The Extender: A Log-Structured Transformer

Anonymous authors
Paper under double-blind review

Abstract

We introduce the Extender, a log-structured variant of the standard Transformer architecture. In a standard Transformer, each layer communicates with subsequent layers exclusively via the residual **h**, a superposition channel. The Extender adds a concatenation channel **x**: each layer ℓ emits both a residual update δ_ℓ which is added to **h**, and a much smaller *extension* ϵ_ℓ which is appended to **x**. While both the FFN and **q** see **h**, the attention **kv** projections take only **x** as input. As a result, the fully *extended* **x** contains the complete input for the **kv** projections of all layers, reducing the persistent attention memory footprint from $2Ld_{model}$ to $\sum|\epsilon_\ell|$. We find that with $|\epsilon_\ell| = 32$, the Extender matches Transformer accuracy on short-context (CORE) tasks at 199M-924M parameters, and exceeds Transformer accuracy on long-context (RULER) workloads, again at 924M parameters. For our 1664-wide, 924M model, the Extender's persistent attention memory footprint is 104× smaller than MHA. The memory savings grow with model width.

1 Introduction

The residual stream is fundamental to the standard Transformer model: it is the sole input to every layer; the sole output of each layer is an update to the residual stream. Arguably, its most important function is that of mitigating vanishing gradients in deep models: each individual layer sees a direct gradient from the final loss computation, enabling stable optimization. In other words, the loss directly teaches each layer how to predict the next token.

However, this design is not without drawbacks. While each layer may in practice only make small changes, the residual update from each layer is full width, making each feature of the input to the subsequent attention layer unique. During inference, this requires each attention layer to maintain its own, full-width residual (**h**) or key-value (**kv**) cache to avoid a very costly recomputation. The memory footprint of these caches grows linearly with model width d_{model}, depth L and context length T. For a large language model, the aggregate **kv**-cache alone can exceed 100 GB before dimensionality reduction, layer reuse, quantization and other space-saving optimizations.

Moreover, beyond predicting the next token, each layer ℓ may also need to communicate its own findings, say ϕ_ℓ to subsequent layers. Here, ϕ_ℓ may not look anything like the next token; for example, it may encode a distance from another token, or a repetition counter, yet this too must be communicated through the residual, potentially in contention with other writes, and against the teachings of the loss.

The *Extender* is a log-structured variation on the classic Transformer architecture that separates the next-token prediction stream from the attention input. Similar to the Transformer, the Extender is made up of a stack of Extender layers, each consisting of several individual sub-layers, but dominated by softmax attention and FFN. However, instead of giving the attention sub-layers full visibility into the output of the previous layer, the Extender provides two *distinct* channels:

h The conventional residual, used primarily for predicting the next token. Each layer contributes a full-width *residual update* δ_ℓ, such that $\mathbf{h}_\ell = \mathbf{h}_{\ell-1} + w_\ell\delta_\ell$, where w_ℓ is a scalar weight. $\mathbf{h}_\ell$ is part of the input to the *FFN sub-layer* of the next block, as well as to attention (**q** only, not **kv**), and the final logit projection. $\mathbf{h}_{-1} = [embed(t)]$.

x Each layer ℓ contributes information to subsequent attention **kv** and **q** projections via a small, read-only *feature extension* ϵ_ℓ to a log-structured token embedding. The extended embedding after layer ℓ is $\mathbf{x}_\ell = [\mathbf{x}_{\ell-1}; RMSNorm(\epsilon_\ell)]$ where $\mathbf{x}_{-1} = [embed(t)]$.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We use $\mathbf{x}_*$ to refer to the *fully extended* embedding after the token has traversed all L layers. As the $*$ in the notation implies, the append-only nature of $\mathbf{x}$ in the Extender architecture means that $\mathbf{x}_*$ contains every $x_\ell, \ell \in [0..L\text{-}1]$ as a prefix. Therefore, while every layer sees a different $\mathbf{x}_\ell$, a single shared $\mathbf{x}_*$-cache suffices across all attention sub-layers.

Thus while the Multi-Head Attention (MHA) Vaswani et al. (2017) Transformer requires **kv**-cache memory proportional to $2TLd_{\text{model}}$ (two d_{model}-wide vectors per layer, per token), the Extender instead requires $\mathbf{x}_*$-cache memory TLd_ϵ (one d_ϵ-wide extension per layer, per token), reducing the persistent attention memory footprint at long context lengths by up to $\frac{2d_{model}}{d_\epsilon}\times$ vs. MHA. The model sizes we tested, 198M/436M/920M, with d_ϵ=32, the Extender matches our Reference Transformer on short-context tasks ($\sim$ +1 on CORE), and leads significantly on long-context (RULER) tasks.

To compute softmax attention without a **kv**-cache, $\mathbf{x}_*$-cache based attention would have to repeatedly re-materialize **kv** from $\mathbf{x}_*$, potentially resulting in significant computational overhead. To resolve this, the Extender features an *ephemeral* **kv**-cache at inference time. The *ephemeral* **kv**-cache of a given layer may be materialized on-demand from the $\mathbf{x}_*$-cache by multiplication with the relevant attention W_k and W_v matrices. Amortized over multiple input and output tokens, this **kv** materialization cost is small. Thus the *ephemeral* **kv**-cache may be released between turns, as needed.

Because of the large attention memory footprint of MHA, it is rarely used in its pure form in large models. Instead, models that retain softmax attention use techniques such as dimensionality reduction Shazeer (2019); Ainslie et al. (2023); DeepSeek-AI (2024), layer reuse Liu et al. (2024a); Sun et al. (2024); Wu & Tu (2024); Brandon et al. (2024); Zuhri et al. (2024) and quantization Xiao et al. (2023); Frantar et al. (2023); Liu et al. (2024b) to limit the footprint. The Extender instead changes the architecture, separating the per-layer next-token prediction δ_ℓ from the per-token features ϵ_ℓ that persist in attention memory. The result is a compact attention memory footprint between turns, while achieving similar accuracy on short-context and better accuracy on long-context tasks. During turns, memory requirements are unchanged vs a standard Transformer. However, existing dimensionality reduction, layer reuse and quantization methods may still be applied to attention during turns, if desired. Our experiments with GQA (See Appendix D) suggest that Grouped Query Attention combines well with the Extender.

The remainder of the paper is organized as follows. §2 briefly introduces past work on reducing attention memory footprint. §3 presents the Extender in more detail, and §4 discusses inference considerations. §5 compares Extender accuracy vs. a Reference Transformer on short-context (DCLM CORE) and long-context (RULER) tasks, and §6 concludes. Additional results and analysis, including GQA, knowledge distillation and scaling results are available in the Appendix.

2 Background

Because the attention memory footprint of a standard softmax multi-head attention Transformer grows linearly with model width d_{model}, layer depth L and context length T, a variety of memory-reduction approaches have been proposed. The Extender reduces *persistent* attention memory: the memory required between turns, while remaining compatible with existing approaches during a turn. That said, because a method that applies during a turn applies equally between turns, we briefly review the related work below.

Dimensionality Reduction With grouped query attention (GQA) Ainslie et al. (2023), an attention layer with H query heads may maintain only $H/4$ or $H/8$ keys and values per token. This in turn reduces attention memory requirements by 4–8×, at a modest accuracy loss vs. full MHA. Multi-head Latent Attention (MLA) DeepSeek-AI (2024) instead stores a compressed version of the **kv** pair, as much as 10× smaller. This is decompressed on-demand before attention. As a result, MLA does not need to reduce the number of **kv** heads, but may suffer compression losses instead.

The Extender instead changes what the attention layer reads. However, GQA can be applied to the Extender to reduce memory footprint during turns (see Appendix D).

Layer Reuse YOCO and LCKV move toward one shared cache Sun et al. (2024); Wu & Tu (2024): YOCO reuses a single global KV cache in its cross-decoder. LCKV has most layers use the same first layer KV cache. Cross-Layer Attention (CLA) and MLKV instead share one cache among a

group of layers Brandon et al. (2024); Zuhri et al. (2024). MiniCache instead merges similar layers' KV states after training Liu et al. (2024a).

The Extender significantly reuses attention input data, but does not change the attention mechanism. If desired, CLA may be applied to the Extender to save attention memory during turns. We do not evaluate this here.

