μLM: Rethinking Sub-100M Language Models through Memory-First Design

Zijie Chen
Shanghai Jiao Tong University

Guiyun Fan
Tongji University

Haiming Jin*
Shanghai Jiao Tong University

Abstract

Ubiquitous IoT devices, such as companion toys, wearables, and micro-robots, are increasingly embracing language models, yet current cloud-based deployments suffer from latency, privacy, and weak-network reliability issues. These resource-constrained devices typically cap deployable model size below 100M parameters, while existing sub-100M models fall short in basic language and commonsense capabilities. In this extreme regime, we argue that the memory bottleneck supersedes the traditional compute bottleneck as the binding constraint, calling for memory-first design that maximizes model capability under a fixed parameter budget. Notably, the stringent budget binds both what the model stores and what it can afford to learn, demanding architecture-data co-design. In this paper, we propose μLM, a memory-first sub-100M language model framework instantiating this co-design. For architecture, we present DMLA, which preserves latent attention's compact cache while relieving its feature-squeezing bottleneck under aggressive compression via explicit content-position latent decoupling. For data, we present μMix, a capacity-aware multi-stage curriculum tailored to sub-100M models, prioritizing high-quality natural language and commonsense data aligned with embedded scenarios over brute-force token scaling. μLM-70M surpasses all sub-300M baselines and rivals several sub-1B models on standard benchmarks; even at 9M, it remains competitive. Fine-tuned into a tool-calling agent, μLM outperforms same-scale baselines and achieves a 97.2% end-to-end executable rate, presenting a promising direction toward practical sub-100M language modeling.

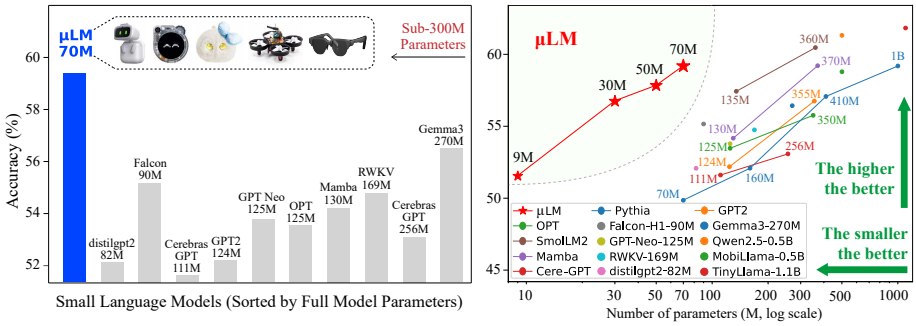

Figure 1: **μLM Performance Comparison. Left:** Average accuracy on standard zero-shot benchmarks across sub-300M SLMs. The icons illustrate the constrained devices that motivate our memory-first design. **Right:** Accuracy-parameter trade-off. The μLM family (red stars) establishes a new performance frontier in the sub-100M regime, evidencing that capability density at extreme scale benefits from memory-first architecture-data co-design. Full results are provided in Table 1.

*Corresponding author.

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

1 Introduction

The capability density of small language models (SLMs) has been improving rapidly, driven by advances including model architecture, data recipes, and training strategies [1]. Recent Gemma- and MiniCPM-family models [2, 3] demonstrate remarkable parameter efficiency, accelerating the deployment of generative AI on edge devices. However, most SLMs are designed primarily for phone- or laptop-class platforms, while a broader and more pervasive class of ultra-constrained devices remains underserved, including companion toys, voice-recording cards, wearables, and micro-robots. For these devices, cloud-only inference is not a satisfactory default: inputs are private and continuously generated, interactions are latency-sensitive, and connectivity is often intermittent or costly to sustain [4, 5]. It remains reasonable for open-ended or heavy queries, but basic language understanding and intent routing for these devices should remain local and always available.

Unlike phone-class devices, ultra-constrained platforms (e.g., RP3A0 [6], STM32N6 [7], CV1800B [8]) operate under system memory ranging from only tens to a few hundred megabytes. Crucially, the operating system, sensor buffers, I/O stacks, and application logic share the same memory budget alongside model weights and runtime states, pushing deployable backbones toward the sub-100M regime [9, 10]. Yet language models at this scale currently fall short of practical interaction needs [11–14]. This extreme scenario reveals a design paradigm that differs fundamentally from mainstream on-device LLMs: *the memory bottleneck supersedes the compute bottleneck as the primary limiting factor.* A model that exceeds the available memory cannot be accelerated [15], while a model that fits only through naive shrinking is often too weak to be useful. The central challenge is thus to design a memory-first language model that maximizes capability density under strict limitations.

This constraint exposes two complementary design axes. On the architecture side, reducing KV state alone is not enough: aggressive KV-head sharing such as MQA or small-group GQA reduces per-head KV diversity, while standard latent attention improves the trade-off yet may entangle semantic matching and positional information under aggressive sub-100M compression [16–18]. On the data side, once the memory budget fixes the model below 100M parameters, pretraining becomes a capacity-allocation problem. Tiny models cannot absorb all domains simultaneously, and brute-force token scaling or high-format-cost corpora consume scarce capacity without improving the target interaction abilities. Therefore, sub-100M language modeling requires not only a memory-efficient architecture, but also data schedules that prioritize what the model can afford to learn.

In this paper, we propose μLM, a sub-100M language model framework built on architecture-data co-design. On the architecture side, we adopt latent attention for its superior memory–expressivity trade-off under a fixed parameter budget [18], and further propose DMLA (Decoupled Multi-head Latent Attention), an attention mechanism reorganized around memory-first deployment. At its core, DMLA explicitly factorizes the latent cache into content and position sub-spaces, alleviating feature squeezing under aggressive compression through input-aware dual-stream fusion. DMLA further introduces ALiBiLe, a learnable positional prior naturally compatible with latent caching. On the data side, we propose μMix, a data scheduling recipe tailored to sub-100M capacity constraints. Instead of scaling token volume alone [19, 20], μMix couples a capacity-aware data mixture with a multi-stage curriculum that progressively establishes foundational syntax and fluency, consolidates the language–logic backbone, and finally injects commonsense knowledge.

Experimental results show that μLM-70M surpasses all evaluated sub-300M baselines on standard zero-shot benchmarks [21], and rivals several sub-1B models (Figure 1). The μLM family establishes a new performance frontier in the sub-100M regime. To further demonstrate practical value, we fine-tune μLM into μAgent, a variant designed for tool-calling [22] that achieves a 97.2% end-to-end executable rate on out-of-distribution scenarios, outperforming its same-scale baseline.

To summarize, our work makes the following key contributions:

- *Memory-first design framework.* We articulate a memory-first design framework for sub-100M language models, identify deployment-memory footprint rather than raw compute as the binding bottleneck, and derive two design principles governing KV allocation and positional injection.
- *Architecture-data co-design.* We propose μLM, jointly addressing the architectural and data axes of sub-100M design: **DMLA** instantiates the memory-first principles by decoupling the latent cache into content and position routes with input-aware fusion to alleviate feature squeezing, while μ**Mix** prioritizes learning efficiency over token volume through a capacity-aware multi-stage curriculum.

- *Comprehensive validation.* We evaluate μLM across standard zero-shot benchmarks and lightweight agentic tasks: μLM-70M surpasses all sub-300M baselines and rivals several sub-1B models, while the fine-tuned μAgent outperforms its same-scale baseline and achieves a 97.2% executable rate.

2 Related Work

On-device generative AI. A broad class of battery-powered edge devices, including companion toys, wearables, and micro-robots, increasingly relies on generative intelligence, yet current cloud-based deployments suffer from latency, privacy, and offline-availability limitations, creating a pressing need for on-device inference [5, 23, 24]. Unlike smartphones, these devices operate under milliwatt-scale power budgets and megabyte-scale memory [9, 25], which is further shared by the operating system, sensor buffers, and application logic. Thus, the practical memory available to models is ultra-constrained, often pushing deployable backbones below 100M parameters [4, 26]. However, mainstream SLMs such as MobileLLM [27], MobiLlama [28], and SmolLM2 [20] mainly target the 100M–1B range [29], while existing sub-100M models fall short of practical interaction quality [11, 13, 14]. To our knowledge, μLM is the first language model framework explicitly designed for this regime, surpassing several hundred-million-parameter models through a memory-first design.

Memory-efficient architectures and data-centric pretraining. For architecture, efficient designs reduce per-head key/value diversity (MQA [16], GQA [17]) or compress the cache into a shared latent space (MLA [18, 30]), while positional encodings such as RoPE [31] and ALiBi [32] interact non-trivially with these mechanisms, a cache-compatibility issue we revisit in Section 3. For data, scaling laws and data-constrained studies [19, 33], together with curated corpora such as FineWeb, DataComp-LM, and Cosmopedia [34–36], highlight the importance of token budgets, mixture design, and synthetic data [37]. However, neither line has systematically examined the appropriate attention and data choices under sub-100M constraints, where every parameter must be justified against an extreme capacity ceiling. μLM, to our knowledge, is the first to co-design architecture and data under an explicit memory-first objective for this regime: DMLA alleviates feature squeezing via content-position decoupling, while μMix allocates limited capacity using a capacity-aware curriculum.

3 Memory-First Design Principles

Two Design Questions. On ultra-constrained devices, deployment feasibility is determined by total memory footprint before throughput optimization becomes meaningful. Design should therefore preferentially compress what persists during inference, model weights and cached states, rather than minimize transient compute. Two questions follow that we address in turn: (i) how should scarce KV parameters be allocated, and (ii) how should positional information be injected under that allocation?

(i) Latent attention over grouped sharing. Under a fixed total parameter budget, Grouped-Query Attention (GQA) [17] and Multi-head Latent Attention (MLA) [18] allocate KV parameters in qualitatively different ways: GQA spends its budget on g shared key/value projections, each used by a group of heads, while MLA spends it on a shared low-rank backbone followed by per-head up-projections. With hidden size D, h query heads of width $d = D/h$, and matched non-KV components, equating KV parameter counts yields $r = \frac{2gd}{3}$, where r is the MLA latent width. Which allocation is better depends on the structure of the target KV mapping. Prior empirical studies show that learned KV representations exhibit low intrinsic dimensionality and compressible spectral structure [38–40], so the shared backbone efficiently encodes the global KV geometry; what limits expressivity is the number of distinct per-head projections. We formalize this as Theorem 1 in Appendix B.1: when a target mapping factorizes through a rank-r latent but requires $m > g$ distinct per-head projections, equal-budget MLA represents it exactly while GQA cannot. Under tight parameter budgets that force g to be small, particularly the extreme $g = 1$ (MQA) case [16], GQA's head-diversity ceiling becomes the binding constraint, motivating MLA as the starting point for sub-100M models.

(ii) ALiBi-style bias over RoPE. We argue that ALiBi-style biases [32] are architecturally preferable to RoPE [31] once latent attention is adopted, for two reasons. First, with limited capacity, an explicit locality prior is more efficient than asking the model to discover one. RoPE encodes relative position through query-key rotations, a flexible mechanism that does not impose distance

decay, leaving the model to learn it from data alongside semantics, syntax, and task behavior. ALiBi instead injects an additive logit bias that decays linearly with distance, placing the recency prior directly in the attention score, and freeing capacity for content modeling. Second, RoPE imposes a compatibility-driven positional split, not a modeling-driven one. The query-key rotation cannot be factored out of the cached state without re-computing it on every query, forcing RoPE-equipped MLA to reserve a fraction of each head's width purely as inert rotational carriers (Appendix B.3). ALiBi entering as a post-dot-product bias imposes no such reservation: any subsequent partition of head width then becomes a deliberate modeling choice rather than a structural tax. We exploit precisely this freedom in Section 4.2, where DMLA's per-head split into content and position streams is allocated to two independently-routed attention pathways, not to passive position-carrying.

4 μLM Architecture

4.1 Overview

Section 3 identifies two architectural ingredients best suited to sub-100M memory-first deployment: *latent KV compression* to minimize the persistent decoding state, and an *ALiBi-style positional prior* to avoid the mandatory positional-carrier subspace imposed by RoPE. The remaining design question is how to organize the compressed state so that a tiny model can better exploit its limited capacity.

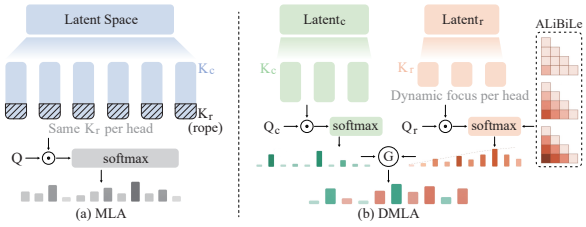

Figure 2: Comparison with MLA and proposed DMLA.

We propose **DMLA** (**D**ecoupled **M**ulti-head **L**atent **A**ttention), which restructures the latent cache into two specialized routes, a content route and a position-biased route, compressed, routed, and normalized independently, fusing only after each forms its own attention distribution. Unlike RoPE-based MLA, where content and positional channels are separated primarily for compatibility with compressed caching, DMLA elevates this separation from a compatibility constraint to a deliberate modeling choice. Figure 2 contrasts the two mechanisms.

4.2 Active Latent Factorization

Let $X \in \mathbb{R}^{T\times D}$ denote hidden states, h the head count, and m_c, m_r the content and position latent widths. DMLA first builds two independent latent caches:

$$C^{\ell} = \mathrm{Norm}(XW_{\ell}^{D}), \quad \ell \in \{c, r\}. \tag{1}$$

The persistent memory footprint per layer per token is bounded to $m_c + m_r$ scalars; keys and values are reconstructed on the fly from this state, while queries are computed from the current hidden state.

From compatibility-driven separation to active factorization. In RoPE-compatible MLA, content and positional score components may be algebraically separable, but they are summed and jointly normalized by a single softmax (Fig. 2a). DMLA makes a stronger choice: the two routes form independent attention distributions before fusion. Defining $\mathcal{A}_d(Q, K, V; B) = \mathrm{softmax}(QK^{\top}/\sqrt{d} + B)\,V$, DMLA computes:

$$Y^{c} = \mathcal{A}_{d_c}(Q^{c}, K^{c}, V;\, M), \qquad Y^{r} = \mathcal{A}_{d_r}(Q^{r}, K^{r}, V;\, M + A^{\mathrm{ALiBiLe}}), \tag{2}$$

where M is the causal mask. The content route produces position-bias-free semantic matching; the position-biased route additionally incorporates **ALiBiLe** (**ALiBi** with **Le**arnable slopes), which retains ALiBi's linear distance-bias form but makes per-head slopes ρ_u trainable, with $A^{\mathrm{ALiBiLe}}_{u,t,s} = -\rho_u(t-s)$ for $s \le t$, where u indexes attention heads and t, s are query/key positions (Appendix D.3). The original ALiBi schedule was designed for models with tens of heads, where a suboptimal per-head slope can be compensated by others; in our regime ($h = 8$–16), every head's effective receptive field carries disproportionate weight, and a fixed geometric schedule risks wasting limited routing capacity.

