MobileMoE: Scaling On-Device Mixture of Experts

Yanbei Chen[1], **Hanxian Huang**[1], **Ernie Chang**[1], **Jacob Szwejbka**[1], **Digant Desai**[1], **Zechun Liu**[1], **Vikas Chandra**[1], **Raghuraman Krishnamoorthi**[1]

[1]Meta AI

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× 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× faster prefill and 2.2-3.4× faster decode than the dense baseline MobileLLM-Pro.

Date: May 27, 2026
Correspondence: Yanbei Chen at yanbeichen@meta.com
Detailed author contributions can be found in the Author Contributions section.

Meta

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 [36], Qwen3 MoE [63]) and proprietary frontier models (e.g., Gemini [55], Grok [61]). However, on-device LLMs remain overwhelmingly dense (e.g., MobileLLM [38], MobileLLM Pro [24]), 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 [63]. 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) [52], 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, 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 [21, 29] and MoEs [8, 30], existing frameworks overwhelmingly focus on scaling models up to tens or hundreds of billions of parameters for deployment on cloud servers. To address practical

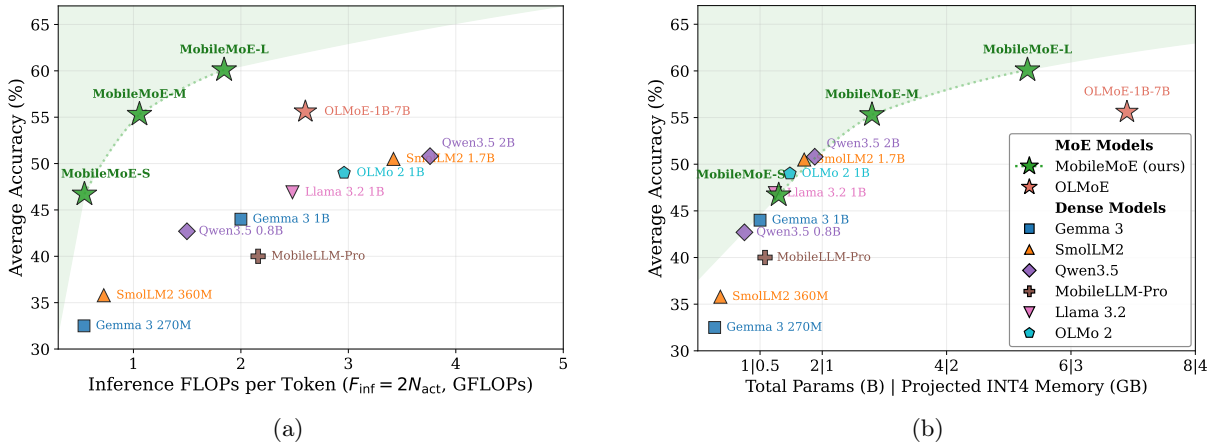


Figure 1 MobileMoE establishes a new Pareto frontier for on-device LLMs. Average benchmark accuracy, computed over 14 benchmarks spanning commonsense, knowledge, science, comprehension, and reasoning, is plotted against (a) per-token inference compute $F_{\text{inf}} = 2N_{\text{act}}$ (GFLOPs) and (b)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

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 ∼6T pre-training tokens, MobileMoE matches or surpasses dense baselines trained on 1.5-2× more tokens (e.g., 9T for Llama 3.2 1B [18], 11T for SmolLM2 1.7B [40]), 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× 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 [43], MobileMoE-M matches its accuracy with ∼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× faster prefill and 2.2–3.4× faster decode than the dense baseline MobileLLM-Pro [24] 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.

Our contributions are three-fold:

[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).

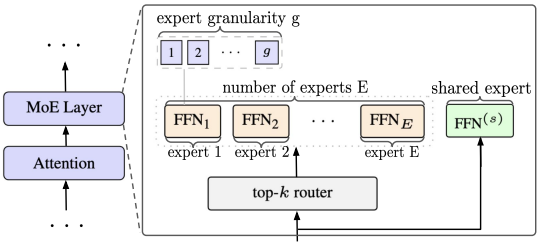
1. We introduce MobileMoE-S/M/L, the first sub-billion-active MoE family for on-device deployment, derived based on a generalized on-device MoE scaling law under joint memory and compute constraints. Guided by this scaling law, we identify the sweet-spot MoE design choices for on-device use cases (moderate sparsity, fine-grained granularity, and a shared expert) that together define MobileMoE.
2. We propose a four-stage training recipe (pre-training → mid-training → SFT → INT4 QAT) with MoE-specific stability and efficiency techniques. The recipe scales MobileMoE to Pareto-leading accuracy at only ∼6T pre-training tokens – substantially fewer than dense baselines (9T for Llama 3.2 1B, 11T for SmolLM2), while surpassing the state-of-the-art MoE OLMoE-1B-7B with fewer total parameters.
3. We deploy MobileMoE on commodity smartphones (Samsung Galaxy S25, iPhone 16 Pro) via a custom fused MoE kernel in ExecuTorch with systematic runtime profiling. MobileMoE-S achieves 1.8–3.8× faster prefill and 2.2–3.4× faster decode than the dense MobileLLM-Pro at comparable INT4 weight memory, establishing MoE as a practical path for efficient on-device LLMs.

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. A growing body of recent work has introduced dense on-device LLMs at sub-billion to few-billion scales: MobileLLM [38], MobileLLM-Pro [24] adopt deep-and-thin architectures to maximize parameter efficiency at sub-billion scales, SmolLM [40], Gemma [56, 57] provide families of small LLMs with competitive quality, and MobileLLM-Flash [23], Nemotron-Flash [16] use architecture search to optimize on-device latency. These efforts focus exclusively on dense architectures, where scaling model quality inherently demands increasing higher active parameter counts and inference compute. We pursue MoE as a complementary path that expands model capacity at minimal per-token compute for efficient on-device deployment.

Mixture of Experts (MoE) offers a parameter-efficient paradigm by routing tokens to a sparse subset of specialized expert networks [15, 26, 32, 52, 68, 69]. Concretely, MoE expands the learning capacity of modern transformers by replacing the dense feed-forward block in each layer with a set of expert subnetworks, increasing total parameters while keeping active parameters compact through sparse routing. Beyond parameter scaling, MoE also enables expert specialization: the routing mechanism learns to assign different token types to dedicated experts, allowing subnetworks to specialize in distinct linguistic tasks [52] and broader multimodal domains [32]. By decoupling total parameters from active inference compute, MoE has driven the scaling of state-of-the-art LLMs, e.g., Mixtral [27], DeepSeek-MoE [12, 36], and Qwen-MoE [62, 63]. While efforts such as OLMoE [43] have explored smaller scales, the sub-billion active-parameter regime – where on-device LLMs operate efficiently [38] – remains unexplored under practical edge constraints. Our work specifically studies MoE at this scale, with systematic analyses of architectural choices under on-device constraints.

