Dynamic Short Convolutions Improve Transformers

Oliver Sieberling[1] **Bharat Runwal**[2] **Rameswar Panda**[2] **Yoon Kim**[1]

[1]Massachusetts Institute of Technology [2]MIT-IBM Watson AI Lab

osieberl@mit.edu

Abstract

Transformers have become the dominant architecture for large language models, largely due to the scalability and flexibility of attention, feed-forward layers, residual connections, and normalization. This paper introduces dynamic short convolutions as an additional neural network primitive for improving Transformers. Unlike static short convolutions, dynamic convolutions use input-dependent filters, which preserves the locality bias of convolution while increasing expressivity. Motivating experiments show that applying dynamic short convolutions to key, query, and value representations improves performance on challenging associative recall tasks compared with static convolutional variants. Across language-modeling experiments ranging from 150M to 2B parameters, dynamic convolutions consistently outperform standard Transformers and Transformers augmented with static short convolutions. Fitting scaling laws indicates a 1.33× compute advantage over compute-matched Transformers when dynamic convolutions are applied to the key, query, value vectors, and a 1.60× advantage when adding dynamic convolutions after every linear layer. Dynamic convolutions also offer improvements on linear RNNs (Mamba-2/Gated DeltaNet) and mixture-of-experts architectures. We make these gains practical with custom Triton kernels[1] that enable efficient training with a manageable end-to-end slowdown. These results suggest that dynamic short convolutions are a scalable, hardware-efficient, and expressive primitive for advancing Transformer-based language models.

1 Introduction

Individual neural network layers form the primitive building blocks of deep learning architectures. Core primitives that have become mainstays include multilayer perceptrons (Rosenblatt, 1958; Rumelhart et al., 1986), convolutions (Fukushima, 1980; LeCun et al., 1998), recurrent layers (Elman, 1990; Hochreiter & Schmidhuber, 1997; Cho et al., 2014) and attention (Bahdanau et al., 2014). Residual connections (He et al., 2016) and normalization techniques (Ioffe & Szegedy, 2015; Ba et al., 2016) are also crucial for practical training of deep architectures built out of such layers.

The Transformer architecture (Vaswani et al., 2017) exemplifies how such primitives can be composed into a flexible and scalable model that is effective across domains. Transformers are built from repeatedly interleaving attention and feed-forward blocks with residual connections and layer normalization. Major refinements to these components since inception include gated and mixture-of-experts feed-forward layers (Shazeer, 2020; Fedus et al., 2022), placement/type of normalization layers (Zhang & Sennrich, 2019; Xiong et al., 2020), key-value sharing techniques (Shazeer, 2019; Ainslie et al., 2023), and relative positional encodings (Su et al., 2021). These architectural advances, combined with improved optimization techniques and higher quality training data, have significantly pushed the performance-efficiency frontier—indeed, modern sub-10B-parameter LLMs routinely outperform older 100B+ parameter LLMs based on older Transformer variants.

[1]The Triton kernels are available at https://github.com/OliverSieberling/dynamic-conv1d.

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

This paper proposes *dynamic convolutions* as an additional primitive for improving the Transformer. Convolution layers, which apply a shared local filter across sequence positions to mix neighboring token representations, have long been used as the "sequence mixing" component in deep models for natural language processing, from early seminal work on word-level tagging (Collobert & Weston, 2008; Collobert et al., 2011), to sentence-level classification (Kalchbrenner et al., 2014; Kim, 2014), sequence-to-sequence learning (Kalchbrenner et al., 2016; Gehring et al., 2017), and language modeling (Dauphin et al., 2017). However, they largely fell out of favor as a primary sequence-mixing mechanism following the introduction of Transformers. In the post-Transformer era, some works have instead found that incorporating lightweight depthwise-separable convolution layers that apply independent local filters within each channel (also called *short convolutions*) into Transformers can improve performance in some settings (So et al., 2021; Allen-Zhu, 2025). To the best of our knowledge, however, such layers are generally not part of frontier open-weight LLMs.[2]

Dynamic (short) convolutions generalize short convolutions by allowing the convolutional filter at each time step to depend on the input, for example by parameterizing it as a learned linear transformation of the current hidden state (Jia et al., 2016; Wu et al., 2019; Chen et al., 2020; Jiang et al., 2020; Gu et al., 2026). This input-dependent parameterization preserves the locality bias of convolutions while increasing their expressivity. For a layer to be practically useful, however, increased expressivity is not enough—it must also be *scalable*. Scalability in the context of modern LLMs means several things. For one, the layer should continue to provide improvements as the model and training data are scaled up. Two, insofar as new layers typically introduce more compute/parameters, the new layer should increase the overall rate at which an architecture can trade off resources for performance, i.e., it should outperform the existing architecture when compute-/parameter-matched. Finally, the layer should be hardware-efficient, i.e., efficiently trainable on modern accelerators such as GPUs and TPUs.

We show that dynamic short convolutions satisfy the above desiderata. Across experiments spanning models with 150M-2B parameters, dynamic convolutions consistently improve upon Transformers with and without short convolutions. Fitting scaling law curves to the results suggests that dynamic convolutions offer a 1.33× compute advantage compared to ordinary Transformers when the dynamic convolutions are applied to the QKV layers, and a 1.60× advantage when they are applied to all linear layers. For wall-clock efficiency, we develop a Triton kernel that results in competitive performance with a well-optimized static short convolution kernel. Combined with an efficient input-dependent filter parameterization, the end-to-end training throughput slowdown is manageable: roughly 8% slowdown for the QKV variant and 22% slowdown for the all-linear variant at the 2B scale. These results collectively position dynamic convolutions as an additional primitive to be considered for improving the Transformer.

2 Dynamic Short Convolutions for Transformers

2.1 Parameterization

A short convolution is a depthwise separable convolution (Chollet, 2017; Howard et al., 2017) with kernel width W (typically $W \in \{3, 4, 5\}$ in language applications), applied along the time dimension. More specifically, a static short convolution computes:

$$y_t := \sum_{k=0}^{W-1} w_k \odot x_{t-k}, \tag{1}$$

where $x_t \in \mathbb{R}^D$ is a sequence of activations, $w \in \mathbb{R}^{W \times D}$ is the convolution filter, and $\odot$ denotes an elementwise product. Note that here the convolution weights w are fixed across the time axis.

Dynamic short convolutions generalize static short convolutions by making the convolution kernel input-dependent. At each position t, a weight generator (e.g., a linear projection) produces the dynamic convolution weights $w^{(t)} \in \mathbb{R}^{W \times D}$, and the convolution is performed with this time-

[2] Short convolution layers are, however, standard in recent linear RNNs such as Mamba (Gu & Dao, 2024; Dao & Gu, 2024) and DeltaNet (Schlag et al., 2021; Yang et al., 2024, 2025b), though even more recent linear RNNs such as Mamba-3 (Lahoti et al., 2026) and Raven (Afzal et al., 2026) eschew the use of short convolutions.

varying filter:

$$y_t := \sum_{k=0}^{W-1} w_k^{(t)} \odot x_{t-k}. \tag{2}$$

Each token thus selects its own filter to retrieve information from the local context. In this respect, dynamic convolutions are reminiscent of attention, but instead of deriving the attention weights from query-key similarity, dynamic convolutions generate them directly from the querying position (Wu et al., 2019). While this mechanism does not reference the content being retrieved, it carries a strong inductive bias toward retrieving by relative position within the filter window.

While dynamic convolutions are expressive, naïvely producing the dynamic convolution weights would require a $D \to W \cdot D$ linear projection, which would roughly double the parameter count of the underlying model (assuming $W = 4$). We therefore consider more parameter-efficient parameterizations. In our first approach, we factorize the projection through a low-rank transformation of rank R. In our second approach, we split the dimensions into different "heads", using the transformation $D \to W \cdot (D/H)$ and broadcasting each weight across a head of size H. While we generally found the low-rank parameterization to perform better, the head-wise variant simplifies the design of an efficient GPU kernel. We also have a bias in the above transformations, and hence the filters are affine transformations of the input.

For placement, for our main experiments we place the dynamic short convolutions on the queries, keys, and values before RoPE, with kernel width $W = 4$. We apply each with a residual, i.e., $X = X + \mathrm{dynamicShortConv}(X)$ for $X \in \{Q, K, V\}$. The projection that generates the dynamic convolution weights takes the post-attention-norm activations as input.[3] We also experiment with placing dynamic convolutions after all linear layers of a Transformer.

2.2 Efficient Training

Dynamic short convolutions have low arithmetic intensity and are therefore bound by memory accesses. Naïve PyTorch implementations repeatedly move intermediate tensors to and from HBM, making dynamic convolutions slow in practice. We address this with a custom Triton (Tillet et al., 2019) kernel that takes the activations and dynamic convolution weights as input, performs the full convolution on-chip, and writes only the final result back to HBM. Each input is read once and each output is written once, so performance is limited primarily by HBM bandwidth.

Since the dynamic-weight tensor of shape $B \times T \times D \times W$ is W times larger than the $B \times T \times D$ activation tensor, it dominates HBM traffic. Therefore, reducing the size of the dynamic convolution weights translates directly into lower latency. Our head-wise dynamic convolution, which shares a single weight filter across H consecutive channels (head-wise tying), reduces the dynamic-weight tensor to $B \times T \times (D/H) \times W$. When $H \gg W$, its IO cost becomes negligible relative to the activations.