Quantization With quantization Xiao et al. (2023); Frantar et al. (2023); Liu et al. (2024b), individual features or weights are represented using fewer bits, leading to both reduced memory footprint and often improved computational throughput. Applied judiciously, the accuracy loss from quantization can be modest, and savings on the order of 2–4× are not uncommon. We do not evaluate quantization in this paper.

2.1 Approaches beyond softmax attention

Softmax attention incurs a cost that is linear in the context length and number of layers, per token. Several alternative approaches Katharopoulos et al. (2020); Sun et al. (2023); Gu & Dao (2023); Peng et al. (2023; 2024); Goldstein et al. (2024) have been proposed which reduce or eliminate the need to revisit each preceding token during attention. The Extender preserves softmax attention, but changes the input to the attention **kv** projections, to achieve a reduction in memory footprint between turns.

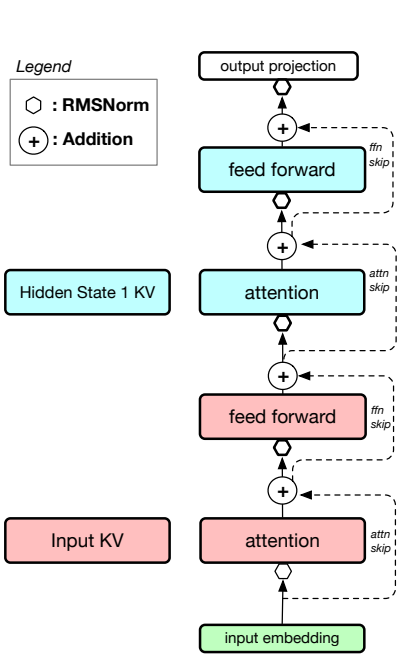
Step	Explanation and Motivation
$\mathbf{x}_{-1} = \mathbf{h}_{-1} = [\text{embed}(\text{token})]$	Both $\mathbf{x}_{-1}$ and $\mathbf{h}_{-1}$ start out as a token embedding, size d_{model}. $embed$ weights tied to $unembed$ below.
$\mathbf{s}_\ell = \text{Last}(\mathbf{x}_{\ell-1}, d_{model})$	Size d_{model} sliding window of $\mathbf{x}_{\ell\text{-}1}$. $\mathbf{s}_0 = \mathbf{x}_{\text{-}1}$.
$\mathbf{a}_\ell = \text{ATTN}(\mathbf{s}_\ell, \mathbf{h})$	The ATTN **kv** projections see only $\mathbf{s}_\ell$, but the **q** projection also sees $\mathbf{h}_\ell$.
$\mathbf{y}_\ell = \mathbf{a}_\ell + \mathbf{h}_{\ell-1}$	$\mathbf{y}_\ell$ is the input to the FFN
$\hat{\delta}_\ell, \epsilon_\ell = \text{FFN}(\text{RMSNorm}(\mathbf{y}_\ell))$	The FFN outputs both the residual update δ_ℓ, and the extension ϵ_ℓ.
$\delta_\ell = \hat{\delta}_\ell + \mathbf{a}_\ell$	$\mathbf{a}_\ell$ allows ATTN to write directly to the residual stream, and provides ATTN a clean gradient.
$\mathbf{x}_\ell = [\mathbf{x}_{\ell-1}; \text{RMSNorm}(\epsilon_\ell)]$	Every layer appends its own read-only ϵ_ℓ to $\mathbf{x}$.
$\mathbf{h}_\ell = \mathbf{h}_{\ell-1} + w_\ell \delta_\ell$	Weighted residual stream update. The learned layer weight w_ℓ tends to give more weight to later layers.
$\text{logits} = \text{unembed}(\text{RMSNorm}(\mathbf{h}_{L-1}))$	Logits from the final hidden state. $unembed$ weights tied to $embed$ above.

Table 1: A compact description of the Extender. The extended embedding $\mathbf{x}$ corresponds to the left part of Figure 1b, while the residual stream $\mathbf{h}$ appears on the right.

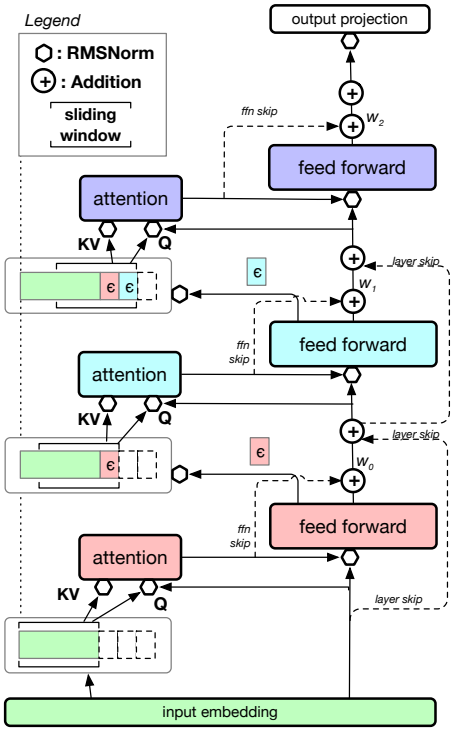
3 The Extender: A Log-Structured Transformer

Figure 1b shows a high-level view of the Extender architecture. Compared to the Transformer in 1a, the biggest difference lies in the separation of the extended embedding $\mathbf{x}$ (left) seen by the attention **kv** projections, and the residual stream $\mathbf{h}$ (right) which is seen only by the FFN sub-layers and the attention **q** projections. Key to the Extender are the *extensions* ϵ_ℓ and *residual updates* δ_ℓ produced by each layer. The extended embedding $\mathbf{x}_\ell$ is *log-structured*, updated by concatenating ϵ_ℓ to the end, rather than the addition used to update $\mathbf{h}$ by δ_ℓ. After $\mathbf{x}$ and $\mathbf{h}$ have passed every layer, logits are produced from $\mathbf{h}$ only. Thus the final layer produces no ϵ_ℓ.

Table 1 describes the model more formally, for $\ell \in 0..L\text{-}1$. Here, $unembed$ and $embed$ have tied weights. $\mathbf{s}_\ell$ is a size d_{model} window over $\mathbf{x}_{\ell-1}$. Shown here is a sliding window using a $Last()$ operator, which takes a slice of the most recently added d_{model} features out of the extended embedding. In §3.1 we explore alternatives to this design choice. Note that while every ϵ persists between turns, the d_{model}-wide initial embedding is fixed and does not have a persistent memory footprint beyond the existing embed table and a token identifier.



(a) 2-layer Reference Transformer: a conventional Llama-style Touvron et al. (2023) Transformer architecture used for comparative evaluation purposes. Each attention layer caches unique KV-pairs derived from the input to the attention layer.



(b) 3-Layer Extender. The extended embedding $\mathbf{x}$ grows through concatenation with the ϵ_ℓ generated by each layer. There are two layer outputs: the extension ϵ_ℓ, and δ_ℓ. Hexagons indicate RMSNorm.

Figure 1: Diagrams describing a conventional Transformer (left), and an Extender (right), using the same visual language and annotations. The Extender attention **kv** projections see the log-structured extended embedding $\mathbf{x}_\ell$, rather than the residual stream $\mathbf{h}_\ell$.

Beyond the extended embedding $\mathbf{x}$, a few aspects of the Extender design bear explicit mention.

Dual FFN outputs the FFN has two outputs: $\hat{\delta_\ell}$ and ϵ_ℓ, thus the output of the FFN is $d_{model} + d_\epsilon$ wide. The hidden dimension of the FFN is governed by the width of its input.

Attention Query the Attention query projection takes $RMSNorm(\mathbf{s}_\ell) + RMSNorm(\mathbf{h}_{\ell-1})$ as input. This does not affect the attention memory footprint, since $\mathbf{q}$ is not retained for future tokens to attend to.

FFN skip The attention output is added after a linear weight to $\mathbf{h}$. This mirrors the design of a Llama-style transformer, and allows the Attention layer to learn directly from the loss.