Scaling Laws characterize power-law relationships between compute, data, and parameters [21, 29], providing a principled foundation for LLM development, covering compute-optimal parameter-data allocation [18, 21], training hyperparameters [4], learning rate schedules [22], and data mixtures [53]. Scaling laws have also been extended to MoE, studying expert count [8], expert granularity [30], and expert allocation under memory constraints [41]. These existing formulations, however, primarily target server-scale LLMs, where abundant hardware resources make large model memory footprints feasible while inference can be parallelized across server GPUs to improve runtime efficiency. By contrast, on-device deployment requires jointly considering inference cost and memory footprint, which are governed by active and total parameters, respectively. 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.



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 $\{\checkmark, \times\}$.

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 [36] uses 256 fine-grained experts with top-8 routing and a shared expert, Qwen3-MoE [63] uses 128 experts with top-8 routing but no shared expert, and Mixtral [27] 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 [6] 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 [38], MobileLLM Pro [24]) 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 law.

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.

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 (i.e., 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 $N_{\text{act}}, N_{\text{total}}$, 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* [41]:

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

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 $\frac{1}{\hat{E}} = \frac{1}{E-1+\left(\frac{1}{E_{\text{start}}} - \frac{1}{E_{\max}}\right)^{-1}} + \frac{1}{E_{\max}}$ [8].

Reduced form II ($\mathcal{L}|_{\hat{E}}$). With expert count E fixed, Eq. (1) reduces to *Chinchilla scaling law* [21]:

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

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

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

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 [14, 24]. Following this practice, we formulate the memory function as

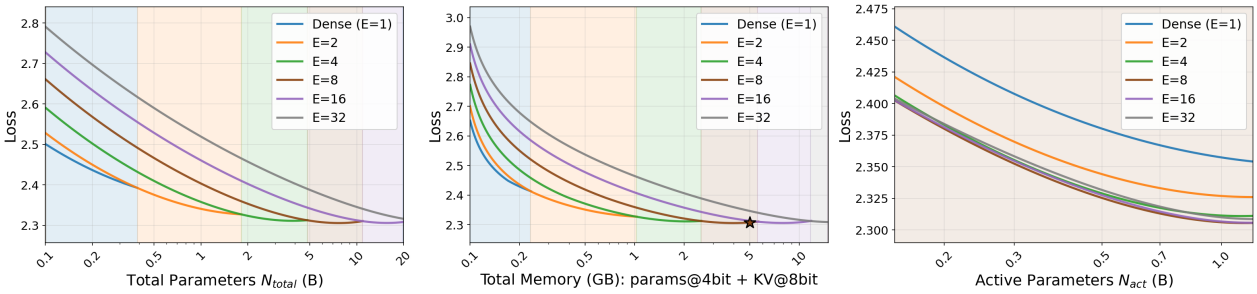
$$\mathcal{M}(N_{\text{total}}, T) = \underbrace{\tfrac{b_w}{8} N_{\text{total}}}_{\mathcal{M}_{\text{weight}}} + \underbrace{\tfrac{b_{\text{kv}}}{8} \cdot 2T n_l n_{\text{kv}} d_h}_{\mathcal{M}_{\text{KV cache}}}, \text{with } \mathcal{M}(N_{\text{total}}, T) \le M \tag{5}$$

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* for the on-device memory required to host the model (static model weights and KV cache, persistent throughout inference), excluding transient activation buffers and runtime overhead which can be optimized via runtime techniques. M is the on-device memory budget.

3.3 Finding the Optimal On-Device MoE

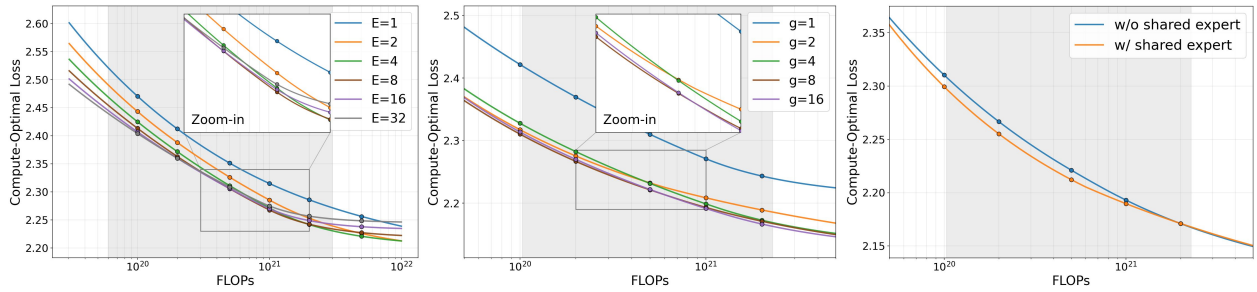
The on-device MoE scaling law (Eq. (1)) and optimization objective (Eq. (4)) serve as the principled foundation to govern our MobileMoE architecture design under compute (F_{train}, F_{inf}) and memory ($\mathcal{M}$) constraints. A naïve joint sweep over the three design axes in Figure 2 (number of experts E, expert granularity g, and shared expert s) would incur a combinatorial 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 changes the expert networks yet preserves both N_{act} and N_{total}, and s adds a shared dense pathway, where the shared expert can be sized to retain both N_{act} and N_{total} – so the memory and compute-optimal E is preserved under any subsequent choice of g or s. We therefore adopt a divide-and-conquer approach, decomposing the architecture optimization into three controlled ablation studies, each isolating one factor while holding the others fixed, to progressively determine the optimal on-device MoE architecture grounded in the on-device MoE scaling law (Section 3.2).

Scaling the number of experts E. Given fixed active parameters N_{act}, the number of experts E 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 $N_{\text{act}} \in \{0.3\text{B}, 0.5\text{B}, 0.9\text{B}\}$ (with the largest (E, N_{act}) combinations exceeding the 5 GB budget); each model is trained



(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 $\mathcal{M} = 5\,\text{GB}$, the optimal E is 8, marked by the star in (b).