For the low-rank prediction of the dynamic convolution weights, we develop a separate Triton kernel that fuses the second projection of a low-rank factorization directly into the convolution kernel. Rather than reading the materialized $B \times T \times D \times W$ dynamic convolution weights, the kernel reads the $B \times T \times R$ low-rank inputs z and the $R \times (D \cdot W)$ second projection U, generates the dynamic weights zU on-chip and immediately applies the convolution. The dynamic weights are never written to HBM, which makes the low-rank kernel significantly faster than the head-size-1 kernel.

Figure 1 compares our custom Triton kernels with PyTorch eager and `torch.compile` baselines (Paszke et al., 2019; Ansel et al., 2024), each selected as the fastest of five mathematically equivalent implementations. We provide a detailed description of the benchmarking setup and the tested baselines in Appendix A. Across all four configurations, our Triton kernels are 1.8–3.9× faster than the best `torch.compile` baseline on the combined forward and backward pass. As expected, the latency decreases as the head-size increases. At $H = 16$, the kernel is even faster than the CUDA-optimized implementation for static convolutions, which we believe is due to the simpler reduction of the convolution-weight gradient. The head-wise kernels sustain 2.6–3.0 TB/s of HBM traffic, compared with a theoretical peak of 3.35 TB/s. Our low-rank kernel has lower latency than the head-size-1 kernel despite fusing an additional linear projection, which demonstrates the benefit

[3] We found this to perform slightly better than taking Q, K, V themselves as input, and it allows the projection to be fused with the `qkv_projection`.

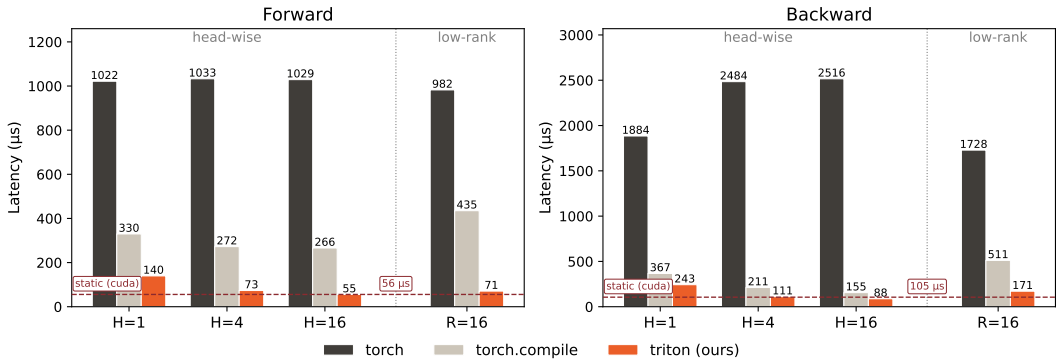


Figure 1: Latency of dynamic short-convolution kernels on an H100 HBM3 80GB GPU ($B = 4$, $T = 4096$, $D = 2048$, $W = 4$, BF16). Triton kernels (orange) vs. the best PyTorch eager (dark grey) and `torch.compile` (light grey) baselines, where each baseline is the fastest of five different implementations. The dashed line is the CUDA-optimized `causal_conv1d` kernel for static short-convolutions of the same width from https://github.com/Dao-AILab/causal-conv1d.

of avoiding materialization of the dynamic convolution weights. Nevertheless, the low-rank kernel remains less optimized than the head-size-16 variant, and an optimized CUDA implementation could further reduce its latency. Overall, our dynamic short convolution kernel is only moderately slower (and for $H \geq 16$ slightly faster) than the CUDA-optimized short convolution kernel[4], which to the best of our knowledge is among the state-of-the-art kernels for static short convolutions.

3 Empirical Study

We experimentally validate augmenting Transformers with dynamic short convolutions in both synthetic benchmarks and real-world language modeling settings.

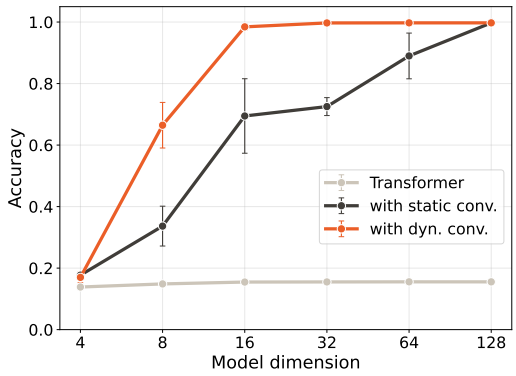
3.1 Synthetic Benchmarks

One motivation for dynamic convolutions in language applications is that language phenomena often require local context-dependent composition to extract meaning from surface form text. For example, consider the phrases "`the old can opener`" and "`the old can swim`". The first phrase is a noun phrase with the syntactic structure `[the [old [can opener]]]` while the second is a verb phrase with the structure `[[the old] [can swim]]`. Even though the prefix of the two phrases is identical, the local composition function over the 4-word window is a function of the last word "`opener`" vs. "`swim`". Successive attention layers can in principle use positional information to compose local context in a context-dependent, dynamic way. However, this is costly and lacks an inductive bias towards locality. Static short convolutions, on the other hand, have a locality bias but do not explicitly model dynamic compositions. Dynamic convolutions are ideally suited for modeling such phenomena.

We study such phenomena in a synthetic setting by considering a modified version of the multi-query associative recall (MQAR; Arora et al., 2023) task. Standard MQAR provides a sequence of (`key`, `value`) pairs, each appearing twice, and supervises the model to predict the value corresponding to a key at the second appearance of each pair. We modify this task by letting each key consist of a variable number of tokens $L_k \in \{1, 2, 3\}$, followed by a delimiter token that encodes L_k, and a single value token. A short illustrative example is given by:

$$\overbrace{\texttt{a b c <3> x}}^{(k_1,v_1)}\ \texttt{x}\ \overbrace{\texttt{b a <2> y}}^{(k_2,v_2)}\ \texttt{a}\ \overbrace{\texttt{b c <2> z}}^{(k_3,v_3)}\ \overbrace{\texttt{b a <2> y}}^{(k_2,v_2)}\ \texttt{a}\ \overbrace{\texttt{b c <2> z}}^{(k_3,v_3)}\ \texttt{x a}\ \overbrace{\texttt{a b c <3> x}}^{(k_1,v_1)}$$

[4] https://github.com/Dao-AILab/causal-conv1d



Task	Transformer	w/ static conv.	w/ dyn. conv.
Compress	0.375	0.417	**0.424**
Fuzzy Recall	0.298	0.505	**0.726**
In-Context Recall	0.942	**1.000**	**1.000**
Memorize	0.791	**0.856**	0.795
Noisy Recall	0.917	**1.000**	**1.000**
Selective Copy	0.930	0.983	**0.988**
Average	0.709	0.793	**0.822**

Figure 2: Left: Performance (median over 5 seeds) on the synthetic variable-key MQAR task. The error bars depict the minimum and maximum values. Right: Performance on the MAD benchmark.

Concretely, in this example, one key (`bc`) is a suffix of another key (`abc`), and therefore a successful retrieval requires a dynamic filter. Here we have three key-value pairs: (`abc`, `x`), (`ba`, `y`), (`bc`, `z`). The second occurrence of each value (underlined) is the supervision target. In between key-value pairs, there can be random filler tokens (grey). The difficulty of this task is that depending on the delimiter token, a different number of preceding tokens must be aggregated to form the key. `<3>` indicates that the key is the previous three tokens, `<2>` the previous two. Because the keys share tokens and have different lengths, no static filter can separate them.

We train Transformers on this task with a single layer and head, varying the model dimension. We compare a vanilla Transformer, the same Transformer with static convolutions on Q, K, V, and our low-rank ($R = 16$) dynamic convolution variant. The convolution widths are all set to $W = 4$, which is just enough to cover the entire key. We train on 100,000 examples, and report the median accuracy over five seeds. As shown in Figure 2, Transformers augmented with dynamic short convolutions outperform Transformers with and without static short convolutions for a given model size, highlighting the benefits of input-dependent and local composition functions.

We next test dynamic convolutions on the mechanistic architecture design benchmark (MAD; Poli et al., 2024), a diagnostic benchmark designed to test the capabilities of different architectures. The results are shown in Figure 2, where we observe dynamic convolutions (also with $R = 16$) to perform well. The improvements are particularly pronounced on the Fuzzy Recall task, where the model must perform in-context recall in a setting where the keys and values consist of a variable number of tokens.

3.2 Language Modeling

We test whether augmenting Transformers (including Mixture of Experts variants) with dynamic short convolutions improves real-world language modeling. We then transfer the same recipe to two strong linear attention variants (Gated DeltaNet (Schlag et al., 2021; Yang et al., 2024, 2025b) and Mamba-2 (Dao & Gu, 2024)).

Experimental Setup. We train all models in the `lm-engine` codebase (Mishra, 2024) on the Nemotron-CC corpus (Su et al., 2025) tokenized with the Granite-4 BPE tokenizer (vocabulary 100,352). All runs use sequence length 4096, RMSNorm (Zhang & Sennrich, 2019), SwiGLU MLPs (Shazeer, 2020), RoPE (Su et al., 2021), and a Llama-style pre-norm block.

For optimization we use AdamW (Loshchilov & Hutter, 2019) with peak learning rate 3×10^{-4}, weight decay 0.1, and learning rate scheduling with 10% warmup and cosine decay to zero. We train dense models at {150M, 300M, 600M, 1B, 2B} parameters, with a token-to-parameter ratio of approximately 50, i.e., 2.5× the compute-optimal recipe recommended by Hoffmann et al. (2022). Additionally, we train a 7B (1B active) parameter Mixture of Experts (MoE) model (Shazeer et al., 2017) on 100B tokens. A more detailed description of the hyperparameter setting can be found in Appendix C.