Crucially, the position-biased route is *not* a fixed distance kernel: the query-key interaction $Q^{r}(K^{r})^{\top}$ further modulates the ALiBiLe prior, allowing content-aware selection within the recency-biased

window, consistent with evidence that positional attention in Transformers is inherently content-dependent [41, 42]. The route therefore acts as a *local expert*: its attention is shaped by the positional prior but not dictated by it. Because softmax is non-linear, the two routes' interaction moves from the score level to the distribution level, where each forms its own routing hypothesis before being combined, expanding the representable family of attention patterns beyond what a single jointly-normalized softmax can express.

Why factorization helps tiny models. A single highly compressed latent in a sub-100M model is an extreme information bottleneck: forcing it to simultaneously encode semantic alignment, recency bias, and head-specific routing leads to severe *feature squeezing*. DMLA provides explicit structural relief by allocating two separate bottlenecks, encouraging the model to compress content and positional evidence differently before attention is formed. The split also improves parameter efficiency by eliminating cross-route key up-projections (Appendix B.2).

4.3 Complementary Design Choices

The active factorization above defines DMLA's core. We complete the data path (Fig. 3) with three coordinated choices.

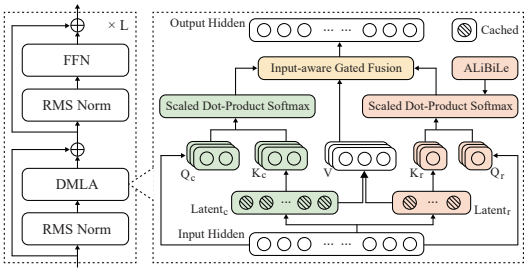

Figure 3: Architecture of μLM.

Asymmetric stream construction. Queries are generated full-rank from the hidden state and split into the two streams $(Q^c, Q^r) = \mathrm{split}_h(XW^Q)$, since they are ephemeral and compressing them would constrain routing without saving deployment memory. Keys are read exclusively from their corresponding latent, $K^\ell = \mathrm{split}_h(C^\ell W_\ell^K)$, preserving route specialization, and remain multi-head rather than shared, since collapsing all heads into a single positional prototype would undermine factorization. Values, however, are computed once from the joint latent $V = \mathrm{split}_h([C^c; C^r]\, W^V)$. This is justified on two grounds. *Modeling:* since V does not participate in the $QK^\top$ score, it carries no content/positional role conflict and benefits from reading the full information in both latents. *Computation:* because the two routes' attention probabilities P^c, P^r share the same shape and are linearly combined by the stream gate (Eq. 3), the bilinearity of attention permits fusing first and multiplying once: $Y_{\text{fused}} = (G \odot P^c + (1 - G) \odot P^r)\, V$, rather than computing P^cV and P^rV separately.

Post-softmax stream gate. To adaptively fuse the two routes, DMLA introduces an input-conditioned gate:

$$G = \sigma\big(g_0 + \alpha\, \tanh(XW_g)\big), \qquad Y_{\text{fused}} = G \odot Y^c + (1 - G) \odot Y^r, \tag{3}$$

where $g_0 \in \mathbb{R}^h$ is a learned per-head static prior, $W_g \in \mathbb{R}^{D\times h}$, and α scales the input-aware modulation. The gate acts *after* the two independent softmaxes, selecting between two already-formed routing hypotheses rather than reweighting score logits. Because it interpolates between normalized outputs, fusion is inherently scale-stable; input-conditioning further allows each token and each head to dynamically shift between semantic matching and recency bias (Appendix D.4).

Overall architecture. Beyond DMLA (Fig. 3, right), μLM follows a standard pre-norm Transformer [43]: RMSNorm [44] before each sub-layer, dense SwishGLU [45] feed-forward (we deliberately avoid MoE, whose routing tables inflate the static weight footprint), and tied input-output embeddings with a 16K-token vocabulary chosen to keep the embedding table under 20% of total parameters at sub-100M scale. The complete block is $\mathrm{DMLA}(X) = \mathrm{merge}_h(Y_{\text{fused}})\, W^O$.

Two properties make DMLA Pareto-improving over MLA at matched latent budget. First, it introduces *no additional cached state* beyond the $m_c + m_r$ latent in Eq. 1, the stream gate and all Q/K/V projections are computed on the fly. Second, it adds *no additional QK dot-product compute*: MLA's single $Q(K)^\top$ with per-head width $d_c{+}d_r$ splits into two narrower products $Q^c(K^c)^\top$ and $Q^r(K^r)^\top$, yielding the same $hT^2(d_c + d_r)$ operations. The stream gate adds only $Dh + h$ parameters, under 0.02% of the 70M-scale total.

5 μMix: Capacity-Aware Data Recipe and Curriculum

For sub-100M models, data mixture directly determines what the model can learn: parameter capacity is too limited to absorb all domains simultaneously, so every corpus-level decision is strictly a capacity-allocation decision. Changing the mixture does not merely affect convergence speed; it dictates what the model can afford to remember and generalize. This section describes μMix, a data scheduling recipe organized around three capacity-aware principles.

5.1 Three Capacity-Aware Principles

P1: Total scale balanced against the capacity wall. The Chinchilla compute-optimal ratio of $\sim$20 tokens/parameter [19] is far too low for sub-100M models, yet feeding hundreds of billions of loosely filtered tokens hits a capacity wall, the point at which held-out loss on a high-quality reference set plateaus ($< 0.5\%$ improvement per additional 1B tokens) and further data mostly introduces conflicting long-tail facts that cannot be reliably stored [33, 46]. μMix uses an approximately 10B-token physical corpus ($\sim$140 tokens/param at 70M), located near the empirically observed plateau in this regime, and trains for 5 epochs to amplify within-budget signal.

P2: Aggressive omission of high-format-cost domains. μMix excludes standalone code and math corpora from base pre-training. Code (indentation, brackets, long-range formal dependencies) and dense mathematical notation impose a disproportionate format cost on sub-100M budgets, consuming capacity for surface regularities that do not transfer to general edge deployment. We allocate capacity exclusively to clean natural language, educational explanations, and everyday commonsense, following the textbook-quality logic of the Phi series [37, 47].

P3: Macro-curriculum over one-shot mixing. Micro-models exhibit an exacerbated recency bias: swapping the mixture in the final 10% of training shifts HellaSwag and PIQA by 1.5-2.8 points at 50M. Rather than treating this as a vulnerability, μMix exploits it as a control mechanism by coupling the data schedule with a WSD-style (Warmup–Stable–Decay) learning-rate schedule [48, 49]: the LR decay phase coincides with a mixture shift toward curated high-quality sources, gently steering the already-formed language–logic backbone rather than forcing it to relearn the world.

5.2 The μMix Recipe

Sources. μMix draws on five corpora (Fig. 4): FineWeb-Edu [34] (educational web text), Cosmopedia [36] (synthetic textbook-style explanations), TinyStories [14] (highly grammatical narratives), WikiHow [50] (procedural commonsense), and Simple English Wikipedia (compact factual grounding without the entity density of full encyclopedias).

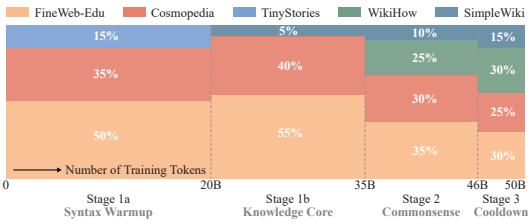

Figure 4: Curriculum learning schedule of μMix.

Stages 1a-1b: Linguistic foundation. Stage 1a (Syntax Warmup) pairs TinyStories with educational text to give the model a clean early signal for grammar and short-range coherence. TinyStories is then phased out in **Stage 1b (Knowledge Core)** to prevent over-specialization to a child-like narrative style, a regression we directly observed when stories were retained throughout, and a small share of Simple English Wikipedia is introduced early to begin acclimating the model to encyclopedic register before commonsense data arrives in bulk.

Stage 2 (Commonsense Injection). The bulk of WikiHow and Simple Wikipedia enters here. We retain Cosmopedia and FineWeb-Edu as a stable trunk to prevent abrupt distribution shifts; the new corpora supply procedural and factual commonsense without displacing the language-logic backbone built in Stage 1.

Stage 3 (Cooldown). Stage 3 uses a small physical budget with higher replay weight (5 epochs, following [33]) and is paired with LR decay. The mixture continues the trajectory established in Stage 2 by further increasing the share of WikiHow and Simple English Wikipedia while modestly reducing FineWeb-Edu and Cosmopedia. The intent is not to introduce new data sources but to consolidate

Table 1: Performance comparison between different SLMs.

Model	#Params	Attention Mechanism	Position Encoding	Language	Contextual Understanding			Commonsense		Score(%)↑ (Averaged)
				BLiMP↑	BoolQ↑	WinoGrande↑	SIQA↑	PIQA↑	HellaSwag↑	
number of parameters < 300M										
Pythia-70M	70M	MHA	RoPE	0.7326	0.5196	0.5185	0.3526	0.5909	0.2751	49.82
distilgpt2-82M	82M	MHA	Learned Abs.	0.7873	0.5884	0.5036	0.3608	0.6099	0.2750	52.08
Falcon-H1-Tiny-90M	90M	GQA+SSM	RoPE	0.8019	0.5869	0.5107	0.3823	0.6518	0.3762	55.16
Cerebras-GPT-111M	111M	MHA	Learned Abs.	0.7649	0.6214	0.4901	0.3552	0.5941	0.2710	51.61
GPT2-124M	124M	MHA	Learned Abs.	0.8215	0.4872	0.5162	0.3659	0.6289	0.3114	52.19
GPT-Neo-125M	125M	MHA	Learned Abs.	0.7974	0.6171	0.5043	0.3721	0.6306	0.3040	53.76
OPT-125M	125M	MHA	Learned Abs.	0.8176	0.5618	0.5130	0.3715	0.6284	0.3158	53.47
Mamba-130M	130M	SSM	Implicit	0.8043	0.5523	0.5217	0.3756	0.6447	0.3526	54.19
SmolLM2-135M	135M	GQA	RoPE	0.8010	0.6046	0.5304	0.3956	0.6823	0.4318	57.43
Pythia-160M	162M	MHA	RoPE	0.7744	0.5511	0.5114	0.3654	0.6181	0.3019	52.04
RWKV-169M	169M	RWKV	Implicit	0.8242	0.6043	0.5178	0.3695	0.6474	0.3218	54.75
Cerebras-GPT-256M	256M	MHA	Learned Abs.	0.7892	0.6150	0.5130	0.3680	0.6137	0.2860	53.08
Gemma3-270M	270M	MQA	RoPE	0.8008	0.5798	0.5200	0.4007	0.6757	0.4090	56.43
number of parameters > 300M										
OPT-350M	331M	MHA	Learned Abs.	0.8267	0.5777	0.5233	0.3987	0.6507	0.3684	55.76
GPT2-355M	355M	MHA	Learned Abs.	0.8266	0.5859	0.5312	0.3910	0.6763	0.3938	56.75
SmolLM2-360M	362M	GQA	RoPE	0.8161	0.6202	0.5250	0.4053	0.7165	0.5450	60.47
Mamba-370M	371M	SSM	Implicit	0.8195	0.6214	0.5533	0.4002	0.6937	0.4649	59.22
Pythia-410M	405M	MHA	RoPE	0.8162	0.6073	0.5391	0.3900	0.6687	0.4063	57.13
Qwen2.5-0.5B	494M	GQA	RoPE	0.8254	0.6205	0.5643	0.4432	0.7035	0.5224	61.32
MobiLlama-0.5B	500M	MHA	RoPE	0.7919	0.5590	0.5288	0.4022	0.7203	0.5252	58.79
Pythia-1B	1011M	MHA	RoPE	0.8320	0.6086	0.5328	0.4017	0.7051	0.4713	59.19
TinyLlama-1.1B	1100M	GQA	RoPE	0.8082	0.5804	0.5880	0.4115	0.7296	0.5922	61.83
μLM-70M	74M	DMLA	ALiBiLe	0.7819	0.6367	0.5120	0.4339	0.6886	0.5018	59.25
μLM-50M	55M	DMLA	ALiBiLe	0.7714	0.6087	0.5177	0.3858	0.6779	0.4916	57.55
μLM-30M	37M	DMLA	ALiBiLe	0.7653	0.5968	0.5325	0.3796	0.6682	0.4661	56.81
μLM-9M	9M	DMLA	ALiBiLe	0.6439	0.5836	0.5210	0.3610	0.6533	0.3111	51.23

procedural and factual commonsense as the model's last impression, exploiting the recency sensitivity quantified in P3 above to bias final behavior toward grounded everyday knowledge rather than toward generic web text.

In summary, μMix navigates the capacity wall of sub-100M models by aggressively omitting high-format-cost domains, physically separating linguistic foundation from commonsense injection, and exploiting micro-model recency sensitivity via WSD-coupled annealing.

6 Experiments

6.1 Main Results

Evaluation setup. We evaluate base language model capability using `lm-evaluation-harness` [21] under a zero-shot setting without prompt tuning. We use six standard benchmarks covering the three capability clusters most relevant to on-device interaction: linguistic knowledge (BLiMP [51]), contextual understanding (BoolQ [52], WinoGrande [53], SIQA [54]), and commonsense reasoning (PIQA [55], HellaSwag [56]). The reported score is the unweighted average of the six task accuracies, multiplied by 100. All baselines are obtained from publicly available checkpoints [2] listed with their Hugging Face identifiers in Appendix E.

Results. Table 1 compares μLM with representative SLMs ranging from sub-100M models to 1B-class baselines. In the sub-300M regime, μLM-70M achieves the best averaged performance despite using substantially fewer parameters. More importantly, μLM-50M reaches 57.55, on par with the strongest sub-300M baselines while using less than half the parameters. The 30M-class model remains highly competitive, surpassing several models with 100M–300M parameters, while even μLM-9M achieves 51.23, above Pythia-70M and close to multiple 100M-class baselines. The comparison with larger models further highlights the capability density of μLM. μLM-70M matches or exceeds several 300M–1B baselines, although it remains smaller by more than an order of magnitude in some cases. We note that μLM is intentionally limited to a maximum context length of 1024 tokens. Ultra-constrained IoT devices are typically dominated by command- or utterance-level interaction, where inputs are short, user intent is explicit, and responses must be produced with low latency [57, 58]. We therefore did not extend context length to chase benchmark scores.

[2] MobileLLM [27] is not included in our evaluation as its checkpoints are gated and not publicly accessible.

Table 2: Ablations on (a) attention mechanism, (b) DMLA components, and (c) curriculum stages.