(a) (b) (c)

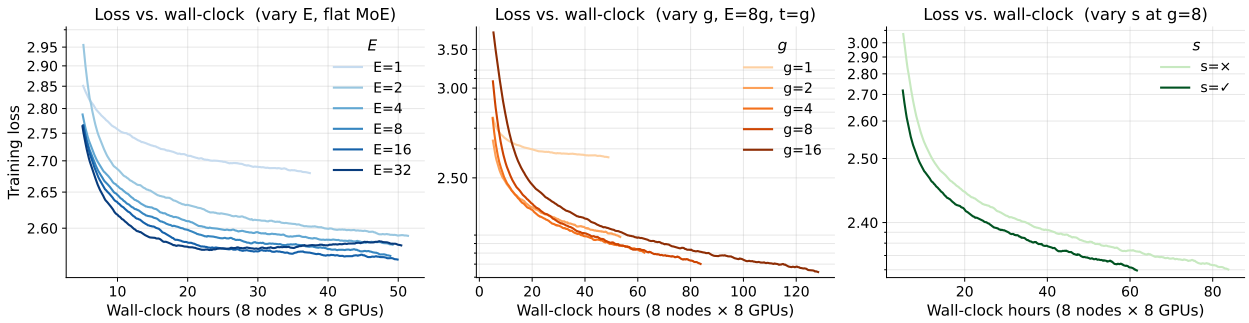
Figure 4 Scaling MoE models with compute optimality. The compute-optimal loss vs. training compute FLOPs for each MoE design factor predicted by the on-device scaling law (Eq. (1)): when varying (a) number of experts E, (b) expert granularity g, (c) with vs without shared expert, the curves with lower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

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 [21], 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 ($M > 0.25\,\text{GB}$), MoE models ($E > 1$) achieve lower loss than dense models (Figure 3(b)). With fixed compute F_{inf} (fixed N_{act}) and F_{train}, increasing E reduces loss with diminishing returns beyond $E = 8$ (Figure 3(c), Figure 4(a)). While the optimal E grows with more memory, moderate sparsity ($E \in \{4, 8\}$) is the practical sweet spot in the 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 3(c)), while remaining within the practical sweet spot under on-device memory constraints (e.g., $\leq 5\,\text{GB}$ in Figure 3(b)).

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 when scaling g. 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 [12]. 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($E = 8$) (derived in Finding 1) under the same experimental regime of N_{act} and D, and solves for the compute-optimal g (Figure 4(b)).



(a) (b) (c)

Figure 5 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.

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

Following Finding 2, we adopt a compute-optimal granularity of $g = 8$ for MoE($E = 8$), which results in 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 [12] and Qwen2MoE [62], yet omitted in OLMoE [43] and Qwen3MoE [63]. To isolate the architectural effect of the shared expert, we compare MoE($E = 8, g = 8$) with and without a shared expert by replacing 4 of the 8 active routed experts with the shared expert (4× the size of a routed fine-grained expert), yielding 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 (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 and g fixed, under the same experimental regime of N_{act} and D to identify the optimal setting of s (Figure 4(c)).

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

Guided by Finding 3, the shared expert (generalist) complements routed experts (specialists). We adopt the shared expert to derive our **MobileMoE** architecture: 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 efficiency of MoE architecture. Figure 5 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 5(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 5(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 (∼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 5(c)), adding the shared expert ($s = \checkmark$) further

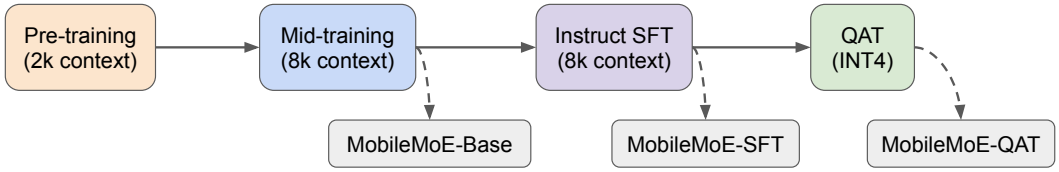


Figure 6 MobileMoE four-stage training recipe: pre-training (PT) → mid-training (MT) → instruct supervised fine-tuning (SFT) → quantization-aware training (QAT) with INT4 precision.

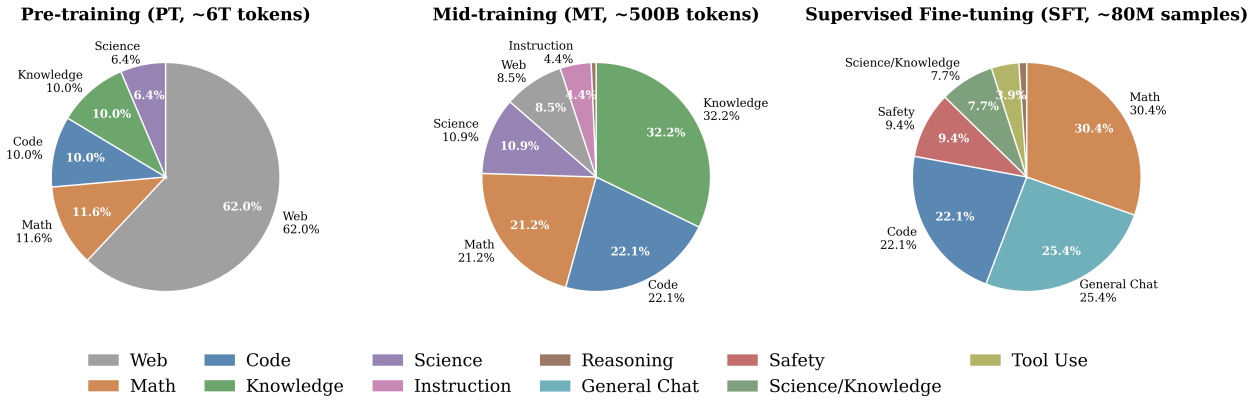


Figure 7 MobileMoE training data mixtures across stages. Domain composition of pre-training (PT, ∼6T tokens), mid-training (MT, ∼500B tokens), and supervised fine-tuning (SFT, >80M samples), with consistent colors per domain across stages. Across stages, the mixture transitions from web-heavy coverage in PT toward domain-specific data (knowledge, code, math) in MT and SFT. Full per-domain datasets are listed in Tables B.1, B.2, and B.3.

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.