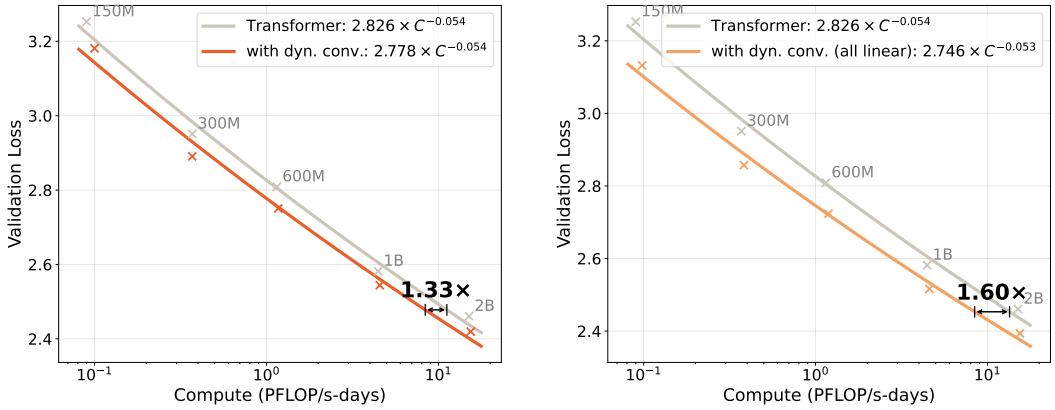


Figure 3: Scaling laws on Transformers with low-rank dynamic convolutions applied to the keys, queries, and values (left) and placed after every linear layer (right).

Scaling laws. We first study the scaling trends of Transformers with and without dynamic convolutions, where for the dynamic convolution we use the low-rank version with ranks $R = \{20, 26, 32, 42, 52\}$ for the $\{150M, 300M, 600M, 1B, 2B\}$-parameter models.[5] Figure 3 (left) shows the validation loss as a function of compute (see Appendix D for FLOP calculations). Fitting a curve to the results suggests that dynamic convolutions offer an approximate $1.33\times$ advantage over compute-matched Transformer baselines.

We also experiment with applying dynamic convolutions to *all linear layers*, instead of placing them only after the `qkv_projection`. To this end, we use the low-rank parameterization with rank $R = 16$ across the $\{150M, 300M, 600M, 1B, 2B\}$-parameter models. We find that placing dynamic short convolutions after every linear layer improves substantially over placing them on the queries, keys, and values alone. Fitting a scaling law for this variant (Figure 3 (right)) yields a $1.60\times$ compute advantage over compute-matched Transformers, up from $1.33\times$ for the $Q/K/V$-only placement.

Training throughput. Figure 4 shows that our dynamic convolution kernels are competitive with static convolution kernels. However, individual kernel efficiency does not always translate to end-to-end model efficiency. We integrate our dynamic short-convolution kernels into the `lm-engine` codebase and measure end-to-end training throughput on a single H100 80GB HBM3 GPU. Figure 4 reports tokens per second at the 300M and 2B parameter scales (sequence length 4096, bf16 precision, with `torch.compile` enabled). Both dynamic-convolution variants stay within 8% overhead compared to the Transformer baseline across configurations, while adding static convolutions gives a slowdown of around 6%. We expect more aggressive kernel fusion to close the gap for both static and dynamic variants. However, even with the current throughput penalty, our scaling law improvement ($1.33\times$) suggests a significant wall-clock time advantage.

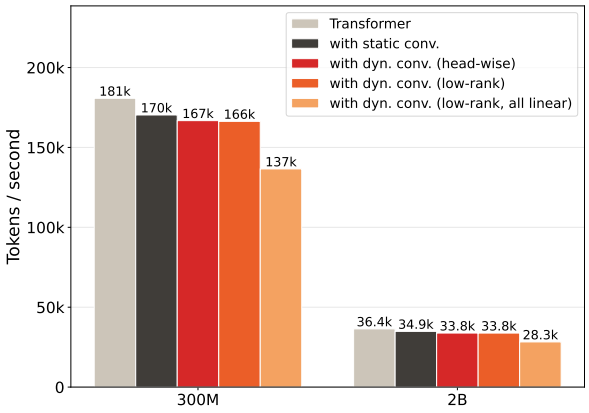


Figure 4: End-to-end training throughput measured on a single H100 80GB HBM3 GPU.

The all-linear variant results in a 22–25% reduction in end-to-end training throughput. Future work could explore fusing the dynamic convolution into the matmul epilogue to further reduce memory I/O. This may substantially decrease this overhead and yield an even larger wall-clock time advantage.

[5] These ranks are selected so that the low-rank version roughly matches the parameters of the head-wise version with head dimension $H = 32$.

Model	Train Tokens	Params	Conv. Location	Perplexity Nemo. ↓	Perplexity LAMB. ↓	Perplexity Wiki. ↓	Task Avg. ↑
MoE Transformer	100B	6.77B	–	9.86	11.55	13.27	62.46
w/ static conv.		6.77B	QKV	9.74	11.20	13.08	62.97
w/ dynamic conv. (head-wise)		6.80B	QKV	9.65	11.61	12.90	**63.43**
w/ dynamic conv. (low-rank)		6.80B	QKV	**9.58**	**10.92**	**12.77**	63.42
Transformer	100B	1.82B	–	11.71	17.28	15.86	58.35
w/ more params (wider FFN)		1.87B	–	11.67	18.49	15.72	58.23
w/ static conv.		1.83B	QKV	11.50	16.21	15.43	58.94
w/ static conv. (param. matched)		1.88B	QKV	11.44	15.61	15.33	58.97
w/ dynamic conv. (head-wise)		1.88B	QKV	11.41	16.34	15.23	58.46
w/ dynamic conv. (low-rank)		1.88B	QKV	11.24	15.43	14.98	59.70
w/ dynamic conv. (low-rank)		1.88B	all linear	**10.95**	**12.51**	**14.43**	**60.70**
Transformer	15B	305.2M	–	19.12	76.62	30.50	47.26
w/ more params (wider FFN)		311.5M	–	18.99	68.05	30.04	46.64
w/ static conv.		305.4M	QKV	18.66	69.64	29.50	46.86
w/ static conv. (param. matched)		311.8M	QKV	18.63	66.53	29.37	46.22
w/ dynamic conv. (head-wise)		311.7M	QKV	18.22	58.54	28.42	47.61
w/ dynamic conv. (low-rank)		311.8M	QKV	18.01	56.66	27.98	**48.90**
w/ dynamic conv. (low-rank)		319.0M	all linear	**17.42**	**46.13**	**26.78**	48.81
Gated DeltaNet (w/o conv.)	15B	305.2M	–	18.93	69.90	30.24	46.93
w/ static conv.		305.4M	QKV	18.75	67.63	29.82	46.76
w/ static conv. (param. matched)		309.6M	QKV	18.74	69.06	29.78	47.60
w/ dynamic conv. (head-wise)		309.6M	QKV	18.03	59.49	28.17	47.27
w/ dynamic conv. (low-rank)		309.5M	QKV	**17.95**	**50.56**	**27.95**	**49.22**
Mamba-2 (w/o conv.)	15B	306.2M	–	20.26	80.81	33.26	46.41
w/ static conv.		306.4M	QKV	19.30	83.50	31.32	45.78
w/ static conv. (param. matched)		309.8M	QKV	19.26	78.96	31.17	46.78
w/ dynamic conv. (head-wise)		309.8M	QKV	**18.69**	**65.80**	30.03	47.24
w/ dynamic conv. (low-rank)		309.8M	QKV	18.72	71.12	**29.90**	**47.34**

Table 1: Main evaluation results of Transformers, Transformers with Static Short Convolutions, and Transformers with Dynamic Short Convolutions. *Param. matched* static baselines widen the MLP intermediate dimension to exactly match the parameter count of the dynamic variants. *Task Avg.* averages 0-shot accuracy over 11 lm-eval-harness tasks.

Evaluation. We next perform downstream evaluations across several baselines. First, Transformer uses the same training recipe and architecture, but without any convolutional layers. Second, Transformer (more params) increases the MLP intermediate dimension to account for the additional parameters introduced through our dynamic convolutions. Third, we compare against a Transformer with static short convolutions on top of the queries, keys, and values, following the "Canon-B" setup from Allen-Zhu (2025). Finally, we compare dynamic short convolutions with both low-rank and head-wise parameterizations. We report perplexity on 25M tokens of held-out Nemotron-CC data, as well as on Wikitext-103 (Merity et al., 2016) and LAMBADA (Paperno et al., 2016). Additionally, we report zero-shot accuracy on various common-sense reasoning tasks via `lm-evaluation-harness` (Gao et al., 2024).

Table 1 reports results for models with (roughly) 300M (15B tokens), 2B (100B tokens), and 7B/1B active Mixture of Experts (100B tokens). Static convolutions generally improve upon ordinary Transformers, but both the head-wise and low-rank variants outperform static convolutions on perplexity and task accuracy at all parameter scales. The all-linear variant gives further gains. See Table 5 of the Appendix for the task performance broken down by benchmark.

We now analyze the capabilities of these models for in-context learning and retrieval on the RULER benchmark (Hsieh et al., 2024). RULER is generally a difficult benchmark for models trained at our scale. We analyze the RULER results for the strongest MoE models in Table 2. We find that

Table 2: Per-subtask RULER accuracy (%) at context length 4096.

