MobileMoE: Scaling On-Device Mixture of Experts

Yanbei Chen[†] **Hanxian Huang** **Ernie Chang** **Jacob Szwejbka**
Digant Desai **Zechun Liu** **Vikas Chandra** **Raghuraman Krishnamoorthi**
Meta AI

Abstract

Mixture-of-Experts (MoE) has become the de facto architecture for hundred-billion-parameter language models, yet its advantages at sub-billion scales for on-device deployment remain largely unexplored. To close this gap, we present **MobileMoE**, a family of on-device MoE language models with sub-billion active parameters (0.3-0.9B active and 1.3-5.3B total) that establish a new Pareto frontier for on-device LLMs. We first formulate an on-device MoE scaling law that jointly optimizes MoE architecture under mobile memory and compute constraints, identifying an on-device sweet spot – moderate sparsity with fine-grained and shared experts – that is simultaneously memory and compute-optimal. Building on the derived architectures, we train MobileMoE with a four-stage recipe covering pre-training, mid-training, instruction fine-tuning, and quantization-aware training, all on open-source datasets. Across 14 benchmarks, MobileMoE matches or exceeds leading on-device dense LLMs with 2-4$\times$ fewer inference FLOPs, and matches or surpasses the state-of-the-art MoE OLMoE-1B-7B with up to 60% fewer parameters. To bridge the last mile to mobile deployment, we provide the first efficient MoE inference on commodity smartphones with comprehensive on-device profiling. At comparable INT4 weight memory, MobileMoE-S delivers 1.8–3.8$\times$ faster prefill and 2.2–3.4$\times$ faster decode than the dense baseline MobileLLM-Pro. We publicly release all MobileMoE model checkpoints at huggingface.co/collections/facebook/mobilemoe.

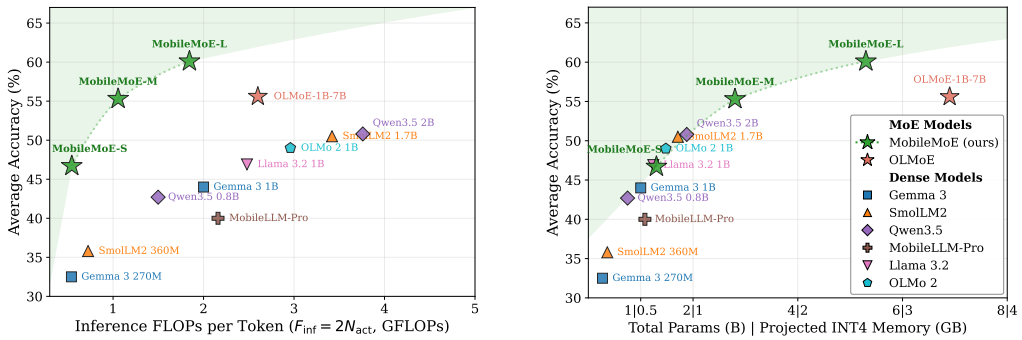
1 Introduction

Mixture-of-Experts (MoE) architectures increasingly dominate state-of-the-art Large Language Models (LLMs), as represented by both open-source models (e.g., DeepSeek V3 [1], Qwen3 MoE [2]) and proprietary frontier models (e.g., Gemini [3], Grok [4]). However, on-device LLMs remain overwhelmingly dense (e.g., MobileLLM [5], MobileLLM Pro [6]), and scaling MoE in the sub-billion active-parameter regime, where on-device LLMs typically operate, remains largely unexplored. Addressing this gap is increasingly crucial for next-generation edge AI: efficient on-device LLMs reduce reliance on cloud compute and enable low-latency, cost-effective, privacy-preserving applications on smartphones, wearables, and embodied agents.

Unlocking the potential of LLMs on edge devices requires overcoming severe compute and memory constraints. MoE architectures address these constraints through three complementary efficiencies. First, *parameter efficiency*: an MoE model expands total capacity through many expert networks while activating only a sparse fraction per token, matching the performance of a much larger dense counterpart at significantly less inference compute [2]. Second, *runtime efficiency*: sparse activation reduces inference FLOPs, lowering runtime latency and conserving mobile battery life. Third, *learning efficiency*: expert networks specialize across distinct domains (e.g., knowledge, code, math) [7], packing broad multi-task capability into one unified model. Crucially, the recent growth of smartphone DRAM in the past few years (e.g., from 4 GB on iPhone 13 to 12 GB on iPhone 17,

[†]Corresponding author. yanbeic@gmail.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) (b)

Figure 1: **MobileMoE establishes a new Pareto frontier for on-device LLMs.** Average benchmark accuracy, computed over 14 benchmarks spanning commonsense, knowledge, science, comprehension, and reasoning, is plotted against (a) per-token inference compute $F_{\text{inf}} = 2N_{\text{act}}$ (GFLOPs) and (b) total parameters N_{total} (B); in (b), x-axis tick labels show total params (B) | projected INT4 memory (GB)[1].

from 8 GB on Samsung Galaxy S21 to 12 GB, 16 GB on S25 and S25 Ultra) provides the memory headroom to host these efficient and capable sparse LLMs directly on mobile devices.

Yet the scaling methodology of on-device MoE, from architectures to training recipes and practical on-device deployment, has yet to be established. While scaling laws have long served as the north star guiding the development of dense LLMs [8, 9] and MoEs [10, 11], existing frameworks overwhelmingly focus on scaling models up to tens or hundreds of billions of parameters for deployment on cloud servers. To address practical edge constraints, we formulate a novel MoE scaling law tailored to the sub-billion active-parameter regime, providing a principled foundation to guide architectural design under joint memory and compute constraints. Building upon this scaling law, we derive **MobileMoE**, the first sub-billion-active MoE family optimized for the edge across three scales (S/M/L): 0.3B/0.5B/0.9B active parameters (1.3B/2.8B/5.3B total) with $<$3 GB INT4 weight footprints to fit in mobile DRAM.

To realize the MoE architectural advantages at scale, we design a comprehensive four-stage training recipe: pre-training, mid-training, instruction fine-tuning, and 4-bit quantization-aware training. Our pipeline explicitly addresses MoE-specific training stability and efficiency, and scales up MobileMoE training with exceptional token efficiency. With only $\sim$6T pre-training tokens, MobileMoE matches or surpasses dense baselines trained on 1.5-2$\times$ more tokens (e.g., 9T for Llama 3.2 1B [12], 11T for SmolLM2 1.7B [13]), validating the learning efficiency of MoE at the sub-billion active scale. Notably, our scaling-law-derived architecture and training recipe enable MobileMoE to establish a new Pareto frontier for on-device LLMs across 14 foundational benchmarks spanning commonsense, knowledge, science, comprehension, and reasoning (Figure 1). Smaller MobileMoE-S/M match or exceed dense baselines using 2-4$\times$ fewer inference FLOPs at comparable memory, while MobileMoE-L pushes the frontier further to state-of-the-art accuracy at sub-billion active scale. Furthermore, compared to the state-of-the-art MoE OLMoE-1B-7B [14], MobileMoE-M matches its accuracy with $\sim$60% fewer active and total parameters, while MobileMoE-L achieves much higher accuracy with 30% fewer active parameters and 23% smaller model memory footprint.

Beyond benchmark performance, we demonstrate the practical on-device runtime benefits by deploying MobileMoE on flagship smartphones: Samsung Galaxy S25 and iPhone 16 Pro. Since most existing mobile inference stacks lack native MoE support, we develop a custom fused MoE kernel to enable efficient MoE inference, providing the first MoE runtime support on commodity smartphone CPUs, with comprehensive runtime profiling across CPU and GPU backends. Powered by this kernel, at comparable INT4 weight memory, MobileMoE-S achieves 1.8–3.8$\times$ faster prefill and 2.2–3.4$\times$ faster decode than the dense baseline MobileLLM-Pro [6] while consuming up to 22% less peak RSS at 8k context. Concurrently, MobileMoE-M matches or outperforms MobileLLM-Pro on runtime with higher accuracy, while MobileMoE-L delivers substantially higher accuracy with moderate runtime cost. The consistency of this Pareto pattern on mobile devices confirms the compute and memory efficiency of MobileMoE holds on real hardware.

[1]Accuracy is reported from public model releases at full precision (BF16). In subplot (b), the x-axis shows total parameters N_{total} in billions alongside the corresponding projected INT4 weight memory, computed as $\mathcal{M}_{\text{weight}} = \frac{1}{2} N_{\text{total}}$ GB under 4-bit weight quantization that is commonly used for on-device LLM deployment (as detailed in Section 3.4, Section 4.3).

2 Related Work

On-Device LLMs enable fast, privacy-preserving, and cost-effective inference at the edge, but must operate under stringent latency and memory constraints distinct from server-side deployments. Recent works introduce compact dense models at sub-billion to few-billion scales, including MobileLLM [5, 6], SmolLM2 [13], and Gemma [15, 16], alongside latency-optimized MobileLLM-Flash [17] and Nemotron-Flash [18]. Another line of work improves hardware efficiency through NAS methods [19–23] that optimize architectures for specific deployment targets. These efforts focus predominantly on dense architectures, where scaling model quality generally requires higher active parameter counts and inference compute. We pursue MoE as a complementary path that expands model capacity at minimal per-token compute.

Mixture of Experts (MoE) offers a parameter-efficient paradigm by routing tokens to a sparse subset of specialized expert networks [24, 7, 25–28]. By decoupling total parameters from active inference compute, MoE has driven the scaling of state-of-the-art LLMs, e.g., Mixtral [29], DeepSeek-MoE [30, 1], Qwen-MoE [31, 2]. While efforts such as OLMoE [14] have explored smaller scales, the sub-billion active-parameter regime, where on-device LLMs operate efficiently [5, 6], remains unexplored under practical edge constraints.

Scaling Laws characterize power-law relationships between compute, data, and parameters [8, 9], providing a principled foundation for LLM development, covering compute-optimal parameter-data allocation [9, 12], training hyperparameters [32], learning rate schedules [33], and data mixtures [34]. Scaling laws have been extended to MoE, studying expert count [10], expert granularity [11], and expert allocation under memory constraints [35]. While existing scaling laws target server-scale LLMs, we formulate an on-device MoE scaling law to derive architecture under mobile memory and compute constraints, with an end-to-end training recipe to scale sub-billion-active MoE on devices.

3 Scaling On-Device MoE

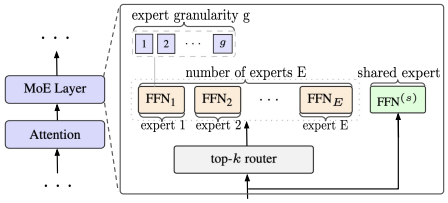
3.1 Preliminaries

Mixture-of-Experts (MoE). Consider a decoder-only transformer with n_l layers of dimension d_{model}. Each layer consists of grouped-query attention (GQA): n_h query heads, n_{kv} key-value heads, followed by a feed-forward network (FFN) of hidden dimension d_{ff}. An MoE model replaces the dense FFN with E routed expert FFNs and a top-k router that selects the k highest-scoring experts per token. State-of-the-art MoE models differ widely in architecture choices: DeepSeek-V3 [1] uses 256 fine-grained experts with top-8 routing and a shared expert, Qwen3-MoE [2] uses 128 experts with top-8 routing but no shared expert, and Mixtral [29] uses 8 coarse-grained experts with top-2 routing. These differences highlight a lack of consensus at scale on key design choices. Crucially, these choices remain largely under-explored for on-device models, where resource constraints differ fundamentally. We therefore study three factors (Figure 2, left): (i) *model sparsity* (E, k), where routed expert count E and active expert count k control the ratio of active to total parameters; (ii) *expert granularity* g, where each routed expert is split into g sub-experts of hidden dimension d_{ff}/g, yielding gE experts with gk activated experts per token; and (iii) *shared expert* s, an always-on expert that bypasses routing. Formally, the MoE layer output is $\mathbf{y} = \sum_{i \in \text{Top-}gk} \text{router}_i(\mathbf{x}) \cdot \text{FFN}_i(\mathbf{x}) + \text{FFN}^{(s)}(\mathbf{x})$.

On-Device LLMs. Existing LLMs follow practical rules of thumb in model design, e.g., GPT-3 [36] uses FFN expansion ratio $d_{\text{ff}}/d_{\text{model}} = 4$ and width-depth aspect ratio $d_{\text{model}}/n_l = 128$, while on-device LLMs (e.g., MobileLLM [5, 6]) adopt a smaller aspect ratio of approximately 40, favoring deeper architectures in the sub-billion-parameter regime. Building on this principle, we instantiate our on-device MoE models with a base backbone defined by $d_{\text{ff}}/d_{\text{model}} = 4$, $d_{\text{model}}/n_l \approx 40$, 4 key-value heads, and optimize MoE-specific choices (Figure 2, right) using our on-device scaling laws.

3.2 On-Device MoE Scaling Law

Unlike server-side deployments with abundant resources, on-device use cases (e.g., smartphones, wearables) face strict hardware constraints, requiring explicit trade-offs among performance, model size, and inference cost. We navigate these trade-offs systematically by jointly optimizing MoE architectures under device memory and compute constraints, over the design space in Figure 2.



Model	Base Arch. d_{model}	d_{ff}	n_h	n_{kv}	n_l	MoE Design E	g	Shared
Small (S)	768	3072	12	4	20	{1–32}	{1–16}	{✓, ×}
Medium (M)	1024	4096	16	4	26	{1–32}	{1–16}	{✓, ×}
Large (L)	1280	5120	20	4	32	{1–32}	{1–16}	{✓, ×}

Figure 2: **MobileMoE architectures.** *Left*: MoE model design space with three design factors: model sparsity (E, k), expert granularity g, shared experts s. *Right*: MobileMoE at three scales: Small (S), Medium (M), Large (L) with 0.3, 0.5, 0.9B active parameters, using a base architecture of expansion ratio $d_{\text{ff}}/d_{\text{model}} = 4$, aspect ratio $d_{\text{model}}/n_l \approx 40$, SwiGLU, GQA with 4 KV heads, along with variants of MoE design choices on number of experts $E \in \{1, 2, 4, 8, 16, 32\}$, fine-grained expert granularity $g \in \{1, 2, 4, 8, 16\}$, and shared expert $\{\checkmark, \times\}$.

On-Device MoE Scaling Law. Formally, we introduce a generalized on-device MoE scaling law:

$$\mathcal{L}(N_{\text{act}}, D, \hat{E}, x) = A_x \hat{E}^{\delta_x} N_{\text{act}}^{\alpha_x + \gamma_x \ln \hat{E}} + B_x \hat{E}^{\omega_x} D^{\beta_x + \zeta_x \ln \hat{E}} + c_x \tag{1}$$

where $\mathcal{L}$ is the model loss, N_{act} is active parameters, D is training data size, $\hat{E}$ is a monotonic transformation of the number of expert E which decides the total parameters N_{total} and the model sparsity: sparsity $= 1 - N_{\text{act}}/N_{\text{total}}$, and x refers to architecture choices: expert granularity g, shared experts s, which does not necessarily change the parameters N_{act}, N_{total}, and c_x is the irreducible loss. This formulation admits two *reduced forms* that recover established scaling laws as special cases.

Reduced form I ($\mathcal{L}|_x$). With architecture choice x fixed, Eq. (1) reduces to *joint MoE scaling law* [35]:

$$\mathcal{L}_x(N_{\text{act}}, D, \hat{E}) = A \hat{E}^{\delta} N_{\text{act}}^{\alpha + \gamma \ln \hat{E}} + B \hat{E}^{\omega} D^{\beta + \zeta \ln \hat{E}} + c \tag{2}$$

which absorbs x as a constant: $A{=}A_x$, $\alpha{=}\alpha_x$, $\delta{=}\delta_x$, $\gamma{=}\gamma_x$, $B{=}B_x$, $\beta{=}\beta_x$, $\omega{=}\omega_x$, $\zeta{=}\zeta_x$, $c{=}c_x$. This reduced form was derived to find *memory-optimal* expert counts in MoE, where $\hat{E}$ is a monotonic transformation of the number of experts E defined as $\frac{1}{\hat{E}} = \frac{1}{E - 1 + \left(\frac{1}{E_{\text{start}}} - \frac{1}{E_{\text{max}}}\right)^{-1}} + \frac{1}{E_{\text{max}}}$ [10].