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 (Figure 6): pre-training → mid-training → instruct SFT → 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 MoE with $E = 8, g = 8, s = \checkmark$ on the base architecture of Figure 2, using tied input-output embeddings, RoPE θ=500,000, and Llama-3 tokenizer with 128K vocabulary [18]. All models are trained on ∼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 across heterogeneous data and tasks [52] (Figure 7, left; full sources in Table B.1). Although our 6T token budget is smaller than those of recent small LLMs (e.g., 9T for Llama 3.2 1B [18], 11T for SmolLM2 [40]), MobileMoE remains both token- and compute-efficient: all three model scales continue to improve through 6T training (Figure 8), yet use far fewer training FLOPs to match or surpass the accuracy of those dense LLMs (detailed in Section 4.2). To ensure MoE training stability and efficiency, we employ the following techniques.

MoE training stability. To ensure stable routing and balanced expert utilization throughout training, we use auxiliary-loss-free balancing [59] (with bias update rate $\lambda_{lb} = 10^{-3}$), which adjusts expert biases based on token load imbalance without backpropagating a loss, combined with router z-loss regularization [69] ($\lambda_z = 10^{-4}$) 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, producing smoother routing score distributions. All router computations are performed in FP32 precision for numerical stability.

MoE training efficiency. With fine-grained experts, each expert's FFN is much smaller than a standard MoE FFN (e.g., 60 routed FFNs of 768×384 in MobileMoE-S vs. 8 FFNs of 4096×14336 in Mixtral 8x7B [27], $\sim 200\times$ smaller), making naïve per-expert computation inefficient. We address this with grouped MLP, which batches all experts into a single fused grouped matrix multiplication (GMM) kernel, replacing small sequential GEMMs with one efficient batched operation. To support this efficient batched operation during pre-training, we adopt drop-and-pad token dispatching (capacity factor 1.5), assigning each expert a fixed-size token buffer to ensure uniform buffer sizes for the batched kernel. We also apply expert parallelism (EP = 4), allowing each GPU to hold and compute $60/4 = 15$ routed experts per GPU in MobileMoE for memory efficiency.

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, domain-specific data (math, code, knowledge, science) (Figure 7, middle; full sources in Table B.2). This domain-concentrated mixture further sharpens MoE expert specialization, allowing routed experts to develop deeper expertise on these domains. With 8K context, MobileMoE's compact KV cache configuration ($n_{kv} = 4$, $d_h = 64$) keeps its KV cache well within the on-device memory budget (Eq. (5)). We train on ∼500B tokens (∼8% of pre-training budget) with linear learning rate annealing [18, 22], 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 ∼80M samples spanning diverse domains (e.g., math, code, instruction-following, science, QA) at 8K context length with sequence packing. The data mix is composed of public datasets (Figure 7, right; full sources listed in Table B.3), where each dataset is sampled in proportion to its size. To ensure expert load balance, we continue to apply the same MoE training stability techniques 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 (e.g., ≤3–5 GB), we apply INT4 QAT [25] to MobileMoE-SFT, following a similar recipe to recent on-device LLMs (e.g., MobileLLM-Pro [24], which shows that direct post-training quantization (PTQ) can degrade quality). Specifically for MoE, we keep the router in FP32 precision throughout QAT to preserve routing stability under quantized gradients, which adds negligible memory cost ($\sim 0.5\%$ overhead). Concretely, all linear weights (attention, MoE FFNs, and embeddings) are quantized with symmetric group-wise INT4 (group size 32), and activations are dynamically quantized to INT8, with router weights kept in FP32. Formally, the linear weights are quantized via:

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

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 quantization range with $b = 4$. Initialized from the SFT checkpoint, we apply QAT on the SFT data with standard cross-entropy loss; this stage produces **MobileMoE-QAT** with model weight footprints of $\mathcal{M}_{\text{weight}} = 0.68/1.48/2.75$ GB on S/M/L (Eq. (5)), all fitting within recent smartphone DRAM budgets for on-device deployment (detailed in Section 4.3).

4 Experiments

4.1 Experimental setup

Training setup. All four training stages of MobileMoE share the same optimizer setup. We use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-15}$) with weight decay 0.1 and gradient clipping at 1.0. Pre-training, mid-training, and SFT use BF16 model weights, with optimizer states, gradients, and MoE router weights 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× from pre-training through SFT, with QAT inheriting the SFT LR. Training is performed on 8-GPU NVIDIA H100 (96 GB) nodes. Full per-stage hyperparameters are summarized in Table 1.

Table 1 MobileMoE training hyperparameters across all stages: pre-training (PT), mid-training (MT), instruct supervised fine-tuning (SFT), and quantization-aware training (QAT).

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	500
Token dispatch	drop-and-pad	drop-and-pad	dropless	dropless
Hardware (H100 nodes)	16–32	16–32	4–8	4–8
Wall-clock time	3–4 weeks	∼2 days	∼2–3 days	∼2–3 days
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-15}$)			
Weight decay	0.1			
Gradient clipping	1.0			
Precision	BF16 (model weights); FP32 (router, optimizer states, gradients)			
Sequence packing	Yes			
QAT-specific				
Weight quantization	—	—	—	INT4 (group size 32)
Activation quantization	—	—	—	INT8
Embedding quantization	—	—	—	INT4

† First value applies to MobileMoE-S and -M; second value to MobileMoE-L.

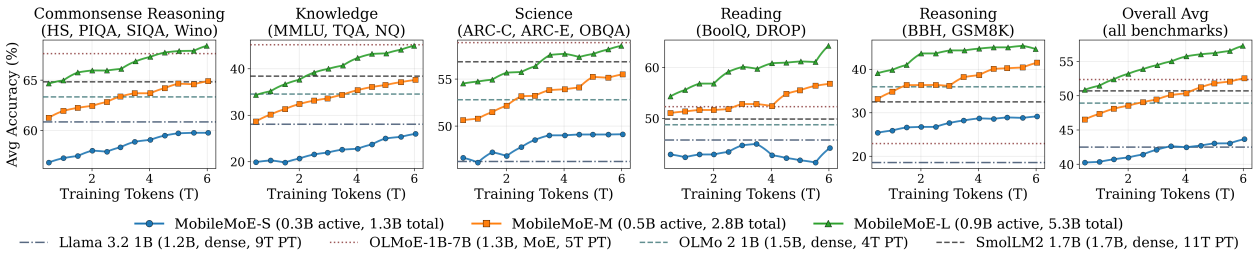


Figure 8 MobileMoE pre-training trajectories. Average benchmark accuracy of MobileMoE-S/M/L across five core competencies plus overall average, plotted against training tokens D up to 6T. Horizontal lines: publicly released pre-trained models with reported pre-training token budgets (i.e., Llama 3.2 1B: 9T, with distillation from Llama 3.1 8B; OLMoE-1B-7B: 5T; OLMo 2-1B: 4T; SmolLM2-1.7B: 11T). Evaluation setups are given in Table C.1.