Model	S1	S2	S3	MK1	MK2	MK3	MQ	MV	CWE	FWE	VT	Avg.
MoE Transformer	99.8	**100.0**	83.8	70.0	4.8	21.2	39.4	32.0	26.4	**43.3**	9.3	48.2
w/ static conv.	**100.0**	**100.0**	73.8	72.0	27.6	6.0	36.5	40.8	13.3	26.8	**23.9**	47.3
w/ dynamic conv. (head-wise)	**100.0**	99.8	81.8	**74.2**	**45.4**	**38.4**	37.4	40.5	15.1	16.3	8.1	50.6
w/ dynamic conv. (low-rank)	99.8	**100.0**	**93.0**	66.2	12.8	11.8	**48.2**	**50.4**	**31.1**	26.4	18.4	**50.7**

the dynamic convolutions perform particularly well on the multi-key (MK), multi-query (MQ), and multi-value (MV) subtasks within RULER, which makes sense given the ability to perform input-dependent local aggregations enabled by the dynamic convolutions. See Table 6 of the Appendix for the RULER results for all models.

Linear attention variants. Modern linear RNN architectures such as Mamba and DeltaNet already include static depthwise separable short convolutions on Q, K, V as part of their sequence mixer (Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2025b). We test whether replacing these static convolutions with our dynamic convolutions further improves the architecture. At the bottom of Table 1 we report the 300M/15B-token results for Mamba-2 (Dao & Gu, 2024) and Gated DeltaNet (Schlag et al., 2021; Yang et al., 2024, 2025b): the default (with static conv.), without any convolutions, and with two variants of our dynamic short convolutions. As expected, removing the static short convolutions increases perplexity on held-out training data. Replacing the static convolutions with their dynamic counterparts significantly improves performance in terms of perplexity across all datasets. Notably, Mamba-2 with dynamic convolutions performs about as well as Gated DeltaNet with static convolutions, which suggests that incorporating dynamic short convolutions may be more beneficial than redesigning the sequence mixer.

3.3 Ablations

We next perform a series of ablations on our architectural decisions. Here we work with the 300M models trained on 15B tokens, and report perplexity (PPL) on the Nemotron-CC corpus.

Width, head dimension, rank. Our main experiments used filter width W of 4, head-size H of 32, and for the low-rank version, rank R such that it is param-matched to the head-wise version. We perform a sweep of these hyperparameters. Table 3a shows the ablation results.

For width, we find that 3 or 4 is generally the sweet spot in terms of providing the best performance. This has generally been found to be the case for static convolutions as well. Widths beyond this sweet spot do not provide additional gains even though they add parameters. For head dimension in the head-wise variant, we used $H = 32$ for our experiments. Making the head dimension smaller does improve performance, but results in many additional parameters. For the low-rank variant, increasing the expressivity via increasing R unsurprisingly leads to improved performance, but at the cost of more parameters. Dynamic convolutions therefore provide another axis with which to trade off compute/parameters for performance. Overall, the results suggest that the low-rank parameterization with $R = 16$ offers a strong trade-off between performance and parameter count.

Layer placement. In our main experiments we placed dynamic convolutions on all components of attention, i.e., queries, keys, and values. We ablate this design choice by placing dynamic convolutions on different subsets of QKV. Table 3b shows that the largest single-projection placement gain comes from the value projection. Applying them to two projections improves performance further, while the best result is achieved when dynamic convolutions are applied to all three projections. We therefore use Q+K+V placement in our main experiments.

QK-norm Transformers. Our main experiments were conducted on Transformers without QK-norm. However, QK-norm (Dehghani et al., 2023) is becoming a popular part of recent frontier open-source LLMs (Team et al., 2025a; Yang et al., 2025a; Team et al., 2025b). Would dynamic convolutions be helpful for Transformers trained with QK-norm? We show the results on QK-norm Transformers in Table 3c, where we indeed find that dynamic convolutions continue to provide

Table 3: Ablations on the 300M models trained on 15B tokens, reporting Nemotron-CC perplexity. (a) Sweep over kernel width W, head size H, and rank R for dynamic convolutions on Q+K+V. (b) Placement of dynamic convolutions inside the attention layer (low-rank R=16, W=4). (c) QK-norm Transformers with and without convolutions.

(a) Width / head size / rank sweep.

Sweep	Params	PPL
Width W (low-rank, R=16)		
$W = 1$	306.8M	18.42
$W = 2$	307.6M	18.17
$W = 3$	308.5M	**18.08**
$W = 4$	309.3M	18.10
$W = 5$	310.1M	18.09
$W = 6$	311.0M	18.10
Head size H (head-wise, W=4)		
$H = 8$	330.5M	**18.03**
$H = 16$	317.9M	18.08
$H = 32$	311.7M	18.21
$H = 64$	308.5M	18.25
$H = 128$	306.9M	18.40
Rank R (low-rank, W=4)		
$R = 4$	306.3M	18.26
$R = 8$	307.3M	18.19
$R = 16$	309.3M	18.10
$R = 32$	313.2M	18.04
$R = 64$	321.1M	17.87
$R = 128$	336.8M	**17.85**

(b) Layer placement.

Placement	Params	PPL
Transformer (w/o conv.)	305.2M	19.12
Q only	306.5M	18.69
K only	306.5M	18.83
V only	306.5M	18.56
Q + K	307.9M	18.44
Q + V	307.9M	18.36
K + V	307.9M	18.35
Q + K + V	309.3M	**18.10**

(c) QK-norm Transformers.

Setup	Params	PPL
Transformer with QK-Norm	305.2M	18.69
w/ static conv.	305.4M	18.56
w/ dynamic conv. (head-wise)	311.5M	18.30
w/ dynamic conv. (low-rank)	311.6M	**17.95**

significant gains for Transformers trained with QK-norm. By contrast, static convolutions provide little benefit when combined with QK-norm in our experiments. For these experiments, we apply per-head RMSNorm to the queries and keys after the convolution (when used) and before RoPE.

4 Discussion and Limitations

Our results suggest that dynamic short convolutions are a useful primitive for improving Transformer-based language models. Unlike static short convolutions, which impose the same local aggregation rule at every position, dynamic convolutions allow each token to choose an input-dependent local composition function. The synthetic experiments support the utility of such a layer, showing gains on tasks that require resolving variable-length local structure before performing recall. The language-modeling results further suggest that these benefits are not limited to toy settings: dynamic convolutions improve dense Transformers, MoE Transformers, and linear attention variants, and the improvements persist under parameter-matched and compute-matched comparisons.

On the limitations side, our scaling study reaches 2B dense parameters and a 7B-parameter MoE with 1B active parameters, which is sufficient to establish consistent trends but does not by itself demonstrate that the same compute advantage will hold at frontier scale, under substantially longer training, or across different data mixtures and tokenizers. Moreover, while our Triton implementations make dynamic convolutions practical on H100 GPUs, additional engineering would be needed to fully optimize inference, support a broader range of hardware, and fuse the dynamic and static components more aggressively. Finally, we only explore a small subset of possible parameterizations, placements, and kernel widths.

5 Related Work

Convolutional networks for sequence modeling. Convolutional networks were a popular class of neural sequence models before the rise of attention-based architectures. Early neural NLP systems used one-dimensional convolutions over word embeddings for tagging and sentence classification

(Collobert & Weston, 2008; Collobert et al., 2011; Kalchbrenner et al., 2014; Kim, 2014). Later work scaled convolutional sequence models to machine translation and language modeling using dilations, gating, and stacked convolutional blocks (Kalchbrenner et al., 2016; Gehring et al., 2017; Dauphin et al., 2017; Bai et al., 2018). More recently, long-convolution models such as S4 and Hyena have revisited convolutions as subquadratic alternatives to attention by using implicit long filters and gating (Gu et al., 2022; Poli et al., 2023).

Dynamic convolutions. Dynamic convolutions have a long history in vision models (Jia et al., 2016; Yang et al., 2019; Li et al., 2019; Chen et al., 2020; Zhou et al., 2021; Li et al., 2022). Their use in language processing and sequence modeling has been more limited. Wu et al. (2019) introduced lightweight and dynamic convolutions that predict convolution kernels from the current time step as efficient alternatives to self-attention, and ConvBERT uses span-based dynamic convolutions to replace a subset of BERT attention heads for local dependency modeling (Jiang et al., 2020).

Convolutions in modern Transformers and linear RNNs. Since the introduction of Transformers, there have been works that combine attention with static convolutions. Conformer combines self-attention with convolutional modules for speech recognition (Gulati et al., 2020), and Lite Transformer allocates separate branches to long-range attention and short-range convolution (Wu et al., 2020). In language modeling, Primer found through architecture search that adding depthwise convolutions after the query, key, and value projections substantially improves training efficiency (So et al., 2021); more recent work emphasizes horizontal information flow among neighboring tokens across multiple sequence architectures (Allen-Zhu, 2025). Static short convolutions are also standard in recent linear RNNs (Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2024, 2025b). Recent work (Gu et al., 2026) uses dynamic convolutions just in the value layer when converting pretrained softmax attention layers to linear attention layers. Our results show that dynamic convolutions improve upon static convolutions in both Transformers and linear RNNs when pretrained from scratch.

6 Conclusion

We introduced dynamic short convolutions as an input-dependent, locality-biased primitive for improving Transformer-based language models. By generating convolutional filters from the current hidden state, dynamic convolutions extend static short convolutions with greater expressivity while retaining efficient local sequence mixing. Across synthetic associative-recall tasks, dense language-modeling experiments from 150M to 2B parameters, a 7B-parameter MoE model, and two linear attention architectures, dynamic convolutions consistently improve over both standard Transformers and static-convolution baselines. Our scaling-law analysis indicates a meaningful compute advantage, and our custom Triton kernels show that these gains can be obtained with modest end-to-end training overhead. Taken together, these results suggest that dynamic short convolutions are a scalable and practical architectural primitive, complementary to attention and promising for LLMs.

