiv:2511.06719.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

M. G. Kendall. 1938. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.

Kaeun Kim, Ghazal Shams, and Kawon Kim. 2026. From seconds to sentiments: differential effects of chatbot response latency on customer evaluations. *International Journal of Human–Computer Interaction*, 42(1):597–612.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham

Kakade, Prateek Jain, and 1 others. 2022. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Barak Lenz, Opher Lieber, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, Daniel Gissin, Daniel Jannai, Dor Muhlgay, Dor Zimberg, Edden M. Gerber, Elad Dolev, Eran Krakovsky, Erez Safahi, Erez Schwartz, and 42 others. 2025. Jamba: Hybrid transformer-mamba language models. In *The Thirteenth International Conference on Learning Representations*.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, and 1 others. 2024. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*.

Meta AI. 2025. Llama 4 maverick and scout. https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E.

Jakob Nielsen. 1994. *Usability engineering*. Morgan Kaufmann.

Miles Olson, Elizabeth Santorella, Louis C. Tiao, Sait Cakmak, David Eriksson, Mia Garrard, Sam Daulton, Maximilian Balandat, Eytan Bakshy, Elena Kashtelyan, Zhiyuan Jerry Lin, Sebastian Ament, Bernard Beckerman, Eric Onofrey, Paschal Igusti, Cristian Lara, Benjamin Letham, Cesar Cardoso, Shiyun Sunny Shen, and 2 others. 2025. Ax: A platform for adaptive experimentation. In *AutoML 2025 ABCD Track*.

Jonathan Pilault, Mahan Fathi, Orhan Firat, Christopher Pal, Pierre-Luc Bacon, and Ross Goroshin. 2023. Block-state transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Liliang Ren, Yang Liu, Yadong Lu, yelong shen, Chen Liang, and Weizhu Chen. 2025. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *The Thirteenth International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106.

Subhajit Sanyal, Srinivas Soumitri Miriyala, Akshay Janardan Bankar, Manjunath Arveti, Sowmya Vajrala, Shreyas Pandith, Sravanth Kodavanti, Abhishek Ameta, Harshit, and Amit Satish Unde. 2026. Nanosd: Edge efficient foundation model for real time image restoration. *Preprint*, arXiv:2601.09823.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

I.M Sobol'. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Mingyu Yang, Mehdi Rezagholizadeh, Guihong Li, Vikram Appia, and Emad Barsoum. 2025b. Zebra-llama: Towards extremely efficient hybrid models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. 2025c. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

A Efficiency Proxies

We present visualization of correlation coefficients between real measured prefill / decode speed vs. parameter count / FLOPs in Fig. 6, Fig. 7,Fig. 8, and Fig. 9 for various context lengths.

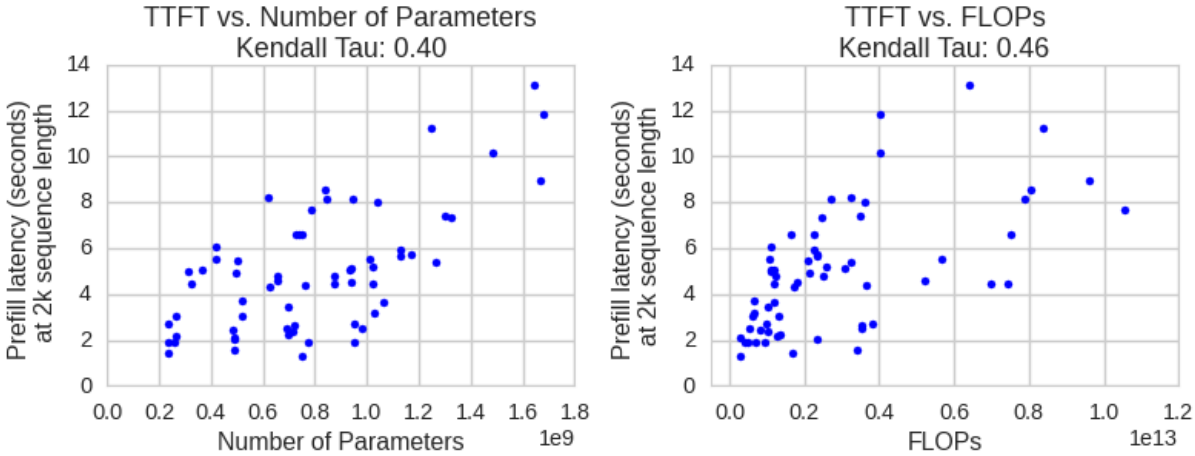

Figure 6: Kendall Tau correlation between TTFT at 2k sequence length vs. model parameter count and FLOPs.