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 [65], PIQA [5], SIQA [51], WinoGrande [50]), knowledge (MMLU [19], NaturalQuestions [31], TriviaQA [28]), science (ARC-C/E [10], OpenBookQA [42]), reading (BoolQ [9], DROP [13]), and reasoning (BBH [54], GSM8K [11]). The advanced tier comprises 8 benchmarks probing frontier capabilities that small LLMs typically struggle with, including math (MATH-500 [20], GSM-Plus [34]), code (HumanEval pass@1 [7], MBPP [1]), instruction following (IFEval [67], IFBench [48]), and knowledge & reasoning (MMLU-Pro [60], GPQA [49]). Detailed evaluation configurations on all benchmarks are given in Appendix C.

Baseline models. We compare MobileMoE against state-of-the-art on-device LLMs with comparable parameter scales: Gemma 3 (270M, 1B) [57], SmolLM2 (360M, 1.7B) [40], MobileLLM-Pro (1.1B) [24], Llama 3.2 (1B) [18], OLMo 2 (1B) [44], Qwen3.5 (0.8B, 2B) [58], and the state-of-the-art MoE model OLMoE-1B-7B (1.3B active, 6.9B total) [43]. 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.

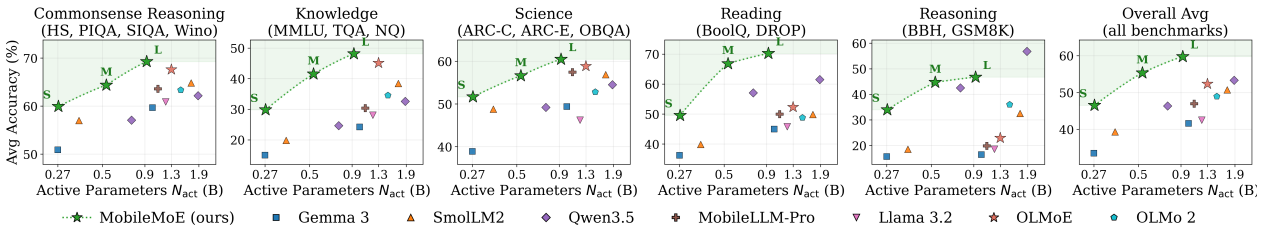


Figure 9 MobileMoE-Base model comparison. Average benchmark accuracy of MobileMoE-S/M/L after mid-training across five core competencies plus overall average, vs. publicly released base models (Gemma 3, SmolLM2, Qwen3.5, MobileLLM-Pro, Llama 3.2, OLMoE, OLMo 2), plotted against active parameters N_{act}. Green stars: MobileMoE-S/M/L; dashed line: MobileMoE Pareto curve. Full table results are given in Table 2.

4.2 Experimental results

4.2.1 Results of Scaling MobileMoE

Data scaling results of MobileMoE. We empirically validate the MobileMoE models derived from on-device scaling laws (Section 3.3) at full-scale pre-training. Figure 8 plots benchmark performance when scaling training data D up to 6T tokens, and compares against existing baselines with reported training token budgets. We have the following observations. **(1)** *MobileMoE benefits from continued data scaling, especially on knowledge.* All three MobileMoE scales improve monotonically through 6T on overall average (MobileMoE-S: 40→44; MobileMoE-M: 46→53; MobileMoE-L: 51→57), with the steepest slope on knowledge (MobileMoE-L: 34→45 on MMLU+TQA+NQ), reflecting MoE's larger total capacity to absorb more knowledge per token than dense models with similar active parameters. **(2)** *MobileMoE-L exhibits superior token efficiency over both dense and MoE baselines.* It surpasses Llama 3.2 1B (9T, distilled, Avg 42) already at ∼0.5T tokens, SmolLM2-1.7B (11T, Avg 51) at ∼1T tokens, and OLMoE-1B-7B (5T, MoE, Avg 52) at ∼2T tokens on overall average. **(3)** *MobileMoE achieves superior parameter efficiency over existing MoE.* By 6T pre-training, MobileMoE-L (Avg 57, 922M active) outperforms OLMoE-1B-7B base model (Avg 52, 1.3B active) with 30% fewer active parameters.

Model scaling results of MobileMoE. We validate the MobileMoE family at full model scales (S/M/L) after pre-training and mid-training, comparing **MobileMoE-Base** against publicly released base models. Figure 9 plots benchmark performance against active parameters N_{act} for MobileMoE-Base vs. existing base models, with detailed per-benchmark numbers in Table 2. We have the following observations. **(1)** *MobileMoE-Base establishes a new Pareto frontier in the sub-billion active-parameter regime.* Overall, MobileMoE-L (922M active, Avg 60) surpasses OLMoE-1B-7B base model (1.3B active, Avg 52) by +8 with 30% fewer active parameters, and outperforms the dense baselines at matched or smaller N_{act}. **(2)** *MobileMoE improves monotonically across S/M/L scales.* 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 (≤500B tokens) to the full pre- and mid-training, confirming the predictive power of our on-device MoE scaling law. **(3)** *MobileMoE outperforms dense baselines across 4 out of 5 capability domains.* At matched N_{act}, the largest gains are on knowledge and reading – reflecting MoE's expanded capacity for knowledge-intensive tasks; the exception is reasoning, where Qwen3.5-2B remains stronger at larger scale with ∼2× more active parameters than MobileMoE-L.

4.2.2 Results of MobileMoE-Base and MobileMoE-SFT

Results of MobileMoE-Base. Table 2 compares MobileMoE-S/M/L against base-model baselines across the 14 foundational benchmarks. We have the following observations.

(1) *MobileMoE-Base achieves state-of-the-art performance among sub-2B active-parameter base models.* 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

Table 2 Base model comparison on foundational benchmarks. N_{act}/N_{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 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 (%). All models are evaluated using lm-eval with the per-task setup in Table C.1; full evaluation details in Appendix C.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	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
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
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
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**

Table 3 Instruct model comparison on foundational benchmarks. Same conventions are adopted following Table 2.