Acknowledgments and Disclosure of Funding

We would like to thank Han Guo, Assaf Ben-Kish, and Yanick Schimpf for valuable discussions and feedback.

This study was supported by MIT-IBM Watson AI Lab and the AI2050 program at Schmidt Sciences (Grant G-25-67980).

Competing interests. The authors declare no competing financial interests.

References

Afzal, A., Bick, A., Xing, E. P., Cevher, V., and Gu, A. Raven: High-recall sequence modeling with sparse memory routing. 2026.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Allen-Zhu, Z. Physics of language models: Part 4.1, architecture design and the magic of canon layers. *arXiv preprint arXiv:2512.17351*, 2025.

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C. K., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Zhang, S., Suo, M., Tillet, P., Zhao, X., Wang, E., Zhou, K., Zou, R., Wang, X., Mathews, A., Wen, W., Chanan, G., Wu, P., and Chintala, S. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 929–947. Association for Computing Machinery, 2024. doi: 10.1145/3620665.3640366.

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/abs/1607.06450.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. URL https://arxiv.org/abs/1409.0473.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., and Liu, Z. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 2011.

Dao, T. and Gu, A. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., Jenatton, R., Beyer, L., Tschannen, M., Arnab, A., Wang, X., Riquelme Ruiz, C., Minderer, M., Puigcerver, J., Evci, U., Kumar, M., Steenkiste, S. V., Elsayed, G. F., Mahendran, A., Yu, F., Oliver, A., Huot, F., Bastings, J., Collier, M., Gritsenko, A. A., Birodkar, V., Vasconcelos, C. N., Tay, Y., Mensink, T., Kolesnikov, A., Pavetic, F., Tran, D., Kipf, T., Lucic, M., Zhai, X., Keysers, D., Harmsen, J. J., and Houlsby, N. Scaling vision transformers to 22 billion parameters. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7480–7512. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

Elman, J. L. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. doi: 10.1007/BF00344251.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. In *International conference on machine learning*, pp. 1243–1252. PMLR, 2017.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. In *Proceedings of CoLM*, 2024.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *Proceedings of ICLR*, 2022.

Gu, Y., Hu, Q., Xi, H., Chen, J., Yang, S., Han, S., and Cai, H. Jet-nemotron: Efficient language model with post neural architecture search. *Advances in Neural Information Processing Systems*, 38:47191–47218, 2026.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. Conformer: Convolution-augmented transformer for speech recognition. In *Interspeech*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? 2024.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456. PMLR, 2015. URL https://proceedings.mlr.press/v37/ioffe15.html.

Jia, X., De Brabandere, B., Tuytelaars, T., and Van Gool, L. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, volume 29, pp. 667–675, 2016.

Jiang, Z., Yu, W., Zhou, D., Chen, Y., Feng, J., and Yan, S. Convbert: Improving bert with span-based dynamic convolution. In *Advances in Neural Information Processing Systems*, 2020.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 655–665, 2014.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Kim, Y. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1746–1751, 2014.

Lahoti, A., Li, K. Y., Chen, B., Wang, C., Bick, A., Kolter, J. Z., Dao, T., and Gu, A. Mamba-3: Improved sequence modeling using state space principles. 2026.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Li, C., Zhou, A., and Yao, A. Omni-dimensional dynamic convolution. *arXiv preprint arXiv:2209.07947*, 2022.

Li, X., Wang, W., Hu, X., and Yang, J. Selective kernel networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 510–519, 2019.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Mishra, M. Lm engine: A hyper-optimized library for pretraining and finetuning, 2024. URL https://github.com/open-lm-engine/lm-engine.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pp. 28043–28078. PMLR, 2023.

Poli, M., Thomas, A. W., Nguyen, E., Ponnusamy, P., Deiseroth, B., Kersting, K., Suzuki, T., Hie, B., Ermon, S., Ré, C., et al. Mechanistic design and scaling of hybrid architectures. *arXiv preprint arXiv:2403.17844*, 2024.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

Schlag, I., Irie, K., and Schmidhuber, J. Linear Transformers Are Secretly Fast Weight Programmers. In *Proceedings of ICML*, 2021.

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

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

So, D., Mänke, W., Liu, H., Dai, Z., Shazeer, N., and Le, Q. V. Primer: Searching for efficient transformers for language modeling. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26053–26066, 2021.

Su, D., Kong, K., Lin, Y., Jennings, J., Norick, B., Kliegl, M., Patwary, M., Shoeybi, M., and Catanzaro, B. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2459–2475, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.123. URL https://aclanthology.org/2025.acl-long.123/.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Team, ., Zeng, A., Lv, X., Zheng, Q., Hou, Z., Chen, B., Xie, C., Wang, C., Yin, D., Zeng, H., Zhang, J., Wang, K., Zhong, L., Liu, M., Lu, R., Cao, S., Zhang, X., Huang, X., Wei, Y., Cheng, Y., An, Y., Niu, Y., Wen, Y., Bai, Y., Du, Z., Wang, Z., Zhu, Z., Zhang, B., Wen, B., Wu, B., Xu, B., Huang, C., Zhao, C., Cai, C., Yu, C., Li, C., Ge, C., Huang, C., Zhang, C., Xu, C., Zhu, C., Li, C., Yin, C., Lin, D., Yang, D., Jiang, D., Ai, D., Zhu, E., Wang, F., Pan, G., Wang, G., Sun, H., Li, H., Li, H., Hu, H., Zhang, H., Peng, H., Tai, H., Zhang, H., Wang, H., Yang, H., Liu, H., Zhao, H., Liu, H., Yan, H., Liu, H., Chen, H., Li, J., Zhao, J., Ren, J., Jiao, J., Zhao, J., Yan, J., Wang, J., Gui, J., Zhao, J., Liu, J., Li, J., Li, J., Lu, J., Wang, J., Yuan, J., Li, J., Du, J., Du, J., Liu, J., Zhi, J., Gao, J., Wang, K., Yang, L., Xu, L., Fan, L., Wu, L., Ding, L., Wang, L., Zhang, M., Li, M., Xu, M., Zhao, M., Zhai, M., Du, P., Dong, Q., Lei, S., Tu, S., Yang, S., Lu, S., Li, S., Li, S., Shuang-Li, Yang, S., Yi, S., Yu, T., Tian, W., Wang, W., Yu, W., Tam, W. L., Liang, W., Liu, W., Wang, X., Jia, X., Gu, X., Ling, X., Wang, X., Fan, X., Pan, X., Zhang, X., Zhang, X., Fu, X., Zhang, X., Xu, Y., Wu, Y., Lu, Y., Wang, Y., Zhou, Y., Pan, Y., Zhang, Y., Wang, Y., Li, Y., Su, Y., Geng, Y., Zhu, Y., Yang, Y., Li, Y., Wu, Y., Li, Y., Liu, Y., Wang, Y., Li, Y., Zhang, Y., Liu, Z., Yang, Z., Zhou, Z., Qiao, Z., Feng, Z., Liu, Z., Zhang, Z., Wang, Z., Yao, Z., Wang, Z., Liu, Z., Chai, Z., Li, Z., Zhao, Z., Chen, W., Zhai, J., Xu, B., Huang, M., Wang, H., Li, J., Dong, Y., and Tang, J. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025a. URL https://arxiv.org/abs/2508.06471.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., Mesnard, T., Cideron, G., bastien Grill, J., Ramos, S., Yvinec, E., Casbon, M., Pot, E., Penchev, I., Liu, G., Visin, F., Kenealy, K., Beyer, L., Zhai, X., Tsitsulin, A., Busa-Fekete, R., Feng, A., Sachdeva, N., Coleman, B., Gao, Y., Mustafa, B., Barr, I., Parisotto, E., Tian, D., Eyal, M., Cherry, C., Peter, J.-T., Sinopalnikov, D., Bhupatiraju, S., Agarwal, R., Kazemi, M., Malkin, D., Kumar, R., Vilar, D., Brusilovsky, I., Luo, J., Steiner, A., Friesen, A., Sharma, A., Sharma, A., Gilady, A. M., Goedeckemeyer, A., Saade, A., Feng, A., Kolesnikov, A., Bendebury, A., Abdagic, A., Vadi, A., György, A., Pinto, A. S., Das, A., Bapna, A., Miech, A., Yang, A., Paterson, A., Shenoy, A., Chakrabarti, A., Piot, B., Wu, B., Shahriari, B., Petrini, B., Chen, C., Lan, C. L., Choquette-Choo, C. A., Carey, C., Brick, C., Deutsch, D., Eisenbud, D., Cattle, D., Cheng, D., Paparas, D., Sreepathihalli, D. S., Reid, D., Tran, D., Zelle, D., Noland, E., Huizenga, E., Kharitonov, E., Liu, F., Amirkhanyan, G., Cameron, G., Hashemi, H., Klimczak-Plucińska, H., Singh, H., Mehta, H., Lehri, H. T., Hazimeh, H., Ballantyne, I., Szpektor, I., Nardini, I., Pouget-Abadie, J., Chan, J., Stanton, J., Wieting, J., Lai, J., Orbay, J., Fernandez, J., Newlan, J., yeong Ji, J., Singh, J., Black, K., Yu, K., Hui, K., Vodrahalli, K., Greff, K., Qiu, L., Valentine, M., Coelho, M., Ritter, M., Hoffman, M., Watson, M., Chaturvedi, M., Moynihan, M., Ma, M., Babar, N., Noy, N., Byrd, N., Roy, N., Momchev, N., Chauhan, N., Sachdeva, N., Bunyan, O., Botarda, P., Caron, P., Rubenstein, P. K., Culliton, P., Schmid, P., Sessa, P. G., Xu, P., Stanczyk, P., Tafti, P., Shivanna, R., Wu, R., Pan, R., Rokni, R., Willoughby, R., Vallu, R., Mullins, R., Jerome, S., Smoot, S., Girgin, S., Iqbal, S., Reddy, S., Sheth, S., Põder, S., Bhatnagar, S., Panyam, S. R., Eiger, S., Zhang, S., Liu, T., Yacovone, T., Liechty, T., Kalra, U., Evci, U., Misra, V., Roseberry, V., Feinberg, V., Kolesnikov, V., Han, W., Kwon, W., Chen, X., Chow, Y., Zhu, Y., Wei, Z., Egyed, Z., Cotruta, V., Giang, M., Kirk, P., Rao, A., Black, K., Babar, N., Lo, J., Moreira, E., Martins, L. G., Sanseviero, O., Gonzalez, L., Gleicher, Z., Warkentin, T., Mirrokni,