Reduced form II ($\mathcal{L}|_{\hat{E}}$). With expert count E fixed, Eq. (1) reduces to *Chinchilla scaling law* [9]:

$$\mathcal{L}_{\hat{E}}(N_{\text{act}}, D, x) = \tilde{A}_x N_{\text{act}}^{\tilde{\alpha}_x} + \tilde{B}_x D^{\tilde{\beta}_x} + c_x \tag{3}$$

which absorbs $\hat{E}$ as constants: $\tilde{A}_x{=}A_x \hat{E}^{\delta_x}$, $\tilde{\alpha}_x{=}\alpha_x{+}\gamma_x \ln \hat{E}$, $\tilde{B}_x{=}B_x \hat{E}^{\omega_x}$, $\tilde{\beta}_x{=}\beta_x{+}\zeta_x \ln \hat{E}$. This reduced form is equivalent to standard scaling laws for finding *compute-optimal* architecture choices.

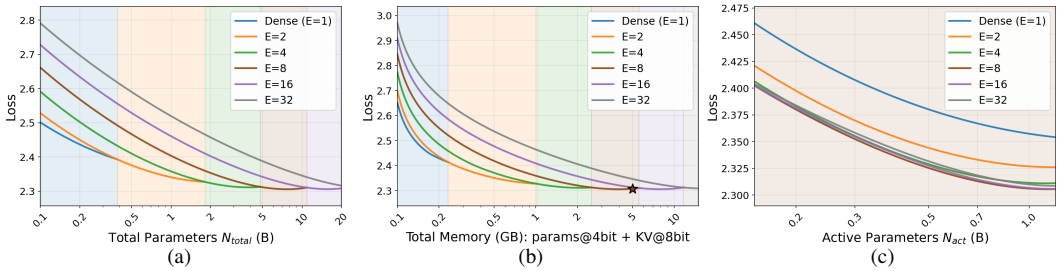
On-Device Optimization Objective. For on-device deployment, the optimization of model architecture includes both compute and memory constraints; thus, we minimize Eq. (1) subject to

$$\begin{aligned} \underset{N_{\text{act}}, D, E, x}{\arg\min} \quad & \mathcal{L}(N_{\text{act}}, D, \hat{E}, x) \\ \text{s.t.} \quad & \text{compute: } F_{\text{train}} = 6 N_{\text{act}} D \text{ (training)}; \quad F_{\text{inf}} = 2 N_{\text{act}} \text{ (per-token inference)} \\ & \text{memory: } \mathcal{M}(N_{\text{total}}, T) \leq M \text{ (device memory budget)} \end{aligned} \tag{4}$$

where F_{train} is the training compute budget in FLOPs, F_{inf} is the per-token inference compute in FLOPs (forward pass only), and M is the device DRAM budget in GB, roughly capped at 5 GB for app usage on current smartphones. The memory function $\mathcal{M}$ accounts for both total parameters N_{total} and the KV cache at context length T. Prior work on on-device LLMs has demonstrated that low-bit quantization (e.g., 4-bit weights, 8-bit KV cache) substantially reduces memory footprint while retaining model quality [6, 37]. Following this practice, we formulate the memory function as

$$\mathcal{M}(N_{\text{total}}, T) = \underbrace{\frac{b_w}{8} N_{\text{total}}}_{\mathcal{M}_{\text{weight}}} + \underbrace{\frac{b_{\text{kv}}}{8} \cdot 2 T n_l n_{\text{kv}} d_h}_{\mathcal{M}_{\text{KV cache}}}, \text{ with } \mathcal{M}(N_{\text{total}}, T) \leq M \tag{5}$$

where $\mathcal{M}_{\text{weight}}$ is the quantized model weight memory with b_w-bit precision (e.g., 4 for INT4), and $\mathcal{M}_{\text{KV cache}}$ is the KV cache memory with b_{kv}-bit precision (e.g., 8 for INT8), T is the context length, d_h is the head dimension, $\mathcal{M}(N_{\text{total}}, T)$ is a tractable *proxy* of on-device memory footprint (excluding transient activation buffers and other runtime overhead), and M is the memory budget.



(a) (b) (c)

Figure 3: **Scaling the number of experts E.** Model loss predicted by the fitted on-device scaling law (Eq. (1)) plotted against (a) total parameters N_{total}, (b) on-device memory $\mathcal{M}$ (Eq. (5)), and (c) active parameters N_{act}, under a training budget of 5×10^{20} FLOPs (our experimental regime). Each curve corresponds to one expert count E, truncated at the crossover with the next-larger E curve. Shaded regions indicate the optimal E along each axis; e.g., at $\mathcal{M}$=5 GB, the optimal E is 8, marked by the star in (b).

3.3 Finding the Optimal On-Device MoE

The on-device MoE scaling law (Eq. (1)) and optimization objective (Eq. (4)) serve as the principled foundation to govern our MobileMoE architecture design under compute (F_{train}, F_{inf}) and memory ($\mathcal{M}$) constraints. A naïve joint sweep over the three design axes in Figure 2 (number of experts E, expert granularity g, and shared experts s) would incur a cubic number of ablation runs, but these axes are *structurally decoupled* at fixed active parameters N_{act}: E alone changes N_{total} (and thus memory), g preserves both N_{act} and N_{total}, and s adds an independent dense pathway – so the memory and compute-optimal E is preserved under any subsequent choice of g or s, where g refines routing and s adds dense capacity at fixed memory footprint. We therefore adopt a divide-and-conquer approach, decomposing the architecture optimization into three controlled studies, each isolating one factor while holding the others fixed, to progressively determine the optimal on-device MoE architecture.

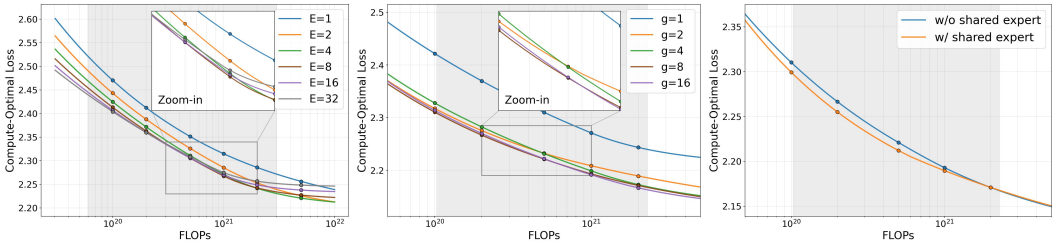
Scaling the number of experts E. Given fixed active parameters N_{act}, the number of experts determines the total parameters N_{total}, which changes the model sparsity ($1 - N_{\text{act}}/N_{\text{total}}$) and model memory (Eq. (5)). To explore the optimal E given fixed device memory constraint, our scaling study sweeps over $E \in \{1, 2, 4, 8, 16, 32\}$ across the three base architectures in Figure 2, spanning the sub-billion active parameter regime $N_{\text{act}} \in \{0.3\text{B}, 0.5\text{B}, 0.9\text{B}\}$; each model is trained across data budgets $D \in \{100, 200, ..., 500\}$ billion tokens. We fit the on-device scaling law (Eq. (1)) on these sweep runs with architecture choice x held fixed, and solve the optimization objective (Eq. (4)). Similar to [9], the scaling coefficients are fitted using LBFGS optimization (detailed in Appendix A.1), which provides the scaling curves in Figure 3, Figure 4(a), and the following finding.

Finding 1: MoE models can be both memory-optimal and compute-optimal over dense models. With fixed memory (M>0.25 GB), MoE models (E>1) achieve lower loss than dense models (Figure 3b). With fixed compute F_{inf} (fixed N_{act}) and F_{train}, increasing E reduces loss with diminishing returns beyond E=8 (Figure 3c, Figure 4a). While the optimal E grows with more memory, moderate sparsity ($E \in \{4, 8\}$) is the practical sweet spot in on-device memory regime.

Based on Finding 1, we construct the on-device MoE with E=8, denoted as MoE(E=8), which achieves near-optimal performance at fixed inference compute with sub-billion active parameters (Figure 3c), while remaining within the practical sweet spot under on-device memory constraints (e.g., $\leq$5 GB in Figure 3b).

Scaling the expert granularity g. Varying expert granularity divides each expert into g fine-grained sub-experts while keeping both total and active parameters intact; thus, on-device memory and inference compute remain constant. Intuitively, finer granularity enables more flexible expert combinations during routing – with $g \cdot E$ fine-grained experts and top-$k \cdot g$ routing, the router can compose more diverse expert combinations, leading to more specialized routing paths [30]. To find the compute-optimal g under the on-device scaling law (Eq. (1)), our scaling study sweeps over $g \in \{1, 2, 4, 8, 16\}$ upon the model MoE(E=8) (Finding 1), under the same experimental regime of N_{act} and D, and solves for the compute-optimal g (Figure 4b).

Finding 2: Fine-grained experts (g>1) achieve substantially lower loss at fixed compute (Figure 4b) due to more diverse top-k routing, but exhibit diminishing returns beyond g=8.



(a) (b) (c)

Figure 4: **Scaling MoE models with compute optimality.** The compute-optimal loss vs. training compute FLOPs for each MoE design factor predicted by the on-device scaling law (Eq. (1)): when varying (a) number of experts E, (b) expert granularity g, (c) with vs without shared expert, the curves with lower model loss indicate the more compute-efficient design choices given a fixed compute budget. Grey area is the experimental regime.

Following Finding 2, we adopt a compute-optimal granularity of $g{=}8$ for $\mathrm{MoE}(E{=}8)$, which results in $\mathrm{MoE}(E{=}8, g{=}8)$, featuring 64 fine-grained experts and top-8 routing. Crucially, this fine-grained expert segmentation maintains the same memory footprint, remaining within on-device limits.

Scaling with shared expert s. Whether to incorporate a shared expert – a dense pathway activated on every token – remains an open design choice: it is adopted in DeepSeekMoE [30] and Qwen2 [31], yet omitted in OLMoE [14] and Qwen3 [2]. To isolate its effect, we compare $\mathrm{MoE}(E{=}8, g{=}8)$ with and without a shared expert by replacing 4 of the 8 active routed experts with a single shared expert ($4\times$ the size of a fine-grained expert). This yields 60 routed experts with top-4 routing and one shared expert. This specific configuration ensures the routed expert count remains divisible by the expert-parallel size ($\mathrm{EP}{=}4$), thereby preserving training efficiency. Notably, it also preserves active and total parameters (and thus memory), enabling a fair ablation on the architectural impact of the shared expert. We fit the on-device scaling law (Eq. (1)) on sweep runs with E, g fixed, under the same experimental regime of N_{act} and D to identify the optimal setting of s (Figure 4c).

Finding 3: With shared expert, on-device MoE model achieves lower loss than the counterparts without shared expert given fixed compute FLOPs (Figure 4c).

Guided by Finding 3, the shared expert (generalist) complements routed experts (specialists). We adopt shared expert to derive our **MobileMoE** architecture: $\mathrm{MoE}(E{=}8, g{=}8, s{=}\checkmark)$ – 60 fine-grained experts, top-4 routing and a shared expert. Applying this to the three base architectures in Figure 2, we obtain **MobileMoE-S/M/L** with $\{0.3, 0.5, 0.9\}$ B active parameters and $\{1.26, 2.82, 5.33\}$ B total parameters, all fitting within a 3-5 GB on-device memory budget under 4-bit quantization (Eq. (5)). Training throughput analysis corroborating these architecture choices is reported in Appendix A.2.

3.4 Scaling MobileMoE with Full Training Recipe

With the MobileMoE architectures derived in Section 3.3, we scale them with a four-stage recipe: pre-training $\rightarrow$ mid-training $\rightarrow$ SFT $\rightarrow$ INT4 QAT, whose design choices are dictated by on-device constraints (Eq. (4), (5)) and the challenges of training MoE models at sub-billion active parameters.

Pre-training. We pre-train all three **MobileMoE-S/M/L** models at context length 2,048, instantiating $\mathrm{MoE}(E{=}8, g{=}8, s{=}\checkmark)$ on the base architecture of Figure 2 (tied input-output embeddings, RoPE $\theta{=}500{,}000$, Llama-3 tokenizer with 128K vocabulary [12]). All models are trained on $\sim$6T tokens from open-licensed data, including a web-heavy mixture ($>60\%$) to provide broad linguistic coverage for general language modeling, and diverse domain coverage for math, code, knowledge, and science to encourage MoE expert specialization. Although our 6T token budget is smaller than those of recent small LLMs (9T for Llama 3.2 1B [12], 11T for SmolLM2 [13]), MobileMoE remains both token- and compute-efficient: all three model scales continue to improve through 6T training (Figure 5), yet use far fewer training FLOPs to match or surpass the accuracy of those dense LLMs (detailed in Section 4.2). For *training stability*, we use auxiliary-loss-free load balancing [38] and router z-loss [25], with FP32 router computations and sigmoid gating, which ensure stable routing and balanced expert utilization throughout training. For *training efficiency* with 60 fine-grained experts per layer (much smaller than in standard MoE), we use grouped-expert MLP with a fused GEMM kernel, expert parallelism ($\mathrm{EP}{=}4$), and drop-and-pad token dispatching that assigns each expert a fixed-size token buffer. More training recipe details (methods, data mixtures) are in Appendix B.

Mid-training. Following pre-training, we perform mid-training to extend context length from 2,048 to 8,192 while shifting the data distribution toward higher-quality data, spanning knowledge-dense and reasoning-heavy domains (code, math, instruction-following) and long documents for the extended context window. With 8K context, MobileMoE's compact d_{model} keeps its KV cache well within the on-device memory budget (Eq. (5)). We train on $\sim$500B tokens ($\sim 8\%$ of pre-training budget) with linear learning rate annealing [12, 33], gradually converging the model on the curated data to produce **MobileMoE-Base** with strengthened downstream capabilities.

Supervised fine-tuning (SFT). We fine-tune MobileMoE-Base on an open-licensed dataset mixture of $\sim$80M samples spanning diverse domains (e.g., math, code, instruction-following, science, QA, and reasoning) at 8K context length with sequence packing. To ensure expert load balance, we continue to apply the same MoE loss terms from pre-training. However, we switch to dropless token dispatching, as the drop-and-pad scheme would discard tokens from structured instruction-response pairs, distorting the learning signal and degrading SFT quality. This stage produces **MobileMoE-SFT** to unlock comprehensive downstream capabilities.

Quantization-aware training (QAT). To fit MobileMoE within on-device memory budgets ($\leq$3–5 GB, Eq. (5)), we apply INT4 QAT [39] to MobileMoE-SFT, following a similar recipe used in recent on-device dense LLMs [6], where linear weights (attention, MoE FFNs, and embeddings) are quantized with symmetric group-wise INT4 (group size 32) and activations are dynamically quantized to INT8 (full formulation in Appendix B). However, specifically for MoE, we keep the router in FP32 throughout QAT to preserve routing stability under quantized gradients, which adds negligible memory cost ($<$0.5% overhead). This stage produces **MobileMoE-QAT** with weight footprints of 0.68/1.48/2.75 GB on S/M/L for on-device deployment (detailed in Section 4.3).

4 Experiments

4.1 Experimental setup

Evaluation protocols. We evaluate MobileMoE on a comprehensive suite of benchmarks across two capability tiers. The *foundational* tier covers 14 widely-used benchmarks spanning five core competencies – *commonsense* (HellaSwag [40], PIQA [41], SIQA [42], WinoGrande [43]), *knowledge* (MMLU [44], NaturalQuestions [45], TriviaQA [46]), *science* (ARC-C/E [47], OpenBookQA [48]), *reading* (BoolQ [49], DROP [50]), and *reasoning & math* (BBH [51], GSM8K [52]). The *advanced* tier comprises 8 benchmarks probing frontier capabilities that small LLMs typically struggle with, including *math* (MATH-500 [53], GSM-Plus [54]), *code* (HumanEval pass@1 [55], MBPP [56]), *instruction following* (IFEval [57], IFBench [58]), and *knowledge & reasoning* (MMLU-Pro [59], GPQA [60]). Detailed evaluation configurations on all benchmarks are given in Appendix C.

Baseline models. We compare MobileMoE against state-of-the-art on-device and efficient LLMs with comparable parameter scales: Gemma 3 (270M, 1B) [16], SmolLM2 (360M, 1.7B) [13], MobileLLM-Pro (1.1B) [6], Llama 3.2 (1B) [12], OLMo 2 (1B) [61], Qwen3.5 (0.8B, 2B) [62], Granite 3.0 (1B, 3B) [63], Jet-Nemotron (2B, 4B) [23], and OLMoE-1B-7B (1.3B active) [14]. To ensure fair comparison, we re-evaluate all baseline models under identical settings with greedy decoding for reproducibility. Model sources are listed in Appendix C.3.