		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
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
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
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
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
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**

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 3) by +4.2, confirming that MobileMoE already excels at pre-training and mid-training.

(2) *MobileMoE-Base 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 $\sim$ 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 with larger total parameter capacity is best exploited on tasks demanding stored factual knowledge and comprehension.

Results of MobileMoE-SFT. Table 3 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 as follows.

(1) *MobileMoE-SFT achieves state-of-the-art performance among sub-2B active-parameter instruct models, preserving the performance advantages of MobileMoE-Base.* MobileMoE-S (272M active, Avg 46.7) surpasses

Table 4 Instruct model comparison on advanced benchmarks. Avg is within-capability mean; **Overall** is mean across all. All models are evaluated in non-thinking mode with 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_{act}/N_{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
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
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
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
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
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**

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). These margins match or exceed the gaps prior to SFT (Table 2), confirming our SFT recipe preserves MobileMoE's architectural advantages.

(2) *MobileMoE-SFT matches dense instruct models using 2–4× fewer active parameters, with the same parameter-efficiency Pareto frontier as MobileMoE-Base.* MobileMoE-S (272M, Avg 46.7) surpasses Qwen3.5 0.8B (42.7) by +4.0 using 2.8× fewer active parameters, and matches Llama 3.2 1B (46.9) with 4.4× fewer active parameters. MobileMoE-M (528M, Avg 55.3) outperforms Qwen3.5 2B (50.8) by +4.5 using 3.6× fewer active parameters. MobileMoE-L (922M, Avg 60.1) exceeds Qwen3.5 2B by +9.3 using ∼2× 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 4 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-SFT shows consistent wins on code and math.* 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-SFT ranks second after Qwen3.5 2B on instruction following and knowledge & reasoning.* MobileMoE-L outperforms all other baselines on these capabilities (e.g., +11.3 over OLMoE-1B-7B on instruction following and +12.0 on knowledge & reasoning), and trails only Qwen3.5 2B. The Qwen3.5 2B advantage likely reflects its more advanced post-training recipe (e.g., distillation, thinking-enabled reasoning). Overall, the MoE benefits of larger total capacity and routed expert specialization yield clear strengths on code and math, while the gap with Qwen3.5 2B on instruction following and knowledge & reasoning motivates enriching our training recipe in future work, e.g., adding distillation and thinking-enabled post-training as in Qwen3/3.5 [58, 63].

Comparison of capabilities across training stages. Figure 10 traces 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 three baselines at comparable scales (SmolLM2 360M for MoE-S, Qwen3.5 0.8B for MoE-M, OLMoE-1B-7B for MoE-L) as references on all 14 foundational benchmarks (Table 5). We highlight four observations.

(1) *Pre-training dominates commonsense reasoning.* HellaSwag, PIQA, SIQA, and WinoGrande saturate

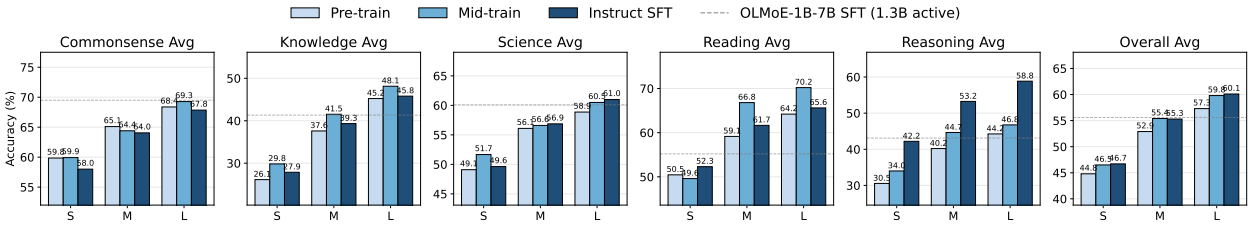


Figure 10 MobileMoE capability progression across training stages. Benchmark accuracy (%) for MobileMoE-S/M/L after each of the three main training stages: pre-training (PT, light), mid-training (MT, medium), and instruction supervised fine-tuning (SFT, dark). Gray dashed line: instruct-tuned OLMoE-1B-7B (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 5.

Table 5 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 for baselines of similar scales (SmolLM2 360M, Qwen3.5 0.8B, OLMoE-1B-7B). Benchmarks and few-shot setup follow Table 2.

			Commonsense Reasoning				Knowledge			Science			Reading		Reasoning		
Model	**Stage**	$N_{\text{act}}/N_{\text{total}}$	HS	PIQA	SIQA	Wino	MMLU[5]	NQ[5]	TQA[5]	ARC-C[25]	ARC-E	OBQA	BoolQ	DROP[3]	BBH[3]	GSM8K[8]	**Avg**
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
MobileMoE-S	PT	**272M/1.3B**	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
	MT		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
MobileMoE-M	PT	**528M/2.8B**	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
	MT		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
MobileMoE-L	PT	**922M/5.3B**	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
	MT		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

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: 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 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. Overall foundational accuracy improves +2 to +3 points from PT to MT across all three MobileMoE scales, while SFT preserves foundational accuracy and unlocks the math/reasoning gains in Table 4.

(4) *MobileMoE-L's advantage over OLMoE-1B-7B is established at PT and compounds through SFT.* As shown in Table 2, MobileMoE-L PT already exceeds OLMoE-1B-7B Instruct on the average benchmark; after SFT, MobileMoE-L further surpasses OLMoE-1B-7B on the majority of benchmarks (dashed reference line in Figure 10), confirming that MobileMoE's advantage establishes during pre-training and retains through subsequent stages.

Table 6 Quantized model comparison with quantization-aware training (INT4 QAT). All models are quantized to 4-bit weights via QAT. **Mem**: INT4 model weight memory in GB. Baseline QAT checkpoint: MobileLLM-Pro (facebook/MobileLLM-Pro-base-int4-accelerator). Benchmark abbreviations follow Table 2.

		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-LB[3]	GSM8K[8]	**Avg**
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**

4.3 On-device deployment of MobileMoE

We deploy MobileMoE to flagship smartphones (*Samsung Galaxy S25* with Snapdragon 8 Elite, *iPhone 16 Pro* with Apple A18 Pro) via INT4 QAT (Section 3.4) and custom MoE inference implemented on ExecuTorch. We benchmark against MobileLLM-Pro [24] – a state-of-the-art on-device LLM purpose-built for mobile deployment, providing a strong baseline at comparable parameter count. In the following section, we evaluate MobileMoE on two key fronts for on-device deployment: (1) benchmark performance with INT4 weight precision, and (2) on-device runtime profiling across different mobile devices and processors.