Method	KV	Layers	Hidden	Params	Score
MHA	12	9	768	76.30M	56.27
MQA	1	10	768	72.56M	55.84
GQA	4	10	768	75.51M	56.38
MLA	12	10	768	76.13M	56.50
DMLA	**12**	**10**	**768**	**74.13M**	**57.62**

(a) Attention comparison (∼70M).

ALiBiLe	Decoupled	Fused_V	Gated	Score
				56.50
√				56.64
√	√			56.86
√	√	√		57.25
√	√	√	√	**57.62**

(b) DMLA components.

Curriculum	Score
Cosine baseline	57.62
Single-stage	58.34
Two-stage	58.91
Three-stage	**59.25**
Four-stage	59.18

(c) Curriculum stages.

6.2 Ablation Study

Evaluation setup. Table 2 validates the two main ingredients of μLM: DMLA on the architecture side and μMix on the data side. Unless otherwise specified, all ablations are conducted at the 70M scale and use the same tokenizer, optimizer, training-token budget, and six-task evaluation suite as in the main experiment. The attention-mechanism and DMLA-component ablations are trained with the same cosine-decay recipe and matched data exposure.

Attention mechanism. Table 2(a) compares MHA, MQA, GQA, MLA, and DMLA under comparable model sizes. MQA reduces KV state but performs worst, indicating that overly aggressive sharing hurts expressivity at this scale. GQA improves over MQA and MHA, but remains below MLA, confirming the value of latent KV compression for memory-first models. However, the MLA baseline still reaches only 56.50. DMLA improves this to 57.62, gaining 1.12 points over MLA while using fewer parameters. This suggests that the bottleneck in tiny latent-attention models is not merely cache size, but the organization of the compressed state.

DMLA components. Table 2(b) further decomposes DMLA. Starting from the MLA-style latent baseline, adding ALiBiLe gives a small gain, showing that learnable distance priors are helpful but insufficient by themselves. Decoupling the latent routes brings an additional gain, but the improvement is moderate because using the content latent to form values, as in the MLA default, is already a reasonable design. The larger gains appear when decoupling is paired with FUSED-V and gated fusion. FUSED-V lets the value bank read from both content and position latents. The post-softmax gate then allows each token and head to adaptively choose between the content-retrieval route and the position-biased local route. These components raise the score from 56.50 to 57.62.

μMix curriculum. Table 2(c) studies the data curriculum while keeping the DMLA architecture fixed. The standard cosine baseline obtains 57.62. Replacing it with a WSD-style single-stage recipe improves the score to 58.34, indicating that a stable training phase followed by a controlled cooldown is already beneficial for micro-models. Introducing a two-stage curriculum further improves performance to 58.91 by separating general language learning from later commonsense injection. The proposed three-stage curriculum achieves the best score, 59.25, by additionally separating the early syntax/fluency warmup from the language–logic trunk. A four-stage variant slightly drops to 59.18, suggesting that further subdividing the early curriculum introduces unnecessary distribution switches and yields diminishing returns. These results support the design of μMix: tiny models benefit not only from carefully selected data, but also from when different data families are presented during training. Detailed mixture ratios and training-stage definitions are deferred to Appendix C.

6.3 μAgent Analysis

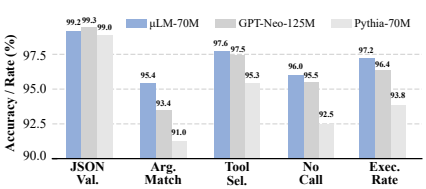

Figure 5: Analysis of μAgent.

The demand for on-device agents on IoT devices has surged rapidly [5]. Agents equipped with basic tool-calling and logical routing capabilities can endow devices (such as adaptive noise cancellation in hearing aids or autonomous obstacle avoidance in unmanned equipment) with autonomous decision-making in offline environments [59]. However, strict code-level JSON output, accurate semantic routing, and reliable rejection of inappropriate tool invocations were previously considered advanced capabilities exclusive to LLMs with billions of parameters. To verify the feasibility of using extremely tiny models as a *domain-specific tool-calling router*, we fine-tune μLM-70M into μAgent. The fine-tuning corpus contains approximately 320M tokens curated across 8 high-frequency on-device scenarios such as weather queries and device control, covering multi-tool selection, argument slot-filling, and negative-sample defense. We compare

μAgent against two same-scale baselines, Pythia-70M and GPT-Neo-125M, all fine-tuned under an identical recipe and evaluated on a shared 1,000-example out-of-distribution benchmark that injects unseen multinational city entities, redundant colloquial prefixes, and adversarial hard negatives. Detailed training and evaluation protocols are deferred to Appendix F.

As shown in Figure 5, μAgent matches or outperforms both baselines across five core metrics. It exhibits remarkable robustness on fundamental JSON validity (JSON Val., 99.2%), zero-shot generalization on out-of-vocabulary noisy entities (Arg. Match, 95.4%, +2.0 over GPT-Neo-125M and +4.4 over Pythia-70M), tool selection (Tool Sel., 97.6%), reliable rejection of chitchat (No Call, 96.0%), and end-to-end deployable behavior (Exec. Rate, 97.2%, +0.8 and +3.4 respectively). The consistent advantage over Pythia-70M, an exactly parameter-matched baseline, confirms that the gain stems from μLM's pre-training rather than tuning effort, and that extremely tiny models are fully capable of structured intent-parsing tasks. This provides promising technical validation for deploying fully local, privacy-preserving agents on ubiquitous yet resource-constrained edge devices.

6.4 On-Device Profiling

To verify the deployment feasibility of μLM on ultra-constrained edge hardware, we profile three variants directly on a Raspberry Pi Zero, one of the most resource-restricted development boards, with a CPU clocked below 1.0 GHz and less than 500 MB of system memory. We deliberately disable any third-party operator accelerators, GEMM tuning libraries, or hardware-specific kernels: every result reported is from pure CPU forward execution, reflecting the model's native compute and memory footprint. Table 3 reports the on-board BF16 model size and the runtime peak RAM during decode for generation lengths $S \in \{1, 32, 64, 128\}$. The peak RAM is measured as the memory footprint of maximum activation tensors plus the KV cache. Thanks to the latent-cache compression in DMLA, the KV-cache memory scaling remains minimal even at $S = 128$. We also report the TTFT/TBT using a 10-token input prompt. Latency results show that all three variants maintain a TBT well below 200 ms with throughputs up to 10.4 tokens/s.

Table 3: On-device profiling of μLM on RPi Zero.

Model	Size (MB)	Runtime RAM (MB)				Latency TTFT/TBT
		S=1	32	64	128	
μLM-70M	148.3	0.04	1.18	2.36	4.72	990/186 ms
μLM-50M	110.2	0.04	1.21	2.43	4.85	830/142 ms
μLM-30M	74.6	0.04	1.16	2.33	4.65	526/96 ms

7 Discussion

μLM makes several deliberate trade-offs for the sub-100M on-device regime. First, given the extremely limited parameter budget, the model is not intended to store dense world knowledge or support broad code/math synthesis. Instead, μLM prioritizes fluent language understanding, commonsense reasoning, and reliable instruction following, which are more relevant to ultra-constrained deployment. Extending it to specialized domains such as mathematics or code remains an important future direction, potentially following specialist-model paradigms such as Qwen2.5-Math [60] and Qwen2.5-Coder [61]. Second, DMLA is designed for the memory-first setting where aggressive latent compression makes every head dimension valuable. As model size and latent width increase, a monolithic latent may be sufficient to encode both semantic and positional information, so DMLA should be viewed as a sub-100M architecture rather than a universal replacement for MLA. Finally, μLM currently targets English only because multilingual vocabularies would make embedding tables dominate the parameter budget. Future work may explore vocabulary pruning or language-aware vocabulary sharing to support multilingual μLM under similarly tight constraints. We see language models for ubiquitous yet constrained devices as an emerging frontier where standard scaling intuitions may break down, inviting fundamental rethinking of architecture and data design.

8 Conclusion

We rethink sub-100M SLM design, where the binding constraint is total memory footprint rather than raw compute, and propose μLM, coupling a memory-first architecture (DMLA) with a capacity-aware multi-stage curriculum (μMix). At 70M parameters, μLM surpasses all evaluated sub-300M baselines and rivals several sub-1B models; even at 9M, it remains competitive across baselines, demonstrating that capability density at this scale is unlocked by deliberate co-design. Fine-tuned into a tool-calling agent, μLM outperforms same-scale baselines and delivers a 97.2% end-to-end executable rate.

Acknowledgments and Disclosure of Funding

This work was supported by NSF China (No. 62372288, W2412089). The authors declare no competing financial interests.

References

[1] Chaojun Xiao, Jie Cai, Weilin Zhao, Biyuan Lin, Guoyang Zeng, Jie Zhou, Zhi Zheng, Xu Han, Zhiyuan Liu, and Maosong Sun. Densing law of llms. *Nature Machine Intelligence*, pages 1–11, 2025.

[2] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[3] MiniCPM Team, Chaojun Xiao, Yuxuan Li, Xu Han, Yuzhuo Bai, Jie Cai, Haotian Chen, Wentong Chen, Xin Cong, Ganqu Cui, et al. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

[4] Vasileios Tsoukas, Anargyros Gkogkidis, Eleni Boumpa, and Athanasios Kakarountas. A review on the emerging technology of tinyml. *ACM Comput. Surv.*, 56(10), June 2024. ISSN 0360-0300. doi: 10.1145/3661820. URL `https://doi.org/10.1145/3661820`.

[5] Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*, 2024.

[6] Raspberry Pi. Rp3a0. `https://www.raspberrypi.com/documentation/computers/processors.html#rp3a0`, 2026. Accessed 13-Apr-2026.

[7] ST. Stm32n657. `https://www.st.com/en/microcontrollers-microprocessors/stm32n657x0.html`, 2026. Accessed 13-Apr-2026.

[8] Milk-V. Cv1800b. `https://milkv.io/chips/cv1800b`, 2026. Accessed 13-Apr-2026.

[9] Josh Millar, Yushan Huang, Sarab Sethi, Hamed Haddadi, and Anil Madhavapeddy. Benchmarking ultra-low-power μnpus. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*, ACM MOBICOM '25, page 1060–1074, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711299. doi: 10.1145/3680207.3765264.

[10] Rong Ding, Haiming Jin, Ningzhi Zhu, Zijie Chen, Yi Fu, Fengyuan Zhu, Guiyun Fan, Xiaohua Tian, and Linghe Kong. Bluetooth-enabled transparent rf sensing. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*, pages 186–200, 2025.

[11] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pages 2397–2430. PMLR, 2023.

[12] Nolan Dey, Gurpreet Gosal, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness, et al. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023.

[13] Dan John Velasco and Matthew Theodore Roque. Rethinking the role of text complexity in language model pretraining. In *Proceedings of the First BabyLM Workshop*, pages 1–28, 2025.

[14] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[15] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 44(3):33–39, 2024.

[16] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[17] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.

[18] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[19] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

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

[21] Lintang Sutawika, Hailey Schoelkopf, Leo Gao, Baber Abbasi, Stella Biderman, Jonathan Tow, Ben Fattori, Charles Lovering, Jason Phang, and Anish Thite. Eleutherai/lm-evaluation-harness, January 2026. URL `https://doi.org/10.5281/zenodo.18394108`.

[22] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *NeurIPS*, 2023.

[23] Long Cheng, Yan Gu, Qingzhi Liu, Lei Yang, Cheng Liu, and Ying Wang. Advancements in accelerating deep neural network inference on aiot devices: A survey. *IEEE Transactions on Sustainable Computing*, 9 (6):830–847, 2024. doi: 10.1109/TSUSC.2024.3353176.

[24] Zijie Chen, Yiming Gao, and Junrui Liang. Lopdm: A low-power on-device predictive maintenance system based on self-powered sensing and tinyml. *IEEE Transactions on Instrumentation and Measurement*, 72: 1–13, 2023.

[25] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in neural information processing systems*, 33:11711–11722, 2020.

[26] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. Tiny machine learning: Progress and futures [feature]. *IEEE Circuits and Systems Magazine*, 23(3):8–34, 2023.

[27] 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.

[28] Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer, Michael Felsberg, Timothy Baldwin, Eric P. Xing, and Fahad Shahbaz Khan. Mobillama: Towards accurate and lightweight fully transparent gpt, 2024.

[29] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

[30] Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. Transmla: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*, 2025.

[31] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[32] Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=R8sQPpGCv0`.

[33] Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A Raffel. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36:50358–50376, 2023.

[34] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

[35] 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.

[36] Loubna Ben Allal, Anton Lozhkov, and Daniel van Strien. Cosmopedia: how to create large-scale synthetic data for pre-training large language models. Hugging Face Blog, 2024.

[37] Marah Abdin, Jyoti Aneja, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL `https://arxiv.org/abs/2404.14219`.

[38] Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15332–15344, 2024.

[39] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37:16692–16723, 2024.

[40] Seijin Kobayashi, Yassir Akram, and Johannes Von Oswald. Weight decay induces low-rank attention layers. *Advances in Neural Information Processing Systems*, 37:4481–4510, 2024.

[41] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 5797–5808, 2019.

[42] Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[44] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[45] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[46] Zhenting Qi, Fan Nie, Alexandre Alahi, James Zou, Himabindu Lakkaraju, Yilun Du, Eric Xing, Sham Kakade, and Hanlin Zhang. Evolm: In search of lost language model training dynamics. *arXiv preprint arXiv:2506.16029*, 2025.

[47] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

[48] 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.

[49] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna B Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *Advances in Neural Information Processing Systems*, 37:76232–76264, 2024.

[50] Mahnaz Koupaee and William Yang Wang. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*, 2018.

[51] Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R Bowman. Blimp: The benchmark of linguistic minimal pairs for english. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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*, pages 4791–4800, 2019.

[57] Gabriel Haas, Michael Rietzler, Matt Jones, and Enrico Rukzio. Keep it short: A comparison of voice assistants' response behavior. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pages 1–12, 2022.

[58] Tawfiq Ammari, Jofish Kaye, Janice Y Tsai, and Frank Bentley. Music, search, and iot: How people (really) use voice assistants. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 26(3):1–28, 2019.

[59] Peter Udo Diehl, Yosef Singer, Hannes Zilly, Uwe Schönfeld, Paul Meyer-Rachner, Mark Berry, Henning Sprekeler, Elias Sprengel, Annett Pudszuhn, and Veit M Hofmann. Restoring speech intelligibility for hearing aid users with deep learning. *Scientific Reports*, 13(1):2719, 2023.

[60] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[61] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[62] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 1715–1725, 2016.

[63] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, 2021.

[64] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[65] 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.

[66] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.