V., Senter, E., Collins, E., Barral, J., Ghahramani, Z., Hadsell, R., Matias, Y., Sculley, D., Petrov, S., Fiedel, N., Shazeer, N., Vinyals, O., Dean, J., Hassabis, D., Kavukcuoglu, K., Farabet, C., Buchatskaya, E., Alayrac, J.-B., Anil, R., Dmitry, Lepikhin, Borgeaud, S., Bachem, O., Joulin, A., Andreev, A., Hardin, C., Dadashi, R., and Hussenot, L. Gemma 3 technical report, 2025b. URL https://arxiv.org/abs/2503.19786.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.

Wu, F., Fan, A., Baevski, A., Dauphin, Y. N., and Auli, M. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*, 2019.

Wu, Z., Liu, Z., Lin, J., Lin, Y., and Han, S. Lite transformer with long-short range attention. In *International Conference on Learning Representations*, 2020.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10524–10533, 2020.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Yang, B., Bender, G., Le, Q. V., and Ngiam, J. Condconv: Conditionally parameterized convolutions for efficient inference. In *Advances in Neural Information Processing Systems*, 2019.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length. In *Proceedings of NeurIPS*, 2024.

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving mamba2 with delta rule. In *Proceedings of ICLR*, 2025b.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32*, 2019.

Zhou, J., Jampani, V., Pi, Z., Liu, Q., and Yang, M.-H. Decoupled dynamic filter networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6647–6656, 2021.

A Kernel Benchmark Setup

All measurements were performed on a single NVIDIA H100 SXM5 80GB HBM3 GPU, which has a theoretical peak HBM bandwidth of 3.35 TB/s and peak BF16 matrix-multiply throughput of 989 TFLOPs. Our benchmarking environment used PyTorch 2.12.0 nightly with CUDA 13.0, cuDNN 9.2.0, Triton 3.7.0, and `causal_conv1d` 1.6.1.

For both the head-wise and the low-rank variant we implement five mathematically equivalent variants in PyTorch. Listing 1 contains the five variants of the head-wise implementation, and Listing 2 contains the five variants of the low-rank implementation. Each variant is benchmarked in PyTorch eager mode and under all four `torch.compile` modes (`default`, `reduce-overhead`, `max-autotune`, `max-autotune-no-cudagraphs`). All `torch.compile` runs are benchmarked with `fullgraph=True`, `dynamic=False`, and Dynamo's recompile limit increased to 64. We report the fastest formulation-mode combination in the main figure. Table 4 contains the winning variant for torch/torch.compile in each setting.

```python
def hw_loop_pad(x, weight):
    B, T, D = x.shape
    _, _, G, W = weight.shape
    H = D // G
    x_h = x.view(B, T, G, H)
    out = torch.zeros_like(x_h)
    for w in range(W):
        x_shift = F.pad(x_h, (0, 0, 0, 0, w, 0))[:, :T]
        out = out + weight[:, :, :, w:w + 1] * x_shift
    return out.reshape(B, T, D)


def hw_unfold(x, weight):
    B, T, D = x.shape
    _, _, G, W = weight.shape
    H = D // G
    x_h = x.view(B, T, G, H)
    x_pad = F.pad(x_h, (0, 0, 0, 0, W - 1, 0))
    windows = x_pad.unfold(1, W, 1).flip(-1)
    return (windows * weight.unsqueeze(-2)).sum(-1).reshape(B, T, D)


def hw_einsum(x, weight):
    B, T, D = x.shape
    _, _, G, W = weight.shape
    H = D // G
    x_h = x.view(B, T, G, H)
    x_pad = F.pad(x_h, (0, 0, 0, 0, W - 1, 0))
    windows = x_pad.unfold(1, W, 1).flip(-1)
    return torch.einsum('btghw,btgw->btgh', windows,
        weight).reshape(B, T, D)


def hw_stack(x, weight):
    B, T, D = x.shape
    _, _, G, W = weight.shape
    H = D // G
    x_h = x.view(B, T, G, H)
    shifts = [x_h]
    for w in range(1, W):
        zero = x_h.new_zeros(B, w, G, H)
        shifts.append(torch.cat([zero, x_h[:, :T - w]], dim=1))
    stacked = torch.stack(shifts, dim=-1)
    return (stacked * weight.unsqueeze(-2)).sum(-1).reshape(B, T, D)


def hw_bmm(x, weight):
    B, T, D = x.shape
    _, _, G, W = weight.shape
    H = D // G
    x_h = x.view(B, T, G, H)
    x_pad = F.pad(x_h, (0, 0, 0, 0, W - 1, 0))
    windows = x_pad.unfold(1, W, 1).flip(-1)
    BT_G = B * T * G
    out = torch.bmm(
        windows.reshape(BT_G, H, W),
        weight.reshape(BT_G, W, 1),
    )
    return out.reshape(B, T, D)
```

Listing 1: Five mathematically equivalent PyTorch implementations of the head-wise dynamic convolution. Here, H denotes the head size and $G = D/H$ the number of groups, consistent with the main-text notation.

```python
def lr_materialize_loop(x, z, U):
    B, T, D = x.shape
    R, WD = U.shape
    W = WD // D
    weight = (z @ U).view(B, T, W, D)
    out = torch.zeros_like(x)
    for w in range(W):
        x_shift = F.pad(x, (0, 0, w, 0))[:, :T]
        out = out + weight[:, :, w, :] * x_shift
    return out


def lr_materialize_unfold(x, z, U):
    B, T, D = x.shape
    R, WD = U.shape
    W = WD // D
    Ur = U.view(R, W, D)
    weight = torch.einsum('btr,rwd->btwd', z, Ur)
    x_pad = F.pad(x, (0, 0, W - 1, 0))
    windows = x_pad.unfold(1, W, 1).flip(-1)
    return (windows * weight.permute(0, 1, 3, 2)).sum(-1)


def lr_unfold_einsum(x, z, U):
    B, T, D = x.shape
    R, WD = U.shape
    W = WD // D
    Ur = U.view(R, W, D)
    x_pad = F.pad(x, (0, 0, W - 1, 0))
    windows = x_pad.unfold(1, W, 1).flip(-1)
    return torch.einsum('btdw,btr,rwd->btd', windows, z, Ur)


def lr_fused_per_tap(x, z, U):
    B, T, D = x.shape
    R, WD = U.shape
    W = WD // D
    Uw = U.view(R, W, D)
    out = torch.zeros_like(x)
    for w in range(W):
        weight_w = z @ Uw[:, w, :]
        x_shift = F.pad(x, (0, 0, w, 0))[:, :T]
        out = out + weight_w * x_shift
    return out


def lr_static_conv_per_rank(x, z, U):
    B, T, D = x.shape
    R, WD = U.shape
    W = WD // D
    Ur = U.view(R, W, D)
    Uf = Ur.flip(1)
    x_bdt = x.transpose(1, 2).contiguous()
    x_pad = F.pad(x_bdt, (W - 1, 0))
    out = torch.zeros(B, T, D, dtype=x.dtype, device=x.device)
    for r in range(R):
        weight_r = Uf[r].T.unsqueeze(1).contiguous()
        conv_r = F.conv1d(x_pad, weight_r, groups=D)
        out = out + z[:, :, r:r + 1] * conv_r.transpose(1, 2)
    return out
```

Listing 2: Five mathematically equivalent PyTorch implementations of the low-rank dynamic convolution. Latency includes the second projection of the low-rank factorization.

We benchmark using Triton's `triton.testing.do_bench` with 500ms warmup and 3000ms measurement window, and record the median. We repeat this 5 times and report the run with the lowest median. We time the forward and the forward+backward independently and report the difference of the medians as the backward latency.

Table 4: Per-configuration latency at B=4, T=4096, D=2048, W=4, BF16. All kernels were tested on $B \times T \times D$ activations with the D dimension contiguous in memory. `torch.compile` cells are labeled (`<variant>, <Inductor mode>`). The last row implements a static convolution (W=4).

