xLSTM: Extended Long Short-Term Memory

Maximilian Beck [*1 2] **Korbinian Pöppel** [*1 2] **Markus Spanring** [1] **Andreas Auer** [1 2] **Oleksandra Prudnikova** [1]
Michael Kopp **Günter Klambauer** [1] **Johannes Brandstetter** [1 2 3] **Sepp Hochreiter** [1 2 3]

Abstract

In the 1990s, the constant error carousel and gating were introduced as the central ideas of the Long Short-Term Memory (LSTM). Since then, LSTMs have stood the test of time and contributed to numerous deep learning success stories, in particular they constituted the first Large Language Models (LLMs). However, the advent of the Transformer technology with parallelizable self-attention at its core marked the dawn of a new era, outpacing LSTMs at scale. We now raise a simple question: How far do we get in language modeling when scaling LSTMs to billions of parameters, leveraging the latest techniques from modern LLMs, but mitigating known limitations of LSTMs? Firstly, we introduce exponential gating with appropriate normalization and stabilization techniques. Secondly, we modify the LSTM memory structure, obtaining: (i) sLSTM with a scalar memory, a scalar update, and new memory mixing, (ii) mLSTM that is fully parallelizable with a matrix memory and a covariance update rule. Integrating these LSTM extensions into residual block backbones yields xLSTM blocks that are then residually stacked into xLSTM architectures. Exponential gating and modified memory structures boost xLSTM capabilities to perform favorably when compared to state-of-the-art Transformers and State Space Models, both in performance and scaling.

1. Introduction

The Long Short-Term Memory (LSTM) ideas (Hochreiter, 1991; Hochreiter & Schmidhuber, 1997b;a), i.e., the constant error carousel and gating, were introduced to overcome the vanishing gradient problem of recurrent neural networks (Hochreiter, 1991; Hochreiter et al., 2000):

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \;, \; h_t = \mathrm{o}_t \; \psi(c_t).$$

The constant error carousel is the additive update of the cell state c_{t-1} (green) by cell inputs z_t and moderated by sigmoid gates (blue). The input gate i_t and the forget gate f_t control this update, while the output gate o_t controls the output of the memory cell, i.e. the hidden state h_t. The cell state is normalized or squashed by ψ and then output gating gives the hidden state.

LSTMs have been successfully applied to various domains (Hochreiter et al., 2001; 2007; Schmidhuber, 2015), and prevailed over text generation until the dawn of Transformers in 2017 (Vaswani et al., 2017).

Despite their tremendous successes, LSTMs have three main limitations: (i) Inability to revise storage decisions. We exemplify this limitation via the *Nearest Neighbor Search* problem (see also Appendix D): With a reference vector given, a sequence must be scanned sequentially for the most similar vector in order to provide its attached value at sequence end. The left panel of Figure 1 shows the mean squared error at this task. LSTM struggles to revise a stored value when a more similar vector is found, while our new xLSTM remediates this limitation by exponential gating. (ii) Limited storage capacities, i.e., information must be compressed into scalar cell states. We exemplify this limitation via *Rare Token Prediction*.

In the right panel of Figure 1, the perplexity of token prediction on Wikitext-103 (Merity et al., 2017) is given for partitions of different token frequency. LSTM performs worse on rare tokens because of its limited storage capacities. Our new xLSTM solves this problem by a matrix memory. (iii) Lack of parallelizability due to memory mixing, i.e., the hidden-hidden connections between hidden states from one time step to the next, which enforce sequential processing.

These limitations of LSTM have paved the way for the emergence of Transformers (Vaswani et al., 2017) in language modeling. What performances can we achieve in language modeling when overcoming these limitations and scaling LSTMs to the size of current Large Language Models?

[*]Equal contribution [1]ELLIS Unit Linz and LIT AI Lab, Institute for Machine Learning, Johannes Kepler University, Linz, Austria [2]NXAI Lab, Linz, Austria [3]NXAI GmbH, Linz, Austria. Correspondence to: Maximilian Beck <beck@ml.jku.at>, Korbinian Pöppel <poeppel@ml.jku.at>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

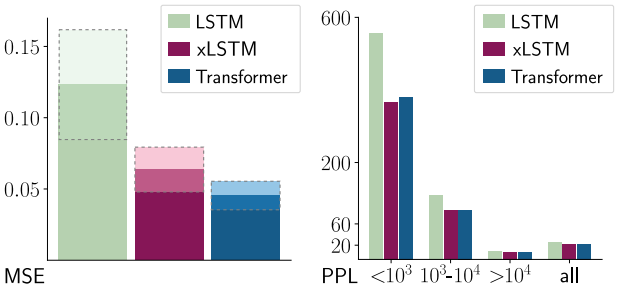

Figure 1: LSTM limitations. **Left**: Nearest Neighbor Search problem in terms of mean squared error (MSE). Given a reference vector, a sequence is scanned sequentially for the most similar vector with the objective to return its attached value at sequence end. LSTM struggles to revise a stored value when a more similar vector is found. Our new xLSTM overcomes this limitation by exponential gating. **Right**: Rare Token Prediction. The perplexity (PPL) of token prediction on Wikitext-103, in partitions of token frequency. LSTM performs worse on predicting rare tokens because of its limited storage capacities, whereas our new xLSTM solves this problem via a matrix memory.

2. Extended Long Short-Term Memory

To overcome the LSTM limitations, Extended Long Short-Term Memory (xLSTM) introduces two main modifications to the original LSTM. Those modifications — exponential gating and novel memory structures — enrich the LSTM family by two members: (i) the new sLSTM with a scalar memory, a scalar update, and memory mixing, and (ii) the new mLSTM with a matrix memory and a covariance (outer product) update rule, which is fully parallelizable. Both sLSTM and mLSTM enhance the LSTM through exponential gating. To enable parallelization, the mLSTM abandons memory mixing, i.e., the hidden-hidden recurrent connections. Both mLSTM and sLSTM can be extended to multiple memory cells, where sLSTM features memory mixing across cells. Further, the sLSTM can have multiple heads without memory mixing across the heads, but only memory mixing across cells within each head. This introduction of heads for sLSTM together with exponential gating establishes a new way of memory mixing. For mLSTM multiple heads and cells are equivalent.

Integrating these new LSTM variants into residual block modules results in xLSTM blocks . Residually stacking those xLSTM blocks in architectures provides xLSTM architectures. See Appendix Figure 4 for the xLSTM architecture with its components.

sLSTM To empower LSTMs with the ability to revise storage decisions, we introduce exponential gates (red) together with normalization and stabilization. For normalization, we introduce a normalizer state that sums up the product of input gate times all future forget gates.

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \tag{1}$$

$$n_t = \mathrm{f}_t \; n_{t-1} + \mathrm{i}_t \tag{2}$$

$$h_t = \mathrm{o}_t \; \tilde{h}_t = \mathrm{o}_t \; c_t \; / \; n_t \tag{3}$$

$$z_t = \varphi\left(\tilde{z}_t\right) \tag{4}$$

$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right) \tag{5}$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right) \tag{6}$$

$$\mathrm{o}_t = \sigma\left(\tilde{\mathrm{o}}_t\right) \,, \tag{7}$$

$$\{\tilde{\mathrm{i}}_t, \tilde{\mathrm{f}}_t, \tilde{z}_t, \tilde{\mathrm{o}}_t\} = \boldsymbol{w}_{\{\mathrm{i,f,z,o}\}}^\top \boldsymbol{x}_t + r_{\{\mathrm{i,f,z,o}\}} \, h_{t-1} + b_{\{\mathrm{i,f,z,o}\}} \tag{8}$$

We transfer the original LSTM gating techniques, i.e., input- and/or hidden-dependent gating plus bias term, to the new architectures. Exponential activation functions can lead to large values that cause overflows. Therefore, we stabilize gates with an additional state m_t (Milakov & Gimelshein, 2018), see Equations (46) – (48) in the appendix.

sLSTM can have multiple memory cells like the original LSTM (see Appendix B.2). Multiple memory cells enable memory mixing via recurrent connections $\boldsymbol{R_z}$, $\boldsymbol{R_\mathrm{i}}$, $\boldsymbol{R_\mathrm{f}}$, $\boldsymbol{R_\mathrm{o}}$ from hidden state vector $\boldsymbol{h}$ to memory cell input $\boldsymbol{z}$ and the gates $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$, respectively. A new aspect in memory mixing is the effect of exponential gating. The new sLSTM can have multiple heads with memory mixing within each head but not across heads. The introduction of heads for sLSTM together with exponential gating establishes a new way of memory mixing.

mLSTM To enhance storage capacities of LSTMs, we increase the LSTM memory cell from a scalar $c \in \mathbb{R}$ to a matrix $\boldsymbol{C} \in \mathbb{R}^{d \times d}$. Hence, retrieval is performed via a matrix multiplication. At time t, we want to store a pair of vectors, the key $\boldsymbol{k}_t \in \mathbb{R}^d$ and the value $\boldsymbol{v}_t \in \mathbb{R}^d$ (we use the Transformer terminology). Later at time $t + \tau$, the value $\boldsymbol{v}_t$ should be retrieved by a query vector $\boldsymbol{q}_{t+\tau} \in \mathbb{R}^d$. This is the setting of Bidirectional Associative Memories (BAMs) (Kohonen, 1972; Anderson, 1972; Nakano, 1972; Anderson et al., 1977). The covariance update rule (Sejnowski, 1977; Dayan & Willshaw, 1991) for storing a key-value pair is $\boldsymbol{C}_t = \boldsymbol{C}_{t-1} + \boldsymbol{v}_t \, \boldsymbol{k}_t^\top$. We assume a layer-norm before projecting inputs to keys and values, therefore they have zero mean. The covariance update rule is optimal (Dayan & Willshaw, 1991) for a maximal separability of retrieved binary vectors, which is equivalent to a maximal signal/noise ratio. Higher separability is possible when limiting retrieval

to pairwise interactions and conceding quadratic complexity like attention (Krotov & Hopfield, 2016; 2017; Ramsauer et al., 2021). The covariance update rule is equivalent to Fast Weight Programmers (Schmidhuber, 1992; Schlag et al., 2021), which have later been equipped with a constant decay rate multiplied to $\boldsymbol{C}_{t-1}$ and a constant learning rate multiplied to $\boldsymbol{v}_t \boldsymbol{k}_t^\top$ (Ba et al., 2016a). In this spirit, we integrate the covariance update rule into the LSTM framework, where the forget gate corresponds to decay rate and the input gate to the learning rate, while the output gate scales the retrieved vector.

For this matrix memory, the normalizer state is the weighted sum of key vectors, where each key vector is weighted by the input gate and all future forget gates. Again, the normalizer state keeps record of the strength of the gates. Since the dot product between query and normalizer state can be close to zero, we use the absolute value of this dot product and lower bound it by a threshold (typically 1.0) as done previously (Sun et al., 2023). The mLSTM forward pass is:

$$\boldsymbol{C}_t = \mathrm{f}_t \; \boldsymbol{C}_{t-1} + \mathrm{i}_t \; \boldsymbol{v}_t \, \boldsymbol{k}_t^\top \tag{9}$$

$$\boldsymbol{n}_t = \mathrm{f}_t \; \boldsymbol{n}_{t-1} + \mathrm{i}_t \; \boldsymbol{k}_t \tag{10}$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t \, , \tilde{\boldsymbol{h}}_t = \frac{\boldsymbol{C}_t \; \boldsymbol{q}_t}{\max \left\{ \left| \boldsymbol{n}_t^\top \; \boldsymbol{q}_t \right| , 1 \right\}} \tag{11}$$

$$\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\} = \boldsymbol{W}_{\{q,k,v\}} \, \boldsymbol{x}_t + \boldsymbol{b}_{\{q,k,v\}} \tag{12}$$

$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right) \, , \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \, \boldsymbol{x}_t + b_\mathrm{i} \tag{13}$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right) , \quad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \, \boldsymbol{x}_t + b_\mathrm{f} \tag{14}$$

$$\mathbf{o}_t = \sigma\left(\tilde{\mathbf{o}}_t\right) \, , \qquad \tilde{\mathbf{o}}_t = \boldsymbol{W}_\mathbf{o} \, \boldsymbol{x}_t + \boldsymbol{b}_\mathbf{o} \tag{15}$$

mLSTM can have multiple memory cells like the original LSTM. For mLSTM, multiple heads and multiple cells are equivalent as there is no memory mixing. In order to stabilize the exponential gates of mLSTM, we use the same stabilization techniques as for sLSTM (see Equation 46). As the mLSTM has no memory mixing, this recurrence can be reformulated in a parallel version (see Appendix B.3).

xLSTM Architecture An xLSTM block should non-linearly summarize the past in a high-dimensional space to better separate different histories or contexts. Separating histories is the prerequisite to correctly predict the next sequence element such as the next token. We resort to Cover's Theorem (Cover, 1965), which states that in a higher dimensional space non-linearly embedded patterns can more likely be linearly separated than in the original space. We consider two residual block architectures: (i) A residual block with post up-projection (like Transformers), which non-linearly summarizes the past in the original space, then linearly maps into a high-dimensional space, applies a non-linear activation function, and linearly maps back to the original space; see Appendix Figure 5 for details. (ii) A residual block with pre up-projection (like State Space Models), which linearly maps to a high-dimensional space, non-linearly summarizes the past in the high-dimensional space and then linearly maps back to the original space. See Appendix Figure 6 for more details. For an xLSTM block containing an sLSTM, we mostly use the post up-projection block. For an xLSTM block containing an mLSTM, we use the pre up-projection block since the memory capacity becomes larger in the high-dimensional space.

An xLSTM architecture is constructed by residually stacking building blocks (Srivastava et al., 2015; He et al., 2016). We rely on the commonly used pre-LayerNorm (Ba et al., 2016b) residual backbones as used in contemporary Large Language Models. See also Appendix Figure 4.

3. Experiments

Here, we compare xLSTM to RWKV-4 (Peng et al., 2023), Llama (Touvron et al., 2023b) and Mamba (Gu & Dao, 2024) after being trained on 300B tokens from SlimPajama (Soboleva et al., 2023) on downstream tasks, and assess their scaling behavior analogous to Kaplan et al. (2020) and Brown et al. (2020). In Appendix Sections D.1 and D.2, we test specific capabilities on synthetic tasks and compare the performance and against more baseline models on a reduced 15B token subset of SlimPajama, namely RWKV-5, RWKV-6 (Peng et al., 2024a), HGRN2 (Qin et al., 2024), Retention (Sun et al., 2023), H3 (Fu et al., 2023), Hyena (Poli et al., 2023). On this dataset, we also perform ablations towards a vanilla LSTM, see Appendix Section D.2. For all experiments, we use the notation xLSTM[a:b] for the ratio a/b of mLSTM-based versus sLSTM-based xLSTM blocks. For example, xLSTM[7:1] means that out of eight blocks, seven are mLSTM-based blocks and one is an sLSTM-based block.

xLSTM as Large Language Model We train the models for next token prediction for 300B tokens from SlimPajama, the same number of tokens as used in e.g., Mamba (Gu & Dao, 2023) and Griffin (De et al., 2024). We select RWKV-4 as RNN representative since for RWKV-5, RWKV-6 and HGRN2 a reasonable training precision setting (Appendix Section D.2) has been found only after the training start of the 300B token experiments (Peng et al., 2024b). We train different model sizes (125M, 350M, 760M, 1.3B), test all models for length extrapolation capabilities and investigate

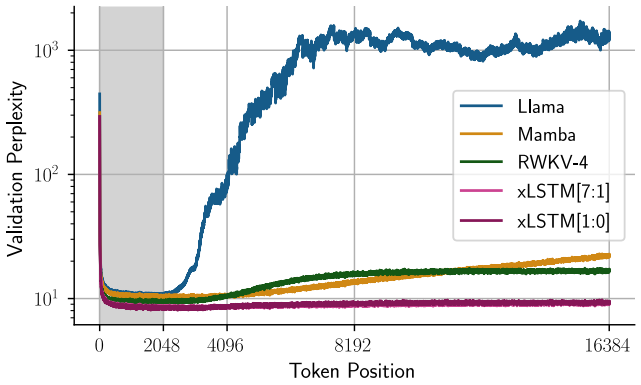

Figure 2: Sequence extrapolation in language modeling. Large models (1.3B) of xLSTM, RWKV-4, Llama, and Mamba compared on the validation set after training on 300B tokens from SlimPajama. Models are trained with context length 2048 (gray) and tested for context lengths up to 16384. In contrast to other methods, xLSTM models remain at low perplexities for longer contexts.

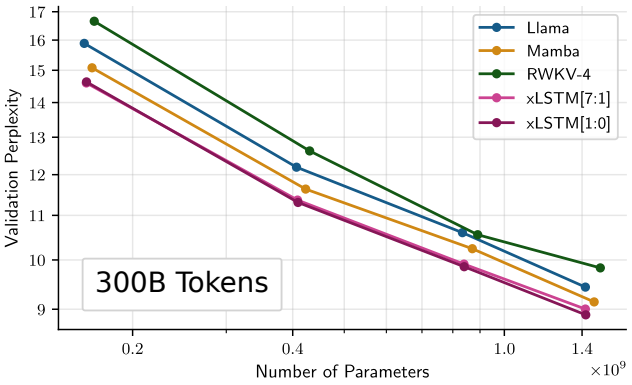

Figure 3: Scaling laws. Next token prediction perplexity of xLSTM, RWKV-4, Llama, and Mamba. The models – with sizes 125M, 350M, 760M, and 1.3B parameters – are trained on 300B tokens from SlimPajama. The scaling laws indicate that for larger models xLSTM will perform well too.

their scaling law behavior. In Appendix Section D.3, we evaluate their performance on downstream tasks and on 571 text domains of the PALOMA benchmark.

Sequence Length Extrapolation. We test the sequence length extrapolation for 1.3B-sized, large models of xLSTM, RWKV-4, Llama, and Mamba. All models are trained on context length 2048, and then tested for context lengths up to 16384. See Figure 2 for the results. In contrast to other methods, xLSTM models maintain low perplexities for longer contexts.

Validation Perplexity and Downstream Tasks. For all model sizes, we evaluate the performance of xLSTM, RWKV-4, Llama, and Mamba models on the SlimPajama validation set for next token prediction and on downstream tasks that measure common sense reasoning. The third column of Appendix Table 7 lists the validation set perplexities of different methods. Both xLSTM[1:0] and xLSTM[7:1] are the best models for all model sizes with respect to the validation set perplexity. The other columns of Appendix Table 7 provide the performance on downstream tasks. In the vast majority of tasks and across all model sizes xLSTM is the best method — only on the ARC task Mamba is in some cases the best method. For details see Appendix D.3.

Performance on PALOMA Language Tasks. For a more detailed understanding, we test the next token prediction performance of xLSTM, RWKV-4, Llama, and Mamba models on PALOMA language tasks (Magnusson et al., 2023). We measure the performance by the perplexity on 571 text domains, ranging from nytimes.com to r/depression on Reddit. Appendix Table 8 shows token prediction perplexity grouped into language modeling (first seven columns) and fine-grained domain benchmarks (last 5 columns). xLSTM[1:0] has in 568 out of 571 (99.5%) text domains a lower perplexity than Mamba, in 486 out of 571 (85.1%) a lower perplexity than Llama, in 570 out of 571 (99.8%) a lower perplexity than RWKV-4, see Appendix D.3.

Scaling Laws. Finally, we assess the power-law scaling behavior, which allows to extrapolate the performance to larger model sizes (Kaplan et al., 2020; Brown et al., 2020). Figure 3 presents the scaling behavior. All models share a similar scaling behavior but with different offsets. RWKV-4 performs worst, followed by Llama and Mamba. xLSTM is better than Mamba with a similar margin to Mamba as Mamba has to Llama. The scaling behavior indicates that for larger models xLSTM will continue to perform favourable compared to Transformers and State-Space models.

4. Conclusion

We have partly answered our simple question: How far do we get in language modeling when scaling LSTM to billions of parameters? So far, we can answer: "At least as far as current technologies like Transformers or State Space Models". We have enhanced LSTM to xLSTM by exponential gating with memory mixing and a new memory structure. xLSTM models perform favorably on language modeling compared to state-of-the-art methods like Transformers and State Space Models. Scaling laws indicate that larger xLSTM models will be serious competitors to current LLMs that are built with the Transformer technology. xLSTM has the potential to considerably impact other fields like Reinforcement Learning, Time Series Prediction, or the modeling of physical systems.

Broader Impacts

Our work introduces novel LSTM architectures that perform favorably compared to Transformers or State Space Models on language modeling. As our novel LSTM models are inherently recurrent and demonstrably extrapolate well to large contexts, they have the potential for near constant scaling at inference. This compares to linear scaling in the context length for Transformer architectures, representing a vast potential in energy and emission savings when deploying such models to real world applications. Thus, in an ever more connected, data-rich world with ubiquitous compute, our work has the potential to increase general accessibility of state-of-the-art machine learning models whilst making them more environmentally sustainable.

However, any novel machine learning technique that has the potential to impact real world applications can be used for harm, not just for good. Our models with their beneficial inference economics and their ability to extrapolate to large contexts at no overhead could be used, for example, to generate and spread disinformation on a grander scale. Moreover, savings in energy and emissions afforded by deploying our novel xLSTM models might be outweighed by an increase in demand for such models, leading to an overall increase in energy consumption and environmentally harmful emissions. Whilst our work focuses on new, more efficient Language Model architectures, it does not directly address the issues of fairness and bias of such models.

References

Achiam, J., Adler, S., Agarwal, S., et al. GPT-4 technical report. *ArXiv*, 2303.08774, 2023.

Anderson, J., Silverstein, J., Ritz, S., and Jones, R. Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, 84:413–451, 1977. doi: 10.1037/0033-295X.84.5.413.

Anderson, J. A. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14, 1972. doi: 10.1016/0025-5564(72)90075-2.

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*, 2312.04927, 2023.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4331–4339. Curran Associates, Inc., 2016a.

Ba, J., Kiros, J. R., and Hinton, G. Layer normalization. *ArXiv*, 1607.06450, 2016b.

Bisk, Y., Zellers, R., LeBras, R., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Blodgett, S. L., Green, L., and O'Connor, B. Demographic dialectal variation in social media: A case study of African-American English. In *Conference on Empirical Methods in Natural Language Processing*, pp. 1119–1130, 2016. doi: 10.18653/v1/D16-1120.

Brown, T., Mann, B., Ryder, N., et al. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Chowdhery, A., Narang, S., Devlin, J., et al. PaLM: scaling language modeling with pathways. *ArXiv*, 2204.02311, 2022.

Chronopoulou, A., Peters, M., and Dodge, J. Efficient hierarchical domain adaptation for pretrained language models. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 1336–1351, 2022. doi: 10.18653/v1/2022.naacl-main.96.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *ArXiv*, 1803.05457, 2018.

Cover, T. M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on*, EC-14(3):326–334, 1965.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, volume 12, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In Oh, A. H., Agarwal, A., Belgrave,

D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://openreview.net/forum?id=H4DqfPSibmx.

Dayan, P. and Willshaw, D. J. Optimising synaptic learning rules in linear associative memories. *Biological Cybernetics*, 65, 1991. doi: 10.1007/bf00206223.