3.1 Choosing d_ϵ and Designing the **kv** Windows

In this paper, we use $|\mathbf{s}|=d_{model}$. This makes the Extender attention layer shape identical to that of the Transformer, making direct comparison easier. Given this constraint, we must choose d_ϵ, or more generally $|\epsilon_\ell|$. Here, we decompose that into setting d_ϵ and ℓ_{max}, where ℓ_{max} is the final layer that emits an ϵ. By default, $\ell_{max}=L-2$. Moreover, since for $\ell > 0$, $|\mathbf{x}_\ell| \geq d_{model}$ each layer $\ell > 0$ can only read a subset of $\mathbf{x}_\ell$.

These two design decisions are closely tied: with a small d_ϵ, it becomes more important that all the features of ϵ are seen by subsequent layers. Similarly, with a large d_ϵ, late layer writes may crowd out the initial embedding and early layer writes, making late layers unable to see critical features. For example our 26-layer model has d_{model}=1664 ($\frac{\text{width}}{\text{height}}$=64). With d_ϵ=32, $\sum |\epsilon_\ell|$=800 features, leaving room for 864 input embedding features in $\mathbf{s}_{25}$. with d_ϵ=64, $\sum |\epsilon_\ell|$=1600 features. Empirically, sizes larger than 64 and smaller than 32 perform significantly worse, and are not evaluated here.

With d_ϵ=64, the choice of read window is particularly important. Here, a sliding window leaves the final layer with only 64 features of the original embedding. Three choices we considered were $First$, $Last$, and $Random$, for both the **k** and the **v** projection. For **k**, we found that $Last$ is always the better choice. For **v** however, the choice is less obvious: many important tasks involve an identity mapping. Thus **v** must be able to recall enough of the input embedding to produce the correct logit. For these, $Last$ would be a poor choice when d_ϵ=64 For many other tasks, $First$ would instead be a poor choice. For d_ϵ=64 and ℓ_{max}=L−2, **k**=$Last$, **v**=$Random$ may be the better compromise. Another possibility is changing ℓ_{max}, preventing later layers from writing to **x**. For example, setting ℓ_{max}=$L/2$ cuts $\sum|\epsilon_\ell|$ by half, leaving room for the embedding. We evaluate a few of these choices in Appendix B. Our default model uses d_ϵ=32, ℓ_{max}=$L-2$, except $|\epsilon_0|$=64.

4 Inference and the $\mathbf{x}_*$-Cache

By replacing the **kv**-cache with an $\mathbf{x}_*$-cache and computing **kv** from $\mathbf{x}_*$ during attention, the Extender could achieve the memory savings claimed above both during and between turns. However, this would incur the compute cost of repeatedly reprojecting **kv** from $\mathbf{x}_*$. To address this, the Extender uses an ephemeral **kv**-cache, which may be freed between turns as desired. This effectively amortizes the cost of reprojecting the **kv**-cache from the $\mathbf{x}_*$-cache. It is also worth noting that because the reprojection cost grows linearly with the total **kv** dimension, using GQA for attention dimensionality reduction **kv**-cache speeds up the reprojection time by a factor similar to the GQA space savings.

5 Evaluation

Below, we present results from Extender and Transformer language models trained on the ClimbMix Diao et al. (2025) and ProLong Gao et al. (2025) datasets, using the short-context DCLM CORE Li et al. (2024) and long-context RULER Hsieh et al. (2024) benchmark suites.

Reference Transformer In order to better understand the strengths and weaknesses of the Extender, we compare it to a *Reference* Transformer. The Reference Transformer is a Llama-style MHA transformer with two small changes, following Andrej Karpathy's nanochat Karpathy (2025) design: the Muon optimizer Jordan et al. (2024) instead of AdamW for the large matrices, and SoftCap Gemma Team (2024) in the attention layer (included with Flash Attention-2 Dao (2024)).

The Reference Transformer is a decoder-only Llama-style Transformer with a standard pre-norm residual stream. Token embeddings of dimension d_{model} are processed by L identical blocks, each of the form

$$\mathbf{h} \leftarrow \mathbf{h} + \mathrm{Attn}(\mathrm{RMSNorm}(\mathbf{h})), \qquad \mathbf{h} \leftarrow \mathbf{h} + \mathrm{FFN}(\mathrm{RMSNorm}(\mathbf{h})),$$

followed by a final RMSNorm before unembedding. RMSNorm uses ε=10^{-5} and a learnable scale.

The Extender mirrors the Reference Transformer as closely as possible in terms of shape, activation functions, optimizer, training schedule, weights, norms etc. Below are several common characteristics.

By default, we use common multi-head causal attention, with Rotary Position Embedding (RoPE, default θ=10^6) applied to queries and keys, using the standard half-dimension split. GQA is addressed briefly in Appendix D Attention logits are soft-capped as $s \cdot \tanh(z/s)$ with default $s = 50$, using a FlashAttention-2 kernel. The feed-forward network is SwiGLU Shazeer (2020),

$$\mathrm{FFN}(h) = W_2\big(\mathrm{SiLU}(W_1\mathbf{h}) \odot W_3\mathbf{h}\big),$$

with hidden width $\lfloor 8d/3 \rfloor$ rounded up to a multiple of 32, and all linear layers bias-free.

The default tokenizer is nanochat's byte-level BPE, vocabulary 32k. Documents are prefixed with a beginning-of-sequence token, and we use document masking for best-fit packed training sequences.

Input embeddings and the output projection share a single weight matrix (tied embeddings). Parameters are initialized as: embedding rows $\mathcal{N}(0, 1/\sqrt{d})$; multi-dimensional weights $\mathrm{Uniform}(-1/\sqrt{\mathrm{fan_in}}, 1/\sqrt{\mathrm{fan_in}})$; RMSNorm scales are initialized to 1.

Models are trained using a WSD training schedule Hägele et al. (2024). We use the Muon optimizer Jordan et al. (2024) for the large weight matrices, and the AdamW optimizer Loshchilov & Hutter (2019) for the embedding weights and other one-dimensional tensors, such as norms. Effective batch for ClimbMix 2k was 256 sequences, scaled peak learning rates were: Muon 2.1×10^{-2}, AdamW 1.41×10^{-2} on embeddings, and 8.5×10^{-4} on other 1-D parameters. For ProLong 64k, effective batch was 4 sequences, scaled peak LR: Muon 8.8×10^{-4} and AdamW 3.5×10^{-5} on both AdamW groups. For efficiency, GEMMs run in bfloat16; RoPE Su et al. (2024) frequencies and RMSNorm Zhang & Sennrich (2019) statistics stay in fp32.

The design intentionally omits QK-norm, value embeddings, attention residuals, and output logit softcap, all of which could be adapted to the Extender, but would not help characterize the difference between the two architectures. The model parameter counts vary slightly between our Extender and Reference Transformer models due to architecture differences, but model width d_{model} and depth L are identical, and as follows. We've made an effort to use the actual sizes in the tables, plots and analysis, but will use the size class name when discussing multiple models of the same size class but different exact sizes.

Size class	**Width** d_{model}	**Depth** L	**Q/K/V heads**	**Head dim**
198M	1024	13	8	128
436M	1280	20	10	128
920M	1664	26	13	128

Table 2: Model shape for each size class. The 436M and 920M models have width-to-depth ratio of 64. The 198M model departs slightly from this ratio, to retain 128-feature heads. The size classes are named after the Transformer size. The Extender versions have approximately 0.5% more parameters.

We apply an auxiliary regularization cost to each token to prevent runaway residual writes: $\frac{1}{L}\sum_\ell \text{relu}(\text{RMS}(\delta_\ell) - \tau)^2$ with τ=8 and coefficient $3{\times}10^{-3}$. After this, later layers still receive more weight (w_ℓ is often 10× that of the earliest layers), but features of δ_ℓ are in a healthy range.

5.1 Short Context: DCLM CORE Benchmark suite

We trained one Extender and one Reference Transformer of each size class, from scratch, using Chinchilla Hoffmann et al. (2022) token budgets of ≈20 tokens per model parameter, or approximately 4B, 9B and 18B tokens for the 198M, 436M and 920M parameter models respectively. No fine-tuning or other task-specific training was applied. Figure 2 shows the performance of these models on the DCLM CORE v1 Li et al. (2024) benchmark suite, using the nanochat Karpathy (2025) CORE harness. We group the 22 CORE benchmarks into categories *T-Mono*/*other*. Here, *T-Mono* is the subset of tasks on which the Transformer score improved monotonically with parameter count under Chinchilla-matched training. Intuitively, monotonic improvement with size suggests a more reliable benchmark in this size range. For reference, we also include Llama-3.2-1B scores. This model was pretrained on up to 9 trillion tokens Meta (2024), explaining the large performance gap on some tasks. On other tasks, such as arc_easy, Llama's 500× more training does not improve performance, suggesting the 1B model size is the primary limitation rather than the training or the architecture.