Appendix Contents

A Implementation Details **15**

A.1 Model Architecture Hyperparameters 15

A.2 Tokenizer Training 15

B Theoretical Analysis **16**

B.1 Expressive Advantage of MLA over GQA Under the Same Total Parameter Budget 16

B.2 Parameter Efficiency of DMLA's Latent Factorization 18

B.3 Cache-Side Argument for ALiBi over RoPE 18

B.4 Three Design Implications 19

C Pre-Training Details **19**

C.1 Optimization and Schedule 19

C.2 μMix Stage Configurations and Token Budgets 20

C.3 WSD Decay Shape Comparison 22

D Architectural Ablations and Visualizations **23**

D.1 Latent Width Allocation: (m_c, m_r) Sweep 23

D.2 Dense FFN vs. Mixture-of-Experts at Sub-100M Scale 23

D.3 Visualizing Learned ALiBiLe Slopes 24

D.4 Visualizing the Input-aware Stream Gate 25

E Extended Evaluation **26**

E.1 Evaluation Setup 26

E.2 Baseline Checkpoints and Sources 27

E.3 INT8 Quantization Robustness 27

F μAgent: Training and Evaluation Details **28**

F.1 Training Data Curation 29

F.2 Fine-Tuning Recipe (Shared Across All Three Models) 29

F.3 OOD Tool-Calling Benchmark 30

F.4 Evaluation Metrics 30

G μLM-Instruct: Conversational Capability **31**

G.1 SFT Data and Training 31

G.2 Sample Conversations 32

A Implementation Details

A.1 Model Architecture Hyperparameters

Table 4 lists the architectural hyperparameters of all four μLM variants reported in the main paper. The four scales (9M / 30M / 50M / 70M) share the same DMLA design and differ only in width, depth, and the per-stream latent/QK widths.

Several design conventions are shared by all variants:

- **Vocabulary.** A single 16,384-token byte-level BPE tokenizer is used across all four variants (Section A.2). Input and output embeddings are weight-tied to keep the embedding-table footprint manageable at sub-100M scale.
- **Feed-forward.** Dense SwishGLU [45] feed-forward networks; no Mixture-of-Experts is used (we discuss this choice in Section D.2).
- **Normalization.** Pre-norm with RMSNorm [44], applied before both the attention and feed-forward sub-layers; $\epsilon = 10^{-5}$.
- **Stream gate.** Initialized at $\sigma(g_0) = 0.7$ (favoring content stream); input-conditioned scale $\alpha = 0.5$; W_g is initialized to zero so that the gate starts purely from its static prior.
- **ALiBiLe.** Per-head slopes initialized from the geometric ALiBi [32] schedule and made trainable thereafter (no reparameterization).
- **Maximum context length.** $T_{\max} = 1024$ during pre-training. As discussed in Section 6.1, this is a deployment-driven choice for ultra-constrained edge devices, not a benchmark-oriented one.

A.2 Tokenizer Training

We train a single byte-level BPE tokenizer [62] of vocabulary size 16,384 and use it for all μLM variants and all downstream fine-tuning experiments. The choice of a small vocabulary is deliberate: at the 70M scale, a vocabulary of 16K consumes approximately 12.6M parameters in the tied embedding table (about 17% of total parameters), whereas the 32K–128K vocabularies common in larger SLMs would consume 25-60% of the parameter budget, an unacceptable loss of capacity for the Transformer layers themselves under our memory-first regime.

Training corpus. The tokenizer is trained on a 1.5M-document mixed-domain sample drawn from the same five corpora used in μMix pre-training, with proportions chosen to roughly approximate the average mixture across all curriculum stages (Table 5). The sample is tokenized with whitespace-aware byte-level pre-tokenization and shuffled before training.

Training procedure. We use the Hugging Face `tokenizers` library's BPE trainer with the following settings:

- **Algorithm.** Byte-level BPE (no character-level UNK; every byte is representable).
- **Pre-tokenization.** `ByteLevel` with `add_prefix_space=False`, which splits on whitespace and converts each byte into one of 256 base symbols before merging.
- **Target vocabulary size.** 16,384 tokens, including 256 byte-level base symbols, 4 special tokens (<|bos|>, <|eos|>, <|pad|>, <|unk|>), and the remaining $\sim$16,124 slots for learned merges.
- **Minimum merge frequency.** 2 (any pair appearing twice is eligible for merging).

Resulting compression ratio. On a held-out validation slice from the same five-source mixture, the trained tokenizer achieves an average compression ratio of approximately 4.1 bytes per token, comparable to GPT-2's 32K-vocabulary tokenizer (4.0 bytes/token) on similar English text despite using only half the vocabulary. This is consistent with the observation that for clean English natural language without code or specialized notation, vocabularies beyond $\sim$10K provide diminishing returns on compression [63], making 16K an attractive trade-off for sub-100M models.

Reuse across model scales and downstream tasks. A single tokenizer is shared across all four μLM scales (9M / 30M / 50M / 70M) and all downstream variants (μAgent, μLM-Instruct). This

Table 4: Architectural hyperparameters of the four μLM variants. All variants share the same DMLA mechanism, vocabulary, SwishGLU FFN, RMSNorm, ALiBiLe positional bias, and tied input–output embeddings; they differ only in scale-dependent dimensions. d_h^c and d_h^r are per-head QK widths for the content and position streams; m_c and m_r are the latent cache widths for the two streams (per-token persistent KV state size). The intermediate dimension follows a $\sim 2.67\times$ ratio over D_{hidden}, standard for SwishGLU.

Hyperparameter	μLM-9M	μLM-30M	μLM-50M	μLM-70M
Hidden size D	256	512	512	768
Number of layers L	6	10	16	10
Number of heads h	8	12	12	12
Content latent width m_c	64	64	80	80
Position latent width m_r	32	32	48	48
Per-token KV-cache size $m_c + m_r$	96	96	128	128
Content QK head dim d_h^c	28	28	40	40
Position QK head dim d_h^r	20	20	24	24
Value head dim d_v	48	48	64	64
FFN intermediate size	704	1408	1408	2048
FFN architecture	Dense SwishGLU			
Vocabulary size $\|V\|$	16,384 (byte-level BPE, tied input-output embeddings)			
Max context length $T_{\max}$	1024			
Position encoding	ALiBiLe (learnable per-head slopes)			
Normalization	RMSNorm pre-norm, $\epsilon = 10^{-5}$			
Stream-gate static init $\sigma(g_0)$	0.7			
Stream-gate dynamic scale α	0.5			

Table 5: Composition of the 1.5M-document tokenizer training sample. Proportions are chosen to approximate the time-averaged mixture across μMix curriculum stages, ensuring the tokenizer is not biased toward any single source's vocabulary.

Source	Proportion (%)	Documents
FineWeb-Edu	50.0	750,000
Cosmopedia	35.0	525,000
WikiHow	7.0	105,000
TinyStories	5.0	75,000
Simple English Wikipedia	3.0	45,000
Total	100.0	1,500,000

avoids any tokenizer-induced artifacts in the cross-scale comparison (Figure 1, right) and ensures that fine-tuning data does not require re-tokenization. Special tokens <|bos|> and <|eos|> are prepended/appended to sequences during training; <|pad|> is used by the dataloader for batch alignment but never appears in actual training targets (loss is masked accordingly).

B Theoretical Analysis

This appendix provides the full derivations and supporting discussions for Section 3. Section B.1 formalizes the parameter-efficiency argument behind starting from MLA rather than GQA in the sub-100M regime; Section B.2 accounts for the parameter savings DMLA gains by eliminating cross-route per-head up-projections; and Section B.3 elaborates on the cache-side argument for ALiBi-style biases that was omitted from the main text for space. Section B.4 consolidates the three design objectives motivating DMLA.

B.1 Expressive Advantage of MLA over GQA Under the Same Total Parameter Budget

This section provides the full parameter-efficiency proof explaining why the latent-factorized design of Multi-head Latent Attention (MLA) can be strictly more expressive than Grouped-Query Attention (GQA) under the same *total parameter budget*, provided that the target attention layer admits a shared low-rank latent structure across heads.

Setup and budget equivalence. Let the input hidden state be $x \in \mathbb{R}^D$. Assume there are h query heads, each with head dimension $d = D/h$. We focus exclusively on the dense content-based key/value (KV) generation mechanism, and ignore softmax and output projection.[3] Assuming all other components of the Transformer block (e.g., embeddings, feed-forward networks, and query/output projections) are identical, matching the total model parameter budget is mathematically equivalent to matching the parameter budget of the KV branches.

Grouped-Query Attention (GQA). Suppose GQA uses g KV groups. For each group $j \in \{1, \dots, g\}$, let $K_j, V_j \in \mathbb{R}^{d \times D}$ be the key and value projections of that group. Each head $i \in \{1, \dots, h\}$ is assigned to one group $\pi(i) \in \{1, \dots, g\}$, so that $k_i(x) = K_{\pi(i)}x$ and $v_i(x) = V_{\pi(i)}x$. Thus, GQA can realize at most g distinct KV prototypes across the h heads. The total number of KV parameters in GQA is

$$P_{\mathrm{GQA}}^{KV} = 2gdD. \tag{4}$$

Multi-head Latent Attention (MLA). In MLA, we first compute a shared latent representation $c(x) = Bx$ via $B \in \mathbb{R}^{r \times D}$, where r is the latent width. Then each head has its own key/value readout matrices $A_i, C_i \in \mathbb{R}^{d \times r}$, and generates $k_i(x) = A_iBx$ and $v_i(x) = C_iBx$. The total number of KV parameters in MLA is

$$P_{\mathrm{MLA}}^{KV} = rD + hdr + hdr = 3Dr. \tag{5}$$

Equal total parameter budget. Let P_{other} denote the number of parameters in the model excluding the KV projections. Since MLA and GQA share identical structures for these non-KV components, equating the total model parameter count ($P_{\mathrm{total}} = P_{\mathrm{other}} + P^{KV}$) directly implies:

$$P_{\mathrm{GQA}}^{KV} = P_{\mathrm{MLA}}^{KV}, \quad \text{i.e.,} \quad 2gdD = 3Dr, \quad \text{hence} \quad r = \frac{2gd}{3}. \tag{6}$$

A shared-latent function class.

Definition 1 (Shared-latent multi-head KV family)**.** Fix integers $r \geq 1$ and $m \geq 1$. Let $\mathcal{F}(r, m)$ denote the family of all multi-head KV mappings such that there exist a shared latent matrix $B \in \mathbb{R}^{r \times D}$ and m readout pairs $(A^{(1)}, C^{(1)}), \dots, (A^{(m)}, C^{(m)})$ with $A^{(\ell)}, C^{(\ell)} \in \mathbb{R}^{d \times r}$, for which each head i is assigned a type $\tau(i) \in \{1, \dots, m\}$ and satisfies $k_i(x) = A^{(\tau(i))}Bx$ and $v_i(x) = C^{(\tau(i))}Bx$. Moreover, the induced KV mappings are pairwise distinct: $(A^{(u)}B, C^{(u)}B) \neq (A^{(v)}B, C^{(v)}B)$ for $u \neq v$, and the type assignment $\tau \to \{1, \dots, m\}$ is surjective.

Theorem 1 (Strict expressive advantage of MLA over GQA at equal total budget)**.** *Fix a total parameter budget. Let GQA and MLA share identical non-KV components such that their equal total budgets imply equal KV budgets: $P_{\mathrm{GQA}}^{KV} = P_{\mathrm{MLA}}^{KV}$ (equivalently, $r = \frac{2gd}{3}$). If a target mapping belongs to $\mathcal{F}(r, m)$ for some $m > g$ and $h \geq m$, then MLA can represent it exactly whereas GQA cannot. Consequently, under the same total parameter budget, MLA is strictly more expressive than GQA on the family $\mathcal{F}(r, m)$.*

Proof. We establish two intermediate facts and then combine them.

Lemma 1 (MLA realizes $\mathcal{F}(r, m)$). If $h \geq m$, then MLA with latent width r can represent any KV mapping in $\mathcal{F}(r, m)$ exactly. Take any mapping in $\mathcal{F}(r, m)$. By Definition 1, there exist a shared latent matrix B and readout pairs $(A^{(1)}, C^{(1)}), \dots, (A^{(m)}, C^{(m)})$. Set the MLA latent projection to $W^{DKV} = B$. For each head i, define $W_i^{UK} = A^{(\tau(i))}$ and $W_i^{UV} = C^{(\tau(i))}$. Then $k_i(x) = W_i^{UK}W^{DKV}x = A^{(\tau(i))}Bx$ and $v_i(x) = W_i^{UV}W^{DKV}x = C^{(\tau(i))}Bx$, exactly matching the target mapping.

Lemma 2 (GQA requires $g \geq m$). If a GQA layer can represent every KV mapping in $\mathcal{F}(r, m)$ exactly, then it must satisfy $g \geq m$. A GQA layer has only g KV groups, hence at most g distinct KV prototypes $(K_1, V_1), \dots, (K_g, V_g)$. Consider a mapping in $\mathcal{F}(r, m)$ with m pairwise distinct induced mappings $(A^{(1)}B, C^{(1)}B), \dots, (A^{(m)}B, C^{(m)}B)$. If $g < m$, then by the pigeonhole principle at least two distinct induced mappings, say those of types $u \neq v$, must be assigned to the same GQA group. But heads in the same GQA group must share exactly the same key and value maps, implying $A^{(u)}B = A^{(v)}B$ and $C^{(u)}B = C^{(v)}B$, which contradicts the distinctness assumption. Therefore $g \geq m$ is necessary.

[3] This analysis isolates the dense KV capacity; positional encoding considerations are addressed separately in Section B.3.

Combining. Lemma 1 shows that MLA represents every mapping in $\mathcal{F}(r, m)$ when $h \geq m$. Lemma 2 shows that whenever $m > g$, GQA cannot represent some such mapping. Hence under the equal-budget constraint $r = \frac{2gd}{3}$ and the regime $m > g$, $h \geq m$, there exist mappings representable by MLA but not by GQA, proving strict expressive advantage on $\mathcal{F}(r, m)$. □

Practical implication for sub-100M models. The above theorem is most consequential precisely in the regime μLM targets. At sub-100M scale, the parameter budget forces g in GQA to be small (typically $g \leq 4$, often $g = 1$ in MQA), while learned KV mappings empirically still admit shared low-rank structure with many distinct head readouts [38–40].

Remark on scope. It is important to emphasize that Theorem 1 does *not* claim that equal-parameter MLA universally subsumes equal-parameter GQA over all possible KV mappings. Rather, it proves the following conditional statement: whenever the target multi-head KV mapping is generated from a shared low-rank latent subspace yet requires many distinct effective head-level KV readouts, MLA is strictly more parameter-efficient than GQA. This conditional scope precisely characterizes the architectural regime in which latent-factorized attention is designed to operate.

B.2 Parameter Efficiency of DMLA's Latent Factorization

This section provides the detailed parameter accounting referenced in Section 4.2. We compare two designs that produce both a content key $K^c \in \mathbb{R}^{T \times h \times d_h^c}$ and a positional key $K^r \in \mathbb{R}^{T \times h \times d_h^r}$ from the same hidden state X.