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., Desjardins, G., Doucet, A., Budden, D., Teh, Y. W., Pascanu, R., DeFreitas, N., and Gulcehre, C. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *ArXiv*, 2402.19427, 2024.

Delétang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Cundy, C., Hutter, M., Legg, S., Veness, J., and Ortega, P. A. Neural networks and the Chomsky hierarchy. In *International Conference on Learning Representations (ICLR)*, volume 11, 2023. URL https://openreview.net/forum?id=WbxHAzkeQcn.

Du, N., Huang, Y., Dai, A. M., et al. GLaM: efficient scaling of language models with mixture-of-experts. *ArXiv*, 2112.06905, 2021.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Re, C. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=COZDy0WYGg.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The Pile: An 800gb dataset of diverse text for language modeling. *ArXiv*, 2101.00027, 2021.

Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with LSTM. *Neural Compututation*, 12(10):2451–2471, 2000.

Google, G. T. Gemini: A family of highly capable multimodal models. *ArXiv*, 2312.11805, 2023.

Greenbaum, S. and Nelson, G. The international corpus of English (ICE) project. *World Englishes*, 15(1):3–15, 1996.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. LSTM: A search space odyssey. *ArXiv*, 1503.04069, 2015.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *ArXiv*, 2312.00752, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=AL1fq05o7H.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *ArXiv*, 2111.00396, 2021.

Gupta, A., Gu, A., and Berant, J. Diagonal state spaces are as effective as structured state spaces. *ArXiv*, 2203.14343, 2022.

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 (CVPR)*, pp. 770–778, 2016.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technische Universität München, 1991.

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

Hochreiter, S. and Schmidhuber, J. LSTM can solve hard long time lag problems. In Mozer, M. C., Jordan, M. I., and Petsche, T. (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 9, pp. 473–479. MIT Press, Cambridge MA, 1997b.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kolen, J. and Kremer, S. (eds.), *A Field Guide to Dynamical Recurrent Networks*. IEEE, 2000.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In Dorffner, G., Bischof, H., and Hornik, K. (eds.), *Proc. Int. Conf. on Artificial Neural Networks (ICANN 2001)*, pp. 87–94. Springer, 2001.

Hochreiter, S., Heusel, M., and Obermayer, K. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

Hoffmann, J., Borgeaud, S., Mensch, A., et al. Training compute-optimal large language models. *ArXiv*, 2203.15556, 2022.

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. *ArXiv*, 2001.08361, 2020.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In III, E. H. D. and Singh, A. (eds.), *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

Katsch, T. GateLoop: Fully data-controlled linear recurrence for sequence modeling. *ArXiv*, 2311.01927, 2023.

Kocetkov, D., Li, R., BenAllal, L., Li, J., Mou, C., nozFerrandis, C. M., Jernite, Y., Mitchell, M., Hughes, S., Wolf, T., Bahdanau, D., vonWerra, L., and deVries, H. The Stack: 3 TB of permissively licensed source code. *ArXiv*, 2211.15533, 2022.

Kohonen, T. Correlation matrix memories. *IEEE Transactions on Computers*, C-21(4), 1972. doi: 10.1109/tc.1972.5008975.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, Deptartment of Computer Science, University of Toronto, 2009.

Krotov, D. and Hopfield, J. J. Dense associative memory for pattern recognition. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, pp. 1172–1180. Curran Associates, Inc., 2016.

Krotov, D. and Hopfield, J. J. Dense associative memory is robust to adversarial inputs. *ArXiv*, 1701.00939, 2017.

Li, Y., Cai, T., Zhang, Y., Chen, D., and Dey, D. What makes convolutional models great on long sequence modeling? *ArXiv*, 2210.09298, 2022.

Liang, P., Bommasani, R., Lee, T., et al. Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 1525:140–146, 2023.

Lin, J., Men, R., Yang, A., Zhou, C., Ding, M., Zhang, Y., Wang, P., Wang, A., Jiang, L., Jia, X., Zhang, J., Zhang, J., Zou, X., Li, Z., Deng, X., Liu, J., Xue, J., Zhou, H., Ma, J., j. Yu, Li, Y., Lin, W., Zhou, J., Tang, J., and Yang, H. M6: A Chinese multimodal pretrainer. *ArXiv*, 2103.00823, 2021.

Linsley, D., Kim, J., Veerabadran, V., Windolf, C., and Serre, T. Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., and Zettlemoyer, L. Mega: Moving average equipped gated attention. *ArXiv*, 2209.10655, 2022.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Annual Meeting of the Association for Computational Linguistics*, volume 49, pp. 142–150, 2011.

Magnusson, I., Bhagia, A., Hofmann, V., et al. Paloma: A benchmark for evaluating language model fit. *ArXiv*, 2312.10523, 2023.

Mehta, H., Gupta, A., Cutkosky, A., and Neyshabur, B. Long range language modeling via gated state spaces. *ArXiv*, 2206.13947, 2022.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICRL)*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Merrill, W. and Sabharwal, A. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. doi: 10.1162/tacl_a_00562.

Merrill, W., Petty, J., and Sabharwal, A. The illusion of state in state-space models. *ArXiv*, 2404.08819, 2024.

Milakov, M. and Gimelshein, N. Online normalizer calculation for softmax. *ArXiv*, 1805.02867, 2018.

Nakano, K. Associatron – a model of associative memory. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):380–388, 1972. doi: 10.1109/TSMC.1972.4309133.

Olsson, C., Elhage, N., Nanda, N., et al. In-context learning and induction heads. *ArXiv*, 2209.11895, 2022.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*. JMLR.org, 2023. doi: 10.5555/3618408.3619518.

Papasavva, A., Zannettou, S., DeCristofaro, E., Stringhini, G., and Blackburn, J. Raiders of the lost KeK: 3.5 years of augmented 4chan posts from the politically incorrect board. In *International AAAI Conference on Web and Social Media (ICWSM)*, volume 14, pp. 885–894, 2020.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N.-Q., Bernardi, R., Pezzelle, S., Baroni, M., G. Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Annual Meeting of the Association for Computational Linguistics*, volume 1, pp. 1525–1534, 2016.

Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., and Launay, J. The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data, and web data only. *ArXiv*, 2306.01116, 2023.

Peng, B., Alcaide, E., Anthony, Q., et al. RWKV: Reinventing RNNs for the transformer era. *ArXiv*, 2305.13048, 2023.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Du, X., Ferdinan, T., Hou, H., Kazienko, P., GV, K. K., Kocoń, J., Koptyra, B., Krishna, S., McClelland, R. J., Muennighoff, N., Obeid, F., Saito, A., Song, G., Tu, H., Woźniak, S., Zhang, R., Zhao, B., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R.-J. Eagle and Finch: RWKV with matrix-valued states and dynamic recurrence. *ArXiv*, 2404.05892, 2024a.

Peng, B., Goldstein, D., Anthony, Q., et al. Eagle and Finch: RWKV with matrix-valued states and dynamic recurrence. *ArXiv*, 2404.05892, 2024b.

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 *Proceedings of the 40th International Conference on Machine Learning (ICML)*. JMLR.org, 2023. doi: 10.5555/3618408.3619572.

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

Qin, Z., Yang, S., and Zhong, Y. Hierarchically gated recurrent neural network for sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2023. URL https://openreview.net/forum?id=P1TCHxJwLB.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. HGRN2: Gated linear RNNs with state expansion. *ArXiv*, 2404.07904, 2024.

Radev, D. R., Muthukrishnan, P., and Qazvinian, V. The ACL anthology network corpus. In *Workshop on Text and Citation Analysis for Scholarly Digital Libraries (NLPIR4DL)*, pp. 54–61. Association for Computational Linguistics, 2009.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. https://openai.com/index/better-language-models, 2019.

Rae, J. W., Borgeaud, S., Cai, T., et al. Scaling language models: Methods, analysis & insights from training Gopher. *ArXiv*, 2112.11446, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, 1910.10683, 2019.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. Hopfield networks is all you need. In *International Conference on Learning Representations (ICLR)*. OpenReview, 2021.

Reid, M., Zhong, V., Gururangan, S., and Zettlemoyer, L. M2D2: A massively multi-domain language modeling dataset. In *Conference on Empirical Methods in Natural Language Processing*, pp. 964–975, 2022.

Reid, M., Savinov, N., Teplyashin, D., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv*, 2403.05530, 2024.

Ribeiro, M. H., Blackburn, J., Bradlyn, B., DeCristofaro, E., Stringhini, G., Long, S., Greenberg, S., and Zannettou, S. The evolution of the manosphere across the web. In *Proceedings of the international AAAI conference on web and social media*, volume 15, pp. 196–207, 2021.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Scao, T. L., Fan, A., Akiki, C., et al. BLOOM: A 176B-parameter open-access multilingual language model. *ArXiv*, 2211.05100, 2022.

Schlag, I., Irie, K., and Schmidhuber, J. Linear transformers are secretly fast weight programmers. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 4 (1):131–139, 1992.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003.

Schulman, J., Zoph, B., Kim, C., Hilton, J., et al. ChatGPT: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/, 2022. OpenAI Research.

Sejnowski, T. J. Storing covariance with nonlinearly interacting neurons. *Journal of Mathematical Biology*, 4, 1977. doi: 10.1007/BF00275079.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training multi-billion parameter language models using model parallelism. *ArXiv*, 1909.08053, 2019.

Smith, J. T. H., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling. *ArXiv*, 2208.04933, 2022.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Soldaini, L., Kinney, R., Bhagia, A., et al. Dolma: an open corpus of three trillion tokens for language model pretraining research. *ArXiv*, 2306.01116, 2023.

Soltan, S., Ananthakrishnan, S., FitzGerald, J., Gupta, R., Hamza, W., Khan, H., Peris, C., Rawls, S., Rosenbaum, A., Rumshisky, A., Prakash, C. S., Sridhar, M., Triefenbach, F., Verma, A., Tur, G., and Natarajan, P. AlexaTM 20B: Few-shot learning using a large-scale multilingual Seq2Seq model. *ArXiv*, 2208.01448, 2022.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28. Curran Associates, Inc., 2015.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *ArXiv*, 2307.08621, 2023.

Sutawika, L., Gao, L., Schoelkopf, H., et al. EleutherAI/lm-evaluation-harness: Major refactor, 2023.

Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., and Zheng, C. Synthesizer: Rethinking self-attention in transformer models. *ArXiv*, 2005.00743, 2020.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations (ICRL)*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Thoppilan, R., deFreitas, D., Hall, J., et al. LaMDA: Language models for dialog applications. *ArXiv*, 2201.08239, 2022.

TogetherComputer. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. *ArXiv*, 2302.1397, 2023a.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., RoziÃ¨re, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. https://arxiv.org/abs/2302.13971, 2023b.

Vadas, D. and Curran, J. R. Parsing noun phrases in the Penn Treebank. *Computational Linguistics*, 37(4):753–809, 2011.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017.

Wang, J., Yan, J. N., Gu, A., and Rush, A. M. Pretraining without attention. *ArXiv*, 2212.10544, 2022.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *ArXiv*, 2006.04768, 2020.

Wang, S., Sun, Y., Xiang, Y., et al. ERNIE 3.0 Titan: Exploring larger-scale knowledge enhanced pre-training for language understanding and generation. *ArXiv*, 2112.12731, 2021.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. mT5: A massively multilingual pre-trained text-to-text transformer. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 483–498, 2021. doi: 10.18653/v1/2021.naacl-main.41.

Yang, S. and Zhang, Y. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism, 2024. URL https://github.com/sustcsonglin/flash-linear-attention.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training. *ArXiv*, 2312.06635, 2023.

Zannettou, S., Bradlyn, B., DeCristofaro, E., Kwak, H., Sirivianos, M., Stringini, G., and Blackburn, J. What is Gab: A bastion of free speech or an alt-right echo chamber. In *The Web Conference*, pp. 1007–1014, 2018. doi: 10.1145/3184558.3191531.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Zeng, A., Liu, X., Du, Z., et al. GLM-130B: An open bilingual pre-trained model. *ArXiv*, 2210.02414, 2022.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. OPT: Open pre-trained transformer language models. *ArXiv*, 2205.01068, 2022.

A. Related Work

Linear Attention. Several methods have been suggested to overcome the quadratic complexity in terms of context length of the Transformer and make attention linear in the context length. The Synthesizer learns synthetic attention weights without token–token interactions (Tay et al., 2020). Linformer realizes self-attention by a low-rank matrix and even linearly approximates it (Wang et al., 2020). Linear Transformer linearizes the attention mechanism (Katharopoulos et al., 2020). Performer linearly approximates the attention softmax by positive orthogonal random features approach (Choromanski et al., 2021). Attention has been replaced by fast long convolutions in the Structured Global Convolution (SGConv) (Li et al., 2022) and the Hyena Hierarchy (Poli et al., 2023).

State Space Models. Recently, State Space Models (SSMs) became very popular since they are linear in the context length and show promising performance compared to Transformers. One of the first proposed models was Structured State Space sequence model (S4) (Gu et al., 2021), followed by Diagonal State Space (DSS) model (Gupta et al., 2022), Gated State Space (GSS) models (Mehta et al., 2022), S5 model (Smith et al., 2022), Bidirectional Gated SSM (BiGS) (Wang et al., 2022), H3 model (Fu et al., 2023), and Mamba (Gu & Dao, 2023).

Recurrent Neural Networks. Recurrent Neural Networks (RNNs) have been suggested to replace Transformer and attention due to their linearity in the context length. RNNs with Deep Linear Recurrent Units (LRUs) showed promising results for language modeling (Orvieto et al., 2023; De et al., 2024), as did Hierarchically Gated Linear RNN (HGRN) (Qin et al., 2023) and HGRN2 (Qin et al., 2024). A well-known RNN approach to large language modeling is RWKV (Peng et al., 2023; 2024b), showcasing competitive performance to Transformers.

Gating. One of the key ideas of LSTM is gating, which was rediscovered and reinterpreted in many recent approaches. Gating was used in HGRN (Qin et al., 2023), HGRN2 (Qin et al., 2024), Gated Linear Attention (GLA) (Yang et al., 2023), Gated State Space (GSS) models (Mehta et al., 2022), Bidirectional Gated SSM (BiGS) (Wang et al., 2022), Moving Average Equipped Gated Attention (MEGA) (Ma et al., 2022), RWKV (Peng et al., 2023), and Mamba (Gu & Dao, 2023).

Covariance Update Rule. To enhance storage capacities, we equipped the mLSTM cell with a matrix memory with a covariance update rule. Other methods which build on such an update mechanism are Fast Weight Programmers (Schmidhuber, 1992; Schlag et al., 2021), RWKV-5 and RWKV-6 (Peng et al., 2024b), Retention (Sun et al., 2023), Linear Transformer (Katharopoulos et al., 2020), and HGRN2 (Qin et al., 2024).

Most Related. Conceptually the closest models to xLSTM are Retention (Sun et al., 2023), RWKV (Peng et al., 2023; 2024b), and HGRN2 (Qin et al., 2024). These models share the concepts matrix memory and/or gating. However, in contrast to the new sLSTM, these approaches do not allow memory mixing. Memory mixing enables to solve state tracking problems, and therefore LSTMs are more expressive than State Space Models (SSMs) and Transformers (Merrill et al., 2024; Delétang et al., 2023). State tracking is required to evaluate code or to track entities in a long narrative.

Residually Stacking Architectures. Like almost all contemporary large deep learning models, xLSTM architectures are constructed by residually stacking building blocks (Srivastava et al., 2015; He et al., 2016). This construction enabled deep convolutional networks (He et al., 2016) and Transformers (Vaswani et al., 2017). Transformers are the ultimate force behind Large Language Models (LLMs) like GPT-3 (Brown et al., 2020), ChatGPT (Schulman et al., 2022), GPT-4 (Achiam et al., 2023), Megatron-LM (Shoeybi et al., 2019), Gopher (Rae et al., 2021), ERNIE 3.0 Titan (Wang et al., 2021), GLaM (Du et al., 2021), Chinese M6 (Lin et al., 2021), mutilingual AlexaTM 20B (Soltan et al., 2022), OPT (Zhang et al., 2022), Chinchilla (Hoffmann et al., 2022), BLOOM (Scao et al., 2022), GLM-130B (Zeng et al., 2022), LaMDA (Thoppilan et al., 2022), PaLM (Chowdhery et al., 2022), Llama (Touvron et al., 2023a), Gemini (Google, 2023; Reid et al., 2024).

B. Extended Long Short-Term Memory

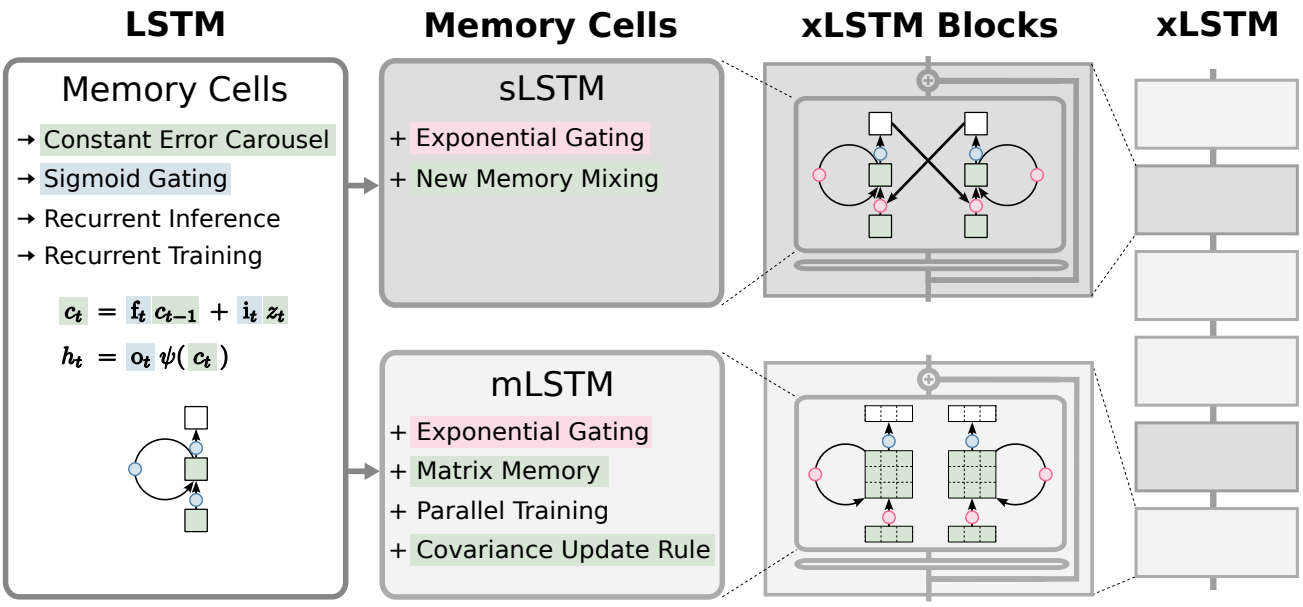

Figure 4: The extended LSTM (xLSTM) family. From left to right: 1. The original LSTM memory cell with constant error carousel and gating. 2. New sLSTM and mLSTM memory cells that introduce exponential gating. sLSTM offers a new memory mixing technique. mLSTM is fully parallelizable with a novel matrix memory cell state and new covariance update rule. 3. mLSTM and sLSTM in residual blocks yield xLSTM blocks. 4. Stacked xLSTM blocks give an xLSTM architecture.

B.1. Review of the Long Short-Term Memory

The original LSTM idea (Hochreiter, 1991; Hochreiter & Schmidhuber, 1997b;a) introduced the scalar memory cell as a central processing and storage unit that avoids vanishing gradients (Hochreiter, 1991; Hochreiter et al., 2000) through the constant error carousel (cell state update). The memory cell contains three gates: input, output, and forget gate. The latter was introduced by Gers et al. (2000). The LSTM memory cell update rules at time step t are:

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \qquad \text{cell state} \qquad (16)$$

$$h_t = \mathrm{o}_t \; \tilde{h}_t \,, \qquad \tilde{h}_t = \psi\left(c_t\right) \qquad \text{hidden state} \qquad (17)$$

$$z_t = \varphi\left(\tilde{z}_t\right) \,, \qquad \tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z \qquad \text{cell input} \qquad (18)$$

$$\mathrm{i}_t = \sigma\left(\tilde{\mathrm{i}}_t\right) \,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \boldsymbol{x}_t + r_\mathrm{i}\, h_{t-1} + b_\mathrm{i} \qquad \text{input gate} \qquad (19)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \,, \qquad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \boldsymbol{x}_t + r_\mathrm{f}\, h_{t-1} + b_\mathrm{f} \qquad \text{forget gate} \qquad (20)$$

$$\mathrm{o}_t = \sigma\left(\tilde{\mathrm{o}}_t\right) \,, \qquad \tilde{\mathrm{o}}_t = \boldsymbol{w}_\mathrm{o}^\top \boldsymbol{x}_t + r_\mathrm{o}\, h_{t-1} + b_\mathrm{o} \qquad \text{output gate} \qquad (21)$$

The weight vectors $\boldsymbol{w}_z$, $\boldsymbol{w}_\mathrm{i}$, $\boldsymbol{w}_\mathrm{f}$, and $\boldsymbol{w}_\mathrm{o}$ correspond to the input weight vectors between inputs $\boldsymbol{x}_t$ and cell input, input gate, forget gate, and output gate, respectively. The weights r_z, r_i, r_f, and r_o correspond to the recurrent weights between hidden state h_{t-1} and cell input, input gate, forget gate, and output gate, respectively. b_z, b_i, b_f, and b_o are the corresponding bias terms. φ and ψ are the cell input and hidden state activation functions (typically $\tanh$). ψ is used to normalize or squash the cell state, which would be unbounded otherwise. All gate activation functions are sigmoid, i.e., $\sigma\left(x\right) = 1/(1+\exp(-x))$. In later formulations, multiple memory cells were combined in a vector, which allows the usage of recurrent weight matrices to mix the cell outputs of memory cells (Greff et al., 2015) Ablation studies showed that all components of the memory cell are crucial (Greff et al., 2015).