4.2 Experimental results

Pre-training results of MobileMoE. We empirically validate the MobileMoE models derived from on-device scaling laws (Section 3.3) at full-scale pre-training. Figure 5 plots benchmark performance when scaling training data D up to 6T tokens, and compares against existing baselines with reported training token budget. We have these key observations: **(1)** all three MobileMoE scales improve monotonically through 6T on overall average (e.g., MobileMoE-L: 51$\rightarrow$57; MobileMoE-M: 46$\rightarrow$53; MobileMoE-S: 40$\rightarrow$44), with the steepest slope on *Knowledge* (MobileMoE-L: 34$\rightarrow$45 on MMLU+TQA+NQ), which aligns with MoE's larger total capacity of absorbing more knowledge per token than dense models with similar active parameters; **(2)** MobileMoE-L surpasses Llama 3.2 1B (9T, distilled, Avg 42) already by 0.5T tokens, SmolLM2-1.7B (11T, Avg 51) by $\sim$1T tokens, and OLMoE-1B-7B (5T, MoE, Avg 52) by $\sim$2T tokens on overall average, demonstrating superior token efficiency against both dense and MoE baselines; **(3)** by 6T pre-training, MobileMoE-L (Avg 57,

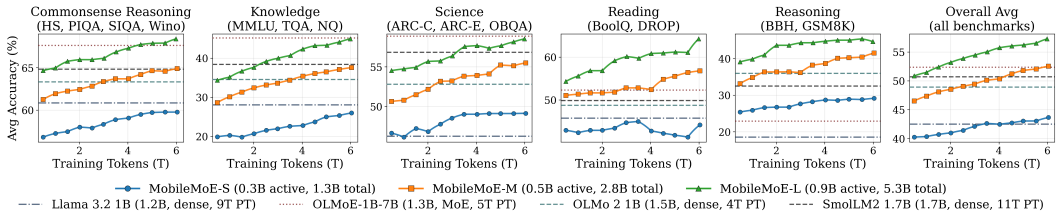


Figure 5: **MobileMoE pre-training trajectories.** Average benchmark accuracy of MobileMoE-S/M/L across five core competencies plus overall average, plotted against training tokens D up to 6T. Horizontal lines: publicly released pre-trained models with reported pre-training token budgets (Llama 3.2 1B: 9T, distilled from 8B; OLMoE-1B-7B: 5T; OLMo 2-1B: 4T; SmolLM2-1.7B: 11T). Evaluation setups are given in Table C.1.

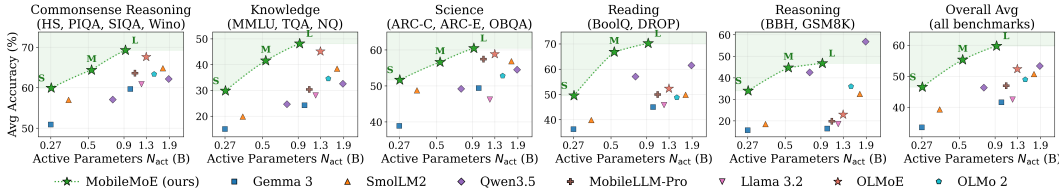


Figure 6: **MobileMoE-Base model comparison.** Average benchmark accuracy of MobileMoE-S/M/L after mid-training across five core competencies plus overall average, vs. publicly released base models (Gemma 3, SmolLM2, Qwen3.5, MobileLLM-Pro, Llama-3.2, OLMoE, OLMo 2), plotted against active parameters N_{act}. Green stars: MobileMoE-S/M/L; dashed line: MobileMoE Pareto curve. Full table results are given in Table D.1.

Table 1: **Instruct model comparison on foundational benchmarks**. Superscripts on benchmarks denote few-shot count (0-shot is omitted). **MobileMoE-S/M/L** refer to 272M/528M/922M active, 1.3B/2.8B/5.3B total params. HS: HellaSwag, Wino: WinoGrande, NQ: NaturalQuestions, TQA: TriviaQA, ARC-C/E: ARC-Challenge/Easy, OBQA: OpenBookQA, BBH: BIG-Bench Hard. All values are benchmark accuracy (%). All models are evaluated by same setup in lm-eval (Table C.1). Similar base model comparison is given in Table D.1.

		Commonsense Reasoning				**Knowledge**			**Science**			**Reading**		**Reasoning**		
Model	N_{act}/N_{total}	HS	PIQA	SIQA	Wino	MMLU^5	NQ^5	TQA^5	ARC-C^{25}	ARC-E	OBQA	BoolQ	DROP^3	BBH^3	GSM8K^8	**Avg**
Gemma 3 270M	270M	39.4	67.1	39.6	53.0	26.5	2.8	9.1	27.7	50.5	35.0	56.1	11.0	31.8	5.8	32.5
SmolLM2 360M	362M	56.9	71.6	40.6	57.4	25.9	6.4	20.4	38.8	49.1	36.2	42.5	15.2	30.5	10.0	35.8
Granite-3.0-1B	400M/1.3B	**64.5**	**75.2**	**49.7**	**61.9**	31.9	8.4	29.3	40.9	64.4	**39.0**	69.8	16.2	32.0	30.0	43.8
MobileMoE-S	**272M/1.3B**	56.5	74.9	42.2	58.4	**42.6**	**10.8**	**30.2**	**43.1**	**73.4**	32.4	**72.3**	**32.3**	**32.2**	**52.2**	**46.7**
Qwen3.5 0.8B	749M	49.7	69.4	38.8	57.6	50.2	3.3	16.1	41.8	61.4	30.8	62.5	33.3	37.8	45.7	42.7
Granite-3.0-3B	800M/3.3B	**71.8**	**77.6**	**54.3**	**66.1**	50.2	16.7	41.9	50.6	74.7	**40.8**	**77.5**	35.6	37.4	52.2	53.4
MobileMoE-M	**528M/2.8B**	66.6	**77.6**	49.0	63.0	**53.9**	**17.5**	**46.6**	**52.5**	**79.9**	38.2	76.7	**46.6**	**39.0**	**67.5**	**55.3**
Gemma 3 1B	1.0B	57.8	72.3	42.0	59.0	40.0	7.6	23.4	40.1	63.1	38.4	75.8	22.0	35.8	38.9	44.0
MobileLLM-Pro	1.1B	52.2	73.2	42.7	51.7	38.7	8.9	19.9	35.8	52.7	29.6	68.8	25.5	29.1	31.8	40.0
Llama 3.2 1B	1.2B	61.7	74.8	43.1	61.5	46.1	14.4	38.4	42.1	63.7	37.8	71.6	21.5	33.9	46.0	46.9
OLMoE-1B-7B	1.3B/6.9B	**78.8**	**79.7**	50.8	**68.7**	52.7	17.2	54.1	57.6	75.9	**46.8**	**81.1**	29.3	37.1	49.1	55.6
OLMo 2 1B	1.5B	67.3	75.2	46.1	63.5	42.9	12.4	37.6	45.1	69.8	42.2	71.0	31.2	35.0	46.9	49.0
SmolLM2 1.7B	1.7B	71.7	76.2	44.6	68.4	49.4	14.2	46.0	53.4	62.9	45.8	68.5	24.8	35.3	46.1	50.5
Qwen3.5 2B	1.9B	62.2	72.8	41.0	63.0	57.4	8.8	28.1	53.2	66.0	35.2	71.7	44.7	45.2	61.3	50.8
Jet-Nemotron-2B	2.0B/2.0B	64.3	73.4	38.9	64.0	54.4	15.1	38.4	54.6	57.4	34.6	77.5	54.4	42.2	**78.3**	53.4
Jet-Nemotron-4B	4.0B/4.0B	70.7	75.0	41.3	68.0	57.8	18.6	48.8	56.7	58.3	38.0	75.3	**55.3**	**46.2**	77.6	56.3
MobileMoE-L	**922M/5.3B**	73.0	78.9	**53.4**	66.1	**60.1**	**22.4**	**54.9**	**57.9**	**81.9**	43.2	**81.1**	50.1	40.1	77.6	**60.1**

922M active) outperforms OLMoE-1B-7B's base model (Avg 52, 1.3B active) with 30% fewer active parameters, validating the parameter efficiency of our MoE architectures.

Mid-training results of MobileMoE-Base. After mid-training, **MobileMoE-Base** represents the fully pre-trained foundation comparable to publicly released base models. Figure 6 plots benchmark performance against active parameters N_{act} for MobileMoE-Base vs. existing base models. We have the following observations. **(1)** MobileMoE-Base establishes a new Pareto frontier in the sub-billion active-parameter regime: on overall, MobileMoE-L (922M active, Avg 60) surpasses OLMoE-1B-7B (1.3B active, Avg 52) by +8 with 30% fewer active parameters and every dense baseline at matched or smaller N_{act}; **(2)** the smooth, monotonic MobileMoE curve across S/M/L (Overall Avg: 47→55→60) validates that the MoE architecture derived in Section 3.3) generalizes consistently from scaling law ablation regime ($\leq$500B tokens) to the full pre- and mid-training, confirming the predictive power of our on-device MoE scaling law; **(3)** at matched N_{act}, MobileMoE outperforms dense baselines across 4 out of 5 capability domains, with the largest gains on *Knowledge* and *Reading* – reflecting MoE's expanded capacity to learn knowledge; the exception is *Reasoning*, where Qwen3.5-2B (1.9B dense) remains stronger at larger scale with 2x more active parameters than MobileMoE-L.

Table 2: **Instruct model comparison on advanced benchmarks. Avg** is within-capability mean; **Overall** is mean across all. All models are evaluated in non-think mode by the same setup, using lm-eval and official packages TIGER-AI-Lab/MMLU-Pro, allenai/IFBench (detailed setup in Table C.2; more analysis in Appendix C.4).

		Math			Code			Instruction Following			Knowledge & Reasoning			
Model	$N_{\text{act}}/N_{\text{total}}$	MATH500[4]	GSM+[5]	**Avg**	HumanEval	MBPP[3]	**Avg**	IFEval	IFBench	**Avg**	MMLU-Pro[5]	GPQA	**Avg**	**Overall**
Gemma 3 270M	270M	7.2	4.3	5.7	12.8	9.8	11.3	31.2	11.2	21.2	11.3	25.8	18.6	14.2
SmolLM2 360M	362M	3.8	4.6	4.2	0.0	22.8	11.4	40.2	**19.1**	29.7	12.0	25.8	18.9	16.0
Granite-3.0-1B	400M/1.3B	11.2	16.0	13.6	20.7	**28.8**	24.8	37.2	11.2	24.2	13.9	25.8	19.8	20.6
MobileMoE-S	**272M/1.3B**	**21.0**	**28.9**	**24.9**	**44.5**	27.8	**36.2**	**54.1**	13.8	**33.9**	**18.2**	**27.8**	**23.0**	**29.5**
Qwen3.5 0.8B	749M	19.6	26.5	23.1	31.1	25.4	28.3	59.9	19.8	39.8	24.0	**26.3**	25.1	29.1
Granite-3.0-3B	800M/3.3B	20.6	31.8	26.2	15.2	41.2	28.2	45.0	16.1	30.5	23.0	24.8	23.9	27.2
MobileMoE-M	**528M/2.8B**	**27.2**	**42.3**	**34.7**	**60.4**	**43.0**	**51.7**	**60.8**	**20.9**	**40.8**	**28.3**	24.8	**26.5**	**38.4**
Gemma 3 1B	1.0B	27.6	25.2	26.4	42.1	40.6	41.3	63.7	17.5	40.6	16.1	25.3	20.7	32.3
MobileLLM-Pro	1.1B	8.8	17.0	12.9	59.8	44.2	52.0	63.1	17.0	40.1	10.9	23.2	17.1	30.5
Llama 3.2 1B	1.2B	19.6	27.8	23.7	36.6	37.4	37.0	58.7	17.4	38.0	20.8	28.8	24.8	30.9
OLMoE-1B-7B	1.3B/6.9B	8.4	28.1	18.2	36.0	30.2	33.1	48.1	16.6	32.4	19.5	24.2	21.9	26.4
OLMo 2 1B	1.5B	10.2	25.1	17.7	29.3	14.8	22.0	53.5	14.5	34.0	16.0	29.8	22.9	24.1
SmolLM2 1.7B	1.7B	15.4	26.6	21.0	1.2	34.6	17.9	54.7	16.5	35.6	19.8	29.8	24.8	24.8
Qwen3.5 2B	1.9B	31.0	42.4	36.7	50.0	41.2	45.6	**73.3**	**30.3**	**51.8**	38.8	34.3	36.6	42.7
Jet-Nemotron-2B	2.0B/2.0B	**43.2**	53.0	48.1	59.1	54.0	56.6	36.5	10.0	23.3	39.0	**37.9**	38.5	41.6
Jet-Nemotron-4B	4.0B/4.0B	43.0	**54.1**	**48.6**	64.6	**57.4**	**61.0**	44.1	11.2	27.7	**44.4**	36.9	**40.7**	**44.5**
MobileMoE-L	**922M/5.3B**	32.2	50.2	41.2	**65.2**	52.4	58.8	67.3	20.1	43.7	34.0	33.8	33.9	44.4

Results of MobileMoE-SFT. Table 1 compares MobileMoE-S/M/L against state-of-the-art baselines across the same benchmarks on diverse domains, under identical evaluation protocols. We report average scores across benchmarks as a holistic measure of capability, and summarize our findings:

(1) *MobileMoE achieves state-of-the-art performance among sub-2B active-parameter models.* MobileMoE-S (272M active, Avg 46.7) surpasses existing instruct models at similar scale: Gemma 3 270M (32.5) by +14.2 and SmolLM2 360M (35.8) by +10.9. MobileMoE-M (528M active, Avg 55.3) outperforms all dense baselines up to 1.9B active parameters, including Qwen3.5 2B (50.8) by +4.5, SmolLM2 1.7B (50.5) by +4.8, and OLMo 2 1B (49.0) by +6.3. MobileMoE-L (922M active, Avg 60.1) further extends this lead, surpassing the larger MoE baseline OLMoE-1B-7B (1.3B active, 55.6) by +4.5 despite using 30% fewer active parameters and 23% fewer total parameters (5.3B vs. 6.9B).

(2) *MobileMoE matches dense models using 2–4× fewer active parameters.* MobileMoE-S (272M, Avg 46.7) surpasses Qwen3.5 0.8B (42.7) by +4.0 using $2.8\times$ fewer active parameters, and matches Llama 3.2 1B (46.9) with $4.4\times$ fewer active parameters. MobileMoE-M (528M, Avg 55.3) outperforms Qwen3.5 2B (50.8) by +4.5 using $3.6\times$ fewer active parameters. MobileMoE-L (922M, Avg 60.1) exceeds Qwen3.5 2B by +9.3 using $2.0\times$ fewer active parameters, with especially strong gains on knowledge, science, and math. These results demonstrate that MobileMoE-SFT delivers comparable or superior quality to larger dense instruct models with 2–4× fewer inference FLOPs, translating to lower latency and reduced power consumption for efficient on-device deployment.

Results of MobileMoE-SFT on additional capabilities. Table 2 probes four advanced capabilities where sub-billion dense baselines often collapse (e.g., SmolLM2-360M HumanEval=0.0, Gemma 3 270M MATH500=7.2). We have these observations: **(1)** *MobileMoE shows strong code and math performance at sub-billion active-parameter scales.* On *Code*, MobileMoE-S/M/L (Avg 36.2/51.7/58.8) outperform scale-matched baselines substantially (e.g., MobileMoE-L vs. Qwen3.5 2B +13.2, vs. OLMoE-1B-7B +25.7). On *Math*, MobileMoE-L (Avg 41.2) leads Qwen3.5 2B (Avg 36.7) by +4.5 and OLMoE-1B-7B (Avg 18.2) by +23.0; MobileMoE-S (Avg 24.9) scores substantially higher than Gemma 3 270M (Avg 5.7) and SmolLM2-360M (Avg 4.2). **(2)** *MobileMoE remains competitive on Instruction Following (IF) and Knowledge & Reasoning (K&R).* MobileMoE-L outperforms OLMoE-1B-7B on both capabilities and both Jet-Nemotron variants on IF; it trails Qwen3.5 2B on IF and Jet-Nemotron-4B on K&R while using substantially fewer active parameters. Overall, the MoE benefits of larger total capacity and routed expert specialization yield clear strengths on code and math, while the IF and K&R gap motivates enriching our training recipe in future work, e.g., adding distillation and thinking-enabled post-training as in Qwen3/3.5 [2, 62].

4.3 On-device deployment of MobileMoE

We benchmark MobileMoE on runtime latency and memory against MobileLLM-Pro [6] – a state-of-the-art dense LLM purpose-built for on-device deployment, providing a strong baseline at comparable parameter count. We conduct our runtime profiling on two flagship smartphones: *Samsung Galaxy S25* (Snapdragon) and *iPhone 16 Pro* (Apple A18 Pro). We describe on-device deployment setups below.