4.3.1 Results of MobileMoE-QAT

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 for direct on-device deployment. Table 6 compares the per-benchmark accuracy of MobileMoE-S/M/L against the publicly released QAT baseline MobileLLM-Pro. 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 3), MobileMoE's QAT incurs 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 compared to the 4× weight-memory reduction from BF16 to INT4, indicating that MoE routing and expert computation remain numerically stable under 4-bit quantization and that our QAT recipe (symmetric group-wise INT4 weights + FP32 router, Section 3.4) 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 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 comfortably within modern mobile DRAM budgets (≤ 5 GB). At comparable INT4 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 knowledge results (MMLU +9.4). The INT4 QAT MobileMoE-L ($\mathcal{M}_{\text{weight}} = 2.75$ GB, Avg 57.8) already exceeds the BF16 SFT OLMoE-1B-7B (Avg 55.6, ∼13.8 GB BF16) at ∼5× smaller memory footprint.

4.3.2 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.

Table 7 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 6). 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

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 D). 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 7 reports the full runtime sweep over $\{256, 512, 1k, 2k, 4k, 8k\}$ 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.

(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 $\geq$ 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 8, 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

Table 8 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 6). 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×

Table 9 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 6). 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

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 9 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, 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× 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 ∼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× at 8k, and MobileMoE-L's from 4.1× at 256 to 2.5× 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.

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.

Author Contributions

Yanbei Chen designed the project scope, developed the MobileMoE model architecture (Sections 3.1, 3.2, 3.3), the end-to-end training recipe (Section 3.4) and data (Appendix B), ran all experiments (Sections 4.1, 4.2), and wrote the paper. Hanxian Huang set up the evaluation pipelines (Section 4.1) and contributed to post-training data (Appendix B). Ernie Chang curated the pre-training and mid-training data (Appendix B). Jacob Szwejbka deployed MobileMoE to iPhone 16 Pro and benchmarked with CPU and GPU backends (Section 4.3). Digant Desai implemented the custom MoE kernel for mobile inference, and deployed MobileMoE to Samsung Galaxy S25 with runtime benchmarks (Section 4.3). Zechun Liu and Vikas Chandra provided feedback through regular meetings and project-planning discussions. Raghuraman Krishnamoorthi advised on research direction, provided feedback through regular discussions, and supported the project with computing and other resources. All authors contributed to discussions that shaped the scope and direction of this work.

Acknowledgements

We thank Wei Wen, Igor Fedorov, Yanghan Wang, Tarek Elgamal, Qi Qian, Kimish Patel, Steven Li, Bilgin Cagatay, Mergen Nachin, Min Guo, Shicong Zhao, Patrick Huber, Rylan Conway, Hakan Boyraz, Amrit Panda, Eugene Gorbatov, and Harshit Khaitan for valuable discussions, help, and feedback on this project.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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*, volume 34, pages 7432–7439, 2020.

[6] 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.

[7] 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.

[8] 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.

[9] 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)*, pages 2924–2936, 2019.

[10] 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.

[11] 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.

[12] 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.

[13] 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)*, pages 2368–2378, 2019.

[14] 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.

[15] 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.

[16] 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.

[17] 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, 07 2024. https://zenodo.org/records/12608602.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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. *arXiv preprint arXiv:2603.15954*, 2026.

[24] 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.

[25] 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*, pages 2704–2713, 2018.

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

[27] 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.

[28] 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)*, pages 1601–1611, 2017.

[29] 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.

[30] 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.

[31] 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*, 7:453–466, 2019.

[32] 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.

[33] 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*, 37:14200–14282, 2024.

[34] 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)*, pages 2961–2984, 2024.

[35] 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.

[36] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[37] Yan Liu, Renren Jin, Ling Shi, Zheng Yao, and Deyi Xiong. Finemath: A fine-grained mathematical evaluation benchmark for chinese large language models. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 24(12):1–15, 2025.

[38] 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.

[39] 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*, pages 22631–22648. PMLR, 2023.

[40] Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, et al. Smollm2: When smol goes big—data-centric training of a fully open small language model. In *Second Conference on Language Modeling*.

[41] 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.

[42] 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*, pages 2381–2391, 2018.

[43] 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.

[44] Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[45] 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.

[46] Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, and Thomas Wolf. Fineweb: decanting the web for the finest text data at scale. *HuggingFace. Accessed: Jul*, 12, 2024.

[47] 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.

[48] 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.

[49] 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.

[50] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[51] 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), pages 4463–4473, 2019.

[52] 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.

[53] 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.

[54] 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*, pages 13003–13051, 2023.

[55] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

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

[57] Gemma Team. Gemma 3. 2025. https://arxiv.org/abs/2503.19786.

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

[59] 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.

[60] 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*, 37:95266–95290, 2024.

[61] xAI. Open release of grok-1. https://x.ai/news/grok-os, 2024.

[62] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[63] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[64] 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.

[65] 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.

[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 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.

[68] 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.

[69] 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.

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 [21]) 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, with the transformed expert count $\hat{E}$ following [8]: $\frac{1}{\hat{E}} = \frac{1}{E-1+\left(\frac{1}{E_{\text{start}}}-\frac{1}{E_{\text{max}}}\right)^{-1}} + \frac{1}{E_{\text{max}}}$. 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.

- **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.

B 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 7 visualizes the data mixture composition across three training stages; per-domain dataset lists are detailed in Tables B.1, B.2, B.3.

Pre-training data. Our pre-training data of MobileMoE comes from two sources: (1) the Dolma3 mix from OLMo-3 [44], which provides quality-stratified Common Crawl, OCR-extracted scientific PDFs [47], multi-language code (Stack-Edu), and curated academic math (FineMath [37]); and (2) a curated data collection from MobileLLM models [24, 66], including quality-filtered web corpora (DCLM [33], FineWeb-Edu [46]), code (StarCoder [35], Stack-Edu, Nemotron Code [3]), math (FineMath [37], OpenWebMath [45], Algebraic Stack [2], Nemotron Math [3]), science (arXiv, peS2o, Nemotron Science [3], Natural Reasoning [64]), and knowledge (FLAN [39], StackExchange, Wiki, 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 [52]. Table B.1 summarizes the domain breakdown.