Overall, we find that the Extender and Transformer models perform similarly on DCLM CORE tasks, given parameter counts within 1% of each other, and equal training budgets.

5.2 Long Context: RULER

In this section, we evaluate Reference Transformer and Extender performance on long-context tasks using the RULER Hsieh et al. (2024) benchmark suite. The purpose of the evaluation is an apples-to-apples comparison between Extender and Transformer architectures, not to maximize RULER scores. Models were not fine-tuned for instruction-following or RULER. As a result, these absolute scores are lower than overtrained or fine-tuned public models.

Here, we focus on the 920M parameter models, and evaluate them at the pretrain stage, after 2.5B tokens of 64k ProLong data Gao et al. (2025), and after 5B tokens of 64k ProLong respectively,

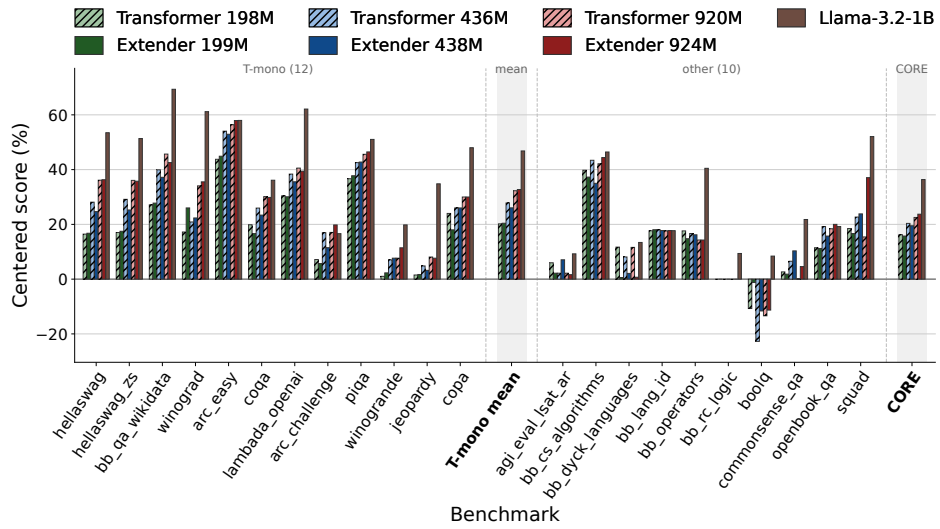


Figure 2: Centered DCLM Core v1 scores. One Transformer (hatched) and one Extender per size class, plus Llama-3.2-1B. 198M/436M/920M trained to Chinchilla token budgets. T-Mono: tasks where the Transformer score increased monotonically with parameter count under Chinchilla.

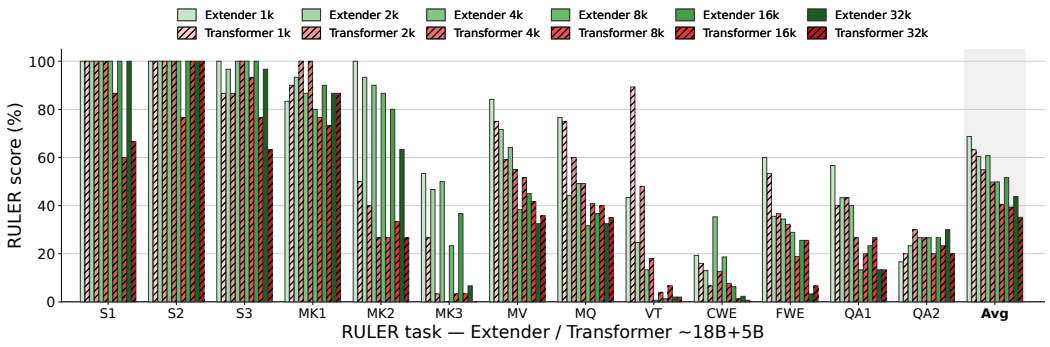


Figure 3: Per-task RULER scores for the ClimbMix 18B → +5B 64k ProLong checkpoints, at evaluation lengths 1k / 2k / 4k / 8k / 16k / 32k. Extender is green, Transformer is red; darker bars are longer contexts. All 13 tasks are shown. Avg is the 13-task RULER mean.

Table 3: Mean accuracy (%) after each mid-train stage, for Extender and Transformer (920M unless noted). SCORE is NIAH-6 (six needle tasks: S1–S3, MK-1, MV, MQ), REST (the other seven RULER tasks: MK-2/3, VT, CWE, FWE, QA-1/2), or RULER (13-task mean).

Model	Stage	SCORE	1k	2k	4k	8k	16k	32k	64k	T-mono	CORE
Extender	ClimbMix 18B	NIAH-6	91.4	82.8	*57.6*	*37.2*	*13.3*	*4.0*	*0.7*	32.7	23.7
		REST	31.2	22.5	*11.6*	*5.0*	*4.1*	*3.7*	*2.9*		
		RULER	59.0	50.3	*32.9*	*19.9*	*8.4*	*3.8*	*1.9*		
Transformer	ClimbMix 18B	NIAH-6	90.0	84.4	*17.5*	*0.0*	–	–	–	32.3	22.5
		REST	33.1	24.1	*5.9*	*0.0*	–	–	–		
		RULER	59.4	51.9	*11.2*	*0.0*	–	–	–		
Extender	ProLong 64k +2.5B	NIAH-6	92.8	87.6	85.1	77.5	78.6	73.2	43.3	30.8	23.5
		REST	51.4	43.1	31.1	23.7	19.2	7.9	4.8		
		RULER	70.5	63.7	56.1	48.5	46.6	38.0	22.6		
Transformer	ProLong 64k +2.5B	NIAH-6	91.0	83.6	75.4	55.7	48.9	42.9	21.0	29.9	21.8
		REST	38.5	18.6	11.1	8.2	7.6	5.0	3.0		
		RULER	62.7	48.6	40.8	30.1	26.7	22.5	11.3		
Extender	prolong 64k +5B	NIAH-6	90.7	84.3	83.3	75.0	78.6	74.7	59.3	31.5	22.8
		REST	49.9	40.0	41.4	28.3	28.6	17.3	9.9		
		RULER	68.7	60.4	60.8	49.9	51.7	43.8	32.7		
Transformer	ProLong 64k +5B	NIAH-6	87.8	84.3	84.0	71.0	65.3	64.6	28.5	30.8	21.8
		REST	42.2	29.7	20.4	14.4	17.2	9.9	7.1		
		RULER	63.2	54.9	49.8	40.5	39.4	35.1	17.0		

using the recommended 60/40 mix of long vs. short context documents. For 2.5B ProLong training, we kept $\theta = 10^6$, and resumed training from the pre-decay pretrain checkpoint, then trained with ProLong tokens finished by a brief decay. For 5B ProLong training, we similarly resumed training from the pre-decay checkpoint of the 2.5B training run. The resulting aggregate token count for the 5B ProLong models is 21.3B tokens (some ClimbMix tokens lost here due to pre-decay resume).

Figure 3 shows how the Extender and the Reference Transformer performed on RULER for context lengths between 1k and 32k tokens. Results are based on n=30 samples per task, with greedy decoding. On average, we find that the Extender outperforms the Reference Transformer on these long-context tasks. Two interesting details stand out: the Extender has a large advantage on the harder multi-key tasks, while the Transformer wins convincingly at the variable tracking task. This suggests a possible difference in inductive bias on these three tasks. However, we did not investigate this further.