Table 3: **On-device runtime latency and memory**: comparing MobileMoE-S/M/L against the dense MobileLLM-Pro on two flagship smartphones – Samsung Galaxy S25 (Snapdragon 8 Elite, 4 CPU threads) and iPhone 16 Pro (Apple A18 Pro, 2 CPU threads), both via ExecuTorch+XNNPACK backend with INT4 weights and INT8 dynamic activations. **Prefill TTFT** (s, ↓): time-to-first-token. **Decode Rate** (tok/s, ↑): generation throughput. **Peak RSS** (GB, ↓): runtime peak RAM measured on real prompts. **Mem** (GB): static INT4 weight memory; **Avg** (%, ↑): mean accuracy over 14 benchmarks after QAT (Table D.3). Results are averaged over multiple runs on real prompts (code, knowledge, math). Best results per column are in **bold**. More runtime results in Appendix E.

			Samsung Galaxy S25												**iPhone 16 Pro**							
	Mem	**Avg**	**Prefill TTFT (s) ↓**				**Decode Rate (tok/s) ↑**				**Peak RSS (GB) ↓**				**Prefill TTFT (s) ↓**				**Decode Rate (tok/s) ↑**			
Model	(GB)	(%)	1k	2k	4k	8k	1k	2k	4k	8k	1k	2k	4k	8k	1k	2k	4k	8k	1k	2k	4k	8k
MobileLLM-Pro	**0.55**	45.5	4.26	14.21	35.62	105.21	45.8	27.6	18.6	10.6	**0.98**	**1.07**	1.35	1.91	6.03	14.13	39.08	122.42	48.5	32.4	17.9	10.6
MobileMoE-S	0.68	44.0	**2.01**	**7.82**	**16.36**	**50.33**	**112.0**	**61.7**	**48.9**	**25.8**	1.02	1.10	**1.23**	**1.49**	**2.14**	**5.23**	**13.86**	**39.91**	**148.4**	**90.2**	**54.2**	**32.2**
MobileMoE-M	1.48	52.5	4.32	16.68	35.02	88.25	64.6	34.5	26.2	15.4	2.12	2.24	2.43	2.77	4.71	10.82	31.43	83.54	76.2	51.0	27.9	17.2
MobileMoE-L	2.75	**57.8**	8.99	25.85	55.12	162.45	36.6	22.9	17.3	8.9	3.87	4.06	4.27	4.71	8.63	21.92	51.98	141.31	43.3	23.5	17.1	10.8

MoE 4-bit quantization. We apply INT4 QAT (Section 3.4, Eq. (6)) to all linear layers, yielding INT4 weight memory footprints of 0.55 / 0.68 / 1.48 / 2.75 GB for MobileLLM-Pro and MobileMoE-S/M/L. Unlike MobileLLM-Pro's publicly released pre-trained QAT model, MobileMoE is quantized on top of its SFT models. Post-QAT average accuracy is in Table 3; full results are in Appendix D.3.

MoE runtime profiling. Since mobile backends provide highly optimized INT4 *dense* matmul kernels but no fused MoE feed-forward operator, we implement a custom ExecuTorch operator for the MoE feed-forward block; while attention and embedding layers use the standard INT4 dense path. We profile MobileMoE on two flagship smartphones across short-context (e.g., 1k input tokens) and long-context (e.g., 2k, 4k, 8k input tokens) regimes. Table 3 reports both smartphones at input lengths $\{1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}\}$; the full per-platform sweep setups, custom MoE kernel design, and additional runtime profiling results are in Appendix E.

Results of on-device latency and memory. Table 3 reports on-device runtime benchmarks, and compares MobileMoE-S/M/L to MobileLLM-Pro. All models are measured under identical quantization, and runtime setups (ExecuTorch + XNNPACK backend) for fair comparison (see Appendix E). We first highlight our findings on Samsung S25. **(1)** *MobileMoE-S achieves Pareto win at comparable size:* at comparable INT4 weights (0.68 vs. 0.55 GB) and similar accuracy (Avg 44.0 vs 45.5), MobileMoE-S is 1.8–$2.2\times$ faster at prefill, 2.2–$2.6\times$ faster at decode, and uses 22% less Peak RSS at 8k context. **(2)** *MobileMoE-M obtains higher accuracy at similar runtime:* with $+7.0$ accuracy boost, it is 0.9–$1.2\times$ at prefill, 1.3–$1.5\times$ at decode, with additional Peak RSS overhead. **(3)** *MobileMoE-L has highest accuracy with moderate runtime and memory cost:* with $+12.3$ accuracy boost, MobileMoE-L is 0.5–$0.6\times$ at prefill and 0.8–$0.9\times$ at decode, with higher Peak RSS across contexts but remaining below 5 GB at 8k context. Overall, MobileMoE establishes a new on-device Pareto frontier across model scales and devices, with consistent gains on iPhone 16 Pro (MobileMoE-S: 2.7–$3.1\times$ prefill, 2.8–$3.1\times$ decode). Our custom MoE kernel turns theoretical FLOPs savings into practical on-device speedups, empirically realizing the memory and compute-efficiency derived in Section 3.3.

5 Conclusion

We presented **MobileMoE**, a family of on-device MoE language models. Our work develops a generalized on-device MoE scaling law that jointly optimizes architecture under mobile memory and compute constraints, an end-to-end recipe that scales MobileMoE training to establish a new Pareto frontier for on-device LLMs in benchmark performance, and the first efficient on-device MoE deployment on commodity smartphone CPUs, with systematic profiling across CPU and GPU backends, which together demonstrate MoE as a practical path at the edge. Several promising directions can further build on this work. On the MoE training side, distillation, reasoning-oriented post-training, and multimodal extensions could unlock better performance and capabilities. On the MoE runtime side, dynamic routing, model compression (e.g., expert pruning, mixed-precision quantization), and mobile NPU deployment could yield further on-device efficiency. While existing on-device models remain predominantly dense, we show that MoE offers a more efficient alternative. We hope MobileMoE opens new directions for next-generation on-device AI, bringing capable, efficient sparse LLMs to edge devices such as smartphones, wearables, and embodied agents, enabling local intelligence that lowers cloud compute demand while delivering private, low-latency inference.

References

[1] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[2] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[3] Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[4] xAI. Open release of Grok-1, 2024.

[5] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Patrick Huber, Ernie Chang, Wei Wen, Igor Fedorov, Tarek Elgamal, Hanxian Huang, Naveen Suda, Chinnadhurai Sankar, Vish Vogeti, Yanghan Wang, et al. MobileLLM-Pro technical report. *arXiv preprint arXiv:2511.06719*, 2025.

[7] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[8] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[9] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *Advances in Neural Information Processing Systems*, 2022.

[10] Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International conference on machine learning*, 2022.

[11] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[13] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

[14] Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

[15] Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[16] Gemma Team. Gemma 3. 2025.

[17] Hanxian Huang, Igor Fedorov, Andrey Gromov, Bernard Beckerman, Naveen Suda, David Eriksson, Maximilian Balandat, Rylan Conway, Patrick Huber, Chinnadhurai Sankar, et al. MobileLLM-Flash: Latency-guided on-device LLM design for industry scale deployment. *arXiv preprint arXiv:2603.15954*, 2026.

[18] Yonggan Fu, Xin Dong, Shizhe Diao, Hanrong Ye, Wonmin Byeon, Yashaswi Karnati, Lucas Liebenwein, Hannah Zhang, Nikolaus Binder, Maksim Khadkevich, et al. Nemotron-Flash: Towards latency-optimal hybrid small language models. *arXiv preprint arXiv:2511.18890*, 2025.

[19] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

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

[21] Xuan Shen, Pu Zhao, Yifan Gong, Zhenglun Kong, Zheng Zhan, Yushu Wu, Ming Lin, Chao Wu, Xue Lin, and Yanzhi Wang. Search for efficient large language models. *Advances in Neural Information Processing Systems*, 2024.

[22] Youpeng Zhao, Ming Lin, Huadong Tang, Qiang Wu, and Jun Wang. Merino: Entropy-driven design for generative language models on IoT devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[23] Yuxian Gu, Qinghao Hu, Haocheng Xi, Junyu Chen, Shang Yang, Song Han, and Han Cai. Jet-Nemotron: Efficient language model with post neural architecture search. *Advances in Neural Information Processing Systems*, 2026.

[24] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 1991.

[25] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

[26] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

[27] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 2022.

[28] Dongxu Li, Yudong Liu, Haoning Wu, Yue Wang, Zhiqi Shen, Bowen Qu, Xinyao Niu, Fan Zhou, Chengen Huang, Yanpeng Li, et al. Aria: An open multimodal native mixture-of-experts model. *arXiv preprint arXiv:2410.05993*, 2024.

[29] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[30] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[31] Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[32] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[33] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[34] Mustafa Shukor, Louis Bethune, Dan Busbridge, David Grangier, Enrico Fini, Alaaeldin El-Nouby, and Pierre Ablin. Scaling laws for optimal data mixtures. *arXiv preprint arXiv:2507.09404*, 2025.

[35] Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, et al. Joint MoE scaling laws: Mixture of experts can be memory efficient. *arXiv preprint arXiv:2502.05172*, 2025.

[36] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

[37] 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. Llama guard 3-1b-INT4: Compact and efficient safeguard for human-AI conversations. *arXiv*, 2024.

[38] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

[39] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

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

[41] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, 2020.

[42] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 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)*, 2019.

[43] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 2021.

[44] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[45] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 2019.

[46] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 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)*, 2017.

[47] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[48] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, 2018.

[49] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 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)*, 2019.

[50] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. 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)*, 2019.

[51] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

[52] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[53] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[54] Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[55] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[56] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[57] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[58] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

[59] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 2024.

[60] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[61] OLMo Team. 2 OLMo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[62] Qwen Team. Qwen3.5-Omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

[63] IBM Granite Team. Granite 3.0 language models. *URL: https://github.com/ibm-granite/granite-3.0-language-models*, 2024.

[64] OLMo Team. OLMo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[65] Jake Poznanski, Aman Rangapur, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Christopher Wilhelm, Kyle Lo, and Luca Soldaini. olmOCR: Unlocking trillions of tokens in PDFs with vision language models. *arXiv preprint arXiv:2502.18443*, 2025.

[66] Changsheng Zhao, Ernie Chang, Zechun Liu, Chia-Jung Chang, Wei Wen, Chen Lai, Sheng Cao, Yuandong Tian, Raghuraman Krishnamoorthi, Yangyang Shi, et al. MobileLLM-R1: Exploring the limits of sub-billion language model reasoners with open training recipes. *arXiv preprint arXiv:2509.24945*, 2025.

[67] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. DataComp-LM: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 2024.

[68] Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, and Thomas Wolf. FineWeb: decanting the web for the finest text data at scale. *HuggingFace. Accessed: Jul*, 2024.

[69] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[70] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-Nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

[71] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations*, 2023.

[72] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

[73] Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Ilia Kulikov, Kyunghyun Cho, Dong Wang, Yuandong Tian, Jason E Weston, et al. NaturalReasoning: Reasoning in the wild with 2.8 m challenging questions. *arXiv preprint arXiv:2502.13124*, 2025.

[74] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The FLAN collection: Designing data and methods for effective instruction tuning. In *International conference on machine learning*, 2023.

[75] 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, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024.

Appendix

A Scaling Law Ablation Details

This appendix provides the detailed configurations, parametric fitting procedure, and training efficiency analysis for the three scaling-law ablations – number of experts E, expert granularity g, and shared expert s – that underpin the on-device MoE architecture derivation in Section 3.3. Table A.1 summarizes the sweep range, fixed settings, and resulting finding of each ablation. All ablations use the base architectures in Figure 2 and train on up to ∼500B tokens.

Table A.1: **Scaling law ablation configurations.** All ablations run on 8 nodes (64 NVIDIA H100 96 GB GPUs) with global batch size 3,072 and sequence length 2,048, which takes 2-10 days to complete each ablation.

Ablation	Sweep range	N_{act} (B/billion)	D (B/billion tokens)	Fixed settings	Finding
Sparsity (E)	$E \in \{1, 2, 4, 8, 16, 32\}$	$\{0.3, 0.5, 0.9\}$	$\{100, 150, 200, \ldots, 500\}$	$g = 1$, $s = \times$	Finding 1: $E = 8$
Granularity (g)	$g \in \{1, 2, 4, 8, 16\}$	$\{0.3, 0.5, 0.9\}$	$\{100, 150, 200, \ldots, 500\}$	$E = 8$, $s = \times$	Finding 2: $g = 8$
Shared expert (s)	$s \in \{\checkmark, \times\}$	$\{0.3, 0.5, 0.9\}$	$\{100, 150, 200, \ldots, 500\}$	$E = 8$, $g = 8$	Finding 3: $s = \checkmark$

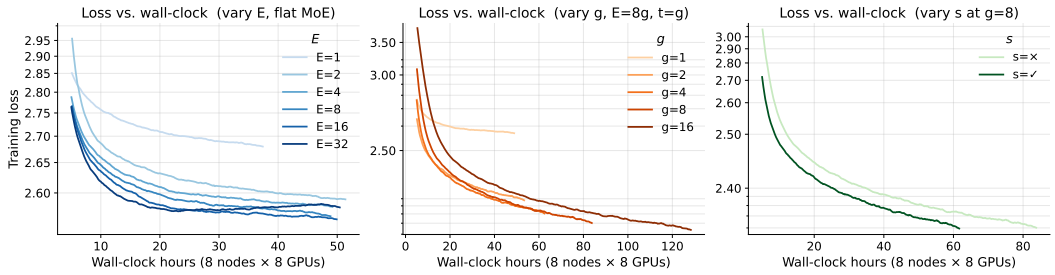
A.1 Parametric Fitting of On-Device MoE Scaling Laws

Parametric fitting procedure. To estimate the coefficients of the on-device MoE scaling law (Eq. (1)), we adopt a two-stage procedure: (i) `scipy.optimize.curve_fit` (nonlinear least-squares with MSE) provides a warm-start initialization, followed by (ii) `scipy.optimize.minimize` with L-BFGS-B optimization (similar to [9]) using an MSE objective, under bounds that keep the irreducible loss $c > 0$. Each ablation fits on the validation loss across runs that sweep over varying active parameters $N_{\text{act}} \in \{0.3, 0.5, 0.9\}$ billion and data tokens $D \in \{100, 150, 200, \ldots, 500\}$ billion tokens. For the parametric fit, we use N_{act} excluding the embedding layer following [8] (i.e., $N'_{\text{act}} \in \{0.17, 0.40, 0.76\}$ B), as embeddings are architecture-invariant across all MoE configurations. The transformed expert count $\hat{E}$ following [10]: $\frac{1}{\hat{E}} = \frac{1}{E - 1 + \left(\frac{1}{E_{\text{start}}} - \frac{1}{E_{\text{max}}}\right)^{-1}} + \frac{1}{E_{\text{max}}}$. We set $E_{\text{start}} = 1$ (the dense baseline) and $E_{\text{max}} = 32$ (the upper bound beyond which total parameters exceed typical mobile DRAM budgets of 5 GB at INT4 for our sub-billion active-parameter regime: $N_{\text{act}} \in \{0.3, 0.5, 0.9\}$ B). We also find empirically that using the simplified form $\hat{E}{=}E$ ($E_{\text{start}}{=}1$, $E_{\text{max}}{=}\infty$; RMSE=0.0089) or fitting $E_{\text{start}}, E_{\text{max}}$ as free parameters (RMSE=0.0060) yields the same optimal E under a 5,GB on-device memory budget; our fitting choice anchors the transformation to the on-device constraint while maintaining competitive fit quality (RMSE=0.0076). We obtain the optimal-E findings by interpolation within the swept range.

Table A.2: **Fitted scaling-law coefficients across the three ablations.** Columns list the nine coefficients of Eq. (1). The E-sweep row directly fits all coefficients of Eq. (1) with x fixed (i.e., $A = A_x$, $\delta = \delta_x$, $\alpha = \alpha_x$, $\gamma = \gamma_x$, $B = B_x$, $\omega = \omega_x$, $\beta = \beta_x$, $\zeta = \zeta_x$, $c = c_x$). The g- and s-rows fit Eq. (1) with $\hat{E}$ fixed; their entries under A, α, B, β are the Chinchilla composites $\tilde{A} = A_x \hat{E}^{\delta_x}$, $\tilde{\alpha} = \alpha_x + \gamma_x \ln \hat{E}$, $\tilde{B} = B_x \hat{E}^{\omega_x}$, $\tilde{\beta} = \beta_x + \zeta_x \ln \hat{E}$ (marked with *), while $\delta, \gamma, \omega, \zeta$ are absorbed ("abs."). The irreducible loss c_x reflects the entropy floor of the validation data and is architecture-independent; the g- and s-sweeps therefore regularize c_x toward the E-sweep estimate (std. err. ± 0.13). RMSE: root-mean-square error of the fit on validation loss.