The vanilla LSTM memory cell update rules (Greff et al., 2015) at time step t extend the scalar cell state formulation to a vector of cell states:

$$\boldsymbol{c}_t = \mathbf{f}_t \odot \boldsymbol{c}_{t-1} + \mathbf{i}_t \odot \boldsymbol{z}_t \qquad \text{cell state} \qquad (22)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t \,, \qquad \tilde{\boldsymbol{h}}_t = \psi\left(\boldsymbol{c}_t\right) \qquad \text{hidden state} \qquad (23)$$

$$\boldsymbol{z}_t = \varphi\left(\tilde{\boldsymbol{z}}_t\right) \,, \qquad \tilde{\boldsymbol{z}}_t = \boldsymbol{W}_{\boldsymbol{z}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\boldsymbol{z}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{z}} \qquad \text{cell input} \qquad (24)$$

$$\mathbf{i}_t = \sigma\left(\tilde{\mathbf{i}}_t\right) \,, \qquad \tilde{\mathbf{i}}_t = \boldsymbol{W}_{\mathbf{i}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{i}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{i}} \qquad \text{input gate} \qquad (25)$$

$$\mathbf{f}_t = \sigma\left(\tilde{\mathbf{f}}_t\right) \,, \qquad \tilde{\mathbf{f}}_t = \boldsymbol{W}_{\mathbf{f}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{f}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{f}} \qquad \text{forget gate} \qquad (26)$$

$$\mathbf{o}_t = \sigma\left(\tilde{\mathbf{o}}_t\right) \,, \qquad \tilde{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{o}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{o}} \qquad \text{output gate} \qquad (27)$$

The matrices $\boldsymbol{W}_{\boldsymbol{z}}$, $\boldsymbol{W}_{\mathbf{i}}$, $\boldsymbol{W}_{\mathbf{f}}$, and $\boldsymbol{W}_{\mathbf{o}}$ correspond to the input weights between inputs $\boldsymbol{x}_t$ and cell input, input gate, forget gate, and output gate, respectively. The matrices $\boldsymbol{R}_{\boldsymbol{z}}$, $\boldsymbol{R}_{\mathbf{i}}$, $\boldsymbol{R}_{\mathbf{f}}$, and $\boldsymbol{R}_{\mathbf{o}}$ correspond to the recurrent weights between hidden state $\boldsymbol{h}_{t-1}$ and cell input, input gate, forget gate, and output gate, respectively. $\boldsymbol{b}_{\boldsymbol{z}}$, $\boldsymbol{b}_{\mathbf{i}}$, $\boldsymbol{b}_{\mathbf{f}}$, and $\boldsymbol{b}_{\mathbf{o}}$ are the corresponding bias vectors. φ and ψ are the cell input and hidden state activation functions (typically $\tanh$). ψ is used to normalize or squash the cell state, which would be unbounded otherwise.

B.2. sLSTM

Similar to the LSTM in Section B.1, also the sLSTM can be vectorized to multiple cells:

$$\boldsymbol{c}_t = \mathbf{f}_t \odot \boldsymbol{c}_{t-1} + \mathbf{i}_t \odot \boldsymbol{z}_t \qquad \text{cell state} \qquad (28)$$

$$\boldsymbol{n}_t = \mathbf{f}_t \odot \boldsymbol{n}_{t-1} + \mathbf{i}_t \qquad \text{normalizer state} \qquad (29)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t \,, \qquad \tilde{\boldsymbol{h}}_t = \boldsymbol{c}_t \odot \boldsymbol{n}_t^{-1} \qquad \text{hidden state} \qquad (30)$$

$$\boldsymbol{z}_t = \varphi\left(\tilde{\boldsymbol{z}}_t\right) \,, \qquad \tilde{\boldsymbol{z}}_t = \boldsymbol{W}_{\boldsymbol{z}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\boldsymbol{z}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{z}} \qquad \text{cell input} \qquad (31)$$

$$\mathbf{i}_t = \exp\left(\tilde{\mathbf{i}}_t\right) \,, \qquad \tilde{\mathbf{i}}_t = \boldsymbol{W}_{\mathbf{i}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{i}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{i}} \qquad \text{input gate} \qquad (32)$$

$$\mathbf{f}_t = \exp\left(\tilde{\mathbf{f}}_t\right) \text{ OR } \sigma\left(\tilde{\mathbf{f}}_t\right) \,, \qquad \tilde{\mathbf{f}}_t = \boldsymbol{W}_{\mathbf{f}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{f}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{f}} \qquad \text{forget gate} \qquad (33)$$

$$\mathbf{o}_t = \sigma\left(\tilde{\mathbf{o}}_t\right) \,, \qquad \tilde{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}}\, \boldsymbol{x}_t + \boldsymbol{R}_{\mathbf{o}}\, \boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathbf{o}} \qquad \text{output gate} \qquad (34)$$

Here, the cell input activation function φ is $\tanh$, the hidden state activation function is the identity. φ helps stabilizing the recurrence.

Considering external gradient contribution $\delta_{\boldsymbol{h}_t}^{\text{ext}}$ from subsequent layers and recurrent gradient contribution $\delta_{\boldsymbol{h}_t}^{\boldsymbol{R}}$ from gradients from future states flowing over the cell interaction matrix $\boldsymbol{R}$, we obtain the recursive backward pass of sLSTM, where δ_a indicates gradients with respect to parameter / internal variable a:

$$\delta_{\boldsymbol{h}_t} = \delta^{ext}_{\boldsymbol{h}_t} + \delta^{\boldsymbol{R}}_{\boldsymbol{h}_t} \tag{35}$$

$$\delta_{\boldsymbol{c}_{t-1}} = \mathbf{f}_t \odot \delta_{\boldsymbol{c}_t} + \mathbf{o}_{t-1} \odot \boldsymbol{n}_{t-1}{}^{-1} \odot \delta_{\boldsymbol{h}_{t-1}} \tag{36}$$

$$\delta_{\boldsymbol{n}_{t-1}} = \mathbf{f}_t \odot \delta_{\boldsymbol{n}_t} - \mathbf{o}_{t-1} \odot \boldsymbol{c}_{t-1} \odot \boldsymbol{n}^{-2}_{t-1} \odot \delta_{\boldsymbol{h}_{t-1}} \tag{37}$$

$$\delta_{\tilde{\mathbf{f}}_t} = \mathbf{f}'_t \odot \boldsymbol{c}_{t-1} \odot \delta_{\boldsymbol{c}_t} + \mathbf{f}'_t \odot \boldsymbol{n}_{t-1} \odot \delta_{\boldsymbol{n}_t} \tag{38}$$

$$\delta_{\tilde{\mathbf{i}}_t} = \mathbf{i}'_t \odot \boldsymbol{z}_t \odot \delta_{\boldsymbol{c}_t} + \mathbf{i}'_t \odot \delta_{\boldsymbol{n}_t} \tag{39}$$

$$\delta_{\tilde{\mathbf{z}}_t} = \mathbf{i}_t \odot \varphi'(\tilde{\mathbf{z}}_t) \odot \delta_{\boldsymbol{c}_t} \tag{40}$$

$$\delta_{\tilde{\mathbf{o}}_t} = \mathbf{o}'_t \odot \boldsymbol{c}_t \odot \boldsymbol{n}_t^{-1} \odot \delta_{\boldsymbol{h}_t} \tag{41}$$

$$\delta_{\mathbf{x}_t} = \sum_{\mathbf{g} \in \{\mathbf{f},\mathbf{i},\mathbf{z},\mathbf{o}\}} \boldsymbol{W}_{\mathbf{g}}^\top \delta_{\tilde{\mathbf{g}}_t} \tag{42}$$

$$\delta^{\boldsymbol{R}}_{\boldsymbol{h}_{t-1}} = \sum_{\mathbf{g} \in \{\mathbf{f},\mathbf{i},\mathbf{z},\mathbf{o}\}} \boldsymbol{R}_{\mathbf{g}}^\top \delta_{\tilde{\mathbf{g}}_t} \tag{43}$$

$$\delta^\top_{\boldsymbol{R}_{\mathbf{g}}} = \sum_t \boldsymbol{h}_{t-1} \delta^\top_{\tilde{\mathbf{g}}_t}\,, \qquad \mathbf{g} \in \{\mathbf{i},\mathbf{f},\mathbf{z},\mathbf{o}\} \tag{44}$$

$$\delta^\top_{\boldsymbol{W}_{\mathbf{g}}} = \sum_t \boldsymbol{x}_t \delta^\top_{\tilde{\mathbf{g}}_t}\,, \qquad \mathbf{g} \in \{\mathbf{i},\mathbf{f},\mathbf{z},\mathbf{o}\} \tag{45}$$

with the derivatives of the respective gate activation function $\mathbf{i}'_t = \exp'(\tilde{\mathbf{i}}_t) = \exp(\tilde{\mathbf{i}}_t) = \mathbf{i}_t$, $\mathbf{o}'_t = \sigma'(\tilde{\mathbf{o}}_t)$, and $\mathbf{f}'_t = \sigma'(\tilde{\mathbf{f}}_t)$ or $\mathbf{f}'_t = \mathbf{f}_t$ depending on the forget gate activation. $\varphi'(z)$ is the derivative of the cell input activation function $\varphi(z)$.

The matrices $\boldsymbol{R}_{\boldsymbol{z}}$, $\boldsymbol{R}_{\mathbf{i}}$, $\boldsymbol{R}_{\mathbf{f}}$, $\boldsymbol{R}_{\mathbf{o}}$ are block-diagonal which is analogous to multiple heads in the mLSTM. This way, the parameters reduce to $d^2/(N_h)$, where N_h is the number of heads, limiting the cell interactions to individual heads. This parameter efficient formulation of cell interactions together with the exponential gating is called the new memory mixing. Finally, to stabilize the backward pass, we clip the magnitude of $\delta^{\boldsymbol{R}}_{\boldsymbol{h}_t}$ to 10, as a means to prohibit exploding gradients for long context lengths.

sLSTM Stabilized Version. The stabilized version of sLSTM introduces a new stabilizer state m, applied as:

$$m_t = \max\left(\log(\mathrm{f}_t) + m_{t-1}, \log(\mathrm{i}_t)\right) \qquad \text{stabilizer state} \tag{46}$$

$$\mathrm{i}'_t = \exp\left(\log\left(\mathrm{i}_t\right) - m_t\right) = \exp\left(\tilde{\mathrm{i}}_t - m_t\right) \qquad \text{stabil. input gate} \tag{47}$$

$$\mathrm{f}'_t = \exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1} - m_t\right) \qquad \text{stabil. forget gate} \tag{48}$$

We show that replacing f_t by f'_t and i_t by i'_t in the forward pass does neither change the output of the whole network nor the derivatives of the loss with respect to the parameters.

The stabilization state m has no gradient, and hence does not influence the other gradients. We re-define $c^{(s)}_t$ and $n^{(s)}_t$ as stabilized cell and normalizer states:

$$c_t = c^{(s)}_t \exp\left(m_t\right) \tag{49}$$

$$n_t = n^{(s)}_t \exp\left(m_t\right) \tag{50}$$

Inserting Equation 46 into Equation 1 yields:

$$\tilde{h}_t^{(s)} = c_t^{(s)} / n_t^{(s)} = \tag{51}$$

$$= \frac{\exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1} - m_t\right) c_{t-1}^{(s)} + \exp\left(\log\left(\mathrm{i}_t\right) - m_t\right) z_t}{\exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1} - m_t\right) n_{t-1}^{(s)} + \exp\left(\log\left(\mathrm{i}_t\right) - m_t\right)} \tag{52}$$

$$= \frac{\exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1}\right) c_{t-1}^{(s)} + \exp\left(\log\left(\mathrm{i}_t\right)\right) z_t}{\exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1}\right) n_{t-1}^{(s)} + \exp\left(\log\left(\mathrm{i}_t\right)\right)} \tag{53}$$

$$= \frac{\exp\left(\log\left(\mathrm{f}_t\right)\right) c_{t-1} + \exp\left(\log\left(\mathrm{i}_t\right)\right) z_t}{\exp\left(\log\left(\mathrm{f}_t\right)\right) n_{t-1} + \exp\left(\log\left(\mathrm{i}_t\right)\right)} \tag{54}$$

$$= \frac{\mathrm{f}_t c_{t-1} + \mathrm{i}_t z_t}{\mathrm{f}_t n_{t-1} + \mathrm{i}_t} = c_t / n_t = \tilde{h}_t \tag{55}$$

Therefore, since the loss solely depends on h_t, there's no dependency on m_t, and consequently, no gradient exists for this stabilization state. Note that m_t can be chosen arbitrarily. We choose $m_t = \max\left(\log\left(\mathbf{f}_t\right) + m_{t-1}, \log\left(\mathbf{i}_t\right)\right)$, which stabilizes the exponential function. One can even find m_t, such that the normalizer state n_t can be eliminated, but this version was experimentally found to be numerically unstable in the backward pass.

B.3. mLSTM

Throughout this section, $\mathbf{1} \in \mathbb{R}^T$ denotes a column vector of ones and $\mathbf{1}^\top \in \mathbb{R}^{1\times T}$ a row vector of ones, where T is the dimension of this vector space.

Recurrent mLSTM Backward Pass. The recurrent formulation of the mLSTM cell in Equation 9 yields the following backward pass recurrence, where δ_a indicates gradients with respect to parameter or internal variable a and $\delta_{\boldsymbol{h}_t}^{\text{ext}}$ denotes gradients from subsequent layers:

$$\delta_{\tilde{\boldsymbol{h}}_t}^\top = \mathbf{o}_t \odot \delta_{\boldsymbol{h}_t}^{\text{ext}} \tag{56}$$

$$\delta_{\boldsymbol{C}_{t-1}} = \mathrm{f}_t \delta_{\boldsymbol{C}_t} + \frac{\boldsymbol{q}_{t-1} \delta_{\tilde{\boldsymbol{h}}_{t-1}}^\top}{\max\left\{\left|\boldsymbol{n}_{t-1}^\top \boldsymbol{q}_{t-1}\right|, 1\right\}} \tag{57}$$

$$\delta_{\boldsymbol{n}_{t-1}} = \mathrm{f}_t \delta_{\boldsymbol{n}_t} - \frac{\boldsymbol{q}_{t-1}^\top \boldsymbol{C}_{t-1}^\top \delta_{\tilde{\mathbf{h}}_{t-1}}}{\max\left\{\left|\boldsymbol{n}_{t-1}^\top \boldsymbol{q}_{t-1}\right|, 1\right\}^2} \Omega\left(\boldsymbol{n}_{t-1}^\top \boldsymbol{q}_{t-1}\right) \boldsymbol{q}_{t-1} \tag{58}$$

$$\delta_{\boldsymbol{v}_t}^\top = \mathrm{i}_t \boldsymbol{k}_t^\top \delta_{\boldsymbol{C}_t}^\top \tag{59}$$

$$\delta_{\boldsymbol{k}_t}^\top = \mathrm{i}_t \left(\boldsymbol{v}_t^\top \delta_{\boldsymbol{C}_t} + \delta_{\boldsymbol{n}_t}^\top\right) \tag{60}$$

$$\delta_{\boldsymbol{q}_t} = \frac{\boldsymbol{C}_t^\top \delta_{\tilde{\boldsymbol{h}}_t}}{\max\left\{\left|\boldsymbol{n}_t^\top \boldsymbol{q}_t\right|, 1\right\}} - \frac{\boldsymbol{q}_t^\top \boldsymbol{C}_t^\top \delta_{\tilde{\boldsymbol{h}}_t}}{\max\left\{\left|\boldsymbol{n}_t^\top \boldsymbol{q}_t\right|, 1\right\}^2} \Omega\left(\boldsymbol{n}_t^\top \boldsymbol{q}_t\right) \boldsymbol{n}_t \tag{61}$$

$$\delta_{\boldsymbol{x}_t} = \sum_{g \in \{q,k,v\}} \boldsymbol{W}_g^\top \delta_{\boldsymbol{g}_t} \tag{62}$$

$$\delta_{\boldsymbol{W}_g}^\top = \sum_t \boldsymbol{x}_t \delta_{\boldsymbol{g}_t}^\top, \qquad g \in \{q,k,v\} \tag{63}$$

$$\delta_{\boldsymbol{b}_g} = \sum_t \delta_{\boldsymbol{g}_t}, \qquad g \in \{q,k,v\} \tag{64}$$

$$\delta_{\tilde{\mathrm{f}}_t} = \left(\mathbf{1}^\top \left(\boldsymbol{C}_{t-1} \odot \delta_{\boldsymbol{C}_t}\right) \mathbf{1} + \mathbf{1}^\top \left(\boldsymbol{n}_{t-1} \odot \delta_{\boldsymbol{n}_t}\right)\right) \gamma\left(\tilde{\mathrm{f}}_t\right) \tag{65}$$

$$\delta_{\tilde{\mathrm{i}}_t} = \left(\mathbf{1}^\top \left(\left(\boldsymbol{v}_t \boldsymbol{k}_t^\top\right) \odot \delta_{\boldsymbol{C}_t}\right) \mathbf{1} + \mathbf{1}^\top \left(\boldsymbol{k}_t \odot \delta_{\boldsymbol{n}_t}\right)\right) \exp\left(\tilde{\mathrm{i}}_t\right) \tag{66}$$

$$\delta_{\tilde{\mathbf{o}}_t} = \tilde{\boldsymbol{h}}_t \odot \sigma'\left(\tilde{\mathrm{o}}_t\right) \odot \delta_{\boldsymbol{h}_t} \tag{67}$$

and $\Omega(z) = \Theta(z-1) - \Theta(-z-1)$, $\Theta(z)$ being the Heaviside step function. $\gamma(z)$ is either $\sigma'(z)$ or $\exp(z)$, depending on the forget gate activation.

Parallel mLSTM Forward Pass. The mLSTM recurrence in Equations (9-15) can be reformulated in a parallel form, which is used to speed up training. After training we can still use the recurrent formulation for fast text generation.

Instead of processing each input $\boldsymbol{x}_t \in \mathbb{R}^d$ at time step t sequentially, the parallel version processes all timesteps of a full sequence $\boldsymbol{X} \in \mathbb{R}^{T \times d}$ at once, where T is the sequence length and d is the head dimension. We present the forward pass of the mLSTM for a single head and drop the head dimension for simplicity.

Let $\tilde{\mathbf{f}} \in \mathbb{R}^T$ be the forget gate pre-activations and $\tilde{\mathbf{i}} \in \mathbb{R}^T$ be the input gate pre-activations for a full sequence. We construct the forget gate activation matrix $\mathbf{F} \in \mathbb{R}^{T \times T}$ by

$$\mathbf{F}_{ij} = \begin{cases} 0 & \text{for } i < j \\ 1 & \text{for } i = j \\ \prod_{k=j+1}^{i} \sigma\left(\tilde{\mathrm{f}}_k\right) & \text{for } i > j \end{cases}, \tag{68}$$

and the input gate pre-activation matrix $\tilde{\mathbf{I}} \in \mathbb{R}^{T \times T}$ by

$$\tilde{\mathbf{I}}_{ij} = \begin{cases} 0 & \text{for } i < j \\ \mathrm{i}_j & \text{for } i \geqslant j \end{cases}. \tag{69}$$

By applying the elementwise exponential input gate activation function naively, we obtain the unstabilized gate activation matrix $\mathbf{D} \in \mathbb{R}^{T \times T}$ as

$$\mathbf{D} = \mathbf{F} \odot \exp(\tilde{\mathbf{I}}) \,. \tag{70}$$

In order to avoid overflow due to the exponential function we apply the same stabilization as in the recurrent sLSTM, see Equation 46. In the parallel formulation of the mLSTM we get a numerically stable gate activation matrix $\mathbf{D}' \in \mathbb{R}^{T \times T}$ by taking the logarithm of $\mathbf{D}$ element-wise and subtracting the row-wise maximum value of $\mathbf{D}$ from each element:

$$\widetilde{\mathbf{D}} = \log \mathbf{D} = \log\left(\mathbf{F} \odot \exp(\tilde{\mathbf{I}})\right) = \log \mathbf{F} + \tilde{\mathbf{I}} \tag{71}$$

$$\mathbf{D}' = \exp(\widetilde{\mathbf{D}} - \max \widetilde{\mathbf{D}}) \tag{72}$$

Given the queries, keys and values $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{T \times d}$, for a full sequence we can compute all hidden pre-activation states $\widetilde{\mathbf{H}} \in \mathbb{R}^{T \times d}$ in parallel for the un-stabilized version by

$$\widetilde{\mathbf{H}} = \boldsymbol{C}\,\boldsymbol{V} \,, \quad \text{with } \boldsymbol{C} = \frac{\widetilde{\boldsymbol{C}}}{\max\{|\sum_{j=1}^{T} \widetilde{\boldsymbol{C}}_{ij}|, 1\}} \,, \quad \text{and } \widetilde{\boldsymbol{C}} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \odot \mathbf{D} \,. \tag{73}$$

Note that we extract the $\frac{1}{\sqrt{d}}$ factor for $\boldsymbol{K}$ explicitly here and further on. For the stabilized version this yields

$$\widetilde{\mathbf{H}} = \boldsymbol{C}\,\boldsymbol{V} \,, \quad \text{with } \boldsymbol{C} = \frac{\widetilde{\boldsymbol{C}}'}{\max\{|\sum_{j=1}^{T} \widetilde{\boldsymbol{C}}'_{ij}|, \exp(-\max \widetilde{\mathbf{D}})\}} \,, \quad \text{and } \widetilde{\boldsymbol{C}}' = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \odot \mathbf{D}' \,, \tag{74}$$

where for both versions the hidden pre-activation states $\widetilde{\mathbf{H}}$ are identical.

With the output gate pre-activations $\widetilde{\mathbf{O}} \in \mathbb{R}^{T \times d}$ we can compute the hidden states $\boldsymbol{H} \in \mathbb{R}^{T \times d}$ for all timesteps by applying the output gate in parallel for each timestep element-wise:

$$\mathbf{H} = \sigma(\widetilde{\mathbf{O}}) \odot \widetilde{\mathbf{H}} \,. \tag{75}$$

This gives the parallel forward pass of the mLSTM for a full input sequence $\boldsymbol{X} \in \mathbb{R}^{T \times d}$.

Parallel mLSTM Backward Pass. We present the backward pass of the mLSTM for the stabilized version only. For completeness we summarize the forward pass in the stabilized version before we present the backward pass.

Given the forget gate matrix $\mathbf{F} \in \mathbb{R}^{T \times T}$, the logarithm of the forget gate matrix $\overline{\mathbf{F}} = \log \mathbf{F} \in \mathbb{R}^{T \times T}$, and the input gate matrix $\mathbf{I} \in \mathbb{R}^{T \times T}$ as introduced above, together with the queries, keys and values $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{T \times d}$, we can write the forward pass of the mLSTM in the stabilized version as:

$$\widetilde{\mathbf{D}} = \overline{\mathbf{F}} + \tilde{\mathbf{I}} \tag{76}$$

$$\boldsymbol{m} = \max_j \widetilde{\mathbf{D}}_{ij} \ , \qquad \text{row-wise maximum} \tag{77}$$

$$\mathbf{D}' = \exp(\widetilde{\mathbf{D}} - \boldsymbol{m}\, \mathbf{1}^\top) \tag{78}$$

$$\widetilde{\boldsymbol{C}}' = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \odot \mathbf{D}' \tag{79}$$

$$\boldsymbol{b} = \sum_{j=1}^{T} \widetilde{\boldsymbol{C}}'_{ij} = \widetilde{\boldsymbol{C}}'\, \mathbf{1} \ , \qquad \text{row-wise sum} \tag{80}$$

$$\boldsymbol{n} = \max\{|\boldsymbol{b}|, \exp(-\boldsymbol{m})\} \tag{81}$$

$$\boldsymbol{C} = \widetilde{\boldsymbol{C}}' \odot \left(\boldsymbol{n}^{-1}\, \mathbf{1}^\top\right) \tag{82}$$

$$\widetilde{\mathbf{H}} = \boldsymbol{C}\, \boldsymbol{V} \tag{83}$$

With this forward pass we can compute the gradients δ_a for all intermediate and input variables to the mLSTM forward pass in the backward pass. We denote the gradient with respect to variable a as δ_a.