Configuration	Implementation	fwd (ms)	bwd (ms)	fwd+bwd (ms)
head-wise: H=1	triton	**0.140**	**0.243**	**0.382**
	torch (unfold)	1.022	1.884	2.906
	torch.compile (stack, max-autotune-no-cudagraphs)	0.330	0.367	0.697
head-wise: H=4	triton	**0.073**	**0.111**	**0.184**
	torch (unfold)	1.033	2.484	3.517
	torch.compile (stack, max-autotune-no-cudagraphs)	0.272	0.211	0.484
head-wise: H=16	triton	**0.055**	**0.088**	**0.143**
	torch (unfold)	1.029	2.516	3.545
	torch.compile (loop_pad, max-autotune-no-cudagraphs)	0.266	0.155	0.421
low-rank: R=16	triton	**0.071**	**0.171**	**0.242**
	torch (materialize_unfold)	0.982	1.729	2.711
	torch.compile (materialize_loop, max-autotune-no-cudagraphs)	0.435	0.511	0.946
static (cuda)	causal_conv1d	**0.056**	**0.105**	**0.161**

B Full Evaluation Results

This section contains the per-task breakdowns for the downstream evaluations summarized in Table 1 of the main paper. The zero-shot common-sense reasoning results are given in Table 5. For ARC-C/E, HellaSwag, OpenBookQA, PIQA, and SciQ we report `acc_norm`, and `acc` for the remaining tasks. Results for all 11 non-QA subtasks of RULER (single/multi-key needle-in-a-haystack, multi-query/value NIAH, common-words extraction, frequent-words extraction, variable tracking) at context length 4096 are provided in Table 6.

Table 5: Per-task 0-shot accuracy on the lm-eval-harness suite. For ARC-C/E, HellaSwag, OBQA, PIQA, SciQ we report `acc_norm`, and for BoolQ, COPA, LAMBADA, RACE, WinoGrande we report `acc`. *Avg.* is the mean of all tasks.

Model	**Params**	**ARC-C**	**ARC-E**	**BoolQ**	**COPA**	**Hella.**	**LAMB.**	**OBQA**	**PIQA**	**RACE**	**SciQ**	**WinoG.**	**Avg.**
MoE Transformer (*100B Tokens*)	6.77B	44.54	72.73	65.29	80.00	68.08	49.66	41.00	77.69	**36.75**	90.80	60.54	62.46
w/ static conv.	6.77B	44.20	73.40	65.66	**82.00**	68.60	50.75	41.60	77.42	36.27	91.90	60.85	62.97
w/ dynamic conv. (head-wise)	6.80B	**45.14**	**74.20**	67.46	80.00	69.12	**51.31**	**43.20**	78.02	35.02	91.10	**63.14**	**63.43**
w/ dynamic conv. (low-rank)	6.80B	44.88	74.03	**68.13**	81.00	**69.22**	51.27	42.80	**78.51**	35.69	**92.20**	59.91	63.42
Transformer (*100B Tokens*)	1.82B	38.14	68.31	60.49	78.00	60.49	44.36	38.60	74.86	34.83	87.00	56.75	58.35
w/ more params	1.87B	37.54	67.85	**63.67**	77.00	60.40	43.51	38.40	74.76	34.74	86.20	56.51	58.23
w/ static conv.	1.83B	37.97	68.27	61.96	**79.00**	61.69	45.06	38.60	74.65	34.55	89.50	57.14	58.94
w/ dynamic conv. (head-wise)	1.88B	38.82	68.69	59.14	75.00	61.80	45.55	37.80	75.35	34.64	88.70	57.62	58.46
w/ dynamic conv. (low-rank)	1.88B	39.16	**69.49**	63.64	**79.00**	62.57	46.09	38.40	75.46	**35.79**	88.70	58.41	59.70
w/ dynamic conv. (all linear)	1.88B	**40.44**	69.36	63.18	78.00	**64.73**	**48.67**	**41.00**	**76.01**	34.83	**91.00**	**60.46**	**60.70**
Transformer (*15B Tokens*)	305.2M	26.79	52.61	**60.52**	63.00	37.40	26.96	30.80	66.27	30.33	73.60	51.62	47.26
w/ more params	311.5M	26.45	51.47	52.23	63.00	37.56	29.17	32.00	**68.34**	29.47	73.10	50.20	46.64
w/ static conv.	305.4M	27.22	51.18	53.79	62.00	38.35	28.66	31.20	66.76	30.33	75.60	50.36	46.86
w/ dynamic conv. (head-wise)	311.7M	27.05	51.01	54.86	67.00	38.93	30.12	31.80	66.70	**31.20**	74.30	50.75	47.61
w/ dynamic conv. (low-rank)	311.8M	**27.90**	52.57	58.01	**73.00**	39.90	29.77	32.00	67.03	30.81	76.00	50.91	**48.90**
w/ dynamic conv. (all linear)	319.0M	27.56	**54.46**	56.18	68.00	**41.02**	**30.51**	**32.20**	67.68	31.10	**76.10**	**52.09**	48.81
Gated DeltaNet (w/o conv.)	305.2M	26.96	51.35	50.46	69.00	38.92	26.26	**31.60**	66.70	29.38	72.50	**53.12**	46.93
w/ static conv.	305.4M	27.22	50.76	53.24	66.00	38.93	26.57	29.80	66.43	28.23	75.70	51.46	46.76
w/ dynamic conv. (head-wise)	309.6M	26.71	53.37	47.89	**70.00**	40.27	26.84	31.20	67.36	29.00	75.50	51.78	47.27
w/ dynamic conv. (low-rank)	309.5M	**27.90**	**53.49**	**58.96**	**70.00**	**40.67**	**30.29**	31.40	**67.74**	**31.10**	**77.30**	52.57	**49.22**
Mamba-2 (w/o conv.)	306.2M	26.96	49.49	57.80	70.00	36.72	24.08	30.60	66.65	29.19	70.10	48.93	46.41
w/ static conv.	306.4M	25.94	**52.19**	48.84	65.00	38.67	23.97	31.40	66.87	28.33	72.50	49.88	45.78
w/ dynamic conv. (head-wise)	309.8M	**27.47**	50.29	**58.56**	66.00	39.09	26.26	31.60	**67.74**	27.85	**74.80**	**49.96**	47.24
w/ dynamic conv. (low-rank)	309.8M	25.94	51.43	54.71	**71.00**	**39.10**	**26.51**	**32.40**	65.89	**29.76**	74.40	49.57	**47.34**

Table 6: Per-subtask RULER accuracy at context length 4096.

Model	Params	S1	S2	S3	MK1	MK2	MK3	MQ	MV	CWE	FWE	VT	Avg.
MoE Transformer (*100B Tokens*)	6.77B	99.8	**100.0**	83.8	70.0	4.8	21.2	39.4	32.0	26.4	**43.3**	9.3	48.2
w/ static conv.	6.77B	**100.0**	**100.0**	73.8	72.0	27.6	6.0	36.5	40.8	13.3	26.8	**23.9**	47.3
w/ dynamic conv. (head-wise)	6.80B	**100.0**	99.8	81.8	**74.2**	**45.4**	**38.4**	37.4	40.5	15.1	16.3	8.1	50.6
w/ dynamic conv. (low-rank)	6.80B	99.8	**100.0**	**93.0**	66.2	12.8	11.8	**48.2**	**50.4**	**31.1**	26.4	18.4	**50.7**
Transformer (*100B Tokens*)	1.82B	**100.0**	**100.0**	80.8	57.8	1.2	3.8	39.6	37.4	10.5	32.1	4.0	42.5
w/ more params	1.87B	**100.0**	97.8	71.6	59.8	0.8	2.0	30.6	35.0	21.5	25.9	3.3	40.8
w/ static conv.	1.83B	99.4	94.8	79.6	68.0	3.0	4.0	**48.2**	**49.5**	6.1	**37.9**	**20.0**	**46.4**
w/ dynamic conv. (head-wise)	1.88B	**100.0**	**100.0**	**95.6**	61.0	**7.4**	6.0	24.9	28.6	13.3	36.3	8.7	43.8
w/ dynamic conv. (low-rank)	1.88B	**100.0**	99.8	80.6	52.6	3.0	3.2	18.0	10.4	25.3	31.9	16.2	40.1
w/ dynamic conv. (all linear)	1.88B	**100.0**	**100.0**	70.8	**77.6**	6.0	**8.4**	33.8	34.9	**30.4**	**37.9**	6.9	46.1
Transformer (*15B Tokens*)	305.2M	67.2	54.2	57.0	31.4	0.4	0.0	18.5	16.4	3.3	0.2	0.0	22.6
w/ more params	311.5M	80.2	66.0	41.2	32.0	0.0	0.0	20.0	19.8	5.6	1.2	0.0	24.2
w/ static conv.	305.4M	78.8	41.0	45.2	21.2	0.0	0.4	12.0	12.3	5.7	**2.3**	0.0	19.9
w/ dynamic conv. (head-wise)	311.7M	97.0	69.4	68.0	31.6	0.2	**1.0**	**22.7**	**21.3**	0.0	1.4	0.5	28.5
w/ dynamic conv. (low-rank)	311.8M	68.4	76.8	**74.4**	33.8	**1.0**	**1.0**	16.9	14.2	**11.6**	2.2	0.0	27.3
w/ dynamic conv. (all linear)	319.0M	**100.0**	**84.8**	73.0	**48.4**	0.2	0.2	10.8	12.4	1.7	0.0	**1.7**	**30.3**
Gated DeltaNet (w/o conv.)	305.2M	90.6	35.8	22.2	**19.6**	0.0	**0.0**	**16.4**	11.1	2.5	0.7	0.1	**18.1**
w/ static conv.	305.4M	99.2	33.0	4.8	19.2	0.0	**0.0**	2.8	2.8	0.9	**5.5**	**5.0**	15.7
w/ dynamic conv. (head-wise)	309.6M	**100.0**	34.2	**31.4**	11.6	0.0	**0.0**	12.2	4.0	0.3	1.9	3.8	**18.1**
w/ dynamic conv. (low-rank)	309.5M	99.8	**36.0**	7.2	18.4	**0.2**	**0.0**	9.7	**16.4**	**5.3**	2.9	2.1	18.0
Mamba-2 (w/o conv.)	306.2M	7.6	1.8	2.8	12.6	**0.0**	**0.0**	7.0	7.2	**2.5**	1.3	**0.0**	3.9
w/ static conv.	306.4M	15.6	2.4	0.4	6.2	**0.0**	**0.0**	2.3	0.7	0.9	0.0	**0.0**	2.6
w/ dynamic conv. (head-wise)	309.8M	30.2	**17.4**	18.4	**21.2**	**0.0**	**0.0**	11.9	9.2	0.6	2.2	**0.0**	10.1
w/ dynamic conv. (low-rank)	309.8M	**50.8**	14.8	**35.2**	21.0	**0.0**	**0.0**	**17.8**	**12.8**	0.4	**3.0**	**0.0**	**14.2**