We define a subset NIAH-6 as the six easier tasks: single-key needle-in-a-haystack variants 1–3 (S1–S3), multivalue (MV), multiquery (MQ), and multikey variant 1 (MK1). We use the label REST to refer to the remaining group of tasks (MK2–3, VT, CWE, FWE, QA1 and QA2). Table 3 reports NIAH-6, REST, and the 13-task RULER mean at each evaluation length, plus T-mono and centered CORE (%) for the ClimbMix 2k base and the 64k ProLong mid-train stages. After the ClimbMix pretrain, and before ProLong training, the Extender is significantly better than the Transformer at context lengths beyond the 2k training context. For 1-2k contexts, any differences are marginal. After ProLong training, the Reference Transformer improves significantly on longer contexts over its pretrain equivalent. However, Transformer performance on long-context tasks consistently trails the Extender, on both the easier NIAH-6 and the harder REST tasks.

For the 920M models shown, the persistent attention memory footprint at 64k context is approximately 54.5 million features (109 MB at bf16) for the Extender, vs. 5.67 billion features (11.3 GB at bf16) for the (MHA) Reference Transformer, a factor 104× reduction. Despite this, we find the Extender to be interchangeable with the Transformer on short context DCLM CORE tasks, and somewhat ahead on long-context RULER tasks.

5.3 Ablation Studies

To better understand how each aspect of the Extender contributes to its performance, we run a series of ablation studies. For these, we train matched ∼200 M Extenders (d_{model}=1024, L=13, d_ϵ=32) on ClimbMix for ≈4 B tokens with a shared WSD recipe, varying one architectural choice at a time

Table 4: Extender at d_ϵ=32 (~200 M, d=1024, L=13, ≈4 B ClimbMix tokens, seed 42). Each later row changes one default choice.

Configuration	Val.	T-mono	CORE	RULER			
				1k	2k	4k	8k
default Extender	2.7547	20.1	14.6	50.1	39.3	25.7	13.5
Q from RMSNorm(h)	2.7555	20.4	15.7	47.0	41.8	12.6	1.2
Q from h	2.7617	19.5	15.1	44.7	41.5	22.1	12.4
Q from x	2.7695	19.2	15.4	44.9	38.5	18.1	8.0
Q from RMSNorm($[x; h]$)	2.7500	20.2	15.6	48.0	37.0	25.9	13.9
Q from $[x; h]$	2.7586	20.0	14.3	48.2	39.4	23.1	10.6
FFN input: RMSNorm($\mathbf{a}_\ell$)	2.8398	17.8	14.5	42.4	35.3	25.3	10.2
FFN input: RMSNorm($\mathbf{a}_\ell$ + RMSNorm($\mathbf{h}$))	2.7727	19.6	14.8	46.0	39.2	33.0	15.4

relative to the default Extender residual graph. Table 4 reports final validation loss, DCLM CORE (T-mono mean and overall CORE, centered) and RULER, for a subset of the ablation tests performed.

Attention Query Input In contrast to the Transformer, Extender has a choice of what input to take for the attention query projection matrix: $\mathbf{x}$, $\mathbf{h}$, or some combination of the two, with or without RMSNorm applied to the input. The default setting is Q from RMSNorm($\mathbf{x}$) + RMSNorm($\mathbf{h}$). In the table, $[x; h]$ denotes concatenation rather than addition of inputs. Many of the differences are marginal, however, two significant results stand out: Q from either h or x alone significantly underperforms on long-context tasks. Note that a concatenated input, with a W_q that is twice as large, did not substantially improve accuracy.

Residual Input into FFN As seen in Table 1, the default configuration passes RMSNorm($\mathbf{a}_\ell + \mathbf{h}_{\ell-1}$) as input to the FFN. Here, we test two ablations: eliminating $\mathbf{h}$ as input, and applying an additional norm on $\mathbf{h}$: RMSNorm($\mathbf{a}_\ell$ + RMSNorm($\mathbf{h}_{\ell-1}$)).

From these ablations, it is clear that including $\mathbf{h}$ in the FFN input lifts performance. However, compared to a conventional Transformer, the loss from dropping $\mathbf{h}$ is not catastrophic: much of the communication between layers is carried by $\mathbf{x}$ rather than relying fully on $\mathbf{h}$. For out-of-domain tasks, having $\mathbf{h}$ in Q is much more important than having $\mathbf{h}$ in the FFN input. The extra RMSNorm ablation reports mixed and likely insignificant results.

5.4 Limitations

While the Extender with $d_\epsilon = 32$ scales well from 199M to 924M parameters, we have not yet been able to test it at much larger model sizes due to resource constraints. Similarly, our token budgets are largely limited to 20× the parameter count.

6 Conclusion

In contrast with prior work on reducing the attention memory footprint, the Extender focuses on the persistent memory footprint: memory required between turns.

The Extender separates the superposition channel $\mathbf{h}$ focused on next-token prediction, from the concatenation channel $\mathbf{x}$ dedicated to attention memory. For every size class we tested, 32 features of attention memory per layer was enough to match the Reference Transformer on short-context tasks, and outperform it on long-context tasks.

7 AI Use Disclosure

In this work, we used generative AI tools to help write the software implementation of both the Extender and Reference Transformer in pytorch, write code to analyze log files, run performance and diagnostic experiments, write code to generate plots from data, and to populate individual table entries, to proofread and suggest edits to the text, to identify relevant related work and create bibliography entries, and to critique figures and writing. We did not use AI to create the original idea, draft or write the text, draw diagrams, interpret results, or create or refine hypotheses.

We have reviewed all AI-assisted work. For example, we have reviewed the implementation source code for errors, checked all plots and numbers for internal consistency and consistency with expected results, and run additional experiments to double-check surprising results. We take responsibility for the final content of this work, including experimental results and artifacts produced with the aid of generative AI.

8 Reproducibility Statement

The results presented here have been vetted for reproducibility: evaluation programs use fixed random seeds, and running the same training or evaluation job twice produces the same results within the bounds of hardware differences, supporting software versions and randomness that may occur beyond the pseudorandom seed. We used only public datasets, publicly available software packages, and a small amount of our own source code.

While the source code is not available for download today, it will be made available upon publication, including complete scripts for training the described models and evaluating them to produce the results shown here. The results in this paper stem from training and evaluation runs on NVIDIA H100 GPUs with SMI driver version 595.84, CUDA Version: 13.2, Python version 3.12.3 and Pytorch 2.5.1+cu121. We have noticed that changes Pytorch version and/or hardware can occasionally result in significant numeric differences for both Extender and Transformer models, and will provide guidance based on our own testing as far as what versions may be used to achieve the closest matching results.

REFERENCES

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 (EMNLP)*, pp. 4895–4901, 2023.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024. NeurIPS 2024.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-CLIMB: CLustering-based iterative data mixture bootstrapping for language model pre-training. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

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

Daniel Goldstein, Fares Obeid, Eric Alcaide, Guangyu Song, and Eugene Cheah. GoldFinch: High performance RWKV/transformer hybrid with linear pre-fill and extreme KV-cache compression. *arXiv preprint arXiv:2407.12077*, 2024.

Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems*, 2024. NeurIPS 2024. Warmup-stable-decay schedule.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. NIPS 2014 Deep Learning Workshop.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling (COLM)*, 2024.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. Blog post, 2024. `https://kellerjordan.github.io/posts/muon/`.

Andrej Karpathy. nanochat. GitHub repository, 2025. `https://github.com/karpathy/nanochat`.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 5156–5165, 2020.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, 2024a. NeurIPS 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 32332–32344, 2024b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Meta. Llama 3.2 model card. Hugging Face, 2024. `https://huggingface.co/meta-llama/Llama-3.2-1B`.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanislaw Wozniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, 2023.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, Kranthi Kiran GV, Jan Kocoń, Bartłomiej Koptyra, Satyapriya Krishna, Ronald McClelland Jr., Jiaju Lin, Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Cahya Wirawan, Stanisław Woźniak, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Peng Zhou, Jian Zhu, and Rui-Jie Zhu. Eagle and finch: RWKV with matrix-valued states and dynamic recurrence. In *First Conference on Language Modeling (COLM)*, 2024.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. NeurIPS 2019 EMC2 Workshop.

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

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. Introduces SwiGLU.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. arXiv:2104.09864.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024. NeurIPS 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017. NIPS 2017.

Haoyi Wu and Kewei Tu. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. ACL 2024.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 38087–38099, 2023.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019. NeurIPS 2019.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. MLKV: Multi-layer key-value heads for memory efficient transformer decoding. *arXiv preprint arXiv:2406.09297*, 2024.