Given the output gradient $\delta_{\widetilde{\mathbf{H}}} \in \mathbb{R}^{T \times d}$ we can compute the backward pass for the intermediate gradients as:

$$\delta_{\boldsymbol{C}}^\top = \boldsymbol{V} \delta_{\widetilde{\mathbf{H}}}^\top \tag{84}$$

$$\delta_{\boldsymbol{n}} = -\left(\widetilde{\boldsymbol{C}}' \odot \left(\boldsymbol{n}^{-2}\, \mathbf{1}^\top\right) \odot \delta_{\boldsymbol{C}}\right)\, \mathbf{1} \tag{85}$$

$$= -\left(\left(\widetilde{\boldsymbol{C}}' \odot \delta_{\boldsymbol{C}}\right) \mathbf{1}\right) \odot \boldsymbol{n}^{-2} \tag{86}$$

$$\delta_{\boldsymbol{b}} = \operatorname{sign}(\boldsymbol{n}) \odot \delta_{\boldsymbol{n}} \odot \begin{cases} 1 & \text{if } |\boldsymbol{b}| > \exp(-\mathbf{m}) \\ 0 & \text{otherwise} \end{cases} \tag{87}$$

$$\delta_{\widetilde{\boldsymbol{C}}', \boldsymbol{C}} = \left(\boldsymbol{n}^{-1}\, \mathbf{1}^\top\right) \odot \delta_{\boldsymbol{C}} \ , \qquad \text{column-wise broadcast} \tag{88}$$

$$\delta_{\widetilde{\boldsymbol{C}}', \boldsymbol{b}}^\top = \mathbf{1}\, \delta_{\boldsymbol{b}}^\top \ , \qquad \text{column-wise broadcast} \tag{89}$$

$$\delta_{\widetilde{\boldsymbol{C}}'} = \delta_{\widetilde{\boldsymbol{C}}', \boldsymbol{C}} + \delta_{\widetilde{\boldsymbol{C}}', \boldsymbol{B}} \tag{90}$$

$$\delta_{\mathbf{D}'} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \odot \delta_{\widetilde{\boldsymbol{C}}'} \tag{91}$$

$$\delta_{\widetilde{\mathbf{D}}} = \exp(\widetilde{\mathbf{D}} - \boldsymbol{m}) \odot \delta_{\mathbf{D}'} = \mathbf{D}' \odot \delta_{\mathbf{D}'} \tag{92}$$

We do not compute the gradients for $\boldsymbol{m}$ as they cancel out (see the proof in the recurrent sLSTM).

With these intermediate gradients the gradients for the logarithmic forget gate matrix $\delta_{\overline{\mathbf{F}}} \in \mathbb{R}^{T \times T}$, the input gate matrix $\delta_{\mathbf{I}} \in \mathbb{R}^{T \times T}$, and the queries, keys and values $\delta_{\boldsymbol{Q}}, \delta_{\boldsymbol{K}}, \delta_{\boldsymbol{V}} \in \mathbb{R}^{T \times d}$ are given by

$$\delta_{\overline{\mathbf{F}}} = \delta_{\widetilde{\mathbf{D}}} \tag{93}$$

$$\delta_{\mathbf{I}} = \delta_{\widetilde{\mathbf{D}}} \tag{94}$$

$$\delta_{\boldsymbol{Q}} = \left(\mathbf{D}' \odot \delta_{\widetilde{\boldsymbol{C}}'}\right) \frac{\boldsymbol{K}}{\sqrt{d}} \tag{95}$$

$$\delta_{\boldsymbol{K}} = \left(\mathbf{D}' \odot \delta_{\widetilde{\boldsymbol{C}}'}\right)^\top \frac{\boldsymbol{Q}}{\sqrt{d}} \tag{96}$$

$$\delta_{\boldsymbol{V}} = \boldsymbol{C}^\top \delta_{\widetilde{\mathbf{H}}} \tag{97}$$

Having computed the gradients for the logarithmic forget gate matrix $\delta_{\overline{\mathbf{F}}}$, we can compute the gradients for the forget gate pre-activations $\delta_{\tilde{\mathbf{f}}} = \left[\delta_{\tilde{\mathrm{f}}_1}, \delta_{\tilde{\mathrm{f}}_2}, ..., \delta_{\tilde{\mathrm{f}}_T}\right]^\top \in \mathbb{R}^T$.

Recall the logarithmic forget gate matrix $\overline{\mathbf{F}} = \log \mathbf{F}$ is computed by

$$\overline{\mathbf{F}}_{ij} = \log \mathbf{F}_{ij} = \begin{cases} -\infty & \text{for } i < j \\ 0 & \text{for } i = j \\ \sum_{k=j+1}^{i} \underbrace{\log \sigma\left(\tilde{\mathrm{f}}_k\right)}_{=:\bar{\mathrm{f}}_k} = \sum_{k=j+1}^{i} \bar{\mathrm{f}}_k & \text{for } i > j \end{cases} . \tag{98}$$

With the substitution $\overline{\mathbf{f}} = \log \sigma(\tilde{\mathbf{f}})$ we compute the gradients for the logarithmic forget gate activations $\delta_{\bar{\mathbf{f}}} = \left[\delta_{\bar{\mathrm{f}}_1}, \delta_{\bar{\mathrm{f}}_2}, ..., \delta_{\bar{\mathrm{f}}_T}\right]^\top \in \mathbb{R}^T$ as

$$\delta_{\bar{\mathrm{f}}_k} = \sum_{j=1}^{k-1} \sum_{i=k}^{T} \left(\delta_{\overline{\mathbf{F}}}\right)_{ij} , \tag{99}$$

$$\delta_{\tilde{\mathrm{f}}_k} = \sigma(-\tilde{\mathrm{f}}_k) \cdot \delta_{\bar{\mathrm{f}}_k} , \tag{100}$$

where the last equation makes use of the following:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} \left(\log \sigma(x)\right) &= -\left(1 + \exp(-x)\right)^{-1} \cdot \exp(-x) \cdot (-1) \\ &= \frac{\exp(-x)}{1 + \exp(-x)} = \frac{1}{1 + \exp(x)} \\ &= \sigma(-x) \end{aligned} \tag{101}$$

Finally, we compute the input gate pre-activations' gradients $\delta_{\tilde{\mathbf{i}}} = \left[\delta_{\tilde{\mathrm{i}}_1}, \delta_{\tilde{\mathrm{i}}_2}, ..., \delta_{\tilde{\mathrm{i}}_S}\right]^\top \in \mathbb{R}^T$ as the column-wise sum over the rows of the input gate matrix $\delta_{\mathbf{I}}$:

$$\delta_{\tilde{\mathrm{i}}_k} = \sum_{i=k}^{T} \left(\delta_{\mathbf{I}}\right)_{ik} \tag{102}$$

This completes the backward pass of the parallel mLSTM for a full input sequence $\boldsymbol{X} \in \mathbb{R}^{T \times d}$.

B.4. Detailed Block Structure

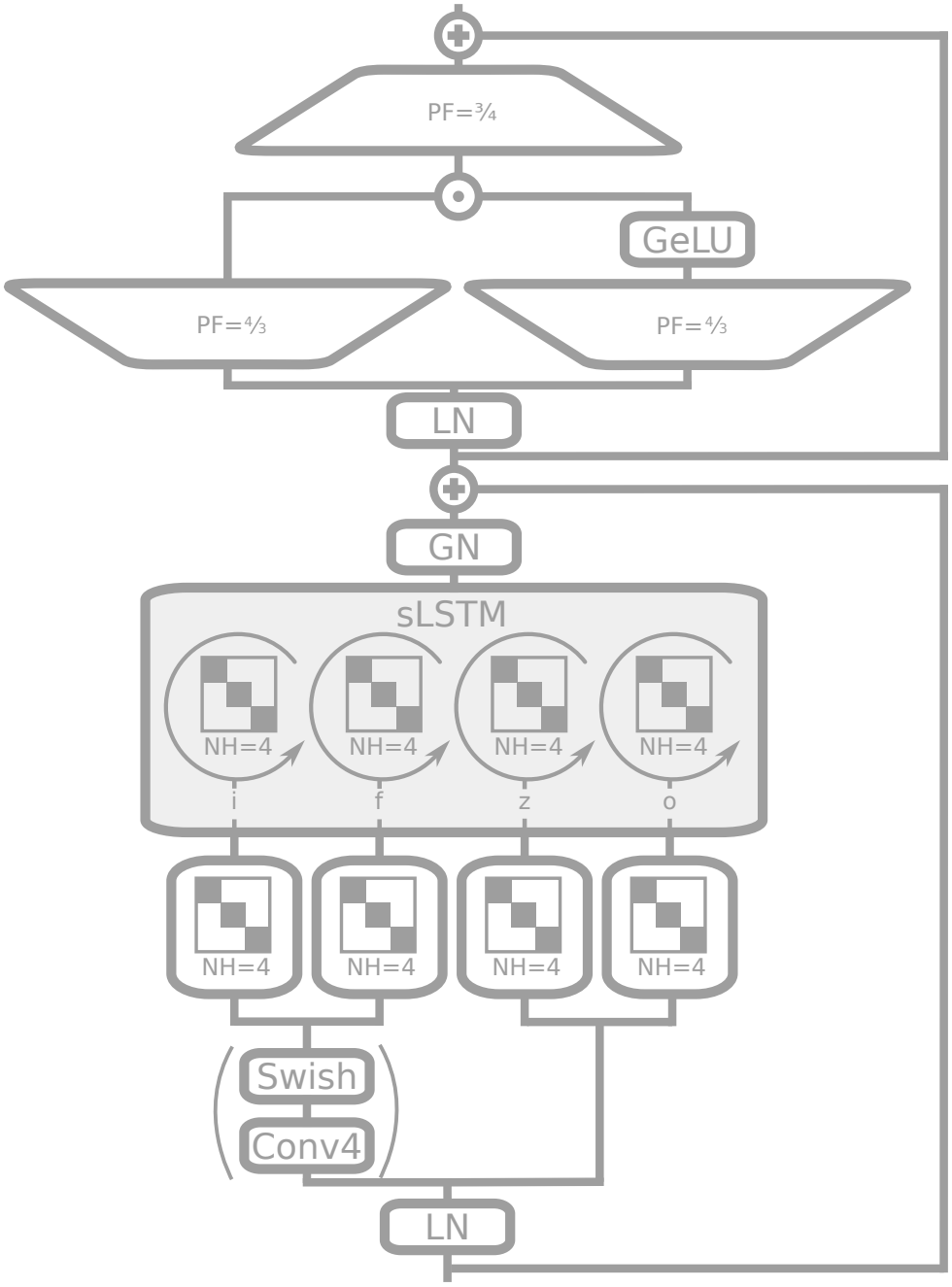

Figure 5: Schematic representation of an sLSTM Block – post up-projection: Embedded in a pre-LayerNorm residual structure, the input is optionally passed through a causal convolution of window size 4 that includes a Swish activation for input and forget gates. Then, for all input, forget and output gates i, f, o, and the cell update z the input is fed through a block-diagonal linear layer with four diagonal blocks or "heads". These diagonal blocks coincide with the recurrent gate pre-activations from the last hidden state, which corresponds to an sLSTM with four heads depicted with the circular arrows. The resulting hidden state goes through a GroupNorm layer (Wu & He, 2018) – a head-wise LayerNorm for each of the four heads. Finally, the output is up- and down-projected using a gated MLP, with GeLU activation function and projection factor $4/3$ to match parameters.

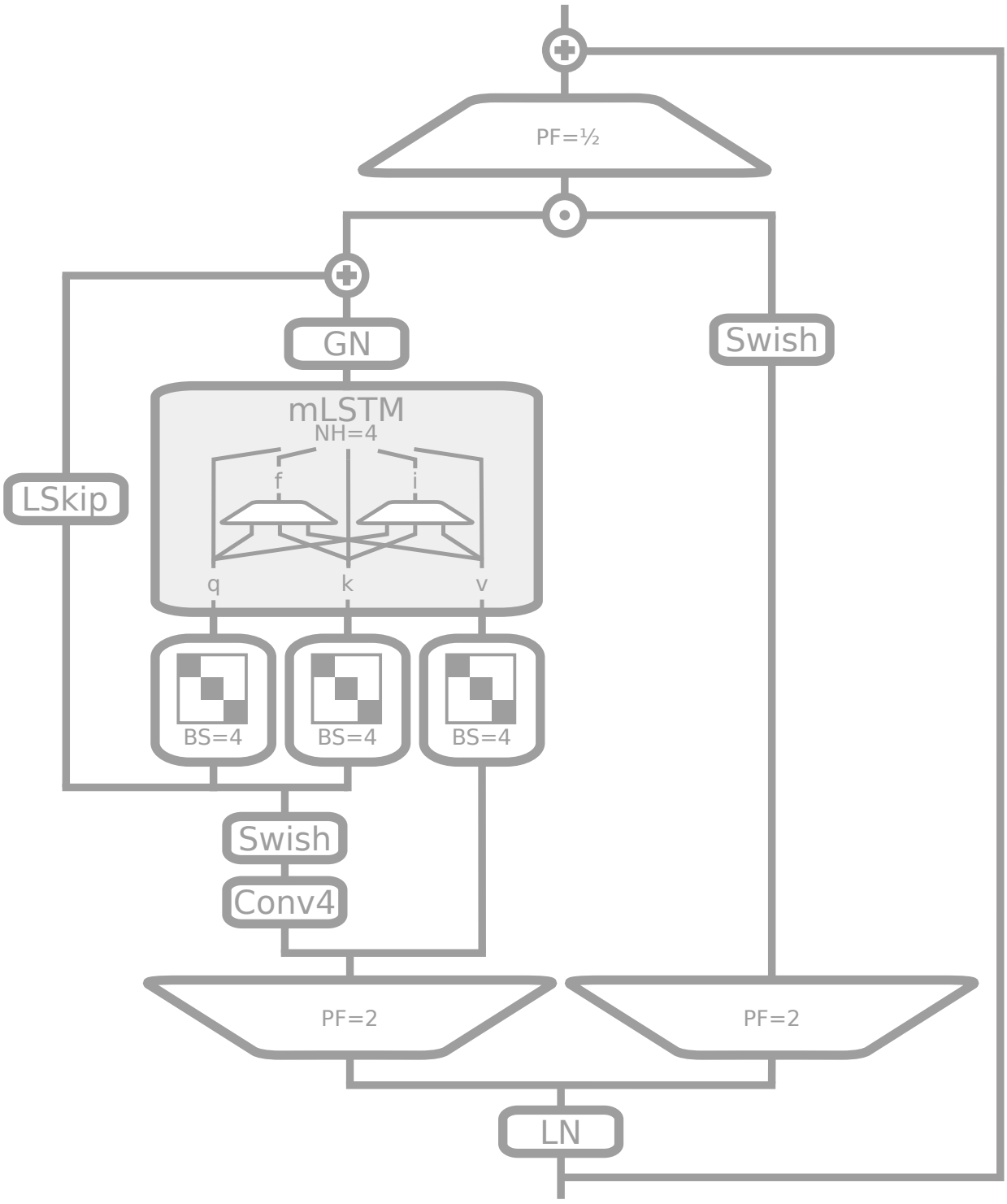

Figure 6: Schematic representation of an mLSTM block – pre up-projection: Embedded in a pre-LayerNorm residual structure, the input is up-projected first with projection factor 2, once for an externalized output gate and once as input for the mLSTM cells. The mLSTM cell input is dimension-wise causally convoluted (kernel size 4), before entering a learnable skip connection. We obtain input q and k via block-diagonal projection matrices of block size 4. The values v are fed directly, skipping the convolution part. After the mLSTM sequence mixing, outputs are normalized via GroupNorm (Wu & He, 2018) – a head-wise layer norm for each of the four heads. Finally, the learnable skip input is added and the result is gated component-wise with the external output gate. The output is down-projected.

B.5. Memory and Speed Considerations

Contrary to Transformers, xLSTM networks have a linear computation and a constant memory complexity with respect to the sequence length. Since the xLSTM memory is compressive, it is well suited for industrial applications and implementations on the edge. The memory of mLSTM does not require parameters, but is computationally expensive through its $d \times d$ matrix memory and $d \times d$ update. We trade off memory capacity against computational complexity. Nevertheless, the computations can be done in parallel on GPUs, therefore these computations have only a minor effect on the wall clock time.

While mLSTM is parallelizable analog to FlashAttention (Dao et al., 2022; Dao, 2024) or GLA (Yang et al., 2023), sLSTM is not parallelizable due to the memory mixing (hidden-hidden connections). However, we developed a fast CUDA implementation with GPU memory optimizations to the register level which is typically less than two times slower than mLSTM.

C. Limitations

(i) In contrast to mLSTM, memory mixing of the sLSTM prohibits parallelizable operations, and thus prevents a fast parallel implementation. However, we developed a fast CUDA kernel for sLSTM, which is currently less than two times slower than the parallel mLSTM implementation. (ii) The mLSTM CUDA kernels are not optimized, and therefore the current implementation is about four times slower than FlashAttention or the scan used in Mamba. Faster CUDA kernels could be obtained in the vein of FlashAttention. (iii) The matrix memory of mLSTM has high computation complexity since $d \times d$ matrices must be processed. Still, the memory update and retrieval is parameter-free, i.e., parallelizable when using standard matrix operations. Thus, the wall clock time overhead due to the complex memory is minor. (iv) The forget gate initialization must be chosen carefully. (v) Since the matrix memory is sequence length independent, increasing the sequence length might overload the memory for longer context sizes. Still, this does not appear to be a limitation for contexts up to 16k, see Section 3. (vi) Due to the expensive computational load for LLM experiments, we did neither fully optimize the architecture nor the hyperparameters, especially for larger xLSTM architectures. We anticipate that an extensive optimization process is needed for xLSTM to reach its full potential.

D. Experiments

Training Setup. For all experiments, we use Python[1] 3.11 with PyTorch 2.2.0[2], and CUDA 12.1[3]. We developed and trained all our models and baselines over the course of three months on a cluster with 128 nodes of eight NVIDIA A100 GPUs each. More than 95% of this compute were used for the Language Modeling experiments in sections D.2 and 3.

Nearest Neighbor Search Task. For this auxiliary task, we use randomly sampled feature vectors of dimension 2 and unit norm. The attached value is a uniformly distributed random number from $[0, 1]$, leading to inputs vectors of dimension 3. The first feature vector serves as search key, with the first value being ignored. Then the model has to predict the value of the nearest neighbor so far in the sequence. We train on 8192 sequences of context length up to 64 (uniformly sampled) and validate on 8192 different samples. All models have two blocks and embedding dimension 128. We use a dropout of 0.1, 10% linear warm-up steps and cosine decay to 1e-7 for 100k total training steps. We sweep over learning rates 1e-4, 1e-3, 1e-2, 1e-1 and 5 seeds each. The reported values in Figure 1 are mean values for the best learning rate and 99% confidence intervals. Note that LSTM requires very high learning rates, whereas Transformers (Llama) perform best at the smallest learning rate. The xLSTM[0:1] reaches similar performance across all learning rates.

Wikitext-103 Rare Token Prediction. For this exemplary experiment on rare token prediction, we trained 125M-sized models on Wikitext-103 (Merity et al., 2017). All models have an embedding dimension of 768 in a post up-projection structure of 12 residual blocks. The Transformer model (Llama) uses Multi-Head Attention, for what is called LSTM the Multi-Head Attention is replaced by an LSTM and the xLSTM[1:0] contains mLSTM layers with matrix memory. Models were trained with maximum learning rate 1e-3, 4k steps linear warm-up and cosine decay for in total 50k steps, using a batch size of 256 and context length of 512. We use the validation perplexity as a stopping criterion and evaluate on the test set.

D.1. Synthetic Tasks and Long Range Arena

Firstly, we test the effectiveness of xLSTM's new exponential gating with memory mixing on formal languages (Delétang et al., 2023). Then, we assess the effectiveness of xLSTM's new matrix memory on the Multi-Query Associative Recall task (Arora et al., 2023). Finally, xLSTM's performance at processing long sequences in the Long Range Arena is evaluated (Tay et al., 2021).

D.1.1. Test of xLSTM's Exponential Gating with Memory Mixing.

We test xLSTM's new exponential gating with memory mixing, which should enable it to solve state tracking problems (Merrill et al., 2024; Merrill & Sabharwal, 2023).

We implement and extend the formal language tasks from Delétang et al. (2023) to enable multi-length training for length extrapolation. Formal languages provide a framework to probe the generalization capabilities of models. They allow to specifically test different expressivity levels, e.g. along the Chomsky hierarchy. Typical language model architectures do not necessarily fit perfectly in these hierarchies (Delétang et al., 2023) — nevertheless these languages allow to illustrate differences in generalization expressivity between different architectures.

Experiment Setup. The different formal language tasks in the experiment (see individual tasks description below) encompass different levels of the Chomsky hierarchy as well as additional counting and memory-focused tasks. We use different lengths per sample, which allows us to validate in a length extrapolation setting. We train on a varying task length up to 40. The evaluation is done for task lengths between 40 and 256 as we are only interested in the "task generalization capabilities" of the models.

In all experiments, we use two blocks (or layers for the pure LSTM) for all models. We compare Llama, Mamba, Retention, Hyena, RWKV-4, RWKV-5, RWKV-6, LSTM, xLSTM[0:1], xLSTM[1:0] and xLSTM[1:1]. The sLSTM block is used without a convolution and with normal weight initialization. LSTM (Block) refers to an architecture where a vanilla LSTM is used instead of self-attention inside a Transformer block.