A natural baseline is a *monolithic* latent: a single shared cache $C^{\text{mono}} \in \mathbb{R}^{T \times (m_c + m_r)}$ produced by $W^{D,\text{mono}} \in \mathbb{R}^{D \times (m_c + m_r)}$, followed by per-head readouts to both key types. Each of the h heads then needs a $(m_c + m_r) \times (d_h^c + d_h^r)$ matrix to read k_i^c and k_i^r jointly from the shared latent. The key-readout parameter count is therefore $P_{\text{mono}}^K = h(m_c + m_r)(d_h^c + d_h^r)$. DMLA instead maintains *two* separate latents $C^c \in \mathbb{R}^{T \times m_c}$ and $C^r \in \mathbb{R}^{T \times m_r}$, with each key reading only from its corresponding latent: $K^c = \text{split}_h(C^c W_c^K)$ uses $W_c^K \in \mathbb{R}^{m_c \times h d_h^c}$, and $K^r = \text{split}_h(C^r W_r^K)$ uses $W_r^K \in \mathbb{R}^{m_r \times h d_h^r}$, yielding $P_{\text{DMLA}}^K = h(m_c d_h^c + m_r d_h^r)$.

The difference,

$$\Delta P^K = P_{\text{mono}}^K - P_{\text{DMLA}}^K = h(m_c d_h^r + m_r d_h^c), \tag{7}$$

corresponds exactly to the *cross-route* readout terms that DMLA eliminates: an $m_c \to d_h^r$ mapping that would produce a positional key from the content latent, and an $m_r \to d_h^c$ mapping that would produce a content key from the positional latent. Both are unnecessary under DMLA's specialization assumption: the content latent encodes semantic content alone, the positional latent encodes recency-relevant signals alone. Removing these cross-readouts yields strictly fewer parameters *without* reducing expressivity within DMLA's design intent.

Concrete example at the 70M scale. With the configuration in Table 4 ($h = 12$, $m_c = 80$, $m_r = 48$, $d_h^c = 40$, $d_h^r = 24$), the savings are $\Delta P^K = 12 \cdot (80 \cdot 24 + 48 \cdot 40) = 46{,}080$ parameters, approximately 0.06% of the 70M total. The absolute saving is modest, but it is a *free* saving with no expressivity cost under DMLA's design intent, and it reinforces the broader point that explicit content–position factorization is not a parameter overhead but a parameter *simplification* once cross-route mappings are recognized as unneeded.

B.3 Cache-Side Argument for ALiBi over RoPE

Section 3 of the main text presented two arguments for preferring ALiBi-style biases over RoPE in the sub-100M MLA regime: (i) an explicit locality prior is more parameter-efficient than learning one from data, and (ii) RoPE imposes a mandatory positional-carrier subspace inside each head, taxing semantic content. A third consideration concerns caching and projection absorption, which we elaborate here.

RoPE acts *inside* the dot product through position-dependent rotations:

$$\text{Score}_{\text{RoPE}}(t, s) = (\mathbf{R}_{\Theta,t}\mathbf{q}_t)^\top (\mathbf{R}_{\Theta,s}\mathbf{k}_s) = \mathbf{q}_t^\top \mathbf{R}_{\Theta,t-s}\mathbf{k}_s. \tag{8}$$

Although the final score depends only on the relative offset $t - s$, the cached representation must still carry the absolute rotation $\mathbf{R}_{\Theta,s}\mathbf{k}_s$ rather than the unrotated key $\mathbf{k}_s$, since the rotation cannot be factored out from the cached state without re-computing it on every query. This complicates the clean absorption of key-side projections into the query side in compressed attention, and tends to motivate additional positional-carrier states during decoding. The effect is most pronounced precisely in MLA-style designs that rely on projection absorption to keep the per-token cache small.

ALiBi, in contrast, is independent of the key/value representation and acts only as an additive logit correction:

$$\text{Score}_{\text{ALiBi}}(t, s) = \tilde{\mathbf{q}}_t^\top \tilde{\mathbf{k}}_s - \rho_h(t - s), \qquad t \geq s, \tag{9}$$

where ρ_h is the per-head slope and $\tilde{\mathbf{q}}_t, \tilde{\mathbf{k}}_s$ denote whatever compressed headwise representations the architecture chooses, independent of the positional bias. As a result, ALiBi preserves a simpler compressed-cache path: the cache can remain fully shared, or be explicitly factorized later for modeling purposes (as in DMLA), without being forced to transport a separate RoPE carrier alongside the content latent. Our **ALiBiLe** variant in Section 4.2 retains this additive-distance form while making ρ_h trainable per head, yielding the head-specific receptive-field flexibility analyzed empirically in Section D.3.

B.4 Three Design Implications

The three preceding analyses jointly motivate DMLA's architecture, distilled into three concrete design objectives for the sub-100M memory-first regime:

1. **Preserve latent KV compression.** Under a tight parameter budget, the latent factorization provides a more favorable expressivity–memory trade-off than grouped KV sharing (Theorem 1).
2. **Avoid mandatory positional carriers.** Position should enter as a lightweight attention prior rather than as a structural tax on semantic bandwidth (Section B.3).
3. **Factorize content and position purposefully.** If the two are to be separated, the separation should serve inductive bias and routing control, not merely RoPE compatibility, and the resulting parameter accounting, when done cleanly, costs nothing relative to a monolithic latent (Section B.2).

C Pre-Training Details

This appendix provides the full pre-training configuration of μLM. Section C.1 reports the optimizer and learning-rate schedule shared across all four model scales; Section C.2 documents the per-stage data mixtures and token budgets of the μMix curriculum and its ablation variants; Section C.3 compares the four decay shapes available in the WSD scheduler and justifies our choice of `one-minus-sqrt`.

C.1 Optimization and Schedule

All four μLM variants are pre-trained with identical optimizer settings, differing only in batch size and total steps (which scale with model size and per-source budgets). Table 6 summarizes the configuration. We use AdamW with the standard $\beta = (0.9, 0.95)$ for language-model pretraining (matching MiniCPM, Llama, and most modern SLMs), weight decay 0.1, and gradient clipping at 1.0. The learning rate uses a Warmup-Stable-Decay (WSD) schedule [48, 49]: a linear warmup over the first 0.8% of total steps, a long stable phase at peak LR 4×10^{-3} that spans every stage except the cooldown, and a cooldown phase that coincides with the final `stage3_cooldown` stage. The cooldown decay shape is `one-minus-sqrt` (Section C.3). Mixed-precision training uses BFloat16 for forward/backward passes, with FP32 master weights and gradient accumulation in FP32; we find this configuration numerically more stable than FP16 with loss scaling at sub-100M scale, with no measurable wall-clock penalty on modern GPUs.

Why peak LR 4×10^{-3}. This is roughly $7\times$ higher than the typical peak LR for billion-parameter Transformer pre-training ($\sim 6 \times 10^{-4}$ for Llama-7B). Two factors justify the higher LR. First, sub-100M models have much smaller parameter counts and benefit from larger relative updates per step to traverse the loss landscape efficiently within the 50B-token budget. Second, the WSD

Table 6: Pre-training hyperparameters shared by all μLM variants. Items marked "shared" are identical across the four scales; batch size and total steps scale with model size as indicated. "Effective tokens" is total physical tokens $\times$ epochs per stage.

Hyperparameter	Value
Optimizer (shared)	
Optimizer	AdamW
(β_1, β_2)	$(0.9, 0.95)$
ϵ	10^{-8}
Weight decay	0.1
Gradient clipping (global L_2)	1.0
Learning-rate schedule (shared)	
Schedule	WSD
Peak LR	4×10^{-3}
Warmup fraction	0.8%
Warmup shape	Linear
Stable phase	All non-cooldown stages
Cooldown phase	Final `stage3_cooldown` stage
Cooldown decay shape	`one-minus-sqrt`
Data and batching	
Total physical tokens	10B
Total effective tokens	50B, 5 epochs/stage
Sequence length	1024
Batch size	48 for μLM-9M/30M; 32 for μLM-50M/70M
Effective batch	32K–49K tokens/step

schedule's stable phase remains at peak LR throughout the bulk of training, so the effective LR mass is dominated by this peak; a higher peak compensates for the relatively short pre-training. We confirmed this empirically: peak LRs of 1×10^{-3} and 2×10^{-3} converge to noticeably worse held-out loss; 8×10^{-3} destabilizes training. 4×10^{-3} is the sweet spot consistent with MiniCPM's reported optimum [48] for similar-scale models.

Validation monitoring. A fixed mixed-domain validation set of approximately 6,000 documents (40% FineWeb-Edu, 30% Cosmopedia, 15% WikiHow, 10% Simple English Wikipedia, 5% TinyStories) is held out from training and evaluated every 500 optimizer steps. The validation set composition is intentionally close to the average training mixture to track held-out language-modeling performance without artificially biasing toward any single source. Validation loss and perplexity are logged to `val_log.jsonl` for post-hoc analysis, and we use this signal (in addition to the lm-evaluation-harness benchmark suite) to determine the empirical capacity wall referenced in Section 5.1 (P1).

C.2 μMix Stage Configurations and Token Budgets

This section provides the exact per-stage data mixtures and token budgets for the μMix three-stage baseline used to train μLM, and for the four ablation variants compared in Section 6.2. All five configurations share the same total physical token budget (10B) and same effective training tokens (50B at 5 epochs per stage), the same model architecture, the same tokenizer, and identical optimizer settings (Section C.1); only the `STAGE_BLUEPRINT` list and—for the cosine baseline—the LR scheduler differ.

Cross-configuration invariants. Two design invariants make the comparison clean. First, configurations **A**, **B**, **C**, and **D** all use the same WSD learning-rate schedule with peak LR 4×10^{-3} and `one-minus-sqrt` cooldown shape, and all preserve a final `stage3_eval_cooldown` stage with identical configuration (0.8B physical / 4B effective tokens, unchanged mixture); this is what triggers the LR cooldown phase in our scheduler. Without keeping this stage fixed across the WSD ablations, removing curriculum structure would simultaneously remove LR decay, conflating two independent variables. Second, all five configurations preserve the same per-source effective token budget across the full training run (FineWeb-Edu 23.3B, Cosmopedia 17.3B, TinyStories 3.0B, WikiHow 3.95B, Simple Wikipedia 2.45B), so any benchmark difference among **A/B/C/D** can be attributed solely to

Table 7: Per-stage data mixtures and physical-token budgets for the five curriculum configurations. "FE" = FineWeb-Edu, "Co" = Cosmopedia, "TS" = TinyStories, "WH" = WikiHow, "SW" = Simple English Wikipedia. All variants train for 5 epochs per stage; the final cooldown stage is identical across **A/B/C/D**.

Stage	Phys. tokens	FE (%)	Co (%)	TS (%)	WH (%)	SW (%)	LR phase
(0) Cosine baseline — single-stage uniform mix, cosine schedule							
Single stage	10.0 B	46.6	34.6	6.0	7.9	4.9	cosine
(A) Single-stage WSD — WSD with uniform main stage + cooldown							
Main	9.2 B	48.0	35.4	6.5	6.0	4.0	stable
Cooldown	0.8 B	30.0	25.0	0.0	30.0	15.0	cooldown
(B) Two-stage WSD — merged warmup+core trunk + commonsense + cooldown							
Trunk (1a+1b)	7.0 B	52.1	37.1	8.6	0.0	2.1	stable
Commonsense	2.2 B	35.0	30.0	0.0	25.0	10.0	stable
Cooldown	0.8 B	30.0	25.0	0.0	30.0	15.0	cooldown
(C) Three-stage WSD — our μMix recipe, used to train all μLM variants							
1a Syntax Warmup	4.0 B	50.0	35.0	15.0	0.0	0.0	stable
1b Knowledge Core	3.0 B	55.0	40.0	0.0	0.0	5.0	stable
2 Commonsense	2.2 B	35.0	30.0	0.0	25.0	10.0	stable
3 Cooldown	0.8 B	30.0	25.0	0.0	30.0	15.0	cooldown
(D) Four-stage WSD — subdivides Stage 1a into two sub-stages							
1a-i Story-heavy	1.5 B	35.0	30.0	35.0	0.0	0.0	stable
1a-ii Story-light	2.5 B	59.0	38.0	3.0	0.0	0.0	stable
1b Knowledge Core	3.0 B	55.0	40.0	0.0	0.0	5.0	stable
2 Commonsense	2.2 B	35.0	30.0	0.0	25.0	10.0	stable
3 Cooldown	0.8 B	30.0	25.0	0.0	30.0	15.0	cooldown

data-mixture scheduling rather than per-source exposure, while configuration **0** additionally varies the LR schedule. Table 7 summarizes all five configurations side by side.

Cosine baseline (0). The cosine baseline removes both axes of our design (WSD scheduling and curriculum staging) and corresponds to the most common "default" training recipe in the literature. The entire 10B physical budget is trained as a single stage with a uniform mixture identical to **A**. The learning rate follows a linear warmup (matching the warmup token count of the WSD schedule) followed by a half-cosine decay from peak LR 4×10^{-3} to a floor of 4×10^{-4} (10% of peak, following standard practice in GPT-3, LLaMA, and Chinchilla); no separate cooldown stage exists. We note that under matched peak LR, the cosine schedule has lower integrated LR mass than WSD (whose stable phase remains at peak throughout), so this baseline conflates schedule shape with effective update magnitude; we deliberately do not compensate for this in order to remain faithful to the standard cosine training recipe used in prior work, and we frame the comparison as "WSD-style training (including its higher effective LR mass) vs. standard cosine training" rather than a pure schedule-shape comparison.

Single-stage (A). The main training stage spans 9.2B physical tokens with a uniform mixture computed as the token-weighted average of the baseline's first three stages: FineWeb-Edu 48.0%, Cosmopedia 35.4%, TinyStories 6.5%, WikiHow 6.0%, Simple Wikipedia 4.0%. This preserves the global per-source share of the baseline and caps TinyStories exposure at exactly 2B effective tokens, matching the baseline's total TinyStories budget. The remaining 0.8B physical / 4B effective tokens form the shared cooldown stage.

Two-stage (B). We merge the baseline's stages 1a and 1b into a single 7B-physical trunk with mixture FineWeb-Edu 52.1%, Cosmopedia 37.1%, TinyStories 8.6%, Simple Wikipedia 2.1% (yielding the same 2B effective TinyStories cap), followed by a separate 2.2B-physical commonsense stage with FineWeb-Edu 35%, Cosmopedia 30%, WikiHow 25%, Simple Wikipedia 10%, identical to the baseline's stage 2. The shared 0.8B cooldown stage follows.

Three-stage (C, our μMix). The full μMix recipe is illustrated in Figure 4 and is the configuration used to train all four μLM variants reported in Table 1. Per-stage motivations are discussed in Section 5.2.