Sweep	Setting	A_x	δ_x	α_x	γ_x	B_x	ω_x	β_x	ζ_x	c_x	RMSE
E-sweep (fixed $g = 1$, $s = \times$)	joint fit	0.2388	0.0906	−0.2833	0.0387	0.6019	1.0593	−0.3210	−0.3684	1.9730	0.0076
g-sweep (fixed $E = 8$, $s = \times$)	$g = 1$	0.2747*	abs.	−0.2265*	abs.	2.4777*	abs.	−0.8296*	abs.	1.9730	0.0037
	$g = 2$	0.1466*	abs.	−0.3243*	abs.	0.3697*	abs.	−0.2492*	abs.	1.9687	0.0031
	$g = 4$	0.1823*	abs.	−0.2880*	abs.	1.1616*	abs.	−0.6188*	abs.	1.9744	0.0036
	$g = 8$	0.1670*	abs.	−0.3006*	abs.	0.5199*	abs.	−0.3870*	abs.	1.9636	0.0029
	$g = 16$	0.1442*	abs.	−0.3377*	abs.	1.1266*	abs.	−0.6204*	abs.	2.0054	0.0034
s-sweep (fixed $E = 8$, $g = 8$)	$s = \times$	0.1670*	abs.	−0.3006*	abs.	0.5199*	abs.	−0.3870*	abs.	1.9636	0.0029
	$s = \checkmark$	0.1224*	abs.	−0.3884*	abs.	0.3487*	abs.	−0.2185*	abs.	1.9636	0.0033

Scaling law coefficients. Table A.2 reports the fitted coefficients across all three ablations.



(a) (b) (c)

Figure A.1: **Training efficiency of MoE architecture** ($N_{\text{act}} = 0.3$B). Training loss vs. wall-clock hours when varying (a) number of experts $E \in \{1, 2, 4, 8, 16, 32\}$, (b) expert granularity $g \in \{1, 2, 4, 8, 16\}$ at $E = 8$, and (c) with vs. without a shared expert $s \in \{\checkmark, \times\}$ at $E = 8, g = 8$. For fair comparison, each run is trained on the same data for 500B tokens, with identical hardware (8 nodes, 64 NVIDIA H100 96 GB GPUs) – see Table A.1 for full ablation details.

- **Scaling the number of experts E:** the E-sweep fits Eq. (1) with x fixed, jointly capturing scaling across $E \in \{1, 2, 4, 8, 16, 32\}$, N_{act}, and D on > 100 datapoints to fit 9 coefficients, and parameterizes the scaling-law curves in Figures 3 and 4(a) underlying Finding 1.
- **Scaling the expert granularity g:** the g-sweep fits Eq. (1) with $\hat{E}$ fixed independently for each $g \in \{1, 2, 4, 8, 16\}$ at $E = 8$ and $s = \times$ (27 datapoints per fit), parameterizing the compute-optimal scaling curves in Figure 4(b) underlying Finding 2.
- **Scaling with shared expert s:** the s-sweep fits Eq. (1) with $\hat{E}$ fixed at $E = 8$, $g = 8$, comparing no shared expert ($s = \times$: 64 routed experts with top-8 routing) against with shared expert ($s = \checkmark$: 60 routed experts plus one always-on shared expert with top-4 routing; see Section 3.3), with 27 datapoints per fit, parameterizing the compute-optimal curves in Figure 4(c) underlying Finding 3.

Note that the E-sweep requires a joint fit given the E-dependent exponents $(\delta_x, \gamma_x, \omega_x, \zeta_x)$ are only identifiable when E varies; while the g- and s-sweeps fit independently since E is held fixed (absorbing those exponents into effective constants) and the scaling dynamics on g and s is characterized by the reduced form in Eq. (3), $\mathcal{L}_{\hat{E}}(N_{\text{act}}, D, x) = \tilde{A}_x N_{\text{act}}^{\tilde{\alpha}_x} + \tilde{B}_x D^{\tilde{\beta}_x} + c_x$, where the architecture choice $x \in \{g, s\}$ modulates only the effective coefficients $(\tilde{A}_x, \tilde{\alpha}_x, \tilde{B}_x, \tilde{\beta}_x, c_x)$, therefore each setting of x can be fitted separately from its own (N_{act}, D) grid.

A.2 Training Efficiency of MoE Architectures

Training efficiency of MoE architecture. Figure A.1 shows training loss versus wall-clock time for the three design factors, complementing the memory and compute-optimal analysis from a training-efficiency perspective. For model sparsity (Figure A.1(a)), the dense baseline ($E = 1$) trains fastest per step but converges to higher loss, while all MoE configurations ($E \geq 2$) share roughly the same training throughput. The final loss exhibits diminishing returns with $E \geq 8$, where $E = 8$ and $E = 16$ converge to similar final loss, but $E = 32$ shows performance regression despite having more total parameters and memory footprint. Taking into account the higher memory cost at larger E (Eq. (5)), $E = 8$ remains a memory- and training-efficient operating point. For expert granularity (Figure A.1(b)), finer-grained experts ($g \geq 2$) achieve lower loss than $g = 1$ (no fine-grained experts) at the same wall-clock budget, while $g = 16$ incurs much higher per-step overhead ($\sim$50% more wall-clock time) than $g = 8$ with negligible loss reduction (< 0.01), making $g = 8$ a training-efficient sweet spot for expert granularity. For the shared expert (Figure A.1(c)), adding the shared expert ($s = \checkmark$) further improves training efficiency over routed-only experts ($s = \times$), achieving higher training throughput with lower final loss at the same N_{act} and N_{total}. Together, these results confirm that the MobileMoE configuration ($E = 8, g = 8, s = \checkmark$) is also training-efficient on real hardware.

B Training Recipe Details

B.1 Training Setup and Hyperparameters

Overall training setup. All four training stages of MobileMoE share the same optimizer setup. We use AdamW (β_1=0.9, β_2=0.95, ϵ=10^{-15}) with weight decay 0.1 and gradient clipping at 1.0. Model

Table B.1: **MobileMoE training hyperparameters across all stages.**

Hyperparameter	Pre-training	Mid-training	SFT	QAT
Context length	2,048	8,192	8,192	8,192
Total tokens	∼6T	∼500B	∼126B	∼21B
Global batch size	2,048 / 3,072†	512 / 768†	256	256
Tokens per step	4.2M / 6.3M†	4.2M / 6.3M†	2.1M	2.1M
Peak learning rate	4×10^{-4}	4×10^{-5}	4×10^{-6}	4×10^{-6}
LR schedule	Cosine	Linear	Cosine	Cosine
LR min ratio	0.1	0.1	0.0	0.0
Warmup steps	8,000	50	3,000 (5%)	500 (5%)
Weight decay	0.1	0.1	0.1	0.1
Gradient clipping	1.0	1.0	1.0	1.0
Adam ϵ	10^{-15}	10^{-15}	10^{-15}	10^{-15}
Adam β_1, β_2	0.9, 0.95	0.9, 0.95	0.9, 0.95	0.9, 0.95
Precision	BF16	BF16	BF16	BF16
Sequence packing	Yes	Yes	Yes	Yes
Optimizer init	From scratch	Fresh	Fresh	Fresh
QAT-specific				
Weight quantization	—	—	—	INT4 (group size 32)
Activation quantization	—	—	—	INT8
Embedding quantization	—	—	—	INT4

† S / M and L model sizes respectively.

weights are stored in BF16 while optimizer states, gradients, and MoE router weights are kept in FP32 for numerical stability. QAT additionally applies INT4 weight and INT8 activation quantization (Section 3.4). Learning rate (LR) schedules are stage-specific: pre-training uses a cosine schedule with peak LR 4×10^{-4}; mid-training uses linear decay from 4×10^{-5}; SFT and QAT use cosine decay from 4×10^{-6}. The peak LR decreases by $10\times$ from pre-training through SFT, with QAT inheriting the SFT LR. Training is performed on 8-GPU NVIDIA H100 (96 GB) nodes. Pre-training uses 16–32 nodes (128–256 GPUs) for 3–4 weeks depending on model size; mid-training uses 16–32 nodes for ∼2 days; SFT and QAT each use 4–8 nodes (32–64 GPUs) for ∼2–3 days. Full per-stage hyperparameters are summarized in Table B.1.

B.2 MoE-Specific Training Methods

MoE routing and load balancing. For load balancing, we use auxiliary-loss-free balancing [38] (λ_{lb}=10^{-3}), which adjusts expert biases based on token load imbalance without backpropagating a loss, combined with router z-loss regularization (λ_z=10^{-4}) [25] to stabilize router logits. We adopt sigmoid gating with per-token top-k normalization, which scores each expert independently rather than forcing competition as in softmax gating, therefore producing smoother routing score distributions for improved training stability. All router computations are performed in FP32 for numerical stability.

MoE training efficiency. With fine-grained experts, each expert's FFN is much smaller than standard MoE FFN (e.g., 60 FFNs of 768×384 in MobileMoE-S vs. 8 FFNs of 4096×14336 in Mixtral 8x7B [29], $\sim200\times$ smaller), making naïve per-expert computation inefficient. We address this with grouped MLP, which batches all local experts into a single fused grouped matrix multiplication (GMM) kernel, replacing small sequential GEMMs with one efficient batched operation, while adopting drop-and-pad token dispatching (capacity factor 1.5) to ensure uniform buffer sizes for the batched kernel. We also apply expert parallelism (EP=4) with all-to-all token dispatch and gather, allowing each GPU to hold and compute only E/EP experts for memory efficiency (i.e., $60/4$=15 routed experts per GPU for MobileMoE). From SFT onward, we switch to dropless dispatching (§3.4) to avoid discarding tokens from structured instruction-response pairs.

MoE quantization-aware training (QAT). Direct post-training quantization (PTQ) could cause quality degradation for on-device LLMs [6]; we therefore apply QAT [39] on MobileMoE-SFT to reduce its memory (Eq. (5)) for on-device use cases. Formally, all linear weights (i.e., attention, MoE FFNs, embeddings) are quantized to INT4 with symmetric group-wise quantization:

$$\tilde{\mathbf{W}}_g = s_g \cdot \text{clamp}\left(\left\lfloor \frac{\mathbf{W}_g}{s_g} \right\rceil, q_{\min}, q_{\max}\right), \quad s_g = \frac{2\ \max(|\mathbf{W}_g|)}{2^b - 1} \tag{6}$$

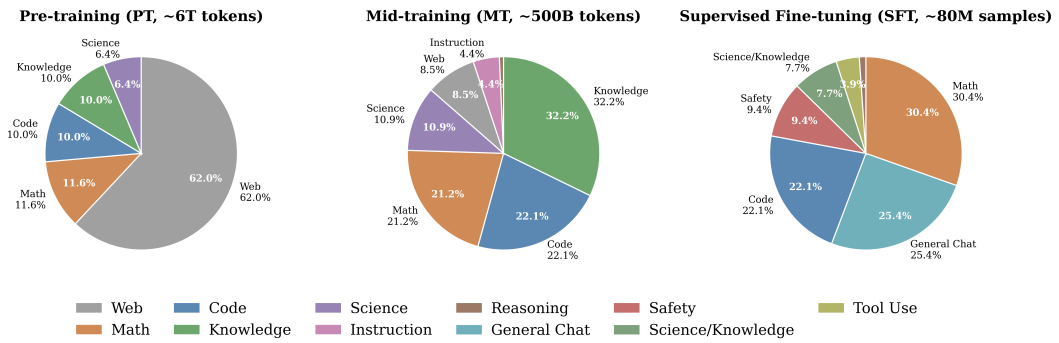


Figure B.1: **MobileMoE training data mixtures across stages.** Domain composition of the pre-training (PT, ∼6T tokens), mid-training (MT, ∼500B tokens), and supervised fine-tuning (SFT, >80M samples) corpora. Colors are consistent across stages for the same domain: web (gray) dominates pre-training but shrinks to <10% in mid-training, while knowledge (green), code (blue), and math (orange) progressively gain share. The full per-domain dataset lists are detailed in Tables B.2, B.3, and B.4.

where $\mathbf{W}_g$ denotes a contiguous group of weights sharing the same scale factor s_g, with group size g=32, and $q_{\min}$=-2^{b-1}, $q_{\max}$=$2^{b-1}-1$ define the INT4 range with b=4; activations are dynamically quantized to INT8, and the MoE router remains in FP32 throughout QAT to preserve routing stability under quantized gradients (negligible memory overhead of $< 0.5\%$). After QAT on the SFT data with standard cross-entropy loss (initialized from the SFT checkpoint; see Table B.1), this stage produces **MobileMoE-QAT** with weight footprint $\mathcal{M}_{\text{weight}}$ of 0.68 GB (S), 1.48 GB (M), 2.75 GB (L), with the theoretical memory proxy $\mathcal{M}$ (Eq. (5), weights + INT8 KV cache at 8k context) of 0.76 GB (S), 1.58 GB (M), 2.88 GB (L), all fitting within recent smartphone DRAM budgets.

B.3 Training Data

All training data across pre-training, mid-training, and SFT stages are publicly available under permissive open-source licenses (CC-BY-4.0, Apache 2.0, ODC-BY, MIT, NVIDIA License). Dataset names below are hyperlinked to their source repositories. Figure B.1 visualizes the data mixture composition across three training stages; per-domain dataset lists are detailed in Tables B.2, B.3, B.4.

Pre-training data. Our pre-training data of MobileMoE comes from two sources: (1) the Dolma3 mix from OLMo-3 [64], which provides quality-stratified Common Crawl, OCR-extracted scientific PDFs [65], multi-language code (Stack-Edu), and curated academic math (FineMath [13]); and (2) a curated data collection from MobileLLM models [66, 6], including quality-filtered web corpora (DCLM [67], FineWeb-Edu [68]), code (StarCoder [69], Stack-Edu, Nemotron Code [70]), math (FineMath [13], OpenWebMath [71], Algebraic Stack [72], Nemotron Math [70]), science (arXiv, peS2o, Nemotron Science [70], Natural Reasoning [73]), and knowledge (FLAN [74], StackExchange, Wikipedia, Cosmopedia). The combined pre-training data mix is web-heavy (62%), as web corpora are abundant and diverse text data, providing broad linguistic coverage to maximize general language modeling capacity. The remaining data is widely distributed across diverse domains: math (11.6%), knowledge (10%), code (10%), and science (6.4%) to build capabilities in reasoning, coding, and factual knowledge during pre-training. This domain diversity is particularly beneficial for MoE models, as exposure to heterogeneous data encourages expert specialization across different token types [7]. Table B.2 summarizes the domain breakdown.

Mid-training data. For mid-training to produce MobileMoE-Base, we shift the data distribution toward higher-quality, domain-specific sources while extending context length to 8,192. The mid-training mix combines two sources: (1) the Dolma3 Dolmino Mix [64], a curated collection spanning synthetic math, code, QA, reasoning traces, instruction-following, high-quality web, and scientific PDFs; and (2) selected subsets from our pre-training data with code, math, and long-document sources upweighted for 8K context learning. While pre-training is web-heavy (62%) for building general language modeling capacity, mid-training shifts toward higher-quality, domain-specific data: web is reduced (62%→9%) while knowledge (10%→32%), code (10%→22%), and math (12%→21%) are upweighted to strengthen downstream capabilities. This domain-concentrated mid-training further sharpens expert specialization in MoE, allowing routed experts to develop deeper expertise on domain-specific tokens. Table B.3 summarizes the mid-training domain breakdown.

Table B.2: **Pre-training data sources by domain.** Dataset names are linked to their source repositories.

Domain	Key Datasets	Weight (%)
Web	DCLM, FineWeb-Edu, Common Crawl	62.0%
Math	FineMath, OpenWebMath, Algebraic Stack, Nemotron Math	11.6%
Code	StarCoder, Stack-Edu, Nemotron Code	10.0%
Knowledge	FLAN, Cosmopedia, Wikipedia	10.0%
Science	OLMoCR Science PDFs, arXiv, peS2o, Nemotron Science, Natural Reasoning	6.4%

Table B.3: **Mid-training data sources by domain.** Dolmino† sources are from the Dolma3 Dolmino Mix.

Domain	Key Datasets	Weight (%)
Knowledge	FLAN, Cosmopedia, Wiki, Dolmino QA†	32.2%
Code	StarCoder, Stack-Edu, Nemotron Code, Dolmino Code†	22.1%
Math	FineMath, OpenWebMath, Algebraic Stack, Nemotron Math, Dolmino Math†	21.2%
Science	OLMoCR Science PDFs, arXiv, peS2o, Nemotron Science	10.9%
Web	DCLM, FineWeb-Edu, Dolmino Web†	8.5%
Instruction†	Dolmino Instruction†	4.4%
Reasoning†	Dolmino Reasoning Traces†	0.7%

Table B.4: **SFT data sources by domain** (>80M samples). Dataset names are linked to their repositories.