All models are trained with 3 different learning rates (1e-2, 1e-3, 1e-4), each with two seeds. Batch size is 256 — cosine

[1] https://python.org
[2] https://pytorch.org
[3] https://docs.nvidia.com/cuda/archive/12.1.0/

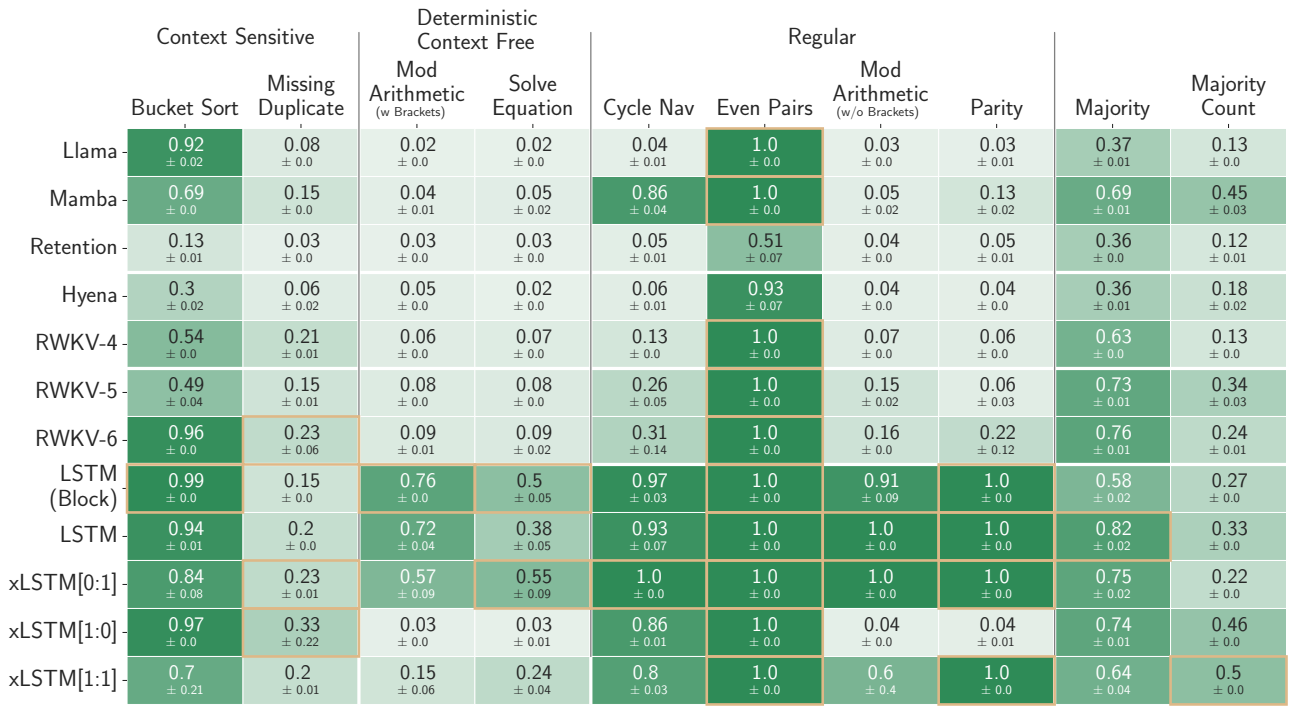

Figure 7: Results given by scaled accuracy of different models at solving formal language tasks. Tasks are grouped by the Chomsky hierarchy.

annealing (min lr: 1e-5) with 10% warm-up steps is applied. We use AdamW (Loshchilov & Hutter, 2019) and a weight decay of 0.1 for training. In each experiment we train for 100k steps — the samples are generated randomly, however, all models are trained and evaluated on the same samples.

The accuracy of the tested methods is evaluated on those tokens relevant to the task. The accuracy is scaled between 0 (random) and 1 (perfect).

Results. The main results of this experiment are shown in Figure 7. Models such as Transformers or State Space Models without memory mixing (no state tracking) cannot solve, e.g. regular grammars like the parity task. This result is in agreement with findings that Transformers and State Space models are fundamentally less powerful than RNNs (Merrill et al., 2024; Merrill & Sabharwal, 2023; Delétang et al., 2023). Figure 8 showcase supplementary results of the formal language task. It details tasks where no model attained a minimum scaled accuracy of 0.3. Although no model achieves proper extrapolation of the task to a larger context length, xLSTM performs best among the evaluated models.

Individual Task Description. The majority of tasks are based on Delétang et al. (2023). We provide the vocabulary size $|V|$ and the random accuracy s_{rand} (for accuracy scaling), used in the evaluation. As we evaluate different task lengths each task has a padding token which is used to pad the sequence to the given context length. In Listing 1 there is an example for each task.

- **Bucket Sort** Given a string of tokens of a sorted alphabet, compute the sorted string.
  $|V| = 11 \quad s_{\text{rand}} = \frac{1}{|V|-1}$
- **Cycle Nav** Given a string of "movement tokens" ($+1$, -1, STAY) compute the end position of the agent with start position 0. The position must be computed modulo the maximum position.
  $|V| = 9 \quad s_{\text{rand}} = \frac{1}{|V|-4}$
- **Even Pairs** Given a binary string of *a* and *b* tokens, compute whether the number of *ab* and *ba* is even. This task can be solved by checking if the first and last token of the string are equal.

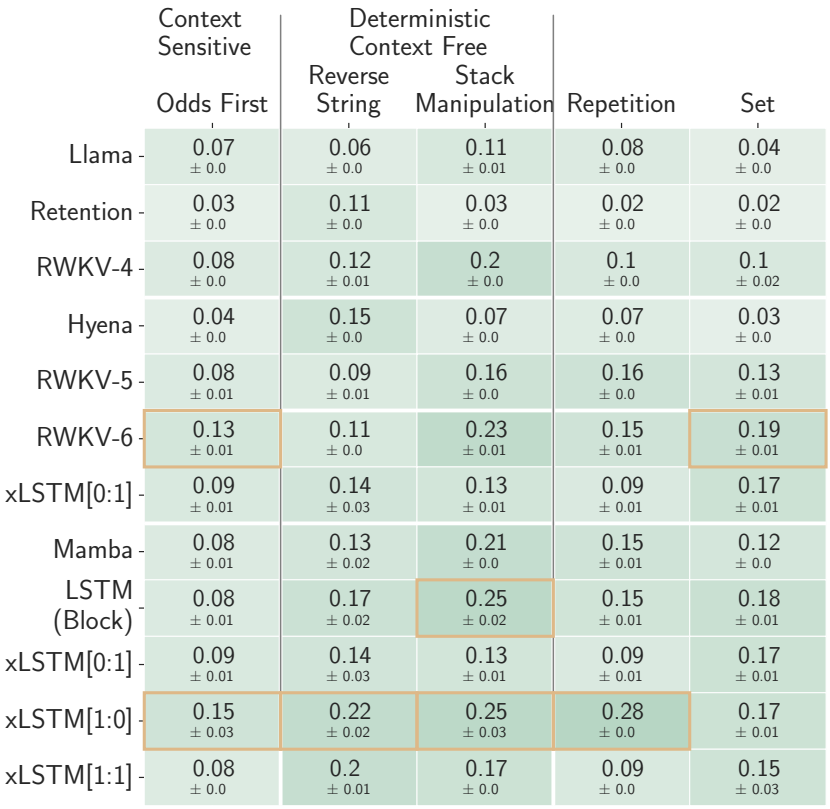

Figure 8: Additional results given by scaled accuracy of different models at solving formal language tasks. Tasks are grouped by the Chomsky hierarchy.

$|V| = 3 \quad s_{\text{rand}} = 0.5$

- **Majority** Given a string of tokens, compute the token that occurred most often in the sequence.
  $|V| = 64 \quad s_{\text{rand}} = \frac{1}{|V|-1}$
- **Majority Count** Given a string of tokens of an ordered alphabet. Compute the count of the token that occurred most often in the sequence. If the count exceeds the vocab size, the highest vocab token should be outputted.
  $|V| = 64 \quad s_{\text{rand}} = \frac{1}{|V|-1}$
- **Missing Duplicate** Given a string of tokens. The string is repeated but one of the tokens is masked in the repetition. Output the token that is masked.
  $|V| = 11 \quad s_{\text{rand}} = \frac{1}{|V|-2}$
- **Mod Arithmetic (w/o Brackets)** Calculate the result — modulo the max number — of the arithmetic operations in the context. The maximum number is the vocabulary size minus the number of special tokens (+,-,*,=, [PAD]).
  $|V| = 10 \quad s_{\text{rand}} = \frac{1}{|V|-5}$
- **Mod Arithmetic (w Brackets)** Calculate the result — modulo the maximum number — of the arithmetic operations in the context. The maximum number is vocabulary size minus the number of special tokens (+,-,*,=,(,), [PAD]).
  $|V| = 12 \quad s_{\text{rand}} = \frac{1}{|V|-7}$
- **Odds First** An string of tokens $t_1, t_2, t_3, ...t_n$ is given. Output all tokens with and odd index ($t_1, t_3, ...$) then the token with an even index (t_2, t_4,..) . Apart from that keep the ordering of the initial string.
  $|V| = 12 \quad s_{\text{rand}} = \frac{1}{|V|-2}$
- **Parity** Given a binary string of *a* and *b* tokens, compute if the number of *b*'s is even. If the number is even output *a* otherwise *b*. This is equivalent to sequentially calculating the half-adder sum.
  $|V| = 3 \quad s_{\text{rand}} = 0.5$
- **Repetition** Given a string of tokens — repeat it.
  $|V| = 12 \quad s_{\text{rand}} = \frac{1}{|V|-2}$
- **Reverse String** Given a string of tokens — repeat it in reverse order.
  $|V| = 12 \quad s_{\text{rand}} = \frac{1}{|V|-2}$
- **Stack Manipulation** An initial stack content is given, followed by a sequence of push and pop operations. Compute the stack content after the operations
  $|V| = 11 \quad s_{\text{rand}} = \frac{1}{\lfloor \frac{|V|-3}{2} \rfloor}$
- **Set** Given a string of tokens, compute the ordered set of the tokens. Keep the ordering so that tokens that occurred first are also outputted first.
  $|V| = 128 \quad s_{\text{rand}} = \frac{1}{|V|-2}$
- **Solve Equation** Given is an equation with the operators {+,-,*,=,(,)}, number, and an unknown variable x. Compute the value of the variable modulo the max number. The maximum number is vocabulary size minus the number of special tokens (+,-,*,=,(,), [PAD], [ACT]).
  $|V| = 14 \quad s_{\text{rand}} = \frac{1}{|V|-9}$

```
Bucket Sort
  Sequence: 1 4 8 6 1 1 1 4 6 8
Cycle Nav
  Sequence: STAY +1 −1 +1 STAY +1 +1 +1 −1 P3
Even Pairs
  Sequence: a b b a a b a b a a
Majority
  Sequence: 1 7 6 4 3 8 1 7 2 1
Majority Count
  Sequence: 1 7 6 4 4 8 1 7 2 2
Missing Duplicate
  Sequence: 4 8 6 2 5 4 8 6 2 [MIS] 5
Mod Arithmetic (w/o Braces)
  Sequence: 0 − 4 + 0 − 2 = 4 [PAD]
Mod Arithmetic (w Braces)
  Sequence: ( ( ( 2 ) * − 2 ) − ( − 4 − 2 ) ) = 2
Odds First
  Sequence: 2 7 3 2 6 9 [ACT] 2 3 6 7 2 9
Parity:
  Sequence: a b b a a b a b
Repetition
  Sequence: 2 4 8 6 2 [ACT] 2 4 8 6 2
Reverse String
  Sequence: 2 4 8 6 2 [ACT] 2 6 8 4 2
Stack Manipulation
  Sequence: ST1 ST1 ST3 POP POP PS3 PS3 [ACT] ST1 ST3 ST3
Set
  Sequence: 8 6 6 3 5 4 5 3 [ACT] 8 6 3 5 4
Solve Equation:
  Sequence: ( ( ( 2 + 0 ) + − x ) − ( 1 ) ) = 2 [ACT] 2
```

Listing 1: Examples of the formal language tasks. Red tokens are evaluated for loss and accuracy metrics, but are padded for the input. The tokens are illustrated in a way that allows easy semantic interpretation for the given task — hence, some tokens are represented by multiple characters.

D.1.2. Test of xLSTM's Memory Capacities on Associative Recall Tasks.

In these experiments, we test xLSTM's new matrix memory in terms of the memory capacity on the Multi-Query Associative Recall task (Arora et al., 2023). Figure 10 illustrates the basic task setup. To enhance the difficulty of the original task, we increase the number of key-value pairs up to 256 and enlarge the context length up to 2048, obtaining extended tests for the memory capacities of different models. We compare 2-block architectures of different models (see Experiment Setup). The models are evaluated by the accuracy at recalling the pairs. Since Transformers (e.g. Llama) have a memory that is exponential in the coding dimension (Ramsauer et al., 2021), they constitute the gold standard at this task. Results for the hardest evaluated task settings are shown in Figure 9. xLSTM[1:1] performs best among all non-Transformer models, also for small models. Interestingly, the sLSTM block does not diminish the memory capacity but rather leverages it, which becomes evident at the most difficult task with 256 key-value pairs.

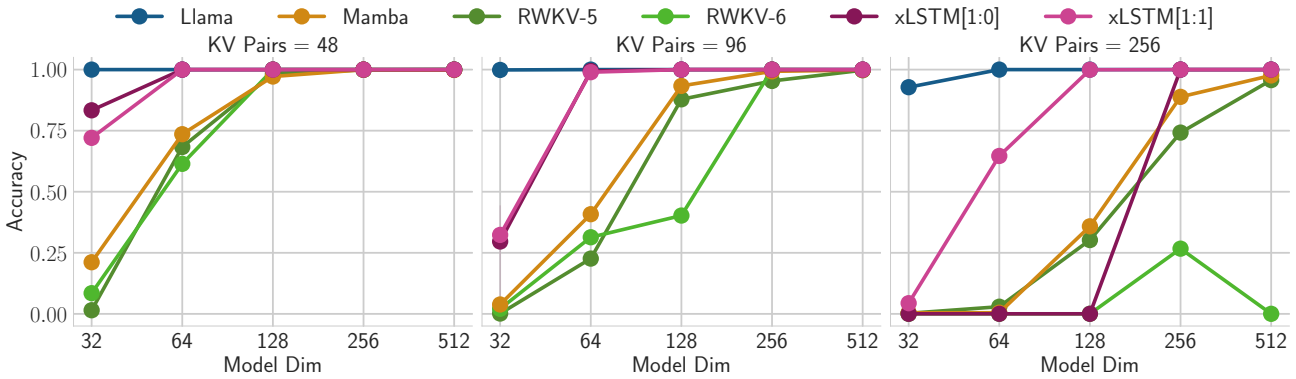

Figure 9: Test of memory capacities of different models at the Multi-Query Associative Recall task with context length 2048. Each panel is dedicated to a different number of key-value pairs. The x-axis displays the model size and the y-axis the validation accuracy.

Why Multi-Query Associative Recall for Memory Tests of LLM Architectures. Associative Recall (AR), the ability to retrieve a specific value (information) associated with a given key (information), constitutes a key capability for LLM to perform well (Poli et al., 2024; Arora et al., 2023; Olsson et al., 2022). Especially its quality of in-context learning seems to be strongly connected to this capability (Olsson et al., 2022). Arora et al. (2023) attribute performance gaps between early non-Transformer and Transformer language models specifically to performance gaps in associative recall. They argue that prior AR evaluations fall short of capturing these differences and propose MQAR, which can show the AR performance differences that translate to performance differences in language modeling performance. Hence, MQAR is especially suitable to analyze the memory capacity of LLM. Transformer (e.g. Llama) models can be seen as the gold standard for this task as their memory is exponential in the coding dimension (Ramsauer et al., 2021).

Experiment Setup. There are two relevant variables that determine different experimental setups. (1) **Context Length (CL)**: Length of the sequence of one sample — this influences the distances between the key-value definition and the recall. (2) **Number Key-Value Pairs (KV)**: Influences how many key-value pairs the model needs to keep track of. The vocabulary size is always 8192.

In all experiments, we use two blocks (or layers for the pure LSTM) for all models. LSTM (Block) model refers to an architecture where a vanilla LSTM is used instead of self-attention inside a Transformer block.

For each task setup, we train each model with 4 different learning rates (batch size > 24: {1e-2, 2.15e-3, 4.6e-4, 1e-4}, batch size 24: {1e-3, 2.2e-4, 5e-5, 1e-5}). The batch size (BS) changes depending on the context length (CL) (CL=64/128: BS=512; CL=256: BS=256; CL=756: BS=128; CL=1024: BS=96; CL=2048: BS=24). We vary the embedding dimension **(Model Dim)** between different experiments – different numbers of heads are used accordingly. For each experiment, we generate 100,000 training samples (validation: 3,000 samples) and train for 64 epochs. We apply cosine annealing (min lr: 1e-4 and 1e-5) with 10% warm-up steps. We use AdamW (Loshchilov & Hutter, 2019) and a weight decay of 0.1 for training.

We conduct three different experiments:

- **MQAR-Experiment 1** evaluates, in the same fashion as Arora et al. (2023), a variety of models (Llama, Mamba, Mamba (noWT) - i.e. without weight tying, Retention, Hyena, H3, RWKV-4, RWKV-5, RWKV-6, LSTM, LSTM (Block), xLSTM[0:1], xLSTM[1:0] and xLSTM[1:1]) on increasing task difficulty by increasing the context length and number of key-value pairs simultaneously. We benchmark three parameter settings: CL,KV={(64,4),(128,8),(256,16)}.
- **MQAR-Experiment 2** increases the task difficulty notably and goes beyond previous evaluations on this task. We individually scale the context length (CL={756, 1024, 2048}) and the key-value pairs (KV={48, 96, 256}) and evaluate all combinations. This experiment especially probes the memory capacity because the number of key-value pairs is high. To reduce the computational burden we only evaluate models that perform flawlessly in Experiment 1 — additionally we evaluate Transformer only in the hardest setting (CL=2048) as sanity check, because no performance decrease is expected.

- **MQAR-Experiment 3** analyzes whether the AR capability learned on a certain context length extrapolates to bigger context lengths. For each KV setting of Experiment 2, we use the models (we select the 3 biggest model dimensions) trained on CL=2048 and evaluate bigger context lengths (CL={4096, 6144, 8192}).

Extended Results. The result of Experiment 1 can be found in Figure 11. In accordance to the results of Arora et al. (2023). H3, Hyena, RWKV-4 fail to solve the task with a smaller model dimension. In contrast, xLSTM[1:1], xLSTM[1:0], Mamba, RWKV-5 and RWKV-6 are able to solve these settings for all model dimensions. The comparison of xLSTM[0:1] with both original LSTM variants indicates that the exponential gating mechanism improves the AR capabilities of the model. However, both fall short because of the reduced memory capacity compared to xLSTM[1:1] and xLSTM[1:0].

The results of Experiment 2 are presented in Figure 12. Scaling the context length has a low impact on the performance of the models. However, while xLSTM[1:1] and xLSTM[1:0] show no clear decay, both RWKV variants slightly, but consistently lose performance with increasing context lengths. The varying number of key-value pairs, which mainly probes the memory capacity of the non-Transformer models, has a more notable impact across all models. RWKV-5 seems to outperform RWKV-6. The latter fails to learn the task at all in some KV=256 settings. Overall xLSTM[1:1] is the best-performing non-Transformer model — suggesting that it provides enhanced memory capacity, also in long contexts.

Figure 13 shows the extrapolation results from Experiment 3. For xLSTM[1:1], xLSTM[1:0], and Mamba the model performance does not change in the extrapolation setting. The RWKV models (especially RWKV5) degrade slightly with increasing context length. xLSTM[1:1] performs best, as it maintains its superior performance of Experiment 2.

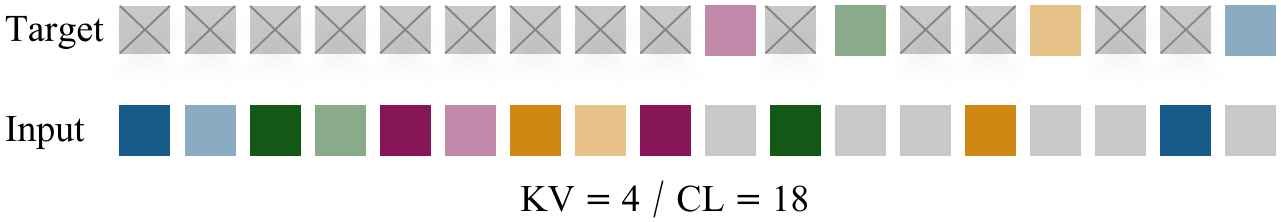

Figure 10: Illustration of the MQAR task. Color pairs represent key-value pairs (keys have darker shade). The first part of the sequence defines the key-value pairs for the respective sample. After that, the keys appear randomly according to a power law distribution [4]. Grey tokens in the input sequence represent a zero token. The "target" sequence contains the value after the respective key appearance — the rest of the tokens are ignored for the accuracy and loss calculation. The model must predict the value tokens given the respective key.

[4]The keys are distributed on the "evaluation part" of the sequence given a power-law distribution. This is motivated by similar structures in natural language text.

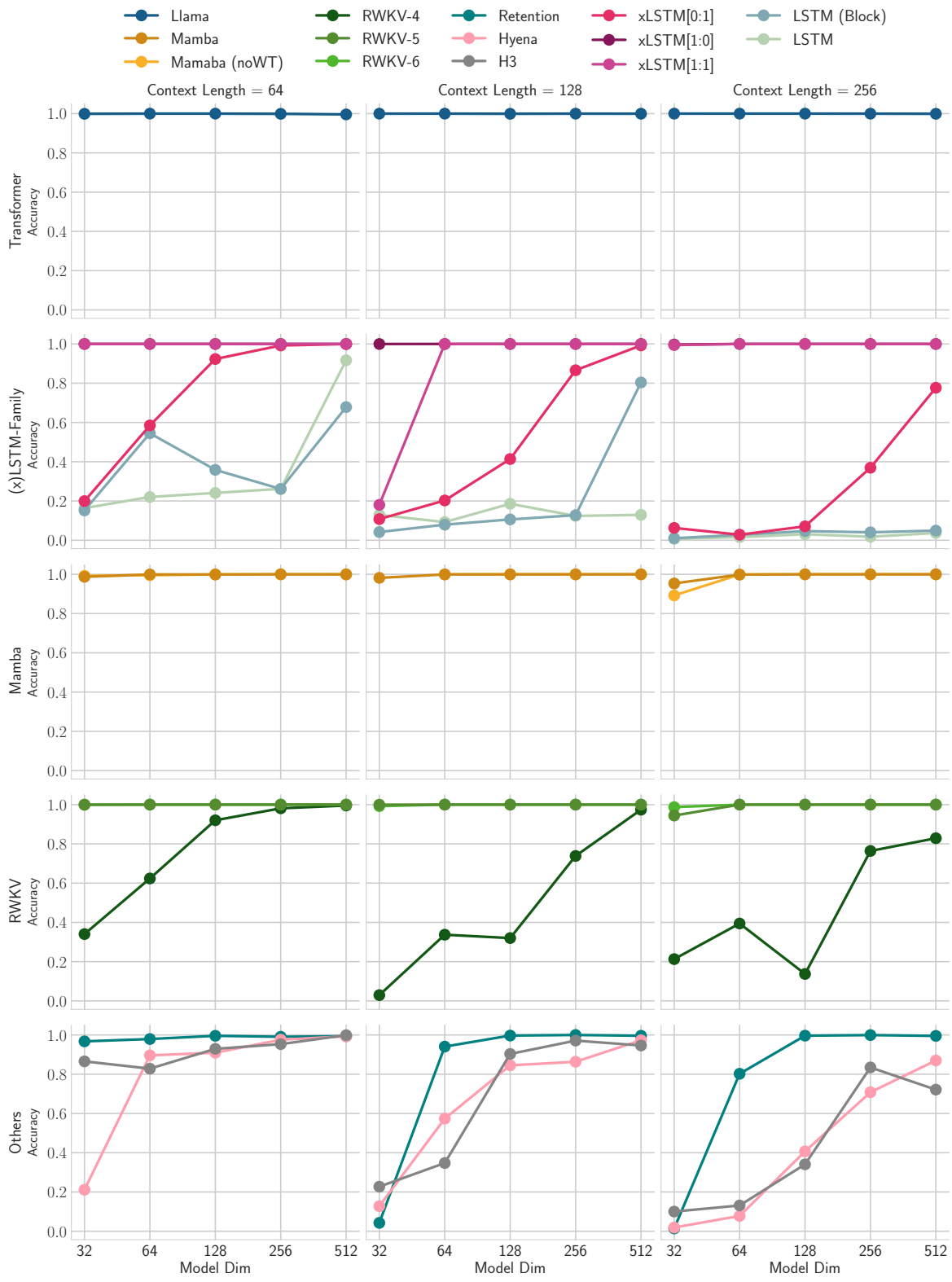

Figure 11: Result of MQAR-Experiment 1. The columns show different task settings (context length and key-value pairs). The rows group related models for better clarity. The x-axis gives the model size and the y-axis the validation accuracy.