C Detailed Experimental Setup

All models are trained in the `lm-engine` codebase (Mishra, 2024) on the Nemotron-CC corpus (Su et al., 2025) tokenized with the Granite-4 BPE tokenizer (vocabulary 100,352). All runs use sequence length $L = 4096$, RMSNorm (Zhang & Sennrich, 2019) with $\varepsilon = 10^{-5}$, SwiGLU MLPs (Shazeer, 2020), RoPE (Su et al., 2021) on the full head dimension, untied input/output embeddings, no biases, and no dropout. For optimization we use AdamW (Loshchilov & Hutter, 2019) with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\varepsilon = 10^{-10}$, weight decay 0.1, and peak learning rate 3×10^{-4} with 10% warm-up and cosine decay to zero. Training uses bf16 mixed precision and `torch.compile` (Ansel et al., 2024).

Weights are initialized from $\mathcal{N}(0, 0.02^2)$ for all linear layers. Static convolution weights are initialized to $\mathcal{U}(-1/\sqrt{W},\ 1/\sqrt{W})$ per element (i.e., the default for `nn.Conv1d`). For the low-rank variant of dynamic convolutions we zero-initialize the second projection of the low-rank factorization and add a bias term to this projection, initialized to $\mathcal{U}(-1/\sqrt{W},\ 1/\sqrt{W})$ per element. We match this for the head-wise variant by zero-initializing the dynamic projection and adding a per-channel bias with the same Kaiming-uniform initialization. Through this, our dynamic convolutions match a static depthwise convolution at initialization.

Per-scale model architecture, batch size, and hardware are listed in Table 7. Dense model experiments use a token-to-parameter ratio of $\sim$50, and the 7B MoE uses 128 experts with top-8 routing, 256 expert intermediate, and 1024 shared-MLP intermediate. For QKV placement the low-rank dynamic-convolution ranks R are chosen so that the parameter count of the low-rank variant roughly matches the head-wise variant at head dimension $H = 32$. For all-linear placement we set the rank to $R = 16$. Convolution kernel width is $W = 4$ throughout.

Table 7: Architecture and training hyperparameters across model scales. "mbs" is micro-batch size per device, "gas" is gradient-accumulation steps. All runs use NVIDIA H100 80GB HBM3 GPUs.

Scale	Architecture				Optimization						Hardware	
	Layers	d_{model}	Heads	MLP int.	mbs	gas	Tot. bs.	Eff. bs. (tok)	Steps	Tokens	GPUs	Nodes
150M	12	768	12	2048	8	4	256	1.05M	8,000	8B	8	1
300M	16	1024	16	2752	8	4	256	1.05M	15,000	15B	8	1
600M	20	1280	20	3456	8	2	512	2.10M	13,000	27B	32	4
1B	26	1664	26	4480	4	4	512	2.10M	25,000	52B	32	4
2B	32	2048	32	5504	2	8	1024	4.19M	24,000	100B	64	8
7B (MoE)	40	1536	24	256 / 1024	2	8	1024	4.19M	25,000	100B	64	8

D Training Compute Convention

We report training compute as

$$C_{\text{full}} = \underbrace{6 \cdot N_{\text{ne}} \cdot D}_{\text{matmul}} + \underbrace{12 \cdot n_{\text{layers}} \cdot d_{\text{model}} \cdot L \cdot D}_{\text{parameter-free attention}}, \tag{3}$$

where N_{ne} is the non-embedding parameter count, D is the number of tokens, n_{layers} is the number of layers, d_{model} is the hidden dimension of the model, and L is the sequence length. Since we are training on sequences of length $L = 4096$, the parameter-free $QK^\top$ increases the total FLOPs significantly beyond the standard $6N$ matmul term (Kaplan et al., 2020; Hoffmann et al., 2022). Across our sweep the attention term contributes 25-47% of C_{full}. We then convert to PFLOP-days via $1\,\text{PFLOP-day} = 8.64\times10^{19}\,\text{FLOPs}$. We report parameter counts and FLOPs of each model in Table 8. The dynamic-convolution variants add $\sim 3\%$ to N_{ne} through the weight-generation projections (low-rank factorizations). Notably, our advantage is robust to the training compute convention. Refitting under $C_{\text{simple}} = 6N_{\text{total}}D$ or $C_{\text{chinchilla}} = 6N_{\text{ne}}D$ gives compute advantages of $1.30\times$ and $1.34\times$ respectively (vs. $1.33\times$ for C_{full}).

Table 8: Parameter counts and training compute for the dense scaling-law sweep.

Model	Parameters		Compute		
	Total	Non-emb.	matmul (FLOPs)	attention (FLOPs)	C_{full} (PFLOP-d)
Transformer (*8B Tokens*)	162.0M	85.0M	4.28×10^{18}	3.80×10^{18}	0.09
w/ dynamic conv. (low-rank)	164.9M	87.8M	4.42×10^{18}	3.80×10^{18}	0.10
w/ dynamic conv. (all linear)	169.8M	92.7M	4.67×10^{18}	3.80×10^{18}	0.10
Transformer (*15B Tokens*)	305.2M	202.4M	1.91×10^{19}	1.27×10^{19}	0.37
w/ dynamic conv. (low-rank)	311.8M	209.0M	1.97×10^{19}	1.27×10^{19}	0.37
w/ dynamic conv. (all linear)	319.0M	216.2M	2.04×10^{19}	1.27×10^{19}	0.38
Transformer (*27B Tokens*)	525.0M	396.5M	6.49×10^{19}	3.43×10^{19}	1.15
w/ dynamic conv. (low-rank)	537.6M	409.1M	6.69×10^{19}	3.43×10^{19}	1.17
w/ dynamic conv. (all linear)	546.7M	418.2M	6.84×10^{19}	3.43×10^{19}	1.19
Transformer (*52B Tokens*)	1.04B	869.5M	2.74×10^{20}	1.11×10^{20}	4.46
w/ dynamic conv. (low-rank)	1.06B	897.3M	2.82×10^{20}	1.11×10^{20}	4.56
w/ dynamic conv. (all linear)	1.07B	906.1M	2.85×10^{20}	1.11×10^{20}	4.59
Transformer (*100B Tokens*)	1.82B	1.62B	9.78×10^{20}	3.24×10^{20}	15.07
w/ dynamic conv. (low-rank)	1.88B	1.67B	1.01×10^{21}	3.24×10^{20}	15.43
w/ dynamic conv. (all linear)	1.88B	1.67B	1.01×10^{21}	3.24×10^{20}	15.46

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 abstract and introduction describe the paper's main contribution as demonstrating that dynamic short convolutions are a promising addition to Transformers. These claims are supported by strong empirical improvements on both synthetic and language modeling tasks. By providing efficient GPU kernels, the paper further shows that the observed scaling-law improvements can be realized in practice.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper includes a discussion of limitations. It is noted that further validation is needed at frontier scale (larger models, more data).

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [N/A]

Justification: The paper does not contain theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: The paper provides the main details required to reproduce the experiments, with detailed hyperparameter settings provided in the appendix. The language modelling experiments are based on a publicly available codebase (`lm-engine`) with publicly available training data (`Nemotron-CC`).

Guidelines:

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

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: The Triton kernels for dynamic short convolutions are released at `https://github.com/OliverSieberling/dynamic-conv1d`. All language-modeling experiments use the publicly available `lm-engine` codebase and the public Nemotron-CC corpus, and the complete architecture, initialization, and training hyperparameters are specified in Appendix C.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: The paper specifies the experimental details such as model sizes, tokenizers, optimization details, model architectures, baselines and evaluation benchmarks. Additional details are in the appendix.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Our synthetic MQAR experiments report median over five seeds and provides minimum/maximum value bars. The language modeling experiments are performed on a single seed, but these large scale runs are known to be seed robust.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We specify the computational requirements and our GPU setup across all language modeling experiments in the appendix. The synthetic experiments have negligible compute cost.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The paper conforms to the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

Answer: [No]

Justification: The research in this paper is foundational and does not impose particular negative societal impacts.

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: No models or data are released.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: The creators of codebases and datasets used in this work are properly credited, such as lm-engine, Nemotron-CC, PyTorch, causal-conv1d, etc.

Guidelines:

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

13. **New assets**

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

    Answer: [N/A]

    Justification: The submission does not contain dataset or model releases.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No crowdsourcing or research with human subjects was performed for this paper.

    Guidelines:

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

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

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

    Answer: [N/A]

Justification: No human-subject research is performed.

Guidelines:

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

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: The method was not developed by an LLM.

Guidelines:

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