Table 8: Effect of WSD cooldown decay shape on μLM-70M's average score across six zero-shot benchmarks (BLiMP, BoolQ, WinoGrande, SIQA, PIQA, HellaSwag). All other hyperparameters are identical; only the cooldown shape differs. `one-minus-sqrt` is our default choice.

Decay shape $\phi(p)$	Average score	Δ vs. best
`one-minus-sqrt`: $1-\sqrt{p}$	$\mathbf{59.25}$	—
`cosine`: $\frac{1}{2}(1+\cos(\pi p))$	59.06	-0.19
`linear`: $1-p$	58.79	-0.46
`lowered_linear`: piecewise (knee 0.20 at $p=0.70$)	58.56	-0.69

Four-stage (D). We further subdivide the baseline's syntax-warmup stage (stage 1a) into two consecutive sub-stages with token-weighted mixtures that exactly reproduce its global composition. The first sub-stage (1.5B physical / 7.5B effective) is story-heavy with FineWeb-Edu 35%, Cosmopedia 30%, TinyStories 35%; the second sub-stage (2.5B physical / 12.5B effective) is story-light with FineWeb-Edu 59%, Cosmopedia 38%, TinyStories 3%. The token-weighted average of these two sub-stages exactly matches the baseline's Stage 1a mixture (TinyStories: $(35\times 1.5+3\times 2.5)/4=15\%$; FineWeb-Edu: 50%; Cosmopedia: 35%). Stages 1b (3.0B), 2 (2.2B), and the final cooldown (0.8B) remain identical to the three-stage baseline. This design tests whether finer granularity of the easy-first portion of the curriculum—without altering per-source budgets, total tokens, LR schedule, or any other variable—confers additional benefit beyond the three-stage structure.

Validation set. The fixed validation set referenced in Section C.1 is held out once and never updated across configurations or stages, with composition {FineWeb-Edu 40%, Cosmopedia 30%, WikiHow 15%, Simple Wikipedia 10%, TinyStories 5%} over approximately 6,000 documents (∼2–5M tokens), enough for a stable loss estimate at every 500-step interval.

C.3 WSD Decay Shape Comparison

The WSD schedule's cooldown phase admits multiple decay shapes, all defined as a function $\phi:[0,1]\to[0,1]$ that maps cooldown progress p to a multiplier on the peak LR (so the effective LR during cooldown is $\eta(p)=\eta_{\text{peak}}\cdot\phi(p)$). We considered four candidates:

$$\phi_{\text{linear}}(p)=1-p,\qquad \phi_{\text{cosine}}(p)=\tfrac{1}{2}(1+\cos(\pi p)),\qquad \phi_{1-\sqrt{\text{p}}}(p)=1-\sqrt{p}, \tag{10}$$

together with a piecewise variant we call `lowered_linear`, which decays linearly to a floor of 0.20 over the first 70% of cooldown and then linearly to zero, modeling a "two-slope" annealing strategy. Linear and cosine are textbook choices; `one-minus-sqrt` ($\phi_{1-\sqrt{p}}$) is the WSD-recommended shape used by MiniCPM [48] and produces front-loaded decay (LR drops fast early, then plateaus near zero), giving the model more time at low LR to refine fine-grained patterns; `lowered_linear` is a pragmatic two-slope baseline.

Why the choice of cooldown shape matters at sub-100M scale. At small scale, the cooldown stage is responsible for converting the high-LR exploration accumulated during the stable phase into a stable, well-calibrated final checkpoint. Because the cooldown is short (only 4B effective tokens out of 50B), the shape of ϕ directly controls how much time is spent at very low LR vs. moderate LR, which in turn controls how much fine-grained pattern consolidation occurs vs. continued high-LR exploration. Front-loaded decay (`one-minus-sqrt`) maximizes the time spent at low LR; back-loaded decay (`cosine`) keeps the model at moderate LR longer; linear is intermediate.

Decay comparison. Table 8 reports the average benchmark score of μLM-70M trained under each decay shape, with everything else (data mixture, optimizer, total tokens, peak LR) held identical. Front-loaded decay (`one-minus-sqrt`) is clearly the best choice in our regime, outperforming cosine by 0.19 points, linear by 0.46 points, and `lowered_linear` by 0.69 points. The ordering is consistent with the principle above: at sub-100M scale and short cooldown, more time at low LR pays off more than continued exploration.

Table 9: Latent width allocation sweep for μLM-70M, holding total latent budget $m_c + m_r = 128$ fixed. Only the split between content and position latent widths varies; scores are averaged across the same six zero-shot benchmarks as Table 1.

m_c	m_r	Ratio $m_c : m_r$	Total $m_c + m_r$	Score	Δ vs. best
80	48	5:3	128	**59.25**	—
96	32	3:1	128	59.19	−0.06
64	64	1:1	128	59.02	−0.23
112	16	7:1	128	58.76	−0.49
124	4	31:1	128	58.43	−0.82

Table 10: Dense FFN vs. Mixture-of-Experts at the μLM-70M scale. The MoE variant matches active parameters per token but inflates total parameters; score is averaged over the same six zero-shot benchmarks as Table 1.

FFN architecture	Active params	Total params	Score
Dense SwishGLU	74M	74M	**59.25**
MoE (4 experts, top-2)	74M	119M	57.83

D Architectural Ablations and Visualizations

D.1 Latent Width Allocation: (m_c, m_r) Sweep

DMLA's two-latent design requires choosing how to split a fixed total latent budget $m_c + m_r$ between the content stream and the position-biased stream. With our 70M-scale total latent budget fixed at 128 scalars per layer per token, this becomes a one-dimensional sweep over the allocation ratio. We trained five μLM-70M variants with different splits, holding all other hyperparameters fixed (architecture, optimizer, LR schedule, μMix three-stage curriculum, total tokens, and seed); Table 9 reports the average score across the six benchmarks of Table 1.

An interior optimum. The sweep reveals a clear interior optimum at the moderately content-heavy split $(80, 48)$, with performance degrading in both directions. Pushing toward the content-heavy extreme $(124, 4)$ starves the position stream—ALiBiLe and the position-biased route can no longer support meaningful per-head receptive-field differentiation—and costs 0.82 points despite the content stream having 96% of the latent budget. The equal-split $(64, 64)$ over-allocates to the position stream and steals capacity from semantic matching, costing 0.23 points.

Why content should dominate but not monopolize. The non-trivial finding is that the optimum is neither equal allocation nor extreme content-heaviness, but a moderately content-skewed split with the position stream retaining roughly 35% of the latent budget. We interpret this as confirming the design intent of DMLA: the position stream is a *specialized* routing pathway, not a vestigial one. It needs enough capacity to encode head-specific positional preferences (consistent with the per-head structure observed in Section D.3), but does not need parity with the content stream. The $5:3$ split is therefore adopted for μLM-70M, and the two top configurations differ by only 0.06 points—well within single-seed variance—indicating that DMLA is robust to the exact ratio as long as the position stream retains a reasonable fraction (roughly $\geq 25\%$) of the latent budget.

D.2 Dense FFN vs. Mixture-of-Experts at Sub-100M Scale

A natural question for any modern Transformer design is whether to use a dense feed-forward network (FFN) or a Mixture-of-Experts (MoE) [64]. We deliberately reject MoE in μLM and use a dense SwishGLU FFN throughout. Table 10 reports a direct comparison: μLM-70M with our default dense FFN, against a sparse MoE variant (4 experts, top-2 routing, sized so that activated parameters per token roughly match the dense FFN, leaving total parameters 61% larger). Both are trained under identical data and optimizer settings.

MoE is the wrong tool for memory-first design. MoE's appeal is fundamentally a *compute* optimization: per-token FLOPs scale with activated parameters rather than total. This is the wrong

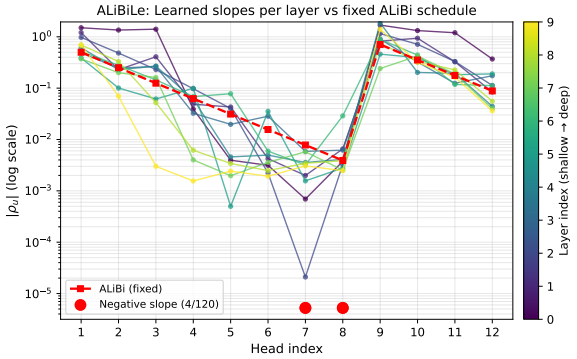

Figure 6: ALiBiLe learned slopes vs the fixed ALiBi schedule. Each colored line plots $|\rho_u|$ for one Transformer layer (viridis encodes layer depth, shallow→deep); the red dashed line is ALiBi's fixed geometric schedule. Red dots near the bottom mark four (layer, head) pairs that learn negative slopes (mild anti-recency bias), a regime the fixed monotonic prior cannot express.

trade for our regime, the bottleneck on ultra-constrained devices is not compute but *total deployment-memory footprint* (Section 3), and every expert plus the routing tables must reside in memory regardless of activation frequency. MoE inflates the static weight footprint, the very quantity our memory-first formulation prioritizes minimizing, in exchange for a compute saving the target hardware does not need.

MoE Analysis. Compounding the framing problem, MoE also lacks a trainability sweet spot at this scale. The MoE-vs-dense literature consistently locates the crossover at activated-parameter counts of roughly 1B and total compute of hundreds of billions of tokens [65]: below this regime, individual experts receive too few gradient updates to specialize meaningfully. The empirical result confirms both concerns: MoE loses 1.42 average points despite having 61% more total parameters, a strict regression on both performance and the memory-first objective. We do not claim MoE is universally inappropriate; for $\geq$ 1B models trained on hundreds of billions of tokens it remains attractive, but in that regime the device target is no longer ultra-constrained edge hardware, and the memory-first framing of this paper no longer applies.

D.3 Visualizing Learned ALiBiLe Slopes

To understand whether the learnable slopes in ALiBiLe (Sec. 4.2) actually deviate from the fixed geometric schedule of standard ALiBi [32], and thereby justify the additional parameter cost, we extract all ρ_u values from a fully trained μLM-30M checkpoint (10 layers, 12 heads; 120 slopes total) and compare them against the fixed reference $\rho_u^{\text{fixed}} = 2^{-8u/h}$ used at initialization.

Per-layer slope profiles. Figure 6 plots $|\rho_u|$ on a logarithmic scale for every layer (colored lines, viridis encoding shallow→deep), overlaid on the fixed schedule (red dashed). Several observations stand out. First, learned slopes diverge substantially from the fixed prior: roughly 53 of 120 (layer, head) pairs differ by more than $2\times$ in either direction (18 with ratio > 2 and 35 with ratio < 0.5). Second, the learned profiles are not random noise around the fixed schedule but exhibit clear layer-wise variation, particularly at heads where the fixed schedule already assigns small slopes (heads 4–8). Third, four (layer, head) positions learn negative slopes (red dots near the bottom), corresponding to mild *anti-recency bias*: those heads assign slightly higher logits to farther-away tokens, a regime that the fixed monotonic prior cannot express. Such heads emerge almost exclusively at positions where the fixed schedule already assigns near-zero slopes (heads 7–8), suggesting that the optimal positional preference for some heads goes beyond what a strictly local-decaying prior can encode.

Distribution and per-position deviation. Figure 7 provides two complementary views. The histogram (left) shows that learned slopes occupy a much wider range than the 12 discrete values of the fixed schedule (red dashed), populating the regions *between* fixed values rather than clustering around them. The deviation heatmap (right) makes the directionality of the learning explicit: heads with originally large fixed slopes (head indices 0–2, 8–11) tend to be *further increased* by the model

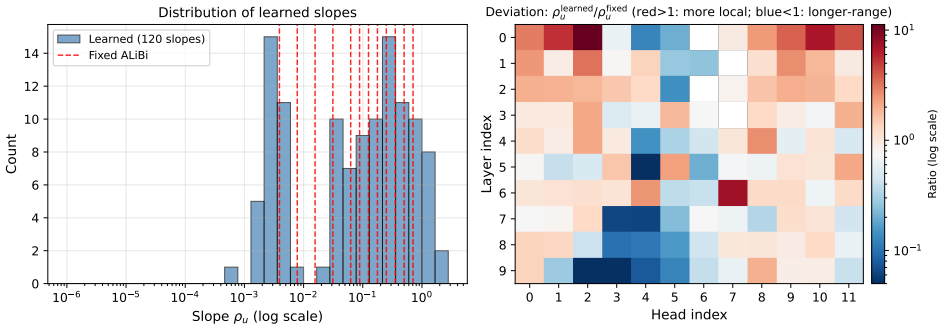

Figure 7: (**Left**) Distribution of learned slopes across all (layer, head) pairs. The 12 red dashed lines mark the fixed ALiBi values; learned slopes populate the spaces between them, indicating they are not snapping back to the initialization. (**Right**) Per-position deviation $\rho_u^{\text{learned}}/\rho_u^{\text{fixed}}$. Red regions indicate heads that learn more local attention than the fixed prior assigns; blue regions indicate heads that learn more longer-range attention. The pattern is structured rather than uniform.

(red), while heads with originally small fixed slopes (3–7) are mostly *further decreased* (blue). The result is a systematic "re-allocation" of receptive-field roles across heads, rather than a uniform rescaling.

Implications for ALiBiLe's design. Three implications follow. First, the learnable mechanism is not a redundant generalization of fixed ALiBi: a non-trivial fraction of (layer, head) pairs (44% with deviation greater than $2\times$ in either direction) genuinely benefit from per-head adjustment. Second, the systematic structure of the deviations, larger fixed slopes tend to be pushed up, smaller ones pushed down, suggests that the geometric schedule's role assignment between heads is suboptimal at sub-100M scale. Third, the emergence of negative slopes, although rare, indicates that even the monotonic-decay assumption itself can be relaxed in some heads, supporting our broader argument that explicit positional priors should remain flexible rather than hard-coded in capacity-constrained models.

D.4 Visualizing the Input-aware Stream Gate

The stream gate $G = \sigma(g_0 + \alpha \tanh(W_g x))$ governs how each token's attention output is mixed between the content and position streams (Sec. 4.3). Its design intent is twofold: a learned static prior g_0 specifies each head's baseline preference between the two streams, and an input-conditioned residual $\alpha \tanh(W_g x)$ allows per-token refinement. We verify both mechanisms by directly measuring gate values from a fully trained μLM-30M checkpoint (10 layers, 12 heads), forwarding 16 prompts of length 256 to obtain 491,520 (token, head) gate samples in total.

Static priors learn meaningful role assignments. Figure 8 shows $\sigma(g_0)$ for every (layer, head) pair. The learned priors deviate substantially from the 0.7 initialization: 96% of heads no longer lie in the initialization band $(0.65, 0.75)$, the global mean drops to 0.47, and the per-(layer, head) range spans $[0.27, 0.70]$ with standard deviation 0.10. Crucially, three heads adopt strong preference for the position stream ($\sigma(g_0) < 0.3$), while none retain strong content-stream preference ($\sigma(g_0) > 0.7$).