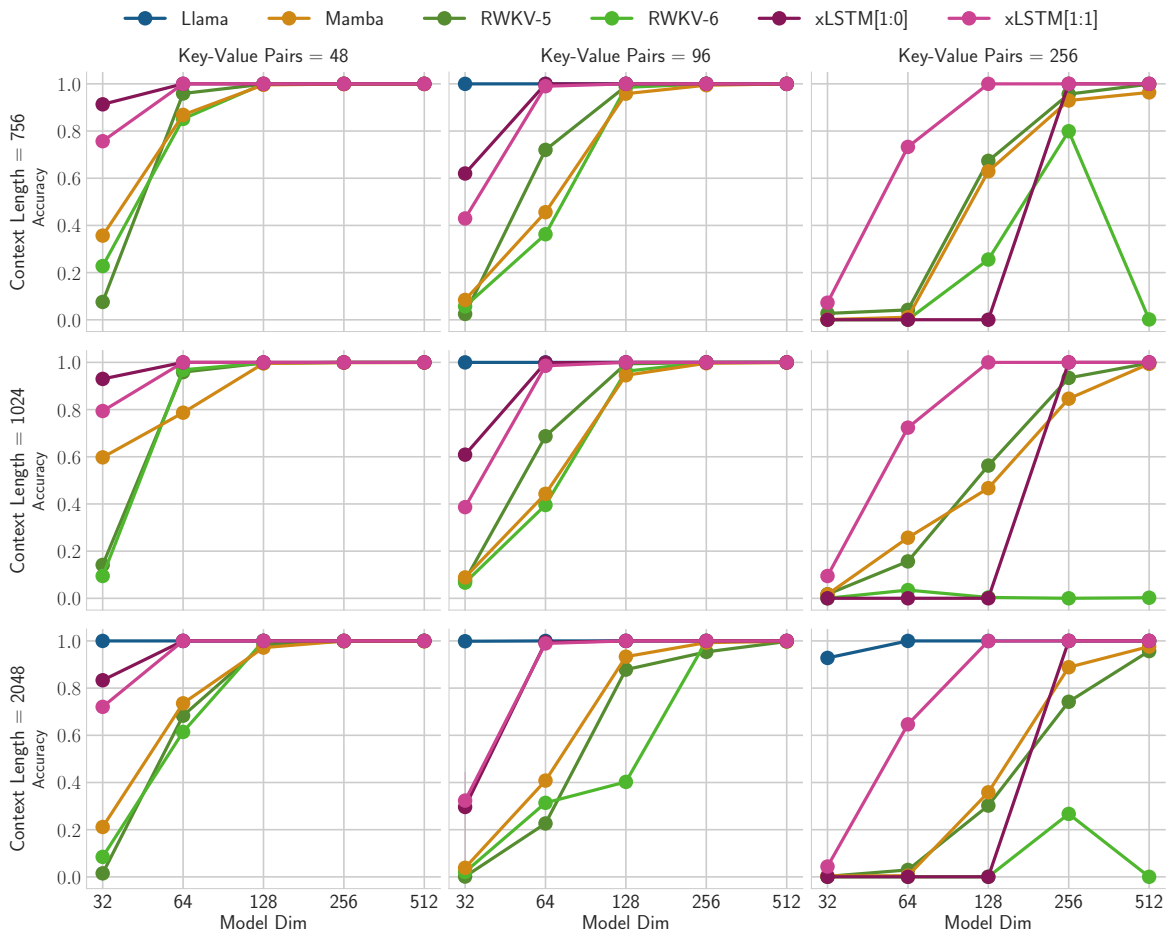

Figure 12: Result of MQAR-Experiment 2. The columns and rows correspond to different numbers of key-value pairs and the context length respectivly. The x-axis gives the model size and the y-axis the validation accuracy.

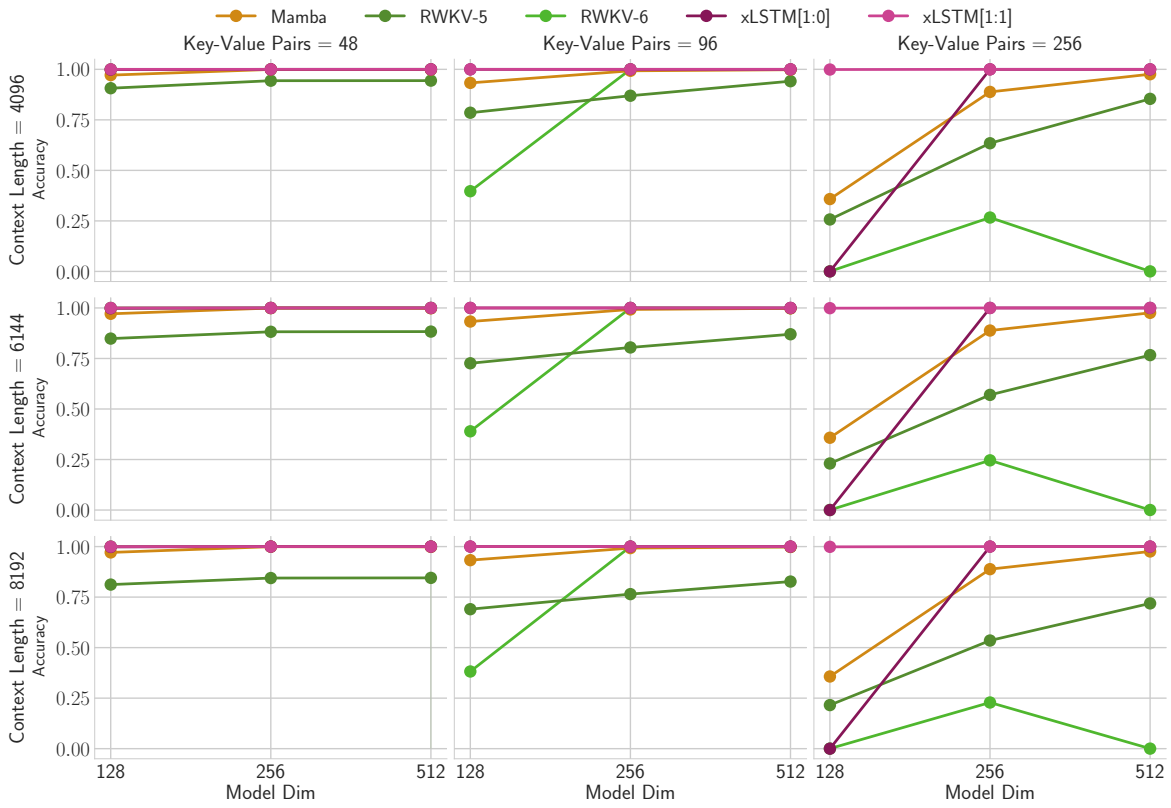

Figure 13: Result of MQAR-Experiment 3 (Extrapolation). All evaluated models were trained on context length 2048 and the number of key-value pairs given by the columns of the plot. The rows show the different context lengths used in the evaluation. The x-axis gives the model size and the y-axis the validation accuracy.

Table 1: Long Range Arena model hyperparameters. These are the model hyperparameters used in each of the Long Range Arena tasks. For each model we used the best learning rate and the better of the two learning rate schedulers.

Task	#Blocks	Embedding Dim	Batch Size	Training Steps
Retrieval	6	128	64	100k
ListOps	8	128	32	80k
Pathfinder	6	192	64	500k
G-Image	6	512	64	180k
RGB-Image	6	512	64	180k

D.1.3. TEST OF XLSTM'S LONG RANGE CAPABILITIES ON THE LONG RANGE ARENA.

We assess the performance of xLSTM across tasks in the Long Range Arena benchmark (Tay et al., 2021), examining its ability to effectively handle longer context lengths and diverse data types.

Our experiments on Long Range Arena benchmark are composed of five tasks:

- **Retrieval**: The task is to predict if two documents have a citation link. The dataset of text documents is derived from the ACL Anthology Network (Radev et al., 2009).
- **ListOps**: This is a set of modular arithmetic tasks including brackets and lists of numbers, using the operations `MIN`, `MAX`, `MEDIAN` and `SUMMOD` (modular sum). A particular example is: `[MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]` $\rightarrow$ `5`
- **Image**: This task is based on a version of the CIFAR dataset (Krizhevsky, 2009), where images are transformed to a sequence of pixels and this sequence has to be classified into the usual CIFAR classes. We test both a gray-scale (G-Image) and RGB (RGB-Image) version of this dataset, as Orvieto et al. (2023) uses colored images contrary to the standard setup.
- **Pathfinder**: The input for this task is a 32x32 gray-scale image, given as pixel sequence, with two dots and several curved lines on it. The task is to predict if the two dots are connected by any of the lines (Linsley et al., 2018).

We omit the **Text** classification task (Maas et al., 2011), as the language modeling experiments already test this kind of data, and the **Pathfinder-X** version of **Pathfinder**.

Experiment Setup. The architectures that are tested in this experiment comprise Llama, Mamba, LSTM, RWKV-4, and xLSTM. LSTM (Block) refers to an architecture where a vanilla LSTM is used inside a post up-projection block (like Transformer with attention replaced by LSTM). For xLSTM we choose the best performing of xLSTM[0:1] or xLSTM[1:0] on the validation set, specifically the former for the Image tasks and the latter for all other ones.

We use the hyperparameter settings of the S5 model (Smith et al., 2022) and Linear Recurrent Unit model (Orvieto et al., 2023), with additional hyperparamter search on learning rates and schedulers for all models. We use two different schedulers: Linear Warm-up Cosine Annealing and Linear Warm-up Cosine Annealing with Restarts. Both learning rate schedulers were evaluated with learning rates of 1e-3, 6e-4 and 1e-4. For the second scheduler, the number of restarts (R) is set to 3. The model hyperparameters for each dataset are displayed in Table 1.

Results. Table 2 shows the result of experiments on the Long Range Arena benchmark. xLSTM demonstrates consistent strong performance on all of the tasks, suggesting that the proposed architecture is remarkably efficient in handling different aspects of long context problems.

Table 2: Long Range Arena test accuracy. Bold highlights the best performing model, underlined the second best. X denotes models that fail to outperform random baselines. xLSTM is the best of xLSTM[1:0], xLSTM[0:1] based on validation dataset accuracy.

	Retrieval acc ↑	ListOps acc ↑	Pathfinder acc ↑	G-Image acc ↑	RGB-Image acc ↑	Ranking acc ↑
Random Baseline	0.500	0.100	0.500	0.100	0.100	
Llama	0.845	0.379	0.887	0.541	0.629	5.2
Mamba	<u>0.902</u>	0.325	**0.992**	0.689	**0.765**	2.2
RWKV-4	0.898	0.389	0.914	<u>0.691</u>	0.757	3.0
LSTM	X	0.275	X	0.675	0.718	5.4
LSTM (Block)	0.880	**0.495**	X	0.690	0.756	3.4
xLSTM	**0.906**	<u>0.411</u>	<u>0.919</u>	**0.695**	<u>0.761</u>	1.6

D.2. Method Comparison and Ablation Study on SlimPajama (15B)

To address the main question of our paper, i.e. what can our new LSTM variants achieve when scaled up in language modelling, we train xLSTMs, Transformers, State Space Models, and other methods on 15B tokens from SlimPajama in the same auto-regressive setting. We compare the trained models on the validation set and perform ablation studies for the xLSTMs.

Comparing xLSTM to Other Methods. We train models on 15B tokens from SlimPajama (Soboleva et al., 2023), and evaluate their perplexity on the validation set. We compare the following methods: xLSTM, GPT-3 (Transformer) (Brown et al., 2020), Llama (Transformer) (Touvron et al., 2023a), H3 (SSM) (Fu et al., 2023), Mamba (SSM) (Gu & Dao, 2023), RWKV-4 (RNN) (Peng et al., 2023), RWKV-5 (RNN) (Peng et al., 2024b), RWKV-6 (RNN) (Peng et al., 2024b), GLA (linear Transformer) (Yang et al., 2023), HGRN2 (RNN) (Qin et al., 2024). RetNet (linear Transformer) (Sun et al., 2023), Hyena (linear Transformer) (Poli et al., 2023), xLSTM[1:0], and xLSTM[7:1]. The models were trained with mixed precision, for RWKV-5, RWKV-6, GLA, HGRN2, the mixed-precision training did not utilize the PyTorch automated mixed precision (see Section D.2).

We categorize the methods into (a) Transformers, (b) State Space Models (SSMs), and (c) Recurrent Neural Networks (RNNs) together with linear Transformers, i.e., linear methods that substitute the Transformer's attention mechanism. The models match a GPT-3 model with 350M parameters in size, i.e. embedding dim 1024 and 24 residual blocks. Only GPT-3 uses shared weights for token and output embeddings, therefore has fewer parameters.

Table 3: Method comparison on next token prediction when trained on 15B tokens from SlimPajama. Best validation perplexities within model classes, i.e., linear Transformers, RNNs, Transformers, SSMs, and xLSTMs are underlined and overall best is in bold. For each model class, the best performing methods are used in Section 3 for LLM training. xLSTMs with new memory (xLSTM[1:0] and xLSTM[7:1]) perform best.

Model	#Params M	SlimPajama (15B) ppl ↓
Hyena	435	17.59
RWKV-4	430	15.62
RWKV-5	456	<u>14.25</u>
RWKV-6	442	15.03
RetNet	431	16.23
GLA	412	16.15
HGRN2	411	14.32

Model	#Params M	SlimPajama (15B) ppl ↓
GPT-3	356	14.26
Llama	407	<u>14.25</u>
H3	420	18.23
Mamba	423	<u>13.70</u>
xLSTM[1:0]	409	**<u>13.43</u>**
xLSTM[7:1]	408	13.48

The results in Table 3 show that xLSTM outperforms all existing methods in validation perplexity, for details see Section D.2. Figure 14 shows the scaling behaviour at 15B training tokens, indicating that xLSTM will also perform favorably for larger models. For the scaling part, we take the best performing methods of this experiment. Note that the 2.7B sized models are strongly under-trained for their size, assuming scaling laws similar to Kaplan et al. (2020).

Ablation Studies. Table 3 and Figure 14 demonstrate that xLSTM achieves excellent results at language modeling when being trained on 15B tokens from SlimPajama. To ablate the changes from LSTM to xLSTM, we morph a vanilla LSTM architecture step-by-step into an xLSTM architecture. Firstly, we integrate LSTM layers into pre-LayerNorm residual backbones. Secondly, we extend this to a post up-projection block. Finally, we add exponential gating and matrix memory. The results are shown in Table 6 (top). The ablation studies attribute the strong performance improvement to both the exponential gating and the matrix memory. Additionally, due to the importance of gating in RNNs and State Space Models, we ablate different gating mechanisms. In Table 6 (bottom), we conclude that having each gate learnable and influenced by the input has an incrementally positive effect. Additional studies on the individual backbone components are discussed in Section D.2.

General Training Procedure. We tokenize our datasets using the HuggingFace GPT-2 tokenizer (Radford et al., 2019; Brown et al., 2020)[5] and use this tokenizer for all models in this paper. In general, we try to follow Brown et al. (2020) for the general training setup, i.e. we choose context length 2048 and batch sizes 256 or 512 for our models. We use the AdamW (Loshchilov & Hutter, 2019) optimizer with beta parameters (β_1, β_2)=(0.9, 0.95) and an epsilon parameter of 1e-5, and gradient clipping at gradient norm 1. As learning rate scheduler we use a linear warm-up with 750 steps and cosine decay to 10% of the peak learning rate. We apply a weight decay of 0.1 to all our models and always exclude the token embedding matrix from weight decay. If not specified otherwise, we do not tie the weights of the token embedding and the language model head. Except for gates we do not use biases in our models, e.g. in other linear layers. We do not apply weight decay to biases and LayerNorm weights. For parallelization, we use PyTorch FSDP in `SHARD_GRAD_OP` mode with mixed precision in `bfloat16`, where applicable. For small models we use `NO_SHARD`. We keep the weights in `float32` and reduce the gradients across GPUs in `float32`. We use `torch.compile` to speed up models where applicable, except for Transformer models as their training curves did not match the non-compiled versions. For xLSTM[7:1], we use positions [3, 5, 7, 40, 42, 44] for sLSTM-based blocks, except for the 125M size, where we use [3, 20] (this is actually a [11:1] ratio). We do not use any positional encoding for our xLSTM models.

Details on Comparison to Other Methods. For the model comparison on 15B training tokens of SlimPajama we train all models with context length 2048 and batch size 256. We use a peak learning rate of 1e-3 for all models for comparability. The learning rate decays over 30k training steps. The models are compared after one epoch at training step 28170. As model implementations we use the original repositories' code for Mamba (Gu & Dao, 2023) [6], RWKV-5, RWKV-6 (Peng et al., 2024b)[7]. For RWKV-4 we use a cleaned and validated re-implementation based on the original repo and kernels (Peng et al., 2023). In our RWKV-4 implementation we enable weight decay on all parameters except biases, the token embedding weight and all LayerNorm weights. For HGRN (Qin et al., 2023), GLA (Yang et al., 2023), HGRN2 (Qin et al., 2024) we use the a re-implementation by the authors of GLA (Yang et al., 2023; Yang & Zhang, 2024)[8]. For GPT-3 and Llama-like Transformers, we use our own implementations based on PyTorch. Note that for all xLSTMs, Transformers, Mamba and RWKV-4, we use Mixed Precision training with `bfloat16` and weights in `float32` precision. Following the general training procedure we use `torch.compile` for all models, except for models using the `flash-linear-attention` (Yang & Zhang, 2024) library because of compilation problems and Transformers as for those training curves deviated.

As RWKV-6 performs worse than RWKV-5, we also train a model with peak learning rate 4e-4, as reported in the original repository for 350M parameter models [9]. This model reaches a perplexity of 16.38, worse than the 15.03 for the standard peak learning rate 1e-3 as reported in Table 3. Similarly, we tested the repository learning rates for other model sizes and all performed worse than the ones we also use for xLSTM (see Table 4).

[5] https://huggingface.co/docs/transformers/en/model_doc/gpt2

[6] https://github.com/state-spaces/mamba

[7] https://github.com/BlinkDL/RWKV-LM/

[8] https://github.com/sustcsonglin/flash-linear-attention

[9] https://github.com/BlinkDL/RWKV-LM/blob/64b7fe4c66fcf7da37019630268075b0558f6dc5/RWKV-v5/train.py#L44

Table 4: Peak learning rates and model dimensions for scaling law plots.

	Model	EmbeddingDim	#Blocks	#Heads/HeadDim	#Params M	Peak LR (15B)	Peak LR (300B)
125M	RWKV-5	768	12	-	176.5	3e-3	-
	RWKV-6	768	12	-	173.6	3e-3	-
	HGRN2	768	12	-	162.2	3e-3	-
	RWKV-4	768	12	-	169.4	3e-3	6e-4
	Llama	768	12	12 / 64	162.2	3e-3	3e-3
	Mamba	768	24	-	167.8	3e-3	3e-3
	xLSTM	768	24	4 / 384	163.8	3e-3	1.5e-3
350M	RKWV-5	1024	24	-	455.7	1e-3	-
	RWKV-6	1024	24	-	441.6	1e-3	-
	HGRN2	1024	24	-	411.4	1e-3	-
	RWKV-4	1024	24	-	430.5	1e-3	4e-4
	Llama	1024	24	16 / 64	406.6	1.5e-3	1.5e-3
	Mamba	1024	48	-	423.1	1.5e-3	1.5e-3
	xLSTM	1024	48	4 / 512	409.3	1e-3	7.5e-4
760M	RWKV-5	1536	24	-	947.8	9e-4	-
	RWKV-6	1536	24	-	907.7	9e-4	-
	HGRN2	1536	24	-	834.2	9e-4	-
	RWKV-4	1536	24	-	891.0	2e-3	2.5e-4
	Llama	1536	24	16 / 96	834.1	1.25e-3	1.25e-3
	Mamba	1536	48	-	870.5	1.25e-3	1.25e-3
	xLSTM	1536	48	4 / 768	840.4	9e-4	6.25e-4
1.3B	RWKV-5	2048	24	-	1616.0	9e-4	-
	RWKV-6	2048	24	-	1537.5	9e-4	-
	HGRN2	2048	24	-	1439.4	9e-4	-
	RWKV-4	2048	24	-	1515.2	1e-3	2e-4
	Llama	2048	24	32 / 64	1420.4	1e-3	1e-3
	Mamba	2048	48	-	1475.3	1e-3	1e-3
	xLSTM	2048	48	4 / 1024	1422.6	9e-4	5e-4
2.7B	RWKV-5	2048	24	-	3194.7	8e-4	-
	RWKV-6	2048	24	-	3021.9	8e-4	-
	HGRN2	2048	24	-	2795.4	8e-4	-
	RWKV-4	2560	32	-	2984.8	8e-4	-
	Llama	2560	32	32 / 80	2779.5	8e-4	-
	Mamba	2560	64	-	2897.2	8e-4	-
	xLSTM	2560	64	4 / 1280	2788.3	8e-4	-

Details on Training Precision for Baselines. For models from `flash-linear-attention` and RWKV-5/6 models we found that PyTorch automatic mixed precision training did not work, but casting the model weights to `float32` initially with FSDP parameter precision `bfloat16` led to a working configuration. In this setting models perform better than in full `bfloat16` training, where the weights are casted to `bfloat16` initially as well. Full `float32` did not work because of the custom kernels.

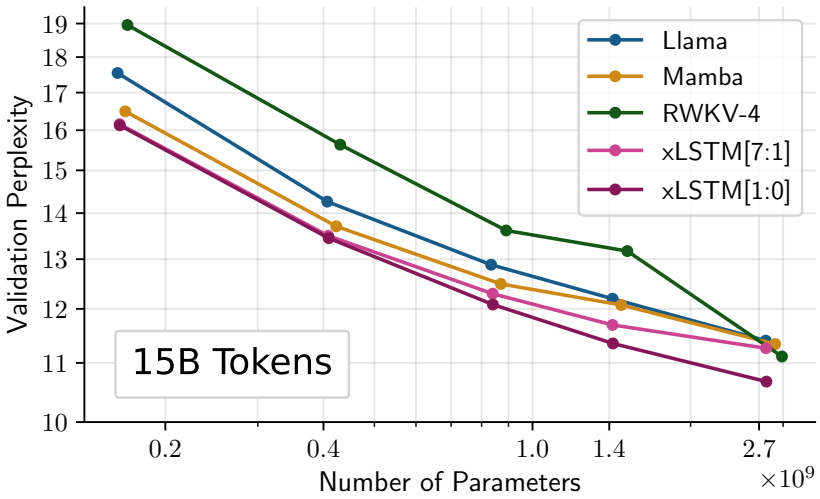

Figure 14: Method comparison on next token prediction when trained on 15B tokens from SlimPajama. Performance measure in validation perplexity for the best methods of each model class (see Table 3) are reported. The performance degradation of xLSTM[7:1] at 2.7B is due to initially slower training convergence that leads to an especially undertrained model. xLSTM is the best method at all sizes.