A Additional Long-Context RULER Evaluation Details

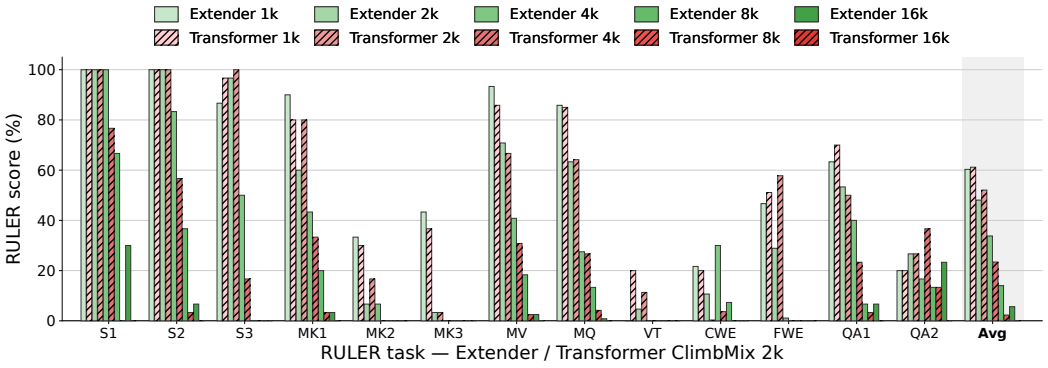


Figure 4: Per-task RULER scores for the ClimbMix 2k pretrain checkpoints, at evaluation lengths 1k / 2k / 4k / 8k / 16k. Extender (d_ϵ=32) is green, Transformer is red.

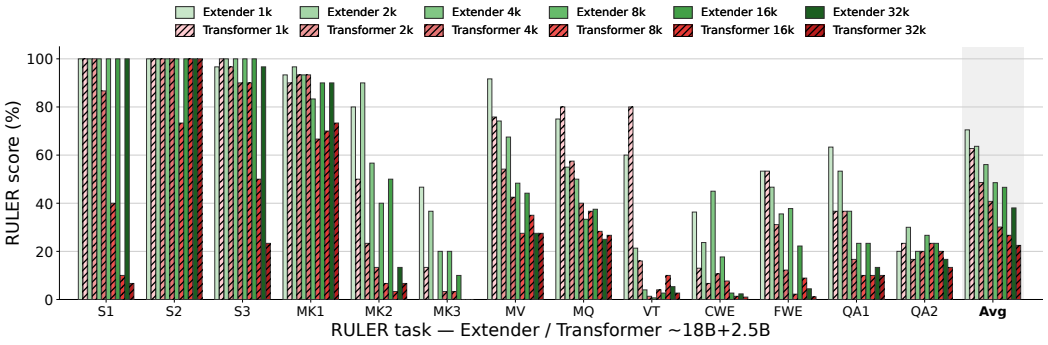
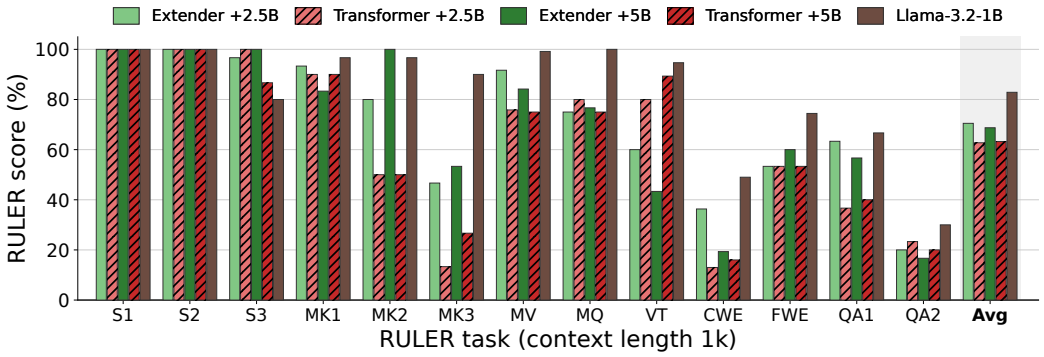


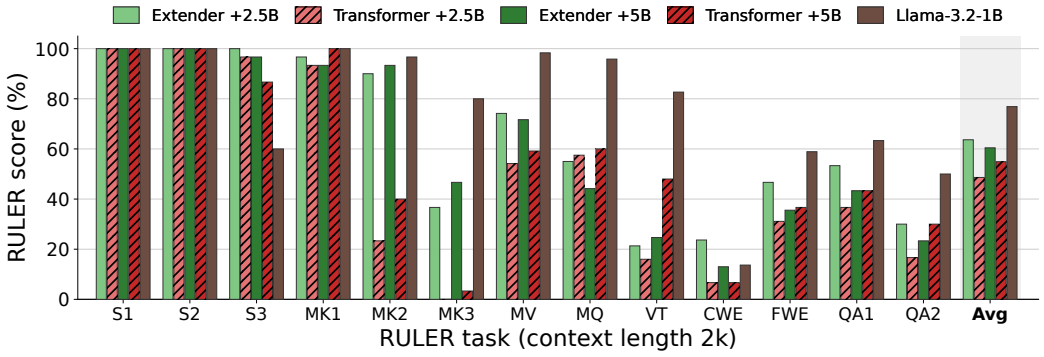
Figure 5: Per-task RULER scores for the ClimbMix 18B → +2.5B 64k ProLong checkpoints, at evaluation lengths 1k / 2k / 4k / 8k / 16k / 32k. All 13 tasks are shown. Avg is the 13-task RULER mean.

Figures 4 and 5 offer additional training stage RULER performance details. In Figure 4, the Extender has a clear advantage at context lengths beyond the 2k training context. Figure 5 is the +2.5B stage of Figure 3, with the Extender advantage already clearly visible.

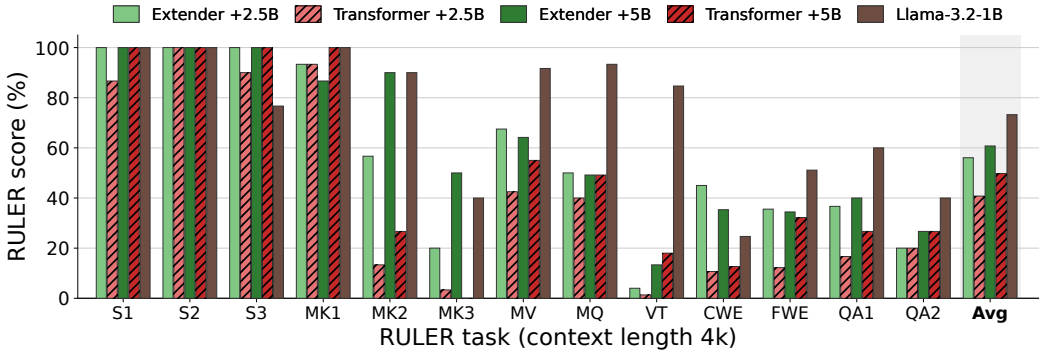
Figures 6a–7c show detailed per-task RULER results per context length, up to 32k.



(a) Context length 1k.

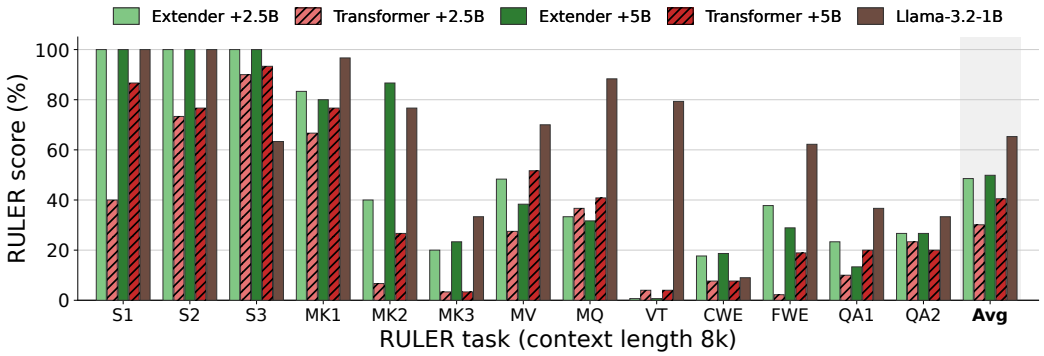


(b) Context length 2k.