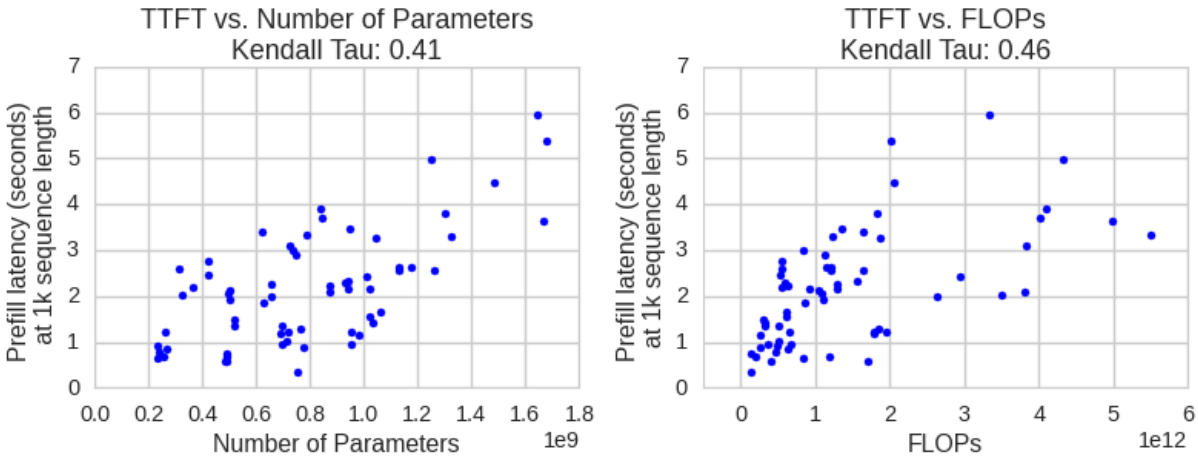

Figure 7: Kendall Tau correlation between TTFT at 1k sequence length vs. model parameter count and FLOPs.

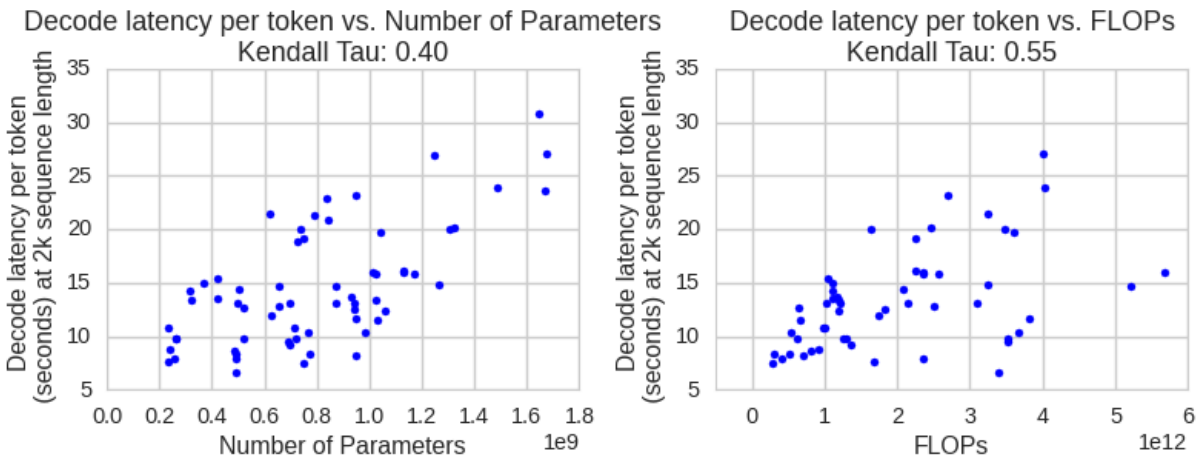

Figure 8: Kendall Tau correlation between decode latency at 2k sequence length vs. model parameter count and FLOPs.

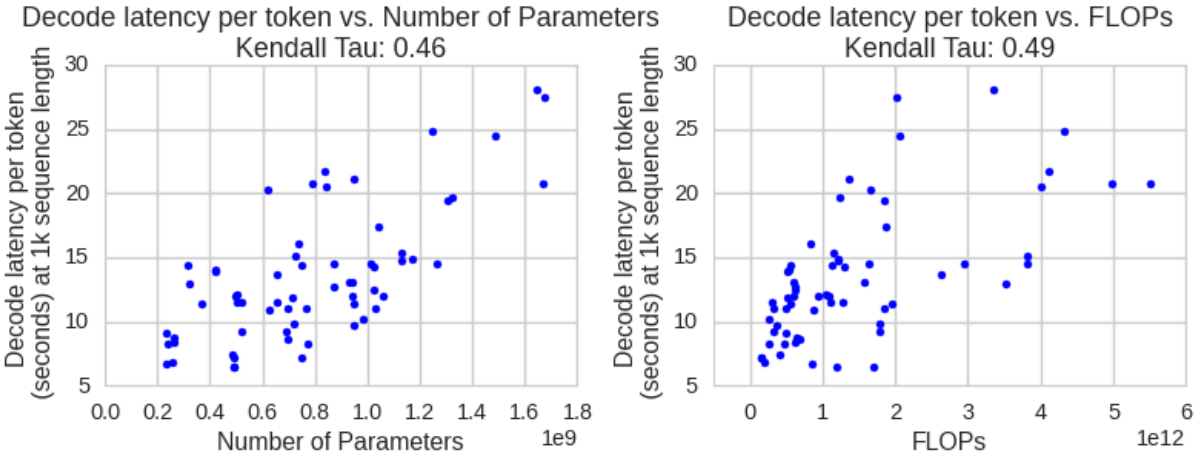

Figure 9: Kendall Tau correlation between decode latency at 1k sequence length vs. model parameter count and FLOPs.

B Explanation for the inefficiency of SWA

A high level explanation for the inefficiency of SWA in our settings is as follows. We find that a prefill chunk size of 1024 produces the lowest time-to-first-token (TTFT), making the best use of parallelism across tokens in the context window. At the same time, Executorch constrains the SWA to be greater than or equal to the prefill chunk size, such that the sliding window is lower-bounded by 1024. Therefore, when prefilling 2k tokens, only the second chunk benefits from the SWA. At the same time, the ring-buffer implementation of SWA in Executorch requires computation of the entire attention matrix, as opposed to just the lower triangular portion as done during standard attention computation. As a result, in the 2k prefill, 1024 chunk size setting, we find that the benefits of SWA are outweighed by the drawback of the slower attention calculation and the model overall achieves worse TTFT with SWA.