Domain	Key Datasets	# Samples	Weight (%)
Math	Nemotron PTD, Nemotron SFT, OpenMathInstruct, Puzzle-KD, Dolci, SmolTalk	39.2M	30.4%
Code	OpenCodeGeneticInstruct, OpenCodeInstruct, Nemotron SFT, Glaive, Dolci, Nemotron-SWE	23.1M	22.1%
General Chat	SmolTalk, Nemotron SFT, Dolci, Tulu-3 SFT, UltraChat, open-perfectblend, Daring-Anteater, FLAN, Orca-AgentInstruct, Retrieval-NVDocs, HelpSteer, OASST2	9.8M	25.4%
Science/Knowledge	OpenScience, Nemotron PTD, Nemotron SFT v2, Dolci	7.6M	7.7%
Reasoning	OmniThought, SmolTalk	0.8M	1.1%
Tool Use	Nemotron SFT, ToolACE, Synth-APIGen, SmolTalk	0.5M	3.9%
Safety	Nemotron PTD, Safety-Guard, PII, Aegis, Dolci	0.5M	9.4%

Supervised fine-tuning (SFT) data. We fine-tune MobileMoE-Base on a diverse mixture of open-licensed instruction-tuning datasets (>80M samples) from 28 public collections at 8K context length with sequence packing. The SFT mix spans multiple domains: math (30.4%), general instruction/chat (25.4%), code (22.1%), safety (9.4%), science/knowledge (7.7%), tool use (3.9%), and reasoning (1.1%), covering a broad range of capabilities for instruction-following. Each dataset is assigned a sampling weight $w_i = \max(1, \lfloor n_i/N \times 100 \rceil)$, where n_i is the dataset sample count and N is the total sample count, which assigns sampling probability proportional to dataset size while guaranteeing that small but important domains (e.g., safety, tool use) receive sufficient representation via the $\max(1, \cdot)$ floor. Table B.4 lists the SFT data sources by domain.

C Evaluation Details

C.1 Evaluation setup of base and instruct models on foundational competencies

We evaluate both pre-trained (base) and instruction-tuned (SFT) models on 14 foundational benchmarks using the Language Model Evaluation Harness (lm-eval) [75] with the vLLM backend (`dtype=auto`, resolved to bfloat16 at model precision; `add_bos_token=True`). All runs use `-batch_size auto:16` and greedy decoding (`temperature=0`, `do_sample=false`) for deterministic results. Instruct models are evaluated in non-thinking mode. We follow standard few-shot settings for both base and instruction models; per-task configurations are summarized in Table C.1.

Table C.1: **Evaluation benchmark configurations on foundational competencies (base and instruct models).** Evaluation is using lm-eval.

Category	Benchmark	Task name	n-shot	Metric
Commonsense Reasoning	HellaSwag	`hellaswag`	0	`acc_norm`
	PIQA	`piqa`	0	`acc_norm`
	SIQA	`social_iqa`	0	`acc`
	WinoGrande	`winogrande`	0	`acc`
Knowledge	MMLU	`mmlu`	5	`acc`
	NQ	`nq_open`	5	`exact_match`
	TQA	`triviaqa`	5	`exact_match`
Science	ARC-C	`arc_challenge`	25	`acc_norm`
	ARC-E	`arc_easy`	0	`acc_norm`
	OBQA	`openbookqa`	0	`acc_norm`
Reading	BoolQ	`boolq`	0	`acc`
	DROP	`drop`	3	`f1`
Reasoning	BBH-LB	`leaderboard_bbh`	3	`acc_norm`
	GSM8K	`gsm8k_cot`	8	`exact_match,flexible-extract`

C.2 Evaluation setup of instruct models on advanced competencies

For instruct-tuned models, we evaluate 8 advanced benchmarks spanning four capability axes: math, code, instruction following, and harder knowledge & reasoning. We use lm-eval [75] for MATH500, GSM-Plus, HumanEval MBPP, IFEval, and GPQA Diamond, and the official packages TIGER-AI-Lab/MMLU-Pro [59] and allenai/IFBench [58] for MMLU-Pro and IFBench. For lm-eval benchmarks, we use the vLLM backend (`dtype=auto`, resolved to bfloat16 at model precision; `add_bos_token=True`), `-batch_size auto:16`, and greedy decoding (`temperature=0`, `do_sample=false`) for deterministic results. Instruct models are evaluated in non-thinking mode. For MMLU-Pro and IFBench, we use the default settings from official packages. Per-task settings are summarized in Table C.2.

Table C.2: **Evaluation benchmark configurations on advanced competencies (instruct models).** Generation tasks use chat templates and greedy decoding ($T = 0$) unless noted; loglikelihood tasks do not generate.

Category	Benchmark	Task name / Script (Package)	Shots	Chat	Metric
Math	MATH500	`minerva_math500` (lm-eval)	4	No	`math_verify`
	GSM-Plus	`gsm_plus` (lm-eval)	5	Yes	`flexible-extract`
Code	HumanEval$_{p@1}$	`humaneval_instruct` (lm-eval)	0	Yes	pass@1 (greedy)
	MBPP	`mbpp_instruct` (lm-eval)	3	Yes	pass@1 (greedy)
IF	IFEval	`ifeval` (lm-eval)	0	Yes	Avg(strict/loose × prompt/inst)
	IFBench	`ifbench_generate.py` (AllenAI)	0	Yes	Avg(strict/loose × prompt/inst)
Knowledge	MMLU-Pro	`evaluate_from_local.py` (TIGER-Lab)	5 (CoT)	No	acc (regex `answer is (X)`)
Reasoning	GPQA Diamond	`gpqa_diamond_zeroshot` (lm-eval)	0	No	acc

C.3 Baseline Model Sources

All baseline models are publicly available. We list the HuggingFace model identifiers used in our evaluation for both base (pre-trained) and instruct (SFT) models in Table C.3.

Table C.3: **Baseline model sources.** HuggingFace identifiers are given for all base and instruct models.

Model	N_{act}	**Base (PT)**	**Instruct (SFT)**
Gemma 3 270M	270M	`google/gemma-3-270m`	`google/gemma-3-270m-it`
SmolLM2 360M	362M	`HuggingFaceTB/SmolLM2-360M`	`HuggingFaceTB/SmolLM2-360M-Instruct`
Granite-3.0-1B	400M/1.3B	–	`ibm-granite/granite-3.0-1b-a400m-instruct`
Qwen3.5 0.8B	749M	`Qwen/Qwen3.5-0.8B-Base`	`Qwen/Qwen3.5-0.8B`
Granite-3.0-3B	800M/3.3B	–	`ibm-granite/granite-3.0-3b-a800m-instruct`
Gemma 3 1B	1.0B	`google/gemma-3-1b-pt`	`google/gemma-3-1b-it`
MobileLLM-Pro	1.1B	`facebook/MobileLLM-Pro-base`	`facebook/MobileLLM-Pro`
Llama 3.2 1B	1.2B	`meta-llama/Llama-3.2-1B`	`meta-llama/Llama-3.2-1B-Instruct`
OLMoE-1B-7B	1.3B/6.9B	`allenai/OLMoE-1B-7B-0924`	`allenai/OLMoE-1B-7B-0924-Instruct`
OLMo 2 1B	1.5B	`allenai/OLMo-2-0425-1B`	`allenai/OLMo-2-0425-1B-Instruct`
SmolLM2 1.7B	1.7B	`HuggingFaceTB/SmolLM2-1.7B`	`HuggingFaceTB/SmolLM2-1.7B-Instruct`
Qwen3.5 2B	1.9B	`Qwen/Qwen3.5-2B-Base`	`Qwen/Qwen3.5-2B`
Jet-Nemotron-2B	2.0B/2.0B	–	`jet-ai/Jet-Nemotron-2B`
Jet-Nemotron-4B	4.0B/4.0B	–	`jet-ai/Jet-Nemotron-4B`

Table C.4: **MMLU-Pro and GPQA Diamond per-protocol ablation.** 5-shot MMLU-Pro and 0-shot GPQA Diamond across 13 instruct-tuned baselines under multiple evaluation protocols from TIGER-AI-Lab/MMLU-Pro and lm-eval. Main-body Table 2 reports MMLU-Pro with TIGER-AI-Lab/MMLU-Pro and GPQA Diamond with lm-eval (LLH). LLH: loglikelihood.

		MMLU-Pro (5-shot)			**GPQA Diamond** (0-shot)	
Model	$N_{\text{act}}/N_{\text{total}}$	TIGER-AI-Lab	lm-eval (LLH)	lm-eval (Chat)	lm-eval (LLH)	lm-eval (Chat)
Gemma 3 270M	270M	11.3	11.3	0.0	25.8	19.2
SmolLM2 360M	362M	12.0	10.9	9.9	25.8	**26.8**
MobileMoE-S	**272M/1.3B**	**18.2**	**18.1**	**14.0**	**27.8**	21.2
Qwen3.5 0.8B	749M	24.0	24.3	**31.2**	**26.3**	20.2
MobileMoE-M	**528M/2.8B**	**28.3**	**25.0**	26.1	24.8	**22.7**
Gemma 3 1B	1.0B	16.1	15.1	0.0	25.3	25.3
MobileLLM-Pro	1.1B	10.9	11.5	15.5	23.2	18.2
Llama 3.2 1B	1.2B	20.8	19.1	13.0	28.8	20.7
OLMo 2 1B	1.5B	16.0	15.9	15.0	29.8	26.3
SmolLM2 1.7B	1.7B	19.8	20.6	20.9	29.8	20.7
Qwen3.5 2B	1.9B	**38.8**	**31.5**	**48.9**	**34.3**	**43.9**
OLMoE-1B-7B	1.3B/6.9B	19.5	18.4	18.7	24.2	24.2
MobileMoE-L	**922M/5.3B**	34.0	29.3	33.9	33.8	26.8

C.4 MMLU-Pro and GPQA Diamond per-protocol ablation

Evaluation protocols. We compare MMLU-Pro and GPQA Diamond evaluation protocols in Table C.4. We report three MMLU-Pro (5-shot) variants: (1) the official package TIGER-AI-Lab/MMLU-Pro, which uses CoT generation, no chat template, and a subject-specific prompt; (2) lm-eval (LLH), which evaluates with deterministic loglikelihood scoring; and (3) lm-eval (Chat), which evaluates with CoT generation and chat template. For GPQA Diamond (0-shot), we report two variants: (1) lm-eval (LLH), multiple-choice loglikelihood scoring, and (2) lm-eval (Chat), CoT generation with chat template.

Analysis on evaluation protocols. As Table C.4 shows, applying the chat template hurts most baselines: MMLU-Pro with lm-eval (Chat) regresses on 9 of 13 models vs. the results with original evaluation package TIGER-AI-Lab (e.g., Gemma 3 1B 0.0 vs. 16.1, Llama 3.2 1B 13.0 vs. 20.8) while substantially benefiting only Qwen3.5 2B (48.9 vs. 38.8). Similarly, GPQA Diamond with lm-eval (Chat) shows the same pattern, helping Qwen3.5 2B ($+9.6$) while degrading 9 of 13 models. To avoid the bias and collapse brought by model-specific chat templates, Table 2 uses the no-chat protocols: TIGER-AI-Lab for MMLU-Pro, lm-eval (LLH) for GPQA Diamond, which give the better score for most models and ensure a consistent comparison among all models.

Table D.1: **Base model comparison**. $N_{\text{act}}/N_{\text{total}}$: active / total parameters including embeddings. Superscripts on benchmarks denote few-shot count (0-shot is omitted). **MobileMoE-S/M/L** refer to 272M/528M/922M active params and 1.3B/2.8B/5.3B total params. HS: HellaSwag, Wino: WinoGrande, NQ: NaturalQuestions, TQA: TriviaQA, ARC-C/E: ARC-Challenge/Easy, OBQA: OpenBookQA, BBH: BIG-Bench Hard. All values are benchmark accuracy (%). Evaluation details are given in Appendix C.

		Commonsense Reasoning				**Knowledge**			**Science**			**Reading**		**Reasoning**		
Model	$N_{\text{act}}/N_{\text{total}}$	HS	PIQA	SIQA	Wino	MMLU^5	NQ^5	TQA^5	ARC-C^{25}	ARC-E	OBQA	BoolQ	DROP^3	BBH^3	GSM8K^8	**Avg**
Gemma 3 270M	270M	41.4	68.3	40.2	53.7	26.7	4.1	14.3	29.4	56.8	30.4	58.3	14.2	29.5	1.8	33.5
SmolLM2 360M	362M	56.5	71.7	40.7	**59.0**	25.2	7.4	26.8	40.5	68.1	**37.6**	**61.8**	17.9	31.7	5.3	39.3
MobileMoE-S	**272M/1.3B**	**58.9**	**75.4**	**46.8**	58.6	**43.7**	**12.6**	**33.2**	**46.5**	**73.9**	34.6	60.2	**39.0**	**31.8**	**36.2**	**46.5**
Qwen3.5 0.8B	749M	54.9	71.3	42.1	59.9	48.2	6.2	19.5	44.0	67.6	36.0	74.6	39.6	**40.9**	44.1	46.4
MobileMoE-M	**528M/2.8B**	**68.3**	**77.5**	**50.2**	**61.6**	**54.7**	**20.9**	**49.0**	**51.0**	**79.4**	**39.4**	**75.1**	**58.5**	37.7	**51.6**	**55.4**
Gemma 3 1B	1.0B	62.2	74.9	42.9	58.8	26.2	10.7	35.7	39.3	72.2	36.8	66.6	23.4	30.5	2.3	41.6
MobileLLM-Pro	1.1B	66.2	76.6	48.4	63.2	32.3	15.6	43.2	52.5	76.6	43.2	**77.5**	22.5	33.0	6.6	47.0
Llama 3.2 1B	1.2B	64.2	75.1	42.8	61.3	31.4	12.0	40.7	40.3	61.8	36.6	63.6	27.9	29.8	7.3	42.5
OLMoE-1B-7B	1.3B/6.9B	**77.0**	**80.5**	43.9	**69.1**	52.6	20.6	**62.3**	55.0	76.6	**45.0**	74.8	29.8	33.5	12.3	52.4
OLMo 2 1B	1.5B	68.4	75.9	44.0	65.0	42.4	14.1	47.1	45.2	73.4	39.8	62.9	34.6	33.3	38.6	48.9
SmolLM2 1.7B	1.7B	71.4	77.6	44.2	66.1	50.2	15.4	49.6	53.3	73.4	43.8	72.4	27.3	34.0	31.0	50.7
Qwen3.5 2B	1.9B	65.9	74.7	43.5	64.6	54.1	10.9	32.6	54.3	71.4	37.8	69.4	53.6	**48.2**	**65.3**	53.3
MobileMoE-L	**922M/5.3B**	74.6	80.0	**54.3**	68.2	**59.6**	**26.7**	58.1	**57.0**	**81.7**	42.8	75.7	**64.7**	37.8	55.7	**59.8**

D Additional Experimental Results

D.1 Base Model Comparison

Results of MobileMoE-Base. Table D.1 compares MobileMoE-S/M/L against base-model baselines across the 14 foundational benchmarks. The base models are obtained after pre-training and mid-training, before instruction supervised fine-tuning. We have the following observations on comparison.

(1) *MobileMoE achieves state-of-the-art performance among sub-2B active-parameter models, at the end of mid-training.* MobileMoE-S (272M active, Avg 46.5) surpasses existing models at similar scale: Gemma 3 270M (33.5) by +13.0 and SmolLM2 360M (39.3) by +7.2. MobileMoE-M (528M active, Avg 55.4) outperforms all baselines up to 1.9B active parameters, including Qwen3.5 2B (53.3) by +2.1, OLMoE-1B-7B (1.3B active, 52.4) by +3.0, and SmolLM2 1.7B (50.7) by +4.7. MobileMoE-L (922M active, Avg 59.8) extends this lead, outperforming OLMoE-1B-7B by +7.4 at 30% fewer active parameters and 23% fewer total parameters (5.3B vs. 6.9B). Notably, MobileMoE-L Base already exceeds the *instruct-tuned* OLMoE-1B-7B (SFT Avg 55.6, Table 1) by +4.2, confirming that the strength of MobileMoE is already shown during pre-training and mid-training stages.