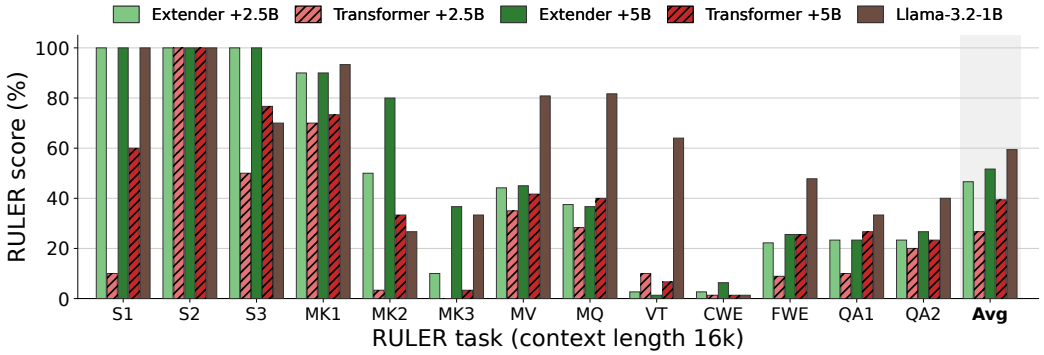


(c) Context length 4k.

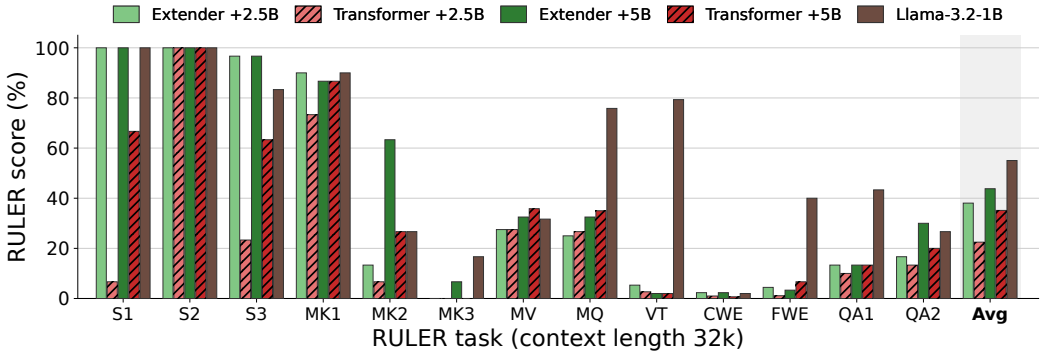
Figure 6: Per-task RULER scores after 64k ProLong mid-train from the ClimbMix 18B checkpoint (+2.5B / +5B), at 1k, 2k, and 4k. Extender / Transformer: d=1664, L=26. Sizes 8k, 16k, and 32k in the next figure.



(a) Context length 8k.



(b) Context length 16k.



(c) Context length 32k.

Figure 7: Same models as Figure 6, at 8k, 16k, and 32k.

B ϵ_ℓ SIZE AND V WINDOW DESIGN

Table 5: Window and ϵ choices at 920M pretrain (~900 M, d=1664, L=26, ClimbMix 2k, 9.6 B tokens). wide0 sets $|\epsilon_0|$=2d_ϵ. ℓ_{max} = 13 uses K and V from Last(d) and emits ϵ_ℓ on only the first 13 layers. RULER is the 13-task mean (%).

Configuration	Val.	T-mono	CORE	RULER			
				1k	2k	4k	8k
d_ϵ=32 wide0, V= Last	2.4983	30.5	23.0	60.3	48.1	*33.8*	*14.0*
d_ϵ=32, V= Last	2.4984	30.5	22.9	57.3	44.3	*36.9*	*14.3*
d_ϵ=64 ℓ_{max}=13	2.4781	30.0	22.9	61.9	47.7	*33.6*	*15.1*
d_ϵ=64, V= Last	2.4934	30.8	22.9	63.0	53.9	*34.6*	*15.3*
d_ϵ=64, V=Rand	2.4773	30.4	23.4	58.6	47.7	*33.8*	*13.3*
d_ϵ=128, V=Rand	2.4842	29.2	20.6	61.4	52.0	*32.8*	*14.7*
Transformer	2.4983	32.3	23.5	58.5	46.2	*19.7*	*0.8*

Table 5 evaluates d_ϵ vs. window design for V on our 920M model class. Due to resource constraints, these ablation models are trained to 9B tokens instead of the full 18B. In our experience, relative 9B token model performance is a good indicator of relative 18B model performance. We evaluate six different choices. Here, *wide0* doubles the width of the first layer: by iteratively zeroing out ϵ_ℓ during evaluations, we have anecdotally found that ϵ_0 is critical to Extender performance on most tasks, while others are task dependent. d_ϵ=32 with *wide0* is the default Extender. However, we find the differences between most models are marginal in these ablations, d_ϵ = 128 being a notable exception for short contexts. Anecdotally, d_ϵ = 64 V=Rand is a strong choice when fully trained. We did not choose it as the default model due to the cleaner and more memory and compute-efficient design of the d_ϵ=32, V=*Last* options.

C DISTILLATION AND EXTENDERS

Table 6: Logit distillation into a 198M Extender. T-mono and CORE are centered DCLM CORE (%); T-mono is the 12-task set in Figure 2. RULER is the 13-task mean (%) at 1k–8k.

Model	Tokens	T-mono	CORE	RULER			
				1k	2k	4k	8k
199M Extender, from scratch	4B	19.9	15.9	46.8	38.5	*24.4*	*12.0*
199M Extender student, Extender teacher	4B	24.1	19.1	51.7	44.0	*29.6*	*14.8*
199M Extender student, Transformer teacher	4B	23.8	17.2	51.9	44.0	*27.8*	*16.0*
924M Extender (teacher)	14.5B	33.2	25.1	70.7	64.1	62.5	53.5
920M Transformer (teacher)	14.5B	32.1	23.9	68.3	58.1	54.7	50.1

Knowledge distillation Hinton et al. (2015); Sanh et al. (2019); Gu et al. (2024) trains a student using a larger teacher model, often achieving higher token efficiency than training from scratch. To better understand differences in inductive bias Goyal & Bengio (2022); Battaglia et al. (2018) between Transformer and Extender models, we train three 198M class extenders: one from scratch, one with a 924M Extender teacher, and one with a 920M Transformer teacher. Here, all three 198M Extenders use d_ϵ=64 and $\mathbf{v}$ selects features from $\mathbf{x}$ using the *Random* policy. All three are trained to 4 Billion ClimbMix tokens with a 2k token sequence length. Both teachers were trained with 9B ClimbMix + 5B ProLong tokens - earlier models than the 18B reported in the main body of the paper. The Extender teacher is somewhat stronger than the Transformer teacher, on both CORE and RULER scores.

Table 6 shows the results of this experiment. The two student models both consistently outperform the from-scratch model, showing that Extenders train well under a distillation objective. The student led by an Extender teacher achieved a CORE score 6 points lower than its teacher, while the Transformer teacher transferred slightly less knowledge, with the student scoring 6.7 points below

the teacher. On long-context RULER tasks, the two students achieved near-identical scores within the 2k training domain despite the Transformer teacher being weaker. These results indicate that under logit distillation, an Extender student learns approximately as well from an Extender teacher as from a Transformer teacher.

D Grouped Query Attention and Extenders

Table 7: GQA vs. MHA Extender and MHA Transformer RULER is the 13-task mean (%). For all models d_{model} = 1664. Head dimension 128 → 13 heads. Thus GQA4/5 is 13 query / 3 KV heads (groups 5, 4, 4). Due to the dimensionality reduction of GQA, the parameter count of the GQA4/5 model is 813M rather than 924M.