General Details on Ablation Studies. We follow our general training procedure and train all models with context length 2048, batch size 256 and peak learning rate 1e-3. We report perplexity values on the validation set.

Additional Ablation Study on Matrix Memory. As default block configuration we use the mLSTM in the pre up-projection block (see Figure 6) and the sLSTM in the post up-projection block (see Figure 5). In this experiment we study the combination of mLSTM with different block variants using the xLSTM[1:0] architecture. We compare the mLSTM in a post up-projection block (see Figure 5) with ReLU^2 activation function and non-gated feed-forward network to mLSTM in a pre up-projection block with and without a dimension-wise causal convolution. Table 5 shows that the matrix memory benefits from the pre up-projection block structure, and that the convolution within this block is important.

Table 5: Matrix Memory variants. We study different configurations for the matrix memory. Matrix memory in the pre up-projection block performs best and gives xLSTM[1:0]. Notably, it seems that the dimension-wise causal convolution within the pre up-projection block is important.

Model	Details	#Blocks	Embedding Dim	#Params M	SlimPajama (15B) ppl ↓
xLSTM[1:0]	Post Up-Projection Block (ReLU2)	24	1024	430.4	13.90
	Pre Up-Projection Block, No Convolution	48	1024	408.8	15.41
	Pre Up-Projection Block, With Convolution	48	1024	409.3	**13.43**

Details on new xLSTM Components Ablation Study. In Table 6 (top), we show our modifications to the vanilla LSTM that transform the vanilla LSTM into the xLSTM. We start with a large default PyTorch LSTM with 24 layers and 1536 hidden size. Due to a lack of skip-connections and LayerNorms, vanilla LSTMs of this size are not trainable. We then add skip-connections and pre-LayerNorms before each LSTM layer corresponding to a residual architecture. This enables training for LSTMs at this scale. Replacing every second LSTM layer by a non-gated feed-forward network with GeLU activation function (similar to Vaswani et al.), which corresponds to the post up-projection backbone (see Figure 5), further boosts performance. Adding Exponential Gating to this architecture yields the sLSTM as depicted in Figure 5, with another large performance improvement. Finally, adding the best Matrix Memory variant found in Table 5 by replacing some sLSTM blocks with the mLSTM (see Figure 6) gives xLSTM[7:1] with the best performance.

Details on Gating Technique Ablation Study. In Table 6 (bottom), we investigate the effect of trainable and input-dependent gates for mLSTM. The results show that, in contrast to other methods (Katharopoulos et al., 2020; Sun et al., 2023; Qin et al., 2023; Katsch, 2023; Yang et al., 2023; Qin et al., 2024; Peng et al., 2024b), having the gates both learnable and input dependent gives the best results.

Details on Scaling Experiments. We follow our general training procedure (see paragraph above) and train all models, including the 1.3B and 2.7B model sizes, with context length 2048 and batch size 256. We use the peak learning rates from Table 4. For Llama and Mamba we use the learning rates reported by Gu & Dao (2023).

Table 6: Ablation studies. **Top:** Ablation studies on the new xLSTM components, contributing the strong performance improvement of xLSTM over vanilla LSTM to both the exponential gating and the matrix memory. **Bottom:** Ablation studies on different gating techniques. We consider an xLSTM[1:0] with sigmoid forget gate and exponential input gate. Bias initialization ∞ means that the forget gate is set to one, $[3, 6]$ indicates that values are taken equidistant in the respective interval, and $\mathcal{N}(0, 0.1)$ that values are randomly chosen from a Gaussian with mean 0 and std 0.1. PPL denotes validation perplexity. The first two lines correspond to models similar to linearized attention, line four to Retention, line five to RWKV-5, and line six to RWKV-6. Dependencies of the gates on the input lead to better performance.

Model	Modification	Exponential Gating	Matrix Memory	#Params M	SlimPajama (15B) ppl ↓
LSTM	Vanilla Multi-Layer LSTM	✗	✗	607.8	2417.86
	Adding Resnet Backbone	✗	✗	506.1	35.46
	Adding Up-Projection Backbone	✗	✗	505.9	26.01
xLSTM[0:1]	Adding Exponential Gating	✓	✗	427.3	17.70
xLSTM[7:1]	Adding Matrix Memory	✓	✓	408.4	**13.48**

Ablation studies on different gating techniques.

Learnable Gates	Forget Gate			Input Gate			SlimPajama (15B) ppl ↓
	Input Dependent	Learnable Bias	Bias Init	Input Dependent	Learnable Bias	Bias Init	
No Gates	✗	✗	$+\infty$	✗	✗	0	NaN
No Gates	✗	✗	$[3, 6]$	✗	✗	0	13.95
Forget Gate	✓	✓	$[3, 6]$	✗	✗	0	13.58
Input Gate	✗	✗	$[3, 6]$	✓	✓	$\mathcal{N}(0, 0.1)$	13.69
Forget Gate Bias	✗	✓	$[3, 6]$	✗	✗	0	13.76
Forget + Input Gate Bias	✗	✓	$[3, 6]$	✗	✓	$\mathcal{N}(0, 0.1)$	13.73
Forget Gate + Input Gate Bias	✓	✓	$[3, 6]$	✗	✓	$\mathcal{N}(0, 0.1)$	13.55
Forget Gate + Input Gate	✓	✓	$[3, 6]$	✓	✓	$\mathcal{N}(0, 0.1)$	**13.43**

D.3. xLSTM Large Language Models – SlimPajama300B

General Training Procedure. We use the same general training procedure as in Section D.2 with peak learning rates from Table 4. For Llama and Mamba we use the learning rates reported by Gu & Dao (2023). All models are trained with context length 2048. The 125M, 350M and 760M models are trained with batch size 256 for 600k training steps, whereas the 1.3B models are trained with batch size 512 for 300k training steps. We keep the same learning rate scheduler across all models.

Details on Downstream Evaluation. We use the LM Evaluation Harness from EleutherAI (Sutawika et al., 2023) for evaluating the following tasks that measure common sense reasoning: LAMBADA (OpenAI version in LM Evaluation Harness) (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-challenge, ARC-easy (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2021). This selection of downstream tasks is also used in previous work by Gu & Dao (2023).

Following Gu & Dao (2023), we report accuracy for LAMBADA, WinoGrande, PIQA, and ARC-easy, and accuracy normalized by sequence length for HellaSwag and ARC-challenge. We evaluate all models in full `float32`, full `bfloat16` and `bfloat16` Mixed Precision with weights in `float32`. For each model we select the best value respectively.

Table 7: Validation set perplexity and downstream tasks. Comparison of xLSTM, RWKV-4, Llama, and Mamba on the validation set at next token prediction and on downstream tasks after training on 300B tokens from SlimPajama. Model sizes are 125M, 350M, 760M, and 1.3B. The first column shows the methods and the second the actual number of parameters. The third column lists the validation set perplexities, while the remaining columns show the performance on downstream tasks. Best model per model size is depicted bold and the second best is underlined. In the vast majority of tasks and across all model sizes xLSTM is the best method — only on the ARC task Mamba is in some cases the best method. xLSTM[1:0] and xLSTM[7:1] are the two best models with respect to validation set perplexity.

	Model	#Params M	SlimPajama (300B) ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	ARC-E acc ↑	ARC-C acc ↑	WinoGrande acc ↑	Average acc ↑
125M	RWKV-4	169.4	16.66	54.72	23.77	34.03	66.00	47.94	24.06	50.91	41.12
	Llama	162.2	15.89	39.21	31.54	34.09	65.45	45.33	23.63	50.67	41.78
	Mamba	167.8	15.08	27.76	34.14	36.47	66.76	**48.86**	24.40	51.14	43.63
	xLSTM[1:0]	163.8	14.63	**25.98**	**36.52**	36.74	65.61	47.81	24.83	**51.85**	43.89
	xLSTM[7:1]	163.7	**14.60**	26.59	36.08	**36.75**	**66.87**	48.32	**25.26**	51.70	**44.16**
350M	RWKV-4	430.5	12.62	21.57	36.62	42.47	69.42	54.46	25.43	51.22	46.60
	Llama	406.6	12.19	15.73	44.19	44.45	69.15	52.23	26.28	53.59	48.32
	Mamba	423.1	11.64	12.83	46.24	47.55	69.70	55.47	27.56	54.30	50.14
	xLSTM[1:0]	409.3	**11.31**	**11.49**	**49.33**	**48.06**	69.59	55.72	26.62	**54.38**	50.62
	xLSTM[7:1]	408.4	11.37	12.11	47.74	47.89	**71.16**	**56.61**	27.82	53.28	**50.75**
760M	RWKV-4	891.0	10.55	10.98	47.43	52.29	72.69	58.84	28.84	55.41	52.58
	Llama	834.1	10.60	9.90	51.41	52.16	70.95	56.48	28.75	56.67	52.74
	Mamba	870.5	10.24	9.24	50.84	53.97	71.16	60.44	29.78	56.99	53.86
	xLSTM[1:0]	840.4	**9.86**	8.09	54.78	55.72	72.69	**62.75**	**32.59**	**58.17**	**56.12**
	xLSTM[7:1]	839.7	9.91	**8.07**	**55.27**	**56.12**	**72.74**	61.36	29.61	56.43	55.26
1.3B	RWKV-4	1515.2	9.83	9.84	49.78	56.20	74.70	61.83	30.63	55.56	54.78
	Llama	1420.4	9.44	7.23	57.44	57.81	73.12	62.79	31.74	59.04	56.99
	Mamba	1475.3	9.14	7.41	55.64	60.45	74.43	**66.12**	**33.70**	60.14	58.41
	xLSTM[1:0]	1422.6	**8.89**	**6.86**	**57.83**	**60.91**	74.59	64.31	32.59	**60.62**	**58.48**
	xLSTM[7:1]	1420.1	9.00	7.04	56.69	60.26	**74.92**	65.11	32.34	59.27	58.10

Details on PALOMA. We use 16 out of the 18 data sources of the PALOMA dataset (Magnusson et al., 2023). We use C4 (Raffel et al., 2019), MC4-EN (Xue et al., 2021), Wikitext-103 (Merity et al., 2017), PennTreebank (Vadas & Curran, 2011), RedPajama (TogetherComputer, 2023), Falcon Refinedweb (Refined Web) (Penedo et al., 2023), Dolma v1.5 (Soldaini et al., 2023), M2D2 S2ORC, M2D2 Wikipedia (Reid et al., 2022), C4-100-Domains (C4 Domains) (Chronopoulou et al., 2022), Dolma-100-Subreddits (Dolma Subreddits) (Soldaini et al., 2023), Dolma-100-Programming Languages (Dolma Coding) (Soldaini et al., 2023; Kocetkov et al., 2022), TwitterAAE (Blodgett et al., 2016; Liang et al., 2023), Manosphere Corpus (Ribeiro et al., 2021), GAB Corpus (Zannettou et al., 2018), 4CHAN Corpus (Papasavva et al., 2020). We leave out ThePile (Gao et al., 2021) and ICE (Greenbaum & Nelson, 1996) as they are not part of Paloma's Huggingface dataset

repository[10]. A detailed description of these datasets can be found in Magnusson et al. (2023, Table 2). All models are evaluated in `bfloat16` Mixed Precision. Results are shown in Table 8.

Results on the data sources TwitterAAE, Manosphere, GAB and 4CHAN are reported in Table 9 and for each individual dataset the results are given in Section E.

In order to evaluate the perplexity values on each data source, we split the text documents into sequences of length 2048, which corresponds to the pre-training context length of all models. For documents longer than 2048 tokens we split each document into non-overlapping input sequences. In this case for the last input sequence, we follow the LM Evaluation Harness and fill up the full 2048 token context window with previous tokens, but compute the perplexity only on the remaining tokens.

We compute the token perplexities per data source in Table 8 as the exponential of the negative loglikelihoods per domain weighted by the number of tokens per domain in that data source as it is defined in Magnusson et al. (2023, Equation 1)

Table 8: Performance on PALOMA Language Modeling Tasks. Comparison of xLSTM, RWKV-4, Llama, and Mamba by the perplexity of next token prediction on the PALOMA language benchmark after training on 300B tokens from SlimPajama. Model sizes are 125M, 250M, 760M, and 1.3B. The second column shows the actual number of parameters. The 571 text domains are grouped into language modeling (next seven columns) and fine-grained domain benchmarks (further 5 columns). The last column shows the average perplexity across all of these tasks. Best model per model size is given in bold and the second best is underlined. xLSTM yields the best performance.

	Model	#Params M	C4	MC4 EN	Wikitext 103	Penn Treebank	Red Pajama	Refined Web	Dolma	M2D2 S2ORC	M2D2 Wikipedia	C4 Domains	Dolma Subreddits	Dolma Coding	Average
125M	RWKV-4	169.4	26.25	22.33	29.18	38.45	8.99	32.47	17.04	23.86	21.42	22.68	37.08	5.12	23.74
	Llama	162.2	24.64	17.23	23.16	31.56	8.26	29.15	15.10	19.71	20.41	21.45	36.73	3.61	20.92
	Mamba	167.8	23.12	17.04	22.49	30.63	7.96	27.73	14.60	19.38	19.36	20.14	34.32	3.77	20.05
	xLSTM[1:0]	163.8	22.54	16.32	21.98	30.47	7.80	27.21	14.35	19.02	19.04	19.65	34.15	3.64	19.68
	xLSTM[7:1]	163.7	**22.39**	**16.13**	**21.47**	**30.01**	**7.75**	**26.91**	**14.13**	**18.6**	**18.84**	**19.52**	**33.9**	**3.59**	**19.44**
350M	RWKV-4	430.5	19.55	15.82	19.64	27.58	6.97	24.28	12.94	17.59	15.96	16.98	29.40	3.90	17.55
	Llama	406.6	18.38	13.28	16.41	**21.82**	6.56	22.09	11.76	15.05	15.25	15.99	28.30	3.12	15.67
	Mamba	423.1	17.33	13.05	16.11	22.24	6.34	21.04	11.42	14.83	14.53	15.16	27.02	3.20	15.19
	xLSTM[1:0]	409.3	17.01	**12.55**	**15.17**	22.51	**6.20**	**20.66**	**11.16**	**14.44**	**14.27**	**14.85**	26.70	**3.08**	**14.88**
	xLSTM[7:1]	408.4	**16.98**	12.68	15.43	21.86	6.23	20.70	11.22	14.62	14.30	**14.85**	**26.61**	3.11	**14.88**
760M	RWKV-4	891.0	15.51	12.76	14.84	21.39	5.91	19.28	10.70	14.27	13.04	13.68	24.22	3.32	14.08
	Llama	834.1	15.75	11.59	13.47	18.33	5.82	19.04	10.33	13.00	13.05	13.76	24.80	2.90	13.49
	Mamba	870.5	15.08	11.54	13.47	19.34	5.69	18.43	10.15	13.05	12.62	13.25	23.94	2.99	13.30
	xLSTM[1:0]	840.4	**14.60**	**11.03**	**12.61**	17.74	**5.52**	**17.87**	**9.85**	**12.50**	**12.20**	**12.81**	23.46	**2.87**	**12.76**
	xLSTM[7:1]	839.7	14.72	11.11	12.68	**17.61**	5.55	18.01	9.87	12.59	12.25	12.89	**23.43**	2.88	12.80
1.3B	RWKV-4	1515.2	14.51	12.04	13.73	19.37	5.62	18.25	10.11	13.46	12.10	12.87	22.85	3.25	13.18
	Llama	1420.4	13.93	10.44	11.74	**15.92**	5.29	17.03	9.35	11.61	11.53	12.24	22.63	2.74	12.04
	Mamba	1475.3	13.35	10.40	11.76	16.65	5.21	16.50	9.17	11.73	11.18	11.83	21.43	2.83	11.84
	xLSTM[1:0]	1422.6	**13.13**	**10.09**	11.41	**15.92**	**5.10**	**16.25**	**9.01**	**11.43**	**10.95**	**11.60**	**21.29**	**2.73**	**11.58**
	xLSTM[7:1]	1420.1	13.31	10.21	**11.32**	16.00	5.16	16.48	9.11	11.61	11.10	11.76	21.50	2.75	11.69

[10] https://huggingface.co/datasets/allenai/paloma

Table 9: Perplexity values per domain.

	Model	#Params M	Twitter AAE	Manosphere	4CHAN	GAB
125M	RWKV-4	169.4	265.80	39.31	18.48	53.89
	Llama	162.2	277.93	32.98	14.03	56.45
	Mamba	167.8	258.17	32.14	14.01	51.58
	xLSTM[1:0]	163.8	244.53	31.45	13.27	51.00
	xLSTM[7:1]	163.7	248.51	30.90	13.45	50.25
350M	RWKV-4	430.5	216.17	30.25	13.82	42.25
	Llama	406.6	231.09	25.90	11.49	43.04
	Mamba	423.1	202.88	25.24	11.60	40.78
	xLSTM[1:0]	409.3	200.61	24.58	11.20	39.83
	xLSTM[7:1]	408.4	206.25	24.73	11.31	39.86
760M	RWKV-4	891.0	195.27	24.66	12.00	35.73
	Llama	834.1	205.50	22.69	10.40	37.68
	Mamba	793.2	182.74	22.58	10.47	36.25
	xLSTM[1:0]	840.4	179.74	21.66	10.11	35.33
	xLSTM[7:1]	839.7	180.19	21.78	10.22	34.89
1.3B	RWKV-4	1515.2	174.87	23.51	11.34	33.18
	Llama	1420.4	192.52	20.67	9.67	34.84
	Mamba	1475.3	171.38	20.37	9.80	32.01
	xLSTM[1:0]	1422.6	166.16	19.94	9.64	31.90
	xLSTM[7:1]	1420.1	171.36	20.28	9.64	32.17

E. Detailed Results on PALOMA Language Model Evaluation

We report the perplexity values on each of the 571 subdomains of PALOMA in Table 10. Note that the aggregated perplexity values in Table 8 are not macro averages of the values shown in Table 10.

Table 10: PPL Evaluations: For the 1.3B sized models trained on 300B SlimPajama tokens, these are the detailed evaluation results on the respective validation datasets.

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
#Params (M)	1420	1475	1515	1420	1423
4chan_meta_sep_val-00000000	9.58	9.72	11.37	9.53	9.55
4chan_meta_sep_val-00000001	9.95	10.06	11.57	9.91	9.88
4chan_meta_sep_val-00000002	9.42	9.53	11.00	9.40	9.38
4chan_meta_sep_val-00000003	9.78	9.93	11.48	9.77	9.77
c4_100dom_val_100_www.ign.com	16.22	15.75	17.10	15.67	15.43
c4_100dom_val_10_www.eventbrite.com	12.72	12.33	13.33	12.30	12.12
c4_100dom_val_11_link.springer.com	8.66	8.54	9.31	8.42	8.33
c4_100dom_val_12_www.chicagotribune.com	12.09	11.60	12.49	11.55	11.37
c4_100dom_val_13_www.foxnews.com	9.59	9.21	9.83	9.16	9.08
c4_100dom_val_14_www.aljazeera.com	10.97	10.61	11.31	10.50	10.40
c4_100dom_val_15_www.dailymail.co.uk	12.42	11.97	12.87	11.85	11.69
c4_100dom_val_16_www.ncbi.nlm.nih.gov	7.39	7.31	7.98	7.11	7.07
c4_100dom_val_17_www.express.co.uk	11.57	11.04	11.84	10.99	10.79
c4_100dom_val_18_en.m.wikipedia.org	9.28	8.95	9.52	8.89	8.80
c4_100dom_val_19_www.cnet.com	12.61	12.23	13.12	12.09	11.97
c4_100dom_val_1_www.nytimes.com	13.13	12.66	14.04	12.68	12.44
c4_100dom_val_20_www.telegraph.co.uk	13.71	13.10	14.28	13.06	12.88
c4_100dom_val_21_www.theatlantic.com	14.70	14.17	15.54	14.17	13.97
c4_100dom_val_22_forums.macrumors.com	17.77	17.34	19.15	17.22	16.95
c4_100dom_val_23_www.oreilly.com	13.36	12.99	14.31	13.02	12.88
c4_100dom_val_24_www.washingtonpost.com	12.06	11.58	12.98	11.64	11.41
c4_100dom_val_25_www.zdnet.com	13.22	12.86	13.80	12.78	12.61
c4_100dom_val_26_www.foxbusiness.com	9.32	9.03	9.58	8.92	8.81
c4_100dom_val_27_www.reuters.com	10.67	10.13	11.16	10.13	9.97
c4_100dom_val_28_www.ibtimes.co.uk	11.36	11.01	11.71	10.89	10.76
c4_100dom_val_29_www.rt.com	13.59	12.96	14.24	12.98	12.74
c4_100dom_val_2_en.wikipedia.org	10.75	10.45	11.32	10.32	10.19
c4_100dom_val_30_www.prweb.com	11.18	10.88	11.92	10.83	10.65
c4_100dom_val_31_www.deviantart.com	21.78	21.05	22.78	21.00	20.69
c4_100dom_val_32_www.si.com	11.49	11.00	11.92	10.90	10.76
c4_100dom_val_33_www.bbc.com	9.35	8.91	9.41	8.80	8.70
c4_100dom_val_34_github.com	11.57	11.49	12.94	11.40	11.28
c4_100dom_val_35_nypost.com	14.31	13.41	15.29	13.62	13.31
c4_100dom_val_36_itunes.apple.com	16.49	15.88	17.15	15.98	15.69
c4_100dom_val_37_www.instructables.com	16.75	16.33	17.73	16.28	15.97
c4_100dom_val_38_www.youtube.com	8.42	8.24	8.83	8.22	8.07
c4_100dom_val_39_www.booking.com	8.84	8.49	8.83	8.41	8.32
c4_100dom_val_40_www.etsy.com	11.93	11.66	12.66	11.52	11.43
c4_100dom_val_41_www.marketwired.com	7.66	7.47	7.88	7.33	7.27
c4_100dom_val_42_sites.google.com	14.23	13.81	14.91	13.68	13.51
c4_100dom_val_43_www.baltimoresun.com	11.57	11.16	11.96	11.09	10.95
c4_100dom_val_44_www.agreatertown.com	13.56	12.94	13.57	12.77	12.64
c4_100dom_val_45_www.npr.org	10.59	10.30	11.14	10.19	10.12
c4_100dom_val_46_www.fool.com	11.03	10.63	11.35	10.56	10.42