(2) *MobileMoE matches larger dense models at 2-4× fewer active parameters, with gains concentrated on knowledge and comprehension.* MobileMoE-S (272M, Avg 46.5) approaches Qwen3.5 0.8B (46.4) using 2.8× fewer active parameters. MobileMoE-M (528M, Avg 55.4) surpasses Qwen3.5 2B (53.3) using 3.6× fewer active parameters. MobileMoE-L (922M, Avg 59.8) outperforms Qwen3.5 2B by +6.5 using ∼2× fewer active parameters. When we compare both MoE models (MobileMoE-L and OLMoE-1B-7B) against the dense 1–2B LLMs (Qwen3.5 2B, SmolLM2 1.7B, Llama 3.2 1B, Gemma 3 1B, MobileLLM-Pro, OLMo 2 1B), the MoE advantage is strongest on *Knowledge* (MMLU, NQ, TQA) and *Reading* (BoolQ, DROP), which suggests the MoE architecture of larger *total* parameter capacity is best exploited on tasks demanding stored factual knowledge and comprehension.

D.2 Training Stage Progression

Comparison of capabilities across training stages. Figure D.1 trace how MobileMoE-S/M/L evolve across three main training stages – pre-training (PT), mid-training (MT), and instruction supervised fine-tuning (SFT). We also compare to the Base and SFT checkpoints of the three baselines of similar scales (SmolLM2 360M for MoE-S, Qwen3.5 0.8B for MoE-M, OLMoE-1B-7B for MoE-L) as references in Table D.2 on all 14 foundational benchmarks. We highlight four observations.

(1) *Pre-training dominates commonsense reasoning.* HellaSwag, PIQA, SIQA, and WinoGrande saturate by the end of PT and remain within ±2 points through MT and SFT across all three MobileMoE scales, indicating that broad linguistic priors are largely acquired from the 6T-token pre-training mixture and do not substantially improve in later mid-training or SFT.

(2) *Mid-training drives the largest knowledge and reading gains.* Upweighting curated knowledge, code, math, and long-document sources yields the largest PT→MT jumps on MMLU (MobileMoE-S:

Table D.2: **MobileMoE training stage progression with scale-matched reference baselines.** Detailed benchmark results for MobileMoE-S/M/L across pre-training (PT), mid-training (MT), and instruction SFT stages, together with Base and SFT results of scale-matched baselines (SmolLM2 360M for MoE-S, Qwen3.5 0.8B for MoE-M, OLMoE-1B-7B for MoE-L). Benchmarks and few-shot setup follow Table D.1.

			Commonsense Reasoning				Knowledge			Science			Reading		Reasoning		
Model	**Stage**	N_{act}/N_{total}	HS	PIQA	SIQA	Wino	MMLU[5]	NQ[5]	TQA[5]	ARC-C[25]	ARC-E	OBQA	BoolQ	DROP[3]	BBH[3]	GSM8K[8]	**Avg**
SmolLM2 360M	Base	362M	56.5	71.7	40.7	59.0	25.2	7.4	26.8	40.5	68.1	37.6	61.8	17.9	31.7	5.3	39.3
	SFT		56.9	71.6	40.6	57.4	25.9	6.4	20.4	38.8	49.1	36.2	42.5	15.2	30.5	10.0	35.8
	PT		60.5	73.1	46.8	59.0	33.5	11.9	33.0	45.0	67.5	34.8	61.9	39.0	31.7	29.4	44.8
MobileMoE-S	MT	**272M/1.3B**	58.9	75.4	46.8	58.6	43.7	12.6	33.2	46.5	73.9	34.6	60.2	39.0	31.8	36.2	46.5
	SFT		56.5	74.9	42.2	58.4	42.6	10.8	30.2	43.1	73.4	32.4	72.3	32.3	32.2	52.2	46.7
Qwen3.5 0.8B	Base	749M	54.9	71.3	42.1	59.9	48.2	6.2	19.5	44.0	67.6	36.0	74.6	39.6	40.9	44.1	46.4
	SFT		49.7	69.4	38.8	57.6	50.2	3.3	16.1	41.8	61.4	30.8	62.5	33.3	37.8	45.7	42.7
	PT		68.7	77.0	51.4	63.3	49.5	16.5	46.8	52.4	74.9	41.0	71.1	47.2	34.0	46.4	52.9
MobileMoE-M	MT	**528M/2.8B**	68.3	77.5	50.2	61.6	54.7	20.9	49.0	51.0	79.4	39.4	75.1	58.5	37.7	51.6	55.4
	SFT		66.6	77.6	49.0	63.0	53.9	17.5	46.6	52.5	79.9	38.2	76.7	46.6	39.0	67.5	55.3
OLMoE-1B-7B	Base	1.3B/6.9B	77.0	80.5	43.9	69.1	52.6	20.6	62.3	55.0	76.6	45.0	74.8	29.8	33.5	12.3	52.4
	SFT		78.8	79.7	50.8	68.7	52.7	17.2	54.1	57.6	75.9	46.8	81.1	29.3	37.1	49.1	55.6
	PT		74.3	79.4	52.5	67.3	55.5	22.6	57.6	57.0	75.6	44.0	74.2	54.2	36.3	52.2	57.3
MobileMoE-L	MT	**922M/5.3B**	74.6	80.0	54.3	68.2	59.6	26.7	58.1	57.0	81.7	42.8	75.7	64.7	37.8	55.7	59.8
	SFT		73.0	78.9	53.4	66.1	60.1	22.4	54.9	57.9	81.9	43.2	81.1	50.1	40.1	77.6	60.1



Figure D.1: **MobileMoE capability progression across training stages.** Benchmark accuracy (%) for MobileMoE-S/M/L after each of the three main training stages: pre-training (PT, light), mid-training (MT, medium), and instruction supervised fine-tuning (SFT, dark). Gray dashed line: instruct-tuned OLMoE-1B-7B (1.3B active) as a scale-matched MoE reference baseline. Each per-competence panel (Commonsense, Knowledge, Science, Reading, Reasoning) reports the average over its constituent benchmarks; the Overall panel reports the average over all 14 benchmarks. Full per-benchmark numerical results are in Table D.2.

33.5→43.7, +10.2; M: 49.5→54.7, +5.2; L: 55.5→59.6, +4.1) and DROP (M: 47.2→58.5, +11.3; L: 54.2→64.7, +10.5), confirming that mid-training is the primary mechanism for strengthening factual recall and long-context comprehension.

(3) *Instruction SFT unlocks math and multi-step reasoning.* GSM8K exhibits the most dramatic MT→SFT jumps (L: 55.7→77.6, +21.9; M: 51.6→67.5, +15.9; S: 36.2→52.2, +16.0), alongside consistent BoolQ and BBH gains, demonstrating that instruction tuning is essential for eliciting multi-step chain-of-thought reasoning that pre-training alone under-realizes. Overall foundational accuracy improves +2 to +3 points from PT to MT across all three MobileMoE scales, while SFT preserves foundational accuracy and unlocks math/reasoning gains.

(4) *MobileMoE-L Base already surpasses the instruct-tuned OLMoE-1B-7B.* MobileMoE-L already surpasses OLMoE-1B-7B Instruct after pre-training (Avg 57.3 vs. 55.6, Table D.2). At the end of mid-training, MobileMoE-L (Avg 59.8) further exceeds instruct-tuned OLMoE-1B-7B (SFT Avg 55.6, Table 1) by +4.2 with 30% fewer active parameters and 23% fewer total parameters; after SFT, MobileMoE-L surpasses OLMoE-1B-7B on the majority of benchmarks (dashed reference line in Figure D.1), confirming that MobileMoE's advantage is established during pre-training and is amplified by mid-training and instruction tuning.

D.3 Quantization-Aware Training (QAT) Results

Results of MobileMoE-QAT. Table D.3 compares INT4 QAT models of MobileMoE-S/M/L against the two publicly released QAT baselines with 4-bit quantization. We highlight our observations.

(1) *QAT preserves nearly all of MobileMoE's bf16 accuracy at 4× weight compression.* Compared against the corresponding bf16 SFT checkpoints (Table 1), MobileMoE's QAT incurs only a 2–3 point drop on overall average: MobileMoE-S 46.7→44.0 (−2.7), MobileMoE-M 55.3→52.5 (−2.8), MobileMoE-L 60.1→57.8 (−2.3). This degradation is small relative to the 4× weight-memory reduction from BF16 to INT4, indicating that the MoE router+experts structure remains

Table D.3: **Quantized model comparison with quantization-aware training (INT4 QAT).** All models are quantized to 4-bit weights via QAT (parameters updated during training to be robust to low-bit weights). Rows are ordered by increasing INT4 weight memory. **Mem**: INT4 model weight memory in GB. Baseline QAT checkpoints: Gemma 3 1B (`google/gemma-3-1b-it-qat-q4_0-gguf`) and MobileLLM-Pro (`facebook/MobileLLM-Pro-base-int4-accelerator`). Benchmark abbreviations follow Table D.1.

		Commonsense Reasoning				Knowledge			Science			Reading		Reasoning		
Model	Mem (GB)	HS	PIQA	SIQA	Wino	MMLU[5]	NQ[5]	TQA[5]	ARC-C[25]	ARC-E	OBQA	BoolQ	DROP[3]	BBH[3]	GSM8K[8]	**Avg**
Gemma 3 1B	0.50	34.8	54.2	33.4	50.9	24.7	3.2	6.9	25.3	32.5	28.4	54.6	1.5	30.8	0.3	27.3
MobileLLM-Pro	0.55	64.7	75.6	47.4	62.8	30.4	13.9	39.9	51.6	75.2	42.8	76.8	20.5	31.4	4.1	45.5
MobileMoE-S	**0.68**	**53.5**	**73.5**	**45.5**	**55.7**	**39.8**	**8.6**	**25.0**	**43.3**	**69.7**	**33.2**	**70.8**	**24.6**	**31.4**	**40.8**	**44.0**
MobileMoE-M	**1.48**	**63.7**	**76.7**	**49.0**	**61.2**	**52.4**	**5.0**	**42.1**	**51.2**	**79.1**	**36.6**	**75.8**	**43.2**	**36.2**	**62.6**	**52.5**
MobileMoE-L	**2.75**	**71.0**	**79.0**	**52.9**	**65.2**	**57.0**	**19.4**	**52.4**	**55.1**	**80.3**	**42.4**	**78.1**	**44.6**	**38.3**	**73.2**	**57.8**

numerically stable under aggressive quantization and that our QAT recipe (symmetric group-wise INT4 weights + FP32 router, Appendix B) effectively recovers the bf16 capability.

(2) *MobileMoE-QAT compresses to on-device memory budgets while remaining competitive with QAT baselines.* After INT4 QAT, MobileMoE-S/M/L have model weight memory ($\mathcal{M}_{\text{weight}}$) of only $0.68/1.48/2.75$ GB, with the theoretical memory proxy $\mathcal{M}$ (Eq. (5), weights + INT8 KV cache at 8k context) of $0.76/1.58/2.88$ GB, all fitting within modern mobile DRAM budgets ($\leq$3–5 GB) – the key prerequisite for on-device deployment. At similar weight memory, MobileMoE-S ($\mathcal{M}_{\text{weight}}$=0.68 GB, Avg 44.0) matches MobileLLM-Pro ($\mathcal{M}_{\text{weight}}$=0.55 GB, 45.5) within 1.5 points on overall average while substantially raising the knowledge and math results (MMLU +9.4, GSM8K +36.7); MobileMoE-M and MobileMoE-L further push the INT4 Pareto to Avg 52.5 ($\mathcal{M}_{\text{weight}}$=1.48 GB) and 57.8 ($\mathcal{M}_{\text{weight}}$=2.75 GB), with MobileMoE-L's INT4 accuracy already exceeding the bf16 SFT OLMoE-1B-7B (Avg 55.6) at less than half its weight memory footprint (2.75 GB INT4 vs. $\sim$13.8 GB bf16).

E Additional On-Device Runtime Profiling

On-device MoE inference. Existing mobile CPU inference backends such as XNNPACK provide highly optimized INT4 dense matmul kernels but lack a fused MoE feed-forward operator. We therefore implement a custom MoE operator in ExecuTorch guided by two principles: (1) *convert sparse expert dispatch into dense grouped GEMMs* – we first reorder tokens (via a counting sort on their assigned expert IDs) so that all tokens routed to the same expert sit contiguously in memory, letting each expert process its slice of tokens as a single dense batched matmul through torchao's INT4 GEMM kernel; and (2) *fuse every sub-operation inside each MoE FFN layer into a single op* – top-k expert selection (over router logits), token dispatch, per-expert gate- and up-projections (fused into one GEMM per expert), SwiGLU activation, down-projection, and weighted-scatter unpermute share one op call, amortizing kernel-launch and activation-quantization overhead. Attention and embedding layers continue to use the XNNPACK INT4 dense path, so MobileLLM-Pro runs end-to-end on XNNPACK while MobileMoE routes its MoE FFN blocks through our custom op.

Runtime profiling setup. We set up runtime profiling on two flagship smartphones: *Samsung Galaxy S25* (Snapdragon 8 Elite, using 4 CPU threads) and *iPhone 16 Pro* (Apple A18 Pro, using 2 CPU threads). We use the same ExecuTorch CPU backend with XNNPACK and batch size 1. Weights are INT4 symmetric with group size 32 (torchao INT4 packing); activations are INT8 dynamic per-row. We sweep input sequence lengths across two generation regimes: (1) short-context generation, e.g., on-device chat, with input $\in \{256, 512, 1024\}$ tokens and output $= 128$ tokens; and (2) long-context generation, e.g., on-device summarization, with input $\in \{2048, 4096, 8192\}$ tokens and output $= 1024$ tokens. The expert utilization ratio of MoE is input-dependent: the portions of activated experts vary according to the input token types (Appendix F). Therefore, rather than following the standard LLM runtime profiling that feeds dummy prompts of fixed sequence lengths (e.g., repeating the same words or using random tokens), we use real prompts across different domains (knowledge, code, math), and run inference 3 times per prompt to compute the average runtime benchmark.

Results of on-device runtime profiling on latency. Table E.1 reports the full runtime sweep over $\{256, 512, 1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}\}$ sequence lengths on Samsung Galaxy S25 and iPhone 16 Pro, comparing MobileMoE-S/M/L to MobileLLM-Pro, deployed with INT4 quantization and ExecuTorch+XNNPACK backend. We highlight these findings.

Table E.1: **On-device runtime latency**: comparing MobileMoE-S/M/L against the dense MobileLLM-Pro on two flagship smartphones – Samsung Galaxy S25 (Snapdragon 8 Elite, 4 CPU threads) and iPhone 16 Pro (Apple A18 Pro, 2 CPU threads), both via ExecuTorch+XNNPACK backend with INT4 weights and INT8 dynamic activations. **Prefill TTFT** (s, ↓): time-to-first-token. **Decode Rate** (tok/s, ↑): generation throughput. Both averaged over multiple runs on real prompts (code, knowledge, math). **Mem** (GB): static INT4 weight memory; **Avg** (%, ↑): mean accuracy over 14 benchmarks after QAT (Table D.3). Best results per column are in **bold**.

			Samsung Galaxy S25												**iPhone 16 Pro**											
	Mem	**Avg**	**Prefill TTFT (s) ↓**						**Decode Rate (tok/s) ↑**						**Prefill TTFT (s) ↓**						**Decode Rate (tok/s) ↑**					
Model	(GB)	(%)	256	512	1k	2k	4k	8k	256	512	1k	2k	4k	8k	256	512	1k	2k	4k	8k	256	512	1k	2k	4k	8k
MobileLLM-Pro	**0.55**	45.5	0.78	1.73	4.26	14.21	35.62	105.21	61.3	56.5	45.8	27.6	18.6	10.6	1.29	2.90	6.03	14.13	39.08	122.42	61.0	55.1	48.5	32.4	17.9	10.6
MobileMoE-S	0.68	44.0	**0.44**	**0.90**	**2.01**	**7.82**	**16.36**	**50.33**	**138.1**	**130.0**	**112.0**	**61.7**	**48.9**	**25.8**	**0.46**	**1.02**	**2.14**	**5.23**	**13.86**	**39.91**	**204.6**	**180.5**	**148.4**	**90.2**	**54.2**	**32.2**
MobileMoE-M	1.48	52.5	0.84	1.79	4.32	16.68	35.02	88.25	83.6	77.1	64.6	34.5	26.2	15.4	1.01	2.23	4.71	10.82	31.43	83.54	106.8	95.3	76.2	51.0	27.9	17.2
MobileMoE-L	2.75	**57.8**	1.53	4.09	8.99	25.85	55.12	162.45	53.4	43.0	36.6	22.9	17.3	8.9	1.86	4.32	8.63	21.92	51.98	141.31	59.4	51.9	43.3	23.5	17.1	10.8

Table E.2: **MoE runtime speedup across mobile devices and processors (CPU, GPU)**: Samsung Galaxy S25 (Snapdragon 8 Elite CPU, XNNPACK), iPhone 16 Pro (Apple A18 Pro CPU, XNNPACK), and iPhone 16 Pro (Apple A18 Pro Metal GPU, MLX). **Prefill TTFT** (s, ↓): time-to-first-token. **Decode Rate** (tok/s, ↑): generation throughput. Both averaged over multiple runs. **Mem** (GB): static INT4 weight memory; **Avg** (%, ↑): mean accuracy over 14 benchmarks after QAT (Table D.3). The **MoE Speedup** row reports MobileMoE-S vs. MobileLLM-Pro per metric per context.