Model	T-mono	CORE	RULER						
			1k	2k	4k	8k	16k	32k	64k
Extender 924M (MHA)	32.7	23.7	59.0	50.3	*32.9*	*19.9*	*8.4*	*3.8*	*1.9*
Extender 813M (GQA4/5)	33.1	22.9	61.2	52.4	*36.3*	*18.4*	*6.6*	*2.1*	*1.2*
Transformer 920M (MHA)	32.3	22.5	59.4	51.9	*11.2*	*0.0*	–	–	–
Extender 924M +2.5B (MHA)	30.8	23.5	70.5	63.7	56.1	48.5	46.6	38.0	22.6
Extender 813M +2.5B (GQA4/5)	31.2	23.3	63.9	57.9	56.3	49.7	45.8	33.8	20.2
Transformer 920M +2.5B (MHA)	29.9	21.8	62.7	48.6	40.8	30.1	26.7	22.5	11.3
Extender 924M +5B (MHA)	31.5	22.8	68.7	60.4	60.8	49.9	51.7	43.8	32.7
Extender 813M +5B (GQA4/5)	31.6	22.5	67.3	61.2	58.4	45.2	40.2	37.7	26.3
Transformer 920M +5B (MHA)	30.8	21.8	63.2	54.9	49.8	40.5	39.4	35.1	17.0

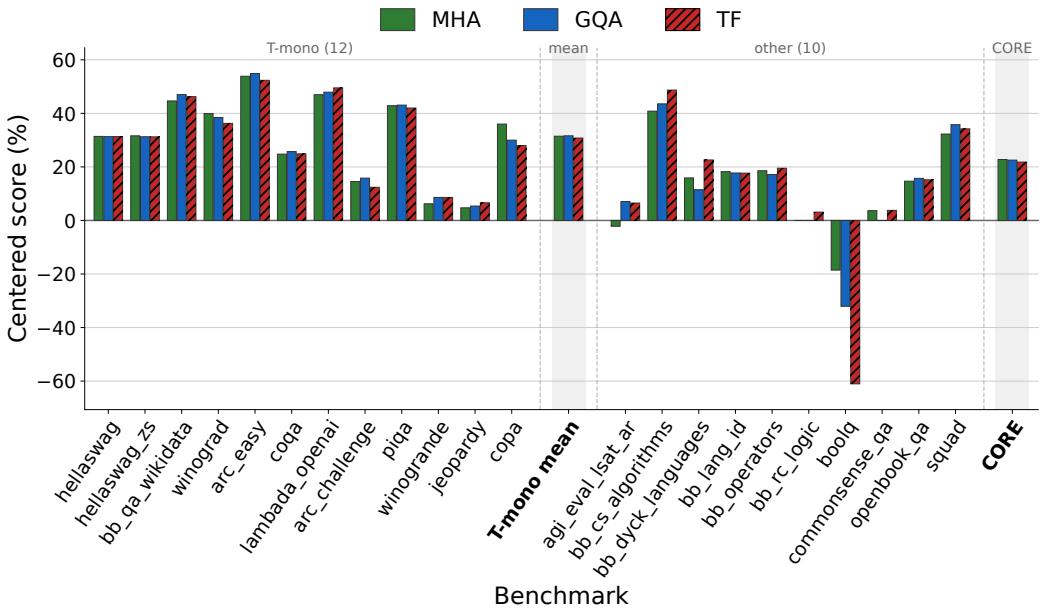


Figure 8: Centered DCLM CORE subscores for the 18B ClimbMix +5B ProLong checkpoints in Table 7. MHA is the full-attention Extender, GQA is 13 query / 3 KV heads (groups 5, 4, 4), and TF is the document-mask Transformer (hatched). T-mono is the 12-task set from Figure 2; CORE is the 22-task mean.

Table 7 shows the performance of GQA on our 924M model, vs. our regular MHA Extender and MHA Transformer, at the 18B ClimbMix pretrain, +2.5B ProLong and +5B ProLong checkpoints. Because our 920M class model has d_{model} = 1664, there are 13 heads each 128 features wide. To accomodate this, we use GQA4/5: two groups of 4 queries, one group of 5 queries. Due to the

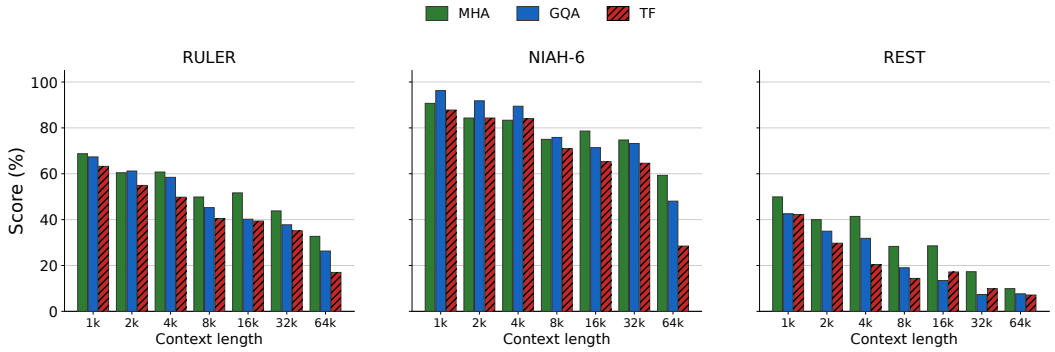


Figure 9: RULER, NIAH-6, and REST versus context length for the +5B checkpoints in Table 7. NIAH-6 is S1–S3, MK1, MV, and MQ; REST is the other seven tasks (MK2–3, VT, CWE, FWE, QA1–2). lengths 1k–64k.

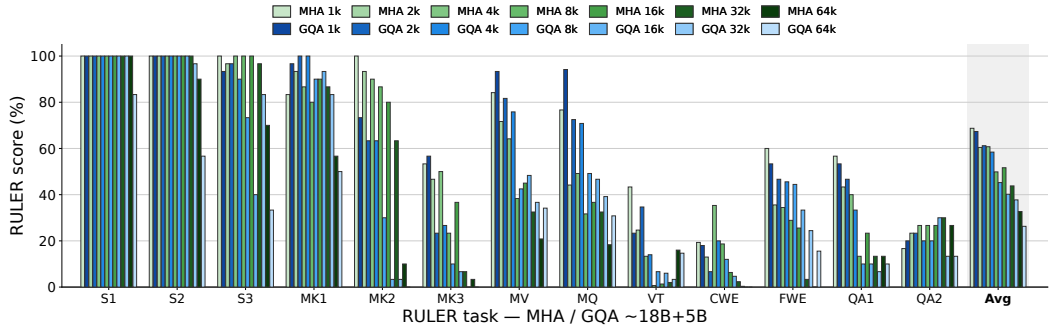


Figure 10: Per-task RULER scores for the 920M 18B ClimbMix +5B ProLong MHA and GQA checkpoints, at 1k / 2k / 4k / 8k / 16k / 32k / 64k. All 13 tasks are shown. Avg is the 13-task RULER mean, n=30.

GQA dimensionality reduction, the actual parameter count of the GQA4/5 model is 813M rather than 924M.

We find that GQA interacts well with Extender, and the Extender GQA4/5 model outperforms the Transformer MHA. Due to resource constraints, we did not train a matching Transformer GQA.

Figure 8 provides per-task performance on the CORE benchmark. GQA tracks MHA performance closely on the more stable T-Mono tasks. On the remaining tasks results are a little noisier, but GQA is largely tracking MHA there as well. Figure 9 shows the RULER performance divided into 13-task mean, NIAH-6, and REST. GQA is overall tracking MHA closely on both the easier NIAH-6 tasks and REST. Finally, Figure 10 shows individual task performance vs. context length.

E Accuracy across model sizes

Table 8 reports centered CORE and RULER for the default Extender for our three size classes, for models trained on ClimbMix only, to a Chinchilla token budget. Accuracy on both CORE and RULER 1–2k improves consistently with model size, particularly on T-mono, where the Transformer is also reliably improving with model size. At context lengths outside of the 2k training context length, RULER scores are less predictable, but suggest an improvement as well.

Table 8: Default Extender at each size class in Figure 2. 199M is 4B tokens, 438M is 9.6B, and 924M is 18B. T-mono and CORE are centered DCLM CORE (%); T-mono is the 12-task set in Figure 2. RULER is the 13-task mean (%). Italics are lengths longer than the 2k training context. "–": not evaluated.

Model	T-mono	CORE	RULER			
			1k	2k	4k	8k
199M	20.4	15.7	49.3	41.0	*28.0*	*16.9*
438M	26.0	19.5	57.3	45.1	*33.4*	*15.6*
924M	32.7	23.7	59.0	50.3	*32.9*	*19.9*