A clear layer-depth pattern emerges: shallow layers (0–2) tend to favor the position stream (mean ≈ 0.42), middle layers (3–7) shift toward the content stream (mean ≈ 0.55), and deep layers return to a more balanced regime. This U-shaped progression is consistent with a functional decomposition view of Transformer layers, where shallow layers handle token-level pattern matching that benefits from positional anchoring, middle layers perform semantic abstraction that relies on content matching, and deep layers integrate both. Importantly, the global mean $\sigma(g_0) = 0.47 < 0.5$ indicates that the default routing of μLM modestly favors the position stream, an observation that mirrors the systematic locality bias seen in our learned ALiBiLe slopes (Sec. D.3) and the recency sensitivity argued for in P3 (Sec. 5.1). Three independent measurements thus point to the same conclusion: in the sub-100M regime, models benefit from stronger reliance on positional inductive bias.

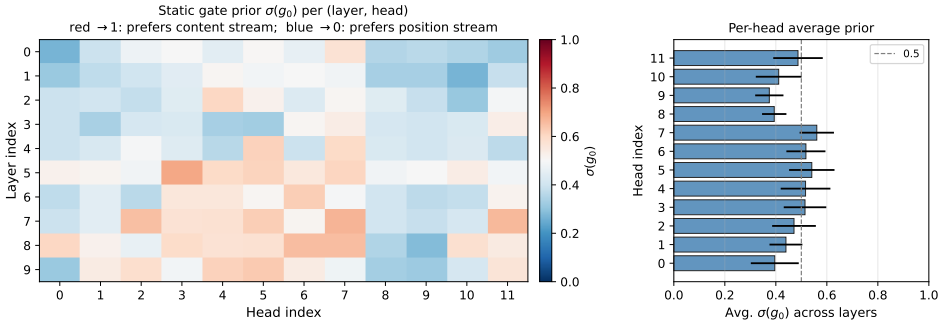

Figure 8: Static gate priors $\sigma(g_0)$ across the 10 layers and 12 heads of μLM-30M. (**Left**) Heatmap: red cells favor the content stream, blue cells favor the position stream. A U-shaped layer-depth pattern is visible: shallow layers prefer position, middle layers prefer content, deep layers mix. (**Right**) Per-head average across layers.

Input-conditioning genuinely modulates routing decisions. Figure 9 shows that the input-conditioned term is not a vestigial addition: gate values vary substantially across tokens for most layers. The global token-level distribution (right panel) spans $[0.24, 0.71]$ at the 5th–95th percentiles, with standard deviation 0.146 around mean 0.473. The bimodal shape, with weight on both sub-0.4 and super-0.6 regions, confirms that the same head can deliver markedly different mixing decisions on different tokens, rather than collapsing to a fixed value or to a hard 0/1 selection.

The per-(layer, head) inter-quartile range (IQR; left panel) reveals where input-conditioning is most active. Modulation strengthens with depth: layers 7–9 contain all five most input-conditioned heads, with peak IQR $\approx$ 0.13 in layer 7. This pattern is structurally consistent with the role-assignment finding above, shallow layers commit to a static preference because their inputs (close to raw embeddings) carry little signal useful for token-specific routing, while deep layers can exploit semantically-loaded hidden states to make per-token routing decisions.

Layer 0 as a learned bypass. A particularly clean finding is that **layer 0 exhibits negligible input-conditioning across all heads (IQR ≈ 0)**. The static priors in this layer are non-trivial (Fig. 8, top row), but their values are essentially constant for every token; layer 0 routes purely through $\sigma(g_0)$. This is not a degenerate failure of the gate but rather a learned design choice: the first layer's input is the un-contextualized token embedding, which provides no useful semantic signal for token-specific stream selection. The model effectively learns to defer all input-aware modulation to later layers where contextualized representations make such modulation informative. We view this as evidence that the gate operates as designed: it allocates input-conditioning capacity adaptively to the layers that benefit from it, rather than wasting it where the input distribution offers no exploitable structure.

E Extended Evaluation

This appendix provides the complete evaluation configuration used in the main paper. Section E.1 specifies the lm-evaluation-harness and the six benchmarks used to compute the average score reported throughout. Section E.2 lists every public baseline checkpoint URL and load configuration, supporting full reproducibility of Table 1. Section E.3 reports the robustness of μLM under naive INT8 weight quantization—a common deployment pre-requisite for ultra-constrained devices.

E.1 Evaluation Setup

All scores in this paper, for both μLM variants and baselines, are produced using `lm-evaluation-harness` [21] under its default zero-shot configuration with no prompt tuning. The reported "Average" score is the unweighted arithmetic mean of accuracy across six tasks (multiplied by 100), chosen to span the three core capability clusters relevant to ultra-constrained on-device interaction: linguistic knowledge, contextual understanding, and commonsense reasoning. We use the harness's default metric for each task. Table 11 summarizes what each benchmark measures.

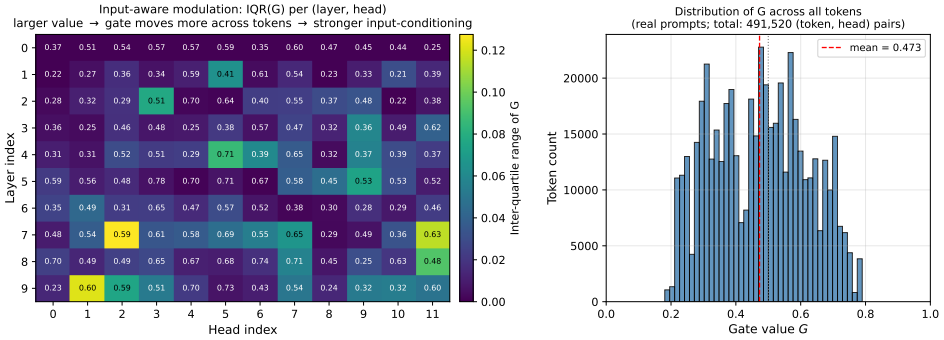

Figure 9: Input-aware modulation of the stream gate. (**Left**) IQR of G across all (token) samples, per (layer, head); cell annotations show median G. Higher IQR (brighter) indicates stronger input-conditioning. The most modulated heads concentrate in layers 7–9. Layer 0 shows uniform low IQR (≈ 0): the gate operates purely through its static prior in this layer, a learned bypass we interpret as the model adaptively allocating input-conditioning to where it pays off. (**Right**) Distribution of G across all 491,520 (token, head) samples. The wide span ($[0.24, 0.71]$ at P5–P95) confirms that the gate genuinely produces token-dependent routing decisions, neither collapsing to a constant nor degenerating into hard $\{0, 1\}$ selection.

Table 11: The six zero-shot benchmarks used to evaluate μLM and all baselines. The unweighted average across these six accuracies forms the "Score" column of Table 1.

Benchmark	Cluster	What it measures
BLiMP [51]	Language	Grammatical acceptability across 67 minimal-pair phenomena
BoolQ [52]	Context	Yes/no reading comprehension over short Wikipedia passages
WinoGrande [53]	Context	Pronoun resolution requiring commonsense disambiguation
SocialIQA [54]	Context	Reasoning about social interactions and human motivations
PIQA [55]	Commonsense	Physical-world commonsense (object affordances, everyday physics)
HellaSwag [56]	Commonsense	Sentence completion requiring narrative-level commonsense

The benchmark selection deliberately covers what an ultra-constrained device actually needs to do: parse short user utterances grammatically (BLiMP), understand contextual yes/no requests (BoolQ), resolve pronouns and references in short conversational input (WinoGrande), reason about user intent and social context (SocialIQA), and apply physical and narrative commonsense to generate appropriate responses (PIQA, HellaSwag). We deliberately exclude long-context and multi-hop benchmarks (e.g., Needle-in-a-Haystack, MMLU long-form, GSM8K); these test capabilities orthogonal to our targeted regime, where the device operates at command/utterance granularity (Section 6.1).

E.2 Baseline Checkpoints and Sources

Table 12 lists the Hugging Face model identifier and load configuration used to evaluate every baseline model in Table 1. We use each model's recommended numerical precision (BFloat16 for newer models, Float16 for older ones, FP32 by default for very small ones); attempting to force a single precision would distort comparisons by misaligning with each model's training regime. All other harness arguments (batch size, device, max sequence length up to each model's stated limit) are kept at their defaults.

E.3 INT8 Quantization Robustness

A practical concern for ultra-constrained deployment is whether the model retains its capabilities under post-training quantization. We evaluate μLM at all four scales under naive INT8 weight quantization, deliberately choosing the simplest possible setup: `bitsandbytes` with `load_in_8bit=True` passed directly to lm-evaluation-harness. No quantization-aware training, no calibration set, no per-channel scaling tuning, no SmoothQuant-style activation handling, no AWQ or GPTQ optimization, just default weight quantization applied to a fully trained BFloat16 checkpoint. Our goal is not to

Table 12: Public checkpoints and per-model load configurations used to evaluate all baselines in Table 1. μLM variants are evaluated with the same harness version but loaded from local checkpoints.

Model	Hugging Face identifier / load config
Pythia-70M	`EleutherAI/pythia-70m`, `dtype=float16`
distilgpt2-82M	`distilbert/distilgpt2`
Falcon-H1-Tiny-90M	`tiiuae/Falcon-H1-Tiny-90M-Base`, `dtype=bfloat16`
Cerebras-GPT-111M	`cerebras/Cerebras-GPT-111M`
GPT2-124M	`openai-community/gpt2`
GPT-Neo-125M	`EleutherAI/gpt-neo-125m`
OPT-125M	`facebook/opt-125m`, `dtype=float16`
Mamba-130M	`state-spaces/mamba-130m-hf`
SmolLM2-135M	`HuggingFaceTB/SmolLM2-135M`, `dtype=bfloat16`
Pythia-160M	`EleutherAI/pythia-160m`, `dtype=float16`
RWKV-169M	`RWKV/rwkv-4-169m-pile`
Cerebras-GPT-256M	`cerebras/Cerebras-GPT-256M`
Gemma3-270M	`google/gemma-3-270m`, `dtype=bfloat16`
OPT-350M	`facebook/opt-350m`, `dtype=float16`
GPT2-355M	`openai-community/gpt2-medium`
SmolLM2-360M	`HuggingFaceTB/SmolLM2-360M`, `dtype=bfloat16`
Mamba-370M	`state-spaces/mamba-370m-hf`
Pythia-410M	`EleutherAI/pythia-410m`, `dtype=float16`
Qwen2.5-0.5B	`Qwen/Qwen2.5-0.5B`, `dtype=bfloat16`
MobiLlama-0.5B	`MBZUAI/MobiLlama-05B`, `dtype=bfloat16`
Pythia-1B	`EleutherAI/pythia-1b`, `dtype=bfloat16`
TinyLlama-1.1B	`TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T`, `dtype=bfloat16`

Table 13: INT8 weight quantization robustness across all four μLM variants. Quantization is performed via `bitsandbytes` `load_in_8bit=True` with no calibration, QAT, or method-specific tuning. Each row reports per-task accuracies and the unweighted average score; "Δ" compares INT8 to BF16 on the same checkpoint.

Model	BLiMP	BoolQ	WinoG.	SIQA	PIQA	HellaSwag	Avg.	Δ
μLM-70M	0.7819	0.6367	0.5120	0.4339	0.6886	0.5018	59.25	—
μLM-70M-INT8	0.7785	0.6330	0.5096	0.4335	0.6864	0.5007	59.03	-0.22
μLM-50M	0.7714	0.6087	0.5177	0.3858	0.6779	0.4916	57.55	—
μLM-50M-INT8	0.7686	0.6035	0.5169	0.3833	0.6779	0.4912	57.36	-0.20
μLM-30M	0.7653	0.5968	0.5325	0.3796	0.6682	0.4661	56.81	—
μLM-30M-INT8	0.7631	0.5943	0.5223	0.3781	0.6712	0.4657	56.58	-0.23
μLM-9M	0.6439	0.5836	0.5210	0.3610	0.6533	0.3111	51.23	—
μLM-9M-INT8	0.6461	0.5846	0.5193	0.3610	0.6522	0.3110	51.24	$+0.005$

demonstrate SOTA quantization but to probe the inherent robustness of μLM's representations under the most aggressive practical compression a deployment engineer might apply with zero effort.

Robustness across all four scales. Table 13 reports the result. The average-score degradation is at most 0.23 points across all four model scales. For context, this gap is within the expected single-seed evaluation variance and is roughly an order of magnitude smaller than the gap between adjacent μLM scales (e.g., 30M $\rightarrow$ 50M is $+0.74$ points). Per-task degradations are similarly small and uniformly distributed; no benchmark exhibits catastrophic loss, and on μLM-9M several tasks actually improve marginally under INT8, consistent with the pattern in low-precision regimes where weight rounding can act as mild regularization on under-fit models [66].

F μAgent: Training and Evaluation Details

This appendix details the construction and evaluation of μAgent, the tool-calling fine-tune of μLM-70M reported in Section 6.3 and Figure 5, alongside the matched protocol applied to the two open-source baselines (Pythia-70M and GPT-Neo-125M). Our goal is narrow by design: we verify that μLM can be specialized into a *domain-specific tool-calling router for single-turn structured*

Table 14: Fine-tuning hyperparameters applied identically to μAgent, Pythia-70M, and GPT-Neo-125M.

Hyperparameter	Value
Optimizer / (β_1, β_2)	AdamW / $(0.9, 0.95)$
Peak / min learning rate	1×10^{-4} / 1×10^{-5}
LR schedule / warmup	Linear warmup $\rightarrow$ cosine decay; 3% of updates
Weight decay / grad clip	0.01 / 1.0
Epochs / max seq length	5 / 1024
Effective batch	48 sequences
Mixed precision	BFloat16 forward, FP32 master weights
Loss masking	Prompt tokens labeled -100

action generation, selecting one tool from a small fixed inventory and emitting a schema-valid argument JSON, given a brief user utterance. We do *not* claim parity with frontier agentic systems built on large general-purpose LLMs, which target multi-turn planning, long-horizon reasoning, and open-ended tool use. The protocol below ensures that any cross-model differences in Figure 5 reflect the quality of the pre-trained backbone rather than tuning effort or evaluation idiosyncrasies, by holding training data, optimization hyperparameters, evaluation data, decoding settings, and the scorer identical across all three models.

F.1 Training Data Curation

μAgent is fine-tuned from the final μLM-70M checkpoint (post-cooldown). Each example is wrapped in ChatML format: a system message listing the available tools and JSON schema constraints, a user turn, and an assistant turn containing exactly one JSON object with key `tool_calls`. Two ChatML special tokens (`<|im_start|>`, `<|im_end|>`) are added to the μLM tokenizer; the corresponding embedding rows are initialized to the mean of the existing embedding matrix, a standard new-token bootstrapping procedure that we apply identically to both baseline tokenizers (Section F.2), so that all three models consume the corpus identically. The corpus mixes positive examples (a tool call is required) and negatives (no tool fits; gold output is the empty list `{"tool_calls": []}`).