			Samsung S25 (Snapdragon CPU, XNNPACK)						**iPhone 16 Pro (Apple CPU, XNNPACK)**						**iPhone 16 Pro (Apple GPU, MLX)**					
	Mem	**Avg**	**Prefill TTFT (s) ↓**			**Decode (tok/s) ↑**			**Prefill TTFT (s) ↓**			**Decode (tok/s) ↑**			**Prefill TTFT (s) ↓**			**Decode (tok/s) ↑**		
Model	(GB)	(%)	512	1k	2k	512	1k	2k	512	1k	2k	512	1k	2k	512	1k	2k	512	1k	2k
MobileLLM-Pro	**0.55**	**45.5**	1.73	4.26	14.21	56.5	45.8	27.6	2.90	6.03	14.13	55.1	48.5	32.4	0.58	1.20	2.49	61.8	59.2	56.3
MobileMoE-S	0.68	44.0	**0.90**	**2.01**	**7.82**	**130.0**	**112.0**	**61.7**	**1.02**	**2.14**	**5.23**	**180.5**	**148.4**	**90.2**	**0.16**	**0.32**	**0.68**	**154.3**	**151.3**	**141.9**
MoE Speedup	—	—	1.9×	2.1×	1.8×	2.3×	2.5×	2.2×	2.8×	2.8×	2.7×	3.3×	3.1×	2.8×	3.6×	3.8×	3.7×	2.5×	2.6×	2.5×

(1) *MobileMoE-S achieves a Pareto win at every context length on both smartphones:* at comparable INT4 weights (0.68 vs. 0.55 GB) and similar accuracy (Avg 44.0 vs. 45.5) to MobileLLM-Pro, on *Samsung S25* MobileMoE-S is 1.8–2.2× faster at prefill and 2.2–2.6× faster at decode. On *iPhone 16 Pro*, the latency speedup widens further to 2.7–3.1× at prefill and 2.8–3.4× at decode. These results demonstrate that the architectural advantage of MobileMoE-S generalizes across mobile devices with different silicon (Qualcomm vs. Apple), attributed to its much smaller compute FLOPs and per-token memory bandwidth.

(2) *MobileMoE-M/L delivers substantially higher accuracy ($+7.0$/$+12.3$ Avg over MobileLLM-Pro) with comparable or modest runtime cost:* on *Samsung S25*, MobileMoE-M (Avg 52.5 vs. 45.5, with +7.0 boost) achieves prefill parity at short context and 1.0–1.2× at ≥ 4k (overall 0.9–1.2×), with a steady 1.3–1.5× decode speedup; MobileMoE-L (Avg 57.8 vs. 45.5, with +12.3 boost), the highest-accuracy variant, delivers this large accuracy gain at a moderate runtime drop (0.4–0.7× prefill, 0.8–0.9× decode) that shrinks with context. On *iPhone 16 Pro*, runtime speedup ratios over the dense baseline strengthen further: MobileMoE-M outperforms MobileLLM-Pro at 1.2–1.5× prefill and 1.6–1.8× decode, while MobileMoE-L narrows its drop to 0.6–0.9× prefill and 0.7–1.0× decode (parity at 8k decode) with substantial performance gain. These results demonstrate that MobileMoE-M/L's substantial accuracy gains at competitive runtime hold consistently across mobile devices, establishing a new on-device Pareto frontier beyond MobileLLM-Pro at higher accuracy.

Why can MoE achieve better on-device latency? As shown in Table E.2, MobileMoE-S yields consistent 1.8–3.8× prefill and 2.2–3.4× decode speedups over the dense MobileLLM-Pro across mobile silicon (Qualcomm vs. Apple), processors (CPU vs. GPU), and inference backends (XNNPACK vs. MLX), at comparable accuracy and INT4 model memory. This advantage stems from MoE's much smaller active-parameter count ($< \frac{1}{3}$ of dense at comparable accuracy), which reduces both per-token compute and memory bandwidth at inference. Prefill is *compute-bound*: per-token FFN matmul scales with active (not total) parameters, so MoE's smaller N_{active} shrinks the dominant prefill cost. Decode is *memory-bandwidth-bound*: per-step weight reads from RAM also scale with active parameters, so MoE transfers fewer bytes per token, yielding higher throughput. Critically, our custom MoE kernel turns these theoretical FLOPs and bandwidth savings into measurable on-device speedups, empirically realizing the compute and memory efficiency derived in Section 3.3.

Results of on-device profiling on peak runtime memory. Table E.3 reports Peak RSS on Samsung Galaxy S25 over different sequence lengths on real prompts and dummy prompts, where Peak RSS is the maximum runtime RAM during inference, including resident weights, KV cache, transient activations, and runtime overhead, which is therefore larger than the static INT4 weight memory and can be optimized and reduced via various runtime memory optimizations (e.g., paged KV cache,

Table E.3: **On-device peak runtime memory**: comparing MobileMoE-S/M/L against the dense MobileLLM-Pro on Samsung Galaxy S25 (Snapdragon 8 Elite, 4 CPU threads), via ExecuTorch+XNNPACK backend with INT4 weights and INT8 dynamic activations. **Peak RSS** (GB, ↓): maximum runtime RAM during inference, including resident weights, KV cache, transient activations, and runtime overhead, averaged over multiple runs. We report Peak RSS separately under *real prompts* (code, knowledge, math) and *dummy prompts* (repeated tokens). **Mem** (GB): static INT4 weight memory; **Avg** (%, ↑): mean accuracy over 14 benchmarks after QAT (Table D.3). Best results per column are in **bold**.

			Peak RSS, real prompts (GB) ↓						Peak RSS, dummy prompts (GB) ↓					
Model	**Mem (GB)**	**Avg (%)**	256	512	1k	2k	4k	8k	256	512	1k	2k	4k	8k
MobileLLM-Pro	**0.55**	45.5	**0.90**	**0.93**	**0.98**	**1.07**	1.35	1.91	0.90	0.92	0.98	1.07	1.32	1.87
MobileMoE-S	0.68	44.0	0.93	0.97	1.02	1.10	**1.23**	**1.49**	**0.63**	**0.61**	**0.75**	**0.85**	**0.86**	**1.11**
MobileMoE-M	1.48	52.5	1.98	2.04	2.12	2.24	2.43	2.77	1.10	1.27	1.15	1.20	1.44	1.91
MobileMoE-L	2.75	**57.8**	3.66	3.75	3.87	4.06	4.27	4.71	1.72	1.99	1.84	2.41	3.49	3.41

activation reuse, kernel fusion). We use Peak RSS here as a conservative upper bound on on-device memory usage. We highlight these findings.

(1) *Real prompts are essential for valid MoE memory profiling:* Recall that our runtime profiling is conducted with real prompts (code, knowledge, math) rather than dummy prompts (repeated tokens). On Samsung S25, Peak RSS under real prompts rises to 1.2–$2.1\times$ that of dummy prompts for MobileMoE-S/M/L, while staying at $\sim 1.0\times$ for the dense MobileLLM-Pro. This asymmetry reflects MoE's input-dependent expert routing at runtime: real prompts activate diverse experts and load more expert weights into RAM, whereas dummy prompts trigger a narrow routing pattern that loads fewer experts. Dummy-prompt RSS thus captures only the *lower bound* of MoE memory usage, while real-prompt RSS faithfully reflects actual runtime behavior.

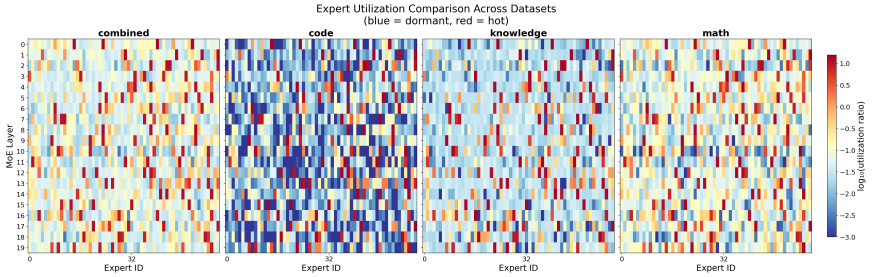
(2) *MobileMoE-S matches dense RAM at short context and saves substantial RAM at long context:* At comparable INT4 weight memory (0.68 vs. 0.55 GB), MobileMoE-S Peak RSS is within $\sim 5\%$ of MobileLLM-Pro at short context ($\leq$ 1k) and substantially lower at long context – 9% lower at 4k (1.23 vs. 1.35 GB) and 22% lower at 8k (1.49 vs. 1.91 GB), which is driven by fewer transformer layers (smaller KV cache), narrower d_{model} (smaller activation buffers), and the mmap loading of only activated experts into RAM. Notably, despite having slightly more total parameters than the dense baseline, the runtime memory and latency advantages of MobileMoE-S hold at comparable INT4 weight footprint.

(3) *MobileMoE-M/L trade extra RAM for substantially higher accuracy, all fitting within commodity on-device DRAM budgets:* relative to MobileLLM-Pro, MobileMoE-M's Peak RSS overhead falls from $\sim 2.2\times$ at short context to $1.45\times$ at 8k, and MobileMoE-L's from $4.1\times$ at 256 to $2.5\times$ at 8k. Even at the largest scale, MobileMoE-L's Peak RSS stays under 5 GB at 8k context (4.71 GB) – comfortably within modern mobile DRAM budgets, confirming that all MobileMoE variants are practical for on-device deployment.

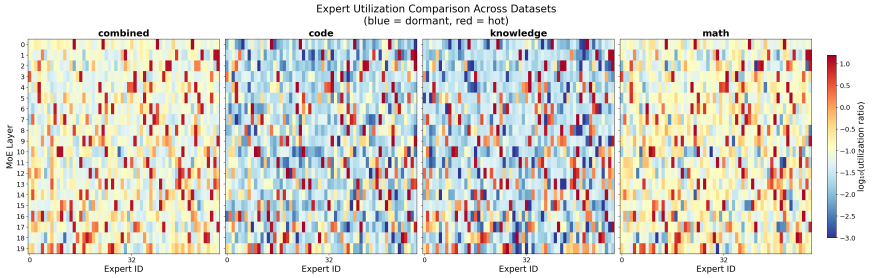
F Quantitative Analysis

Visualization of MobileMoE expert utilization. We visualize per-layer expert utilization patterns across downstream tasks: code, math, knowledge for MobileMoE-S after pre-training (PT), mid-training (MT), and supervised fine-tuning (SFT) stages, where lower utilization (blue) indicates dormant experts and higher utilization (red) indicates frequently activated experts. Figure F.1 reveals these patterns: **(1)** *Expert specialization differs across tasks:* on different domains, different subsets of experts are activated, suggesting the 60 fine-grained experts specialize across distinct domains. **(2)** *Expert utilization broadens through training:* at PT, fewer experts are highly utilized; through MT and SFT, more experts are progressively activated, indicating that downstream training broadens expert utilization while maintaining cross-task specialization.

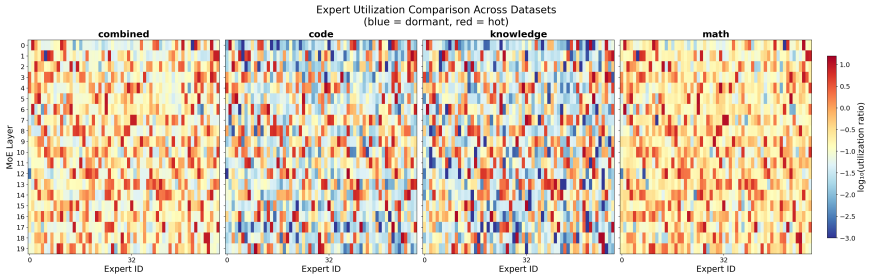
Expert utilization statistics. The distribution of expert utilization ratios varies across downstream tasks (Figure F.2): math activates a broader set of experts, while code or knowledge tasks concentrate on a narrower subset. This task-dependent sparsity indicates that not every expert weight needs to be loaded at inference, opening a path to save on-device memory via selective expert loading or task-conditional pruning.



(a) Pre-training (PT) stage



(b) Mid-training (MT) stage



(c) Supervised fine-tuning (SFT) stage

Figure F.1: **MobileMoE expert utilization heatmaps across training stages and domains.** Each heatmap shows per-layer (rows) per-expert (columns) activation utilization on the $\log_{10}$ scale (blue: dormant, red: hot) for MobileMoE-S evaluated on three downstream domains. Two patterns emerge: (i) different domains activate distinct expert subsets (cross-task specialization), and (ii) expert utilization broadens progressively from PT through MT to SFT.

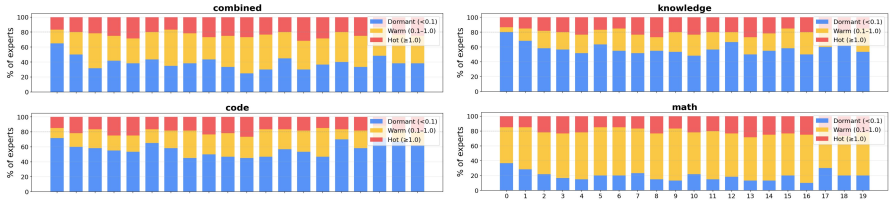


Figure F.2: **Distribution of expert utilization across tasks: math, code, knowledge, combined (all tasks).** Statistics of expert utilization ratios ($\log_{10}$ scale) are shown across all MoE layers for MobileMoE-S (post SFT).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The three main contributions claimed in the abstract and introduction — (i) the on-device MoE scaling law, (ii) the end-to-end training recipe yielding strong benchmark performance, and (iii) the first efficient MoE inference on commodity smartphones — are supported by the experimental results in Section 4 and the Appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: We discuss the limitations in the Conclusion section: our training recipe can be further improved with distillation from larger teacher models and reasoning-focused post-training, which are adopted in recent state-of-the-art LLMs but not yet incorporated in our work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: The paper states all assumptions underlying the on-device MoE scaling law, including the parametric functional form (Eq. 1), the memory function definition with INT4 weights and INT8 KV cache (Eq. 5), and the optimization objective with mobile DRAM and compute constraints (Eq. 4). The reduced forms of the scaling law are derived as special cases of established prior work (Joint MoE law [35] and Chinchilla scaling law [9]). Parametric fitting and empirical validation details are provided in Appendix A.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: All information including implementation details and dataset construction are disclosed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: We use publicly available datasets and publicly release the MobileMoE model checkpoints at `https://huggingface.co/collections/facebook/mobilemoe`. Implementation and evaluation details are provided in the paper and appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: We add detailed implementation details seen in Experiments section and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes].

   Justification: For scaling-law fitting, we report root-mean-square error (RMSE) of the fit across all sweep runs (Appendix A.1). For on-device runtime profiling, we report mean over multiple independent runs (3 runs in Appendix E). For benchmark accuracy, we follow

standard practice of reporting point estimates from the official evaluation protocols of lm-eval-harness, TIGER-AI-Lab/MMLU-Pro, and allenai/IFBench.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Training experiments (scaling-law ablations and MobileMoE pre-training, mid-training, SFT, and QAT) are conducted on NVIDIA H100 96 GB GPUs, with per-experiment compute (number of GPUs, batch size, training tokens, and wall-clock time) reported in Appendix B. On-device runtime profiling is conducted on Samsung Galaxy S25 (Snapdragon 8 Elite) and iPhone 16 Pro (Apple A18 Pro), with details in Section 4.3 and Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: We follow the rule.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

    Justification: As discussed in Section 2, on-device LLMs enable fast, private, and cost-effective inference at the edge – positive impacts MobileMoE inherits. On the negative side, MobileMoE inherits standard LLM concerns (e.g., bias, hallucination, potential misuse) common to all language models, not specific to our MoE approach.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not introduce new datasets or models with elevated risk of misuse beyond standard LLM concerns. MobileMoE is a small (sub-billion active) on-device language model trained on publicly available open-licensed data; potential general LLM concerns (bias, hallucination, misuse) are already addressed under Broader Impacts.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: We cite all models, datasets used in the paper, with full lists in Appendix B and Appendix C (training data sources, evaluation benchmarks, baseline model HuggingFace identifiers). All assets are used under permissive open-source licenses (e.g., CC-BY-4.0, Apache 2.0, ODC-BY, MIT, NVIDIA License).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: We introduce and publicly release MobileMoE model checkpoints at `https://huggingface.co/collections/facebook/mobilemoe`.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: No research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: No research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: LLM is only for paper formatting.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.