Table B.1 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%

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 [44], 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.2 summarizes the mid-training domain breakdown.

Table B.2 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%

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.3 lists the SFT data sources by domain.

Table B.3 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%

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) [17] 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 [17] for MATH500, GSM-Plus, HumanEval MBPP, IFEval, and GPQA Diamond, and the official packages TIGER-AI-Lab/MMLU-Pro [60] and allenai/IFBench [48] 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	$\text{HumanEval}_{p@1}$	`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.

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.

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
Qwen3.5 0.8B	749M	Qwen/Qwen3.5-0.8B-Base	Qwen/Qwen3.5-0.8B
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

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 4 reports MMLU-Pro with TIGER-AI-Lab/MMLU-Pro and GPQA Diamond with lm-eval (LLH). LLH: loglikelihood.

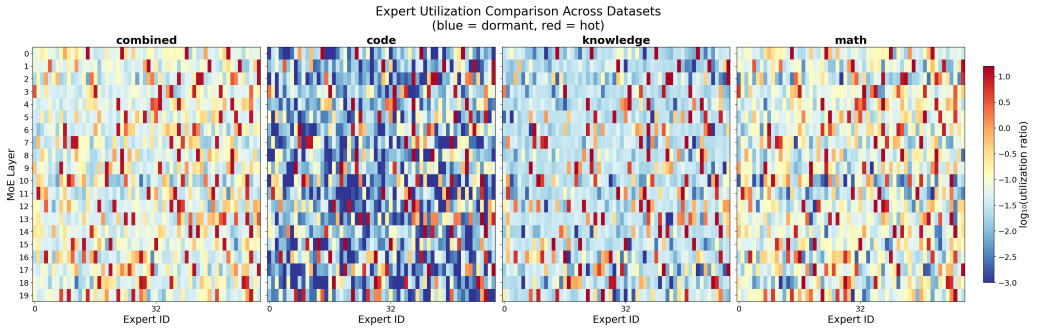
		MMLU-Pro (5-shot)			**GPQA Diamond** (0-shot)	
Model	N_{act}/N_{total}	TIGER-AI-Lab	lm-eval (LLH)	lm-eval (Chat)	lm-eval (LLH)	lm-eval (Chat)
Gemma 3 270M	270M	11.3	11.3	0.0	25.8	19.2
SmolLM2 360M	362M	12.0	10.9	9.9	25.8	**26.8**
MobileMoE-S	**272M/1.3B**	**18.2**	**18.1**	**14.0**	**27.8**	21.2
Qwen3.5 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

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 only Qwen3.5 2B (+9.6) while degrading 10 of 13 models. To avoid the bias and collapse brought by model-specific chat templates, Table 4 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.

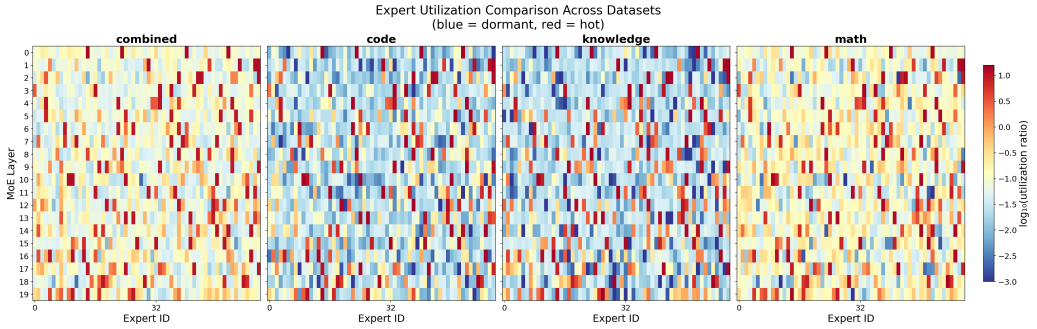
D Quantitative Analysis

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 D.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.

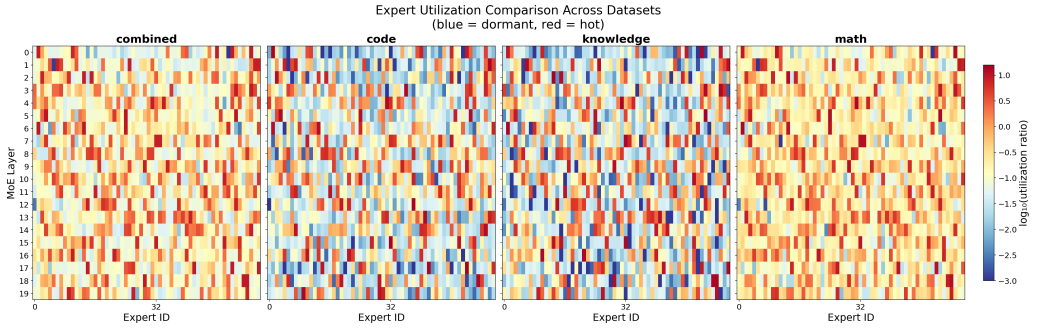
Expert utilization statistics. The distribution of expert utilization ratios varies across downstream tasks (Figure D.2): math activates a broader set of experts, while code or knowledge tasks concentrate on a narrower subset. This task-dependent sparsity indicates that not every expert weight needs 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 D.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.

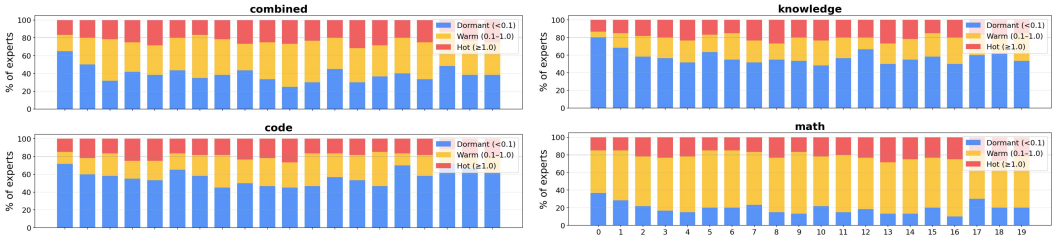


Figure D.2 Distribution of expert utilization across tasks: math, code, knowledge, combined (all tasks). Statistics of expert utilization ratios ($\log_{10}$ scale) are shown across all MoE layers for MobileMoE-S (post SFT).