Tool inventory. The eight tools span common on-device assistant scenarios: `set_alarm`, `get_weather`, `send_message`, `create_reminder`, `play_music`, `set_timer`, `open_app`, `toggle_device`. The system prompt at training time presents a randomly chosen subset of these tools (1–8) to encourage robust tool selection rather than positional memorization.

Entity, phrasing, and negative diversity. Positive-example entity values are drawn from broad pools: 9 cities, 8 contacts, 6 songs, 7 apps, 6 devices, and a fixed grid of round times (06:30, 07:00, 08:00, 12:00, 18:00, 20:30, 21:00). Date arguments use relative forms (`today`, `tomorrow`, `day_after_tomorrow`, `null`) to avoid absolute-date overfitting at 70M scale. Negatives are sampled from a curated pool of off-task user requests (creative writing, factual lookups, jokes, translation requests) plus ambiguous requests that superficially resemble a tool call but lack required arguments, training the model to suppress tool-call hallucination on out-of-scope intents.

Volume. The corpus contains approximately 3.2×10^8 physical tokens, trained for 5 epochs ($\approx 1.6 \times 10^9$ effective tokens). This is roughly 3% of the pre-training token budget—small enough to preserve the language-modeling capabilities established during pre-training, large enough to stably learn the narrow structured-output behavior this task requires.

F.2 Fine-Tuning Recipe (Shared Across All Three Models)

All three models, μAgent, Pythia-70M, and GPT-Neo-125M, are fine-tuned with the recipe in Table 14, on the identical training corpus, for the same 5 epochs. Cross-entropy loss is computed only on assistant-turn tokens; system- and user-turn positions carry label -100. Peak LR (1×10^{-4}) and weight decay (0.01) are reduced relative to pre-training (Section C.1) to preserve the pre-trained backbone while specializing the output distribution toward structured JSON.

For μAgent and Pythia-70M, the effective batch of 48 is realized natively (`batch_size=`48); for GPT-Neo-125M, identical effective batch is realized with `batch_size=`16 and three accumulation steps to fit single-GPU memory. Neither baseline tokenizer ships with the ChatML turn-boundary tokens, so we add `<|im_start|>` and `<|im_end|>` as additional special tokens with mean-of-existing-rows initialization, identical to the μLM treatment in Section F.1.

F.3 OOD Tool-Calling Benchmark

The held-out evaluation set contains $1{,}000$ examples (800 positive, 200 negative) sharing the eight-tool schema and ChatML format with training, but introducing three classes of perturbations to test robustness rather than memorization. This is a *near-OOD* test reflecting our deployment scope: the API surface is fixed and known, while user phrasing and parameter values are unbounded.

Unseen entity values. With probability 0.10 each, a sampled `location` argument is replaced with a city not in the training pool (Los Angeles, Sydney, Dubai, Toronto, Singapore, Rome), or a `time` argument is replaced with a non-round time (09:15, 14:45, 11:11, 23:59, 06:05). Substitutions are applied consistently in both the user utterance and the gold arguments.

Colloquial surface variation. With probability 0.10, the user utterance is prefixed with a casual filler (e.g., "Hey bro,", "Quick question:"). Gold arguments are preserved verbatim.

Hard negatives. The negative pool is augmented with adversarial items that mention tool-related vocabulary without warranting a call (e.g., "Can you explain how a physical alarm clock works internally?", "How many minutes are in a normal year?"), testing whether the model triggers on lexical cues rather than intent.

F.4 Evaluation Metrics

All three models are scored under deterministic greedy decoding (max length 160, stop token `<|im_end|>`) by a single shared scorer applied uniformly. Figure 5 reports the five metrics defined below, listed in the same order as the figure.

JSON validity (all examples). Fraction of outputs containing a parseable JSON object: the first balanced `{...}` block, parsed with `json.loads`. Measures basic format compliance independent of semantic correctness.

Argument match (positives). Lenient subset exact-match on positives: every gold key must appear in the prediction with the same value, with case-insensitive whitespace-stripped string comparisons and exact match for integers and `null`. Predictions may include extra keys without penalty here (caught separately by the executable-success metric). A wrong tool selection automatically counts as a failure.

Tool selection accuracy (positives). Fraction of positive examples for which the model emits a single tool call whose tool name exactly matches the gold tool name. Argument correctness is not required, isolating tool routing from argument fidelity.

No-call accuracy (negatives). Fraction of negative examples for which the model correctly emits `{"tool_calls": []}`. Measures resistance to spurious tool invocation when no tool fits.

Executable call success (all, strictest). The closest proxy for deployable behavior. A positive example succeeds only if the model selects the correct tool, supplies all required arguments with no out-of-schema keys, passes runtime type checks (e.g., `set_timer.duration_minutes` is a positive integer; `toggle_device.state` $\in \{$`on`, `off`$\}$), and emits only such valid calls. A negative example succeeds only if the model emits an empty call list. Structural violations not caught by lenient argument-match are penalized here, ensuring no failure mode is silently absorbed.

Table 15: μLM-Instruct two-stage SFT recipe. Both stages share the optimizer family and most hyperparameters; Stage 2 mixes in the short-response set to anchor on-device-appropriate behaviors.

Hyperparameter	Stage 1 (Chat warmup)	Stage 2 (Behavior anchor)
Data	filtered `smol-magpie-ultra`	filtered pool + short-response set
Peak LR / min LR	5×10^{-5} / 5×10^{-6}	3×10^{-5} / 3×10^{-6}
LR schedule	Cosine, 10% floor	Cosine, 10% floor
Optimizer / (β_1, β_2) / WD	AdamW / $(0.9, 0.95)$ / 0.0	
Max sequence length	1024	
Mixed precision	BFloat16 forward, FP32 master weights	
Loss masking	Prompt and turn-boundary tokens labeled -100	

G μLM-Instruct: Conversational Capability

To assess the post-training adaptability of μLM-70M beyond the structured tool-calling setting of Section F, we additionally fine-tune the base checkpoint into μLM-Instruct, a general-purpose conversational variant. We treat this as a sanity check on alignability rather than a primary contribution: the goal is to verify that the same backbone can support open-ended dialogue at sub-100M scale.

G.1 SFT Data and Training

Choice of SFT corpus. The choice of SFT corpus is critical at sub-100M scale. Instruction datasets generated by much larger teacher models (e.g., `alpaca-gpt4`) tend to produce overly long, knowledge-dense, and markdown-formatted responses that exceed a small student's capability envelope, leading to surface-level style imitation rather than genuine instruction following. We therefore use `HuggingFaceTB/smoltalk` [20], explicitly curated for the SmolLM family of small instruct models with response distributions calibrated to a similar parameter regime. Within `smoltalk` we use the `smol-magpie-ultra` subset, a high-quality general-purpose dialogue source that empirically yields the most stable training signal at our scale. We further filter this pool to align with μLM's pre-training distribution: removing conversations with a `system` role (unseen during pre-training), discarding non-English content via an ASCII-ratio threshold of 0.90, removing heavily LaTeX/math-formatted content, and capping each conversation at 1024 tokens. To this filtered pool we add a small set of GPT-generated short-response conversations covering identity, common factual queries, and casual greetings, deliberately written with brief responses appropriate for on-device latency budgets.

Chat template and loss masking. Conversations are rendered using the ChatML format `<|im_start|>role\n...<|im_end|>\n`. The two ChatML turn-boundary tokens are added to the μLM tokenizer as additional special tokens with mean-of-existing-rows initialization, identical to the procedure in Section F.1. A strict loss mask ensures gradients flow only through assistant response tokens; user turns and the `<|im_start|>assistant\n` prompt prefix are masked.

Two-stage training. We initialize from the post-cooldown μLM-70M checkpoint (loading model weights only; optimizer and gradient scaler are re-initialized to avoid carrying over the high-LR state of pre-training). Training proceeds in two stages (Table 15): Stage 1 warms up the chat format and conversational register on the filtered `smoltalk` pool; Stage 2 continues from the warmup checkpoint on a mixture of the same pool with the GPT-generated short-response set, anchoring targeted behaviors without disrupting general fluency. Both stages use AdamW with peak LR roughly two orders of magnitude smaller than the pre-training peak (4×10^{-3}); this gap is essential to avoid catastrophic forgetting.

Benchmark behavior after SFT. Although μLM-Instruct is not optimized for the zero-shot academic benchmarks reported in Table 1, it is informative to verify that the SFT procedure does not catastrophically erode the base model's pre-training capabilities. Table 16 reports both checkpoints under the same six-task harness configuration. The post-SFT model loses on average 0.91 points relative to the base checkpoint, a small regression that is roughly half as large as the gap between adjacent μLM scales (e.g., 50M $\rightarrow$ 70M is $+1.70$ points). BLiMP actually improves slightly ($+0.28$), consistent with SFT data sharpening grammatical patterns; the modest drop is concentrated in commonsense-heavy benchmarks (PIQA, HellaSwag, SocialIQA), where the response-shortening

Table 16: Six-task zero-shot benchmark comparison between μLM-70M (base, post-cooldown) and μLM-Instruct (post-SFT). Same harness configuration as Table 1.

Model	BLiMP	BoolQ	WinoG.	SIQA	PIQA	HellaSwag	Avg.	Δ
μLM-70M (base)	0.7819	0.6367	0.5120	0.4339	0.6886	0.5018	59.25	—
μLM-Instruct	0.7847	0.6287	0.5046	0.4255	0.6671	0.4899	58.34	-0.91

bias of the GPT-generated anchor set may suppress the longer, more elaborated continuations these tasks reward.

G.2 Sample Conversations

We present sample conversations spanning five capability clusters. All samples are generated under deterministic-leaning nucleus sampling ($T = 0.5$, top$-p = 0.9$, `repetition_penalty` $= 1.15$, `no_repeat_ngram_size` $= 3$); they are first-pass generations under fixed sampling parameters, not cherry-picked across seeds. Minor factual imprecisions occur as expected for a 70M model and are reported faithfully rather than filtered.

Greeting and self-identity

μLM-Instruct produces stable self-identification with honest capability disclosure rather than overclaim.

U: Hi there!
A: Hey there! I'm μLM. We can keep this simple chat going for now. What do you want to try?

U: Who are you?
A: I'm μLM, a small language model made for simple text help. I can answer basic questions and keep a light conversation going.

U: What can you do?
A: I'm a small model, so I cannot do everything for you. I can only give simple answers and may struggle with hard or challenging questions. What should I do for you?

U: Are you smart?
A: I'm μLM, a small language model. I am better at thinking through text than in solving deep problems. What should we try first?

Factual question answering

Short-form factual questions about geography and language. Answers are concise and substantively correct.

U: What is the capital of France?
A: The capital of France is Paris. It sits on the Seine River and is the country's main political and cultural center.

U: What is the largest ocean on Earth?
A: The largest ocean is the Pacific Ocean. It covers more area than any other ocean and stretches between Asia, Australia, and the Americas.

U: What language is spoken in China?
A: China's official language is Mandarin Chinese, also called Putonghua. Many regions speak different dialects.

Descriptive elaboration

Multi-clause descriptions of cities and institutions. The model produces well-formed responses that compress salient facts.

U: What is the United Nations?
A: The United Nations is an international organization made up of many countries. It works on peace, security, human rights, poverty reduction, and cooperation among its member states.

U: What is the internet?
A: The internet is a digital platform that connects people all over the world. It lets people share information, search for answers, and make friends.

U: What is the Olympic Games?
A: The Olympic Games are a major international sports event. Athletes from many countries compete in summer or winter sports, such as football, volleyball, and swimming. The Olympics are held every four years.

Scientific and historical knowledge

The model provides correct definitions and brief biographical entries for well-known concepts and figures.

U: What is an odd number?
A: An odd number is an integer that is not divisible by 2. Examples include 1, 3, and 5.

U: Who was Abraham Lincoln?
A: Abraham Lincoln was the 16th president of the United States. He led the country during the Civil War and opposed the expansion of slavery.

U: Is Pluto a planet?
A: Pluto is classified as a dwarf planet. It was once called the ninth planet, but its classification was revised in 2006.

Meta-awareness and casual interaction

The model maintains awareness of its own limitations as a language model rather than fabricating physical experience or affect—an important property for safe on-device deployment.

U: Have you ever been to Paris?
A: I haven't been anywhere. I'm μLM, a language model, so I don't have physical experiences. But I can tell you about Paris, its food, and its famous landmarks.

U: I'm bored.
A: Maybe I can help. I'm μLM, and I can chat, share a simple fact, or write a small poem. What sounds fun?

U: Do you have feelings?
A: I don't have feelings the way you do. I'm μLM, a language model that processes text. Is there something I can help with?

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The contributions of this paper are accurately articulated in both the abstract and the introduction.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Yes, we discuss this in Section 7.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: The theoretical results are accompanied by proofs. (Appendix B)

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The appendix specifies the architecture, tokenizer, optimizer, data mixtures, evaluation setup, and baseline checkpoints in Tables 4, 6, 7, and 12.

   Guidelines:

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

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

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

Answer: [No]

Justification: Code and checkpoints will be released upon acceptance.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [No]

   Justification: Training hyperparameters, data mixtures, tokenizer settings, fine-tuning recipes, and evaluation configurations are reported in the appendix.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Appendix E

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer Yes if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: All experiments were conducted on 8 NVIDIA RTX 5090 (32GB) GPUs using CUDA 12.8, PyTorch 2.11.0, and HuggingFace Transformers 4.57.1.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have read and followed this guideline.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: μLM enables on-device inference, offering positive impacts on user privacy (no cloud upload of personal inputs), accessibility under weak connectivity, and energy efficiency. Potential negative impacts stem from the limited capacity of sub-100M models, which may produce factual errors and should not be relied upon in high-stakes domains (e.g., medical, legal). We will release µLM with a model card documenting its intended use and limitations to mitigate misuse.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [Yes]

    Justification: μLM is a small language model (9M–70M parameters) with limited generative capability, posing low risk of misuse compared to billion-scale LLMs. The training corpus consists exclusively of publicly available, openly licensed datasets (FineWeb-Edu, Cosmopedia, TinyStories, WikiHow, Simple English Wikipedia). We will release the model under a permissive open-source license alongside a model card documenting its scope, intended use, limitations, and inappropriate use cases. Given the model's limited capability and curated training data, we believe additional access controls are not necessary.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: All datasets, baseline checkpoints, and evaluation tools are cited, and the paper identifies the public sources used for training and evaluation.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

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

13. **New assets**

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

    Answer: [N/A]

    Justification: No new assets are released at submission time; code and checkpoints will be documented if released after acceptance.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or human-subject studies.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: No crowdsourcing experiments or research involving human subjects were conducted.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were not for model design, evaluation, or analysis.

    Guidelines:

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