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
c4_100dom_val_47_www.tripadvisor.com	15.80	15.26	16.26	15.10	14.93
c4_100dom_val_48_www.bbc.co.uk	12.55	12.10	13.02	12.00	11.85
c4_100dom_val_49_lists.w3.org	18.75	18.24	19.89	18.05	17.84
c4_100dom_val_4_www.latimes.com	11.88	11.46	12.40	11.39	11.24
c4_100dom_val_50_mashable.com	12.44	11.95	12.85	11.90	11.76
c4_100dom_val_51_disneyparksmomspanel.dis	11.99	11.29	11.98	11.16	11.00
c4_100dom_val_52_www.cnbc.com	10.65	10.32	10.99	10.24	10.10
c4_100dom_val_53_answers.sap.com	23.59	23.09	25.71	22.99	22.55
c4_100dom_val_54_homestars.com	14.13	13.70	14.51	13.65	13.52
c4_100dom_val_55_www.hindustantimes.com	12.13	11.60	12.74	11.60	11.37
c4_100dom_val_56_www.reference.com	11.57	11.04	11.75	10.92	10.79
c4_100dom_val_57_www.city-data.com	18.38	17.94	19.61	17.73	17.62
c4_100dom_val_58_medium.com	15.50	15.09	16.58	15.18	15.01
c4_100dom_val_59_app-wiringdiagram...	9.74	9.10	9.68	8.88	8.75
c4_100dom_val_5_www.theguardian.com	14.78	14.09	15.47	14.08	13.86
c4_100dom_val_60_www.csmonitor.com	15.35	14.85	15.92	14.75	14.57
c4_100dom_val_61_www.adweek.com	14.55	13.95	15.58	14.09	13.81
c4_100dom_val_62_docs.microsoft.com	7.69	7.79	8.86	7.68	7.58
c4_100dom_val_63_www.yahoo.com	9.29	8.88	9.71	8.89	8.77
c4_100dom_val_64_www.thesun.co.uk	12.18	11.66	12.74	11.59	11.39
c4_100dom_val_65_www.nydailynews.com	12.15	11.60	12.61	11.56	11.36
c4_100dom_val_66_www.dailystar.co.uk	10.65	10.17	11.03	10.09	9.92
c4_100dom_val_67_fineartamerica.com	12.06	11.58	12.29	11.46	11.36
c4_100dom_val_68_www.kickstarter.com	13.85	13.58	15.38	13.55	13.38
c4_100dom_val_69_uk.reuters.com	9.54	9.13	9.90	9.07	8.92
c4_100dom_val_6_www.huffpost.com	13.45	13.03	13.96	12.99	12.83
c4_100dom_val_70_www.insiderpages.com	13.24	12.84	13.55	12.77	12.64
c4_100dom_val_71_www.inquisitr.com	12.12	11.58	12.86	11.71	11.38
c4_100dom_val_72_lists.debian.org	18.18	17.81	19.62	17.67	17.30
c4_100dom_val_73_www.straitstimes.com	11.51	11.06	11.91	10.94	10.79
c4_100dom_val_74_www.cbsnews.com	10.29	9.91	10.60	9.82	9.72
c4_100dom_val_75_simple.wikipedia.org	8.25	7.85	8.37	7.78	7.67
c4_100dom_val_76_deadline.com	14.75	13.83	15.48	13.92	13.51
c4_100dom_val_77_www.androidheadlines.com	11.11	10.74	11.43	10.72	10.59
c4_100dom_val_78_www.wired.com	14.42	13.88	15.14	13.87	13.68
c4_100dom_val_79_www.bustle.com	12.79	12.33	13.19	12.25	12.09
c4_100dom_val_7_patents.google.com	7.59	7.84	9.33	7.72	7.59
c4_100dom_val_80_premium.wpmudev.org	16.86	16.63	18.13	16.50	16.29
c4_100dom_val_81_www.librarything.com	14.36	13.98	15.42	13.91	13.75
c4_100dom_val_82_mail-archives.apache.org	5.67	5.61	6.17	5.56	5.49
c4_100dom_val_83_scholars.duke.edu	8.72	8.43	9.03	8.32	8.21
c4_100dom_val_84_www.glassdoor.com	16.64	15.97	16.99	16.00	15.83
c4_100dom_val_85_www.pcworld.com	12.34	11.95	12.95	11.90	11.72
c4_100dom_val_86_www.shutterstock.com	8.70	8.89	10.75	8.62	8.52
c4_100dom_val_87_myemail.constantcontact.co	14.59	14.24	15.32	14.18	13.98
c4_100dom_val_88_www.eventbrite.co.uk	14.47	13.99	14.89	13.98	13.79
c4_100dom_val_89_www.fastcompany.com	14.24	13.75	15.52	13.82	13.56
c4_100dom_val_8_www.businessinsider.com	10.97	10.69	11.35	10.52	10.46
c4_100dom_val_90_www.firstpost.com	11.71	11.24	12.08	11.12	10.96
c4_100dom_val_91_www.entrepreneur.com	13.10	12.68	13.65	12.72	12.54
c4_100dom_val_92_www.breitbart.com	13.47	12.67	14.29	12.84	12.56
c4_100dom_val_93_techcrunch.com	14.20	13.68	15.18	13.82	13.58

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
c4_100dom_val_94_www.nme.com	14.12	13.28	15.06	13.43	13.12
c4_100dom_val_95_www.ndtv.com	10.66	10.26	10.90	10.10	10.00
c4_100dom_val_96_finance.yahoo.com	9.96	9.55	10.22	9.43	9.34
c4_100dom_val_97_archives.lib.state.ma.us	6.53	6.12	7.09	6.27	5.85
c4_100dom_val_98_www.gsmarena.com	23.21	22.15	24.52	22.10	21.76
c4_100dom_val_99_www.lonelyplanet.com	11.33	10.92	12.28	10.84	10.69
c4_100dom_val_9_www.forbes.com	13.72	13.31	14.63	13.34	13.13
c4_en_val-00000000	14.34	13.70	14.87	13.67	13.46
c4_en_val-00000001	14.86	14.28	15.51	14.21	14.09
c4_en_val-00000002	15.29	14.71	15.95	14.71	14.51
c4_en_val-00000003	12.95	12.28	13.32	12.23	12.06
c4_en_val-00000004	12.56	12.13	13.27	12.05	11.87
c4_en_val-00000005	12.77	12.35	13.26	12.32	12.18
dolma-v1_5_val_books	13.00	12.44	13.64	12.44	12.27
dolma-v1_5_val_common-crawl	16.86	16.37	18.00	16.35	16.10
dolma-v1_5_val_pes2o	9.42	9.56	11.25	9.41	9.29
dolma-v1_5_val_reddit_uniform	23.04	21.97	23.84	22.05	21.80
dolma-v1_5_val_stack_uniform	2.30	2.33	2.53	2.30	2.29
dolma-v1_5_val_wiki	10.86	10.48	11.25	10.41	10.31
dolma_100_proglang_val_00_text	5.61	6.30	6.94	5.67	5.69
dolma_100_proglang_val_01_markdown	3.16	3.16	3.56	3.15	3.11
dolma_100_proglang_val_02_c	1.84	1.91	2.23	1.86	1.85
dolma_100_proglang_val_03_php	1.75	1.75	1.83	1.73	1.72
dolma_100_proglang_val_04_java	1.96	1.99	2.18	1.95	1.95
dolma_100_proglang_val_05_c++	2.19	2.25	2.53	2.21	2.19
dolma_100_proglang_val_06_python	2.35	2.39	2.62	2.36	2.34
dolma_100_proglang_val_07_javascript	2.54	2.59	2.83	2.53	2.53
dolma_100_proglang_val_08_html	1.92	1.94	2.13	1.91	1.91
dolma_100_proglang_val_09_c#	2.23	2.28	2.45	2.19	2.24
dolma_100_proglang_val_10_yaml	2.93	3.01	3.71	2.94	2.92
dolma_100_proglang_val_11_go	1.75	1.78	1.97	1.77	1.75
dolma_100_proglang_val_12_typescript	2.17	2.20	2.41	2.18	2.16
dolma_100_proglang_val_13_xml	2.44	2.50	2.78	2.46	2.48
dolma_100_proglang_val_14_css	2.25	2.25	2.34	2.21	2.20
dolma_100_proglang_val_15_jupyter-nb	1.57	1.60	1.75	1.58	1.58
dolma_100_proglang_val_16_rust	1.96	2.01	2.23	1.97	1.96
dolma_100_proglang_val_17_unity3d-asset	4.01	4.17	4.56	4.10	4.05
dolma_100_proglang_val_18_gettext-catalog	2.84	2.87	3.53	2.86	2.83
dolma_100_proglang_val_19_ruby	2.41	2.44	2.70	2.39	2.38
dolma_100_proglang_val_20_vue	1.95	1.95	2.10	1.94	1.93
dolma_100_proglang_val_21_sql	2.18	2.23	2.46	2.17	2.16
dolma_100_proglang_val_22_swift	1.86	1.88	2.04	1.86	1.84
dolma_100_proglang_val_23_kotlin	2.05	2.07	2.29	2.07	2.04
dolma_100_proglang_val_24_scala	2.24	2.28	2.64	2.25	2.23
dolma_100_proglang_val_25_scss	2.26	2.27	2.38	2.24	2.24
dolma_100_proglang_val_26_tex	4.04	4.21	4.97	4.10	4.04
dolma_100_proglang_val_27_dart	1.79	1.82	2.01	1.80	1.78
dolma_100_proglang_val_28_kicad	2.57	2.79	3.86	2.68	2.67
dolma_100_proglang_val_29_shell	3.71	3.74	4.31	3.69	3.63
dolma_100_proglang_val_30_smali	1.38	1.39	1.45	1.38	1.37
dolma_100_proglang_val_31_lua	5.65	6.01	7.18	5.33	5.45
dolma_100_proglang_val_32_restructuredtext	4.01	4.05	4.66	3.97	3.92

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
dolma_100_proglang_val_33_perl	2.57	2.62	3.01	2.59	2.55
dolma_100_proglang_val_34_diff	2.87	2.95	3.43	2.89	2.86
dolma_100_proglang_val_35_ini	3.91	4.16	4.90	4.05	3.98
dolma_100_proglang_val_36_jsx	1.83	1.84	1.95	1.83	1.82
dolma_100_proglang_val_37_haskell	2.94	3.07	3.73	3.02	2.95
dolma_100_proglang_val_38_gnuplot	2.65	2.88	3.36	2.81	2.77
dolma_100_proglang_val_39_postscript	19.09	19.52	19.56	18.66	18.64
dolma_100_proglang_val_40_groff	6.13	6.32	7.45	6.22	6.21
dolma_100_proglang_val_41_turtle	2.35	2.45	3.17	2.39	2.35
dolma_100_proglang_val_42_fortran	2.32	2.39	2.83	2.35	2.31
dolma_100_proglang_val_43_makefile	2.93	3.01	3.51	2.86	2.82
dolma_100_proglang_val_44_mathematica	10.34	11.34	13.24	10.49	10.71
dolma_100_proglang_val_45_pascal	4.18	4.81	5.49	4.17	4.27
dolma_100_proglang_val_46_common-lisp	2.56	2.71	3.32	2.62	2.58
dolma_100_proglang_val_47_gas	2.49	2.73	3.59	2.57	2.53
dolma_100_proglang_val_48_vhdl	3.91	4.06	4.69	3.92	3.90
dolma_100_proglang_val_49_julia	3.25	3.36	4.05	3.30	3.26
dolma_100_proglang_val_50_edn	1.99	2.10	2.67	2.04	2.03
dolma_100_proglang_val_51_visual-basic	2.42	2.49	2.72	2.37	2.38
dolma_100_proglang_val_52_powershell	4.08	4.16	4.50	3.86	3.89
dolma_100_proglang_val_53_g-code	2.26	2.66	3.29	2.44	2.37
dolma_100_proglang_val_54_ocaml	3.06	3.29	4.22	3.19	3.13
dolma_100_proglang_val_55_java-server-p	2.10	2.11	2.31	2.06	2.09
dolma_100_proglang_val_56_solidity	4.09	4.41	5.28	4.05	4.10
dolma_100_proglang_val_57_graphviz-dot	2.17	2.48	3.54	2.32	2.29
dolma_100_proglang_val_58_less	2.24	2.26	2.33	2.22	2.22
dolma_100_proglang_val_59_twig	1.81	1.81	1.91	1.80	1.79
dolma_100_proglang_val_60_asciidoc	5.33	5.50	6.84	5.43	5.34
dolma_100_proglang_val_61_groovy	2.12	2.15	2.41	2.13	2.11
dolma_100_proglang_val_62_llvm	2.26	2.40	3.25	2.31	2.23
dolma_100_proglang_val_63_hcl	2.52	2.56	2.96	2.52	2.48
dolma_100_proglang_val_64_html+erb	2.10	2.09	2.23	2.08	2.07
dolma_100_proglang_val_65_erlang	2.84	2.98	3.87	2.88	2.85
dolma_100_proglang_val_66_elixir	2.93	2.99	3.58	2.91	2.90
dolma_100_proglang_val_67_eagle	5.35	6.90	10.75	5.64	5.76
dolma_100_proglang_val_68_arduino	3.37	3.40	3.81	3.28	3.28
dolma_100_proglang_val_69_coffeescript	2.80	2.85	3.27	2.80	2.77
dolma_100_proglang_val_70_toml	7.76	7.62	8.44	7.53	7.58
dolma_100_proglang_val_71_cuda	2.15	2.21	2.56	2.19	2.16
dolma_100_proglang_val_72_nix	7.80	7.84	9.03	7.88	7.83
dolma_100_proglang_val_73_smalltalk	9.32	9.61	12.60	9.47	9.20
dolma_100_proglang_val_74_cmake	1.87	1.86	2.02	1.84	1.81
dolma_100_proglang_val_75_actionscript	2.45	2.54	2.88	2.46	2.46
dolma_100_proglang_val_76_glsl	2.40	2.42	2.72	2.36	2.32
dolma_100_proglang_val_77_systemverilog	2.53	2.66	3.17	2.58	2.55
dolma_100_proglang_val_78_haxe	2.74	2.81	3.20	2.77	2.76
dolma_100_proglang_val_79_f#	2.89	3.02	3.53	2.93	2.88
dolma_100_proglang_val_80_max	1.59	1.62	1.80	1.61	1.61
dolma_100_proglang_val_81_objective-c++	2.18	2.19	2.40	2.17	2.16
dolma_100_proglang_val_82_standard-ml	3.57	4.05	4.79	3.81	3.77
dolma_100_proglang_val_83_dockerfile	4.08	4.17	4.37	4.01	4.05
dolma_100_proglang_val_84_emacs-lisp	3.83	3.83	4.44	3.80	3.72

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
dolma_100_proglang_val_85_scheme	2.78	2.86	3.40	2.84	2.77
dolma_100_proglang_val_86_clojure	3.18	3.30	4.00	3.26	3.17
dolma_100_proglang_val_87_handlebars	1.79	1.79	1.88	1.78	1.78
dolma_100_proglang_val_88_smarty	2.30	2.35	2.58	2.29	2.30
dolma_100_proglang_val_89_logos	2.37	2.58	2.98	2.46	2.44
dolma_100_proglang_val_90_stata	4.67	5.08	6.85	4.85	4.81
dolma_100_proglang_val_91_yacc	2.42	2.48	2.87	2.44	2.43
dolma_100_proglang_val_92_nimrod	2.75	2.87	3.63	2.81	2.77
dolma_100_proglang_val_93_tcl	3.00	3.16	3.95	3.07	3.02
dolma_100_proglang_val_94_viml	5.56	5.76	7.21	5.59	5.55
dolma_100_proglang_val_95_asp	1.79	1.79	1.90	1.77	1.77
dolma_100_proglang_val_96_protocol-buffer	1.32	1.31	1.38	1.31	1.32
dolma_100_proglang_val_97_r	2.80	2.92	3.66	2.86	2.81
dolma_100_proglang_val_98_cython	2.34	2.39	2.69	2.36	2.35
dolma_100_proglang_val_99_mediawiki	2.01	2.10	2.48	2.12	2.04
dolma_100_subreddits_val_00_AskReddit	20.25	19.29	20.38	19.28	19.14
dolma_100_subreddits_val_01_politics	22.08	20.70	22.07	20.83	20.61
dolma_100_subreddits_val_02_AmItheAsshole	22.49	21.30	22.89	21.60	21.27
dolma_100_subreddits_val_03_worldnews	22.57	21.43	22.77	21.50	21.23
dolma_100_subreddits_val_04_relationships	18.64	17.80	18.89	17.86	17.67
dolma_100_subreddits_val_05_relationship_adv	19.40	18.53	19.68	18.63	18.46
dolma_100_subreddits_val_06_news	22.49	21.25	22.51	21.49	21.17
dolma_100_subreddits_val_07_leagueoflegends	34.45	32.41	35.13	32.46	32.04
dolma_100_subreddits_val_08_todayilearned	22.53	21.30	22.68	21.28	21.10
dolma_100_subreddits_val_09_TwoXChromoso	20.20	19.16	20.25	19.20	19.02
dolma_100_subreddits_val_10_personalfinance	18.62	17.65	18.82	17.73	17.64
dolma_100_subreddits_val_11_changemyview	20.02	19.10	20.50	19.17	18.99
dolma_100_subreddits_val_12_unpopularopinio	23.39	22.16	23.63	22.32	22.04
dolma_100_subreddits_val_13_movies	21.62	20.52	21.79	20.64	20.35
dolma_100_subreddits_val_14_Games	22.26	21.15	22.52	21.18	20.87
dolma_100_subreddits_val_15_nba	23.28	21.93	23.60	22.10	21.85
dolma_100_subreddits_val_16_pics	21.84	20.56	21.82	20.64	20.47
dolma_100_subreddits_val_17_gaming	24.45	23.13	24.61	23.15	22.86
dolma_100_subreddits_val_18_soccer	23.38	22.12	23.61	22.19	22.03
dolma_100_subreddits_val_19_nfl	19.86	18.76	20.17	18.81	18.62
dolma_100_subreddits_val_20_explainlikeimfiv	18.35	17.21	18.59	17.32	17.03
dolma_100_subreddits_val_21_conspiracy	23.86	22.53	24.09	22.67	22.54
dolma_100_subreddits_val_22_atheism	21.23	20.18	21.43	20.23	20.13
dolma_100_subreddits_val_23_AskMen	20.00	19.04	20.11	19.10	18.94
dolma_100_subreddits_val_24_videos	22.26	21.24	22.51	21.29	21.04
dolma_100_subreddits_val_25_sex	21.13	20.13	21.30	20.09	19.98
dolma_100_subreddits_val_26_raisedbynarcissi	22.07	21.08	22.48	21.20	21.02
dolma_100_subreddits_val_27_NoStupidQuesti	19.66	18.59	19.87	18.68	18.52
dolma_100_subreddits_val_28_DestinyTheGam	35.27	33.58	36.13	33.78	33.37
dolma_100_subreddits_val_29_anime	23.21	22.04	23.46	22.12	21.77
dolma_100_subreddits_val_30_DnD	28.22	26.71	28.78	26.72	26.39
dolma_100_subreddits_val_31_ukpolitics	22.35	21.19	22.80	21.31	21.10
dolma_100_subreddits_val_32_funny	20.78	19.45	20.70	19.40	19.23
dolma_100_subreddits_val_33_europe	21.76	20.59	22.10	20.72	20.52
dolma_100_subreddits_val_34_canada	22.44	21.21	22.44	21.30	21.09
dolma_100_subreddits_val_35_Christianity	17.88	17.02	18.10	17.04	16.94
dolma_100_subreddits_val_36_SquaredCircle	25.87	24.31	25.83	24.34	24.03

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
dolma_100_subreddits_val_37_AskWomen	17.72	16.81	17.77	16.85	16.72
dolma_100_subreddits_val_38_legaladvice	18.66	17.75	18.92	17.74	17.64
dolma_100_subreddits_val_39_JUSTNOMIL	24.25	23.16	24.86	23.32	23.02
dolma_100_subreddits_val_40_technology	23.39	22.09	23.52	22.21	21.95
dolma_100_subreddits_val_41_IAmA	19.83	18.83	19.86	18.71	18.56
dolma_100_subreddits_val_42_wow	31.26	29.25	31.44	29.39	28.82
dolma_100_subreddits_val_43_Parenting	20.15	19.11	20.43	19.30	19.06
dolma_100_subreddits_val_44_exmormon	23.12	21.90	23.44	21.99	21.84
dolma_100_subreddits_val_45_AdviceAnimals	22.14	20.96	22.14	20.98	20.79
dolma_100_subreddits_val_46_childfree	21.87	20.85	22.13	20.89	20.72
dolma_100_subreddits_val_47_unitedkingdom	23.27	22.00	23.40	22.00	21.85
dolma_100_subreddits_val_48_ffxiv	32.53	30.79	33.33	31.01	30.62
dolma_100_subreddits_val_49_dndnext	29.67	28.03	30.53	28.26	27.63
dolma_100_subreddits_val_50_ADHD	20.75	19.83	21.14	19.95	19.78
dolma_100_subreddits_val_51_loseit	19.36	18.39	19.49	18.52	18.33
dolma_100_subreddits_val_52_asoiaf	25.28	23.99	25.63	23.94	23.69
dolma_100_subreddits_val_53_BabyBumps	20.96	19.82	21.11	19.92	19.76
dolma_100_subreddits_val_54_Advice	19.17	18.29	19.35	18.38	18.19
dolma_100_subreddits_val_55_australia	23.97	22.51	24.06	22.61	22.40
dolma_100_subreddits_val_56_CFB	20.45	19.41	20.92	19.49	19.23
dolma_100_subreddits_val_57_offmychest	19.63	18.79	19.77	18.93	18.77
dolma_100_subreddits_val_58_PublicFreakout	25.96	24.49	26.02	24.65	24.39
dolma_100_subreddits_val_59_TrueOffMyChes	21.53	20.63	21.70	20.73	20.54
dolma_100_subreddits_val_60_science	20.44	19.46	20.64	19.51	19.38
dolma_100_subreddits_val_61_magicTCG	28.82	26.79	28.94	26.69	26.38
dolma_100_subreddits_val_62_asktransgender	20.72	19.86	21.07	19.83	19.62
dolma_100_subreddits_val_63_DotA2	34.35	32.38	34.74	32.57	32.16
dolma_100_subreddits_val_64_neoliberal	21.74	20.59	22.26	20.64	20.45
dolma_100_subreddits_val_65_whowouldwin	29.18	27.81	30.08	27.63	27.30
dolma_100_subreddits_val_66_depression	18.28	17.52	18.31	17.50	17.41
dolma_100_subreddits_val_67_WTF	22.30	21.18	22.38	21.17	20.99
dolma_100_subreddits_val_68_pathofexile	40.48	38.59	41.43	38.75	38.43
dolma_100_subreddits_val_69_PoliticalDiscussi	20.01	18.92	20.16	18.97	18.82
dolma_100_subreddits_val_70_Libertarian	22.97	21.77	23.15	21.87	21.75
dolma_100_subreddits_val_71_PurplePillDebate	24.94	23.66	25.44	23.85	23.55
dolma_100_subreddits_val_72_Fitness	21.57	20.35	21.48	20.34	20.11
dolma_100_subreddits_val_73_books	21.12	20.02	21.31	20.09	19.82
dolma_100_subreddits_val_74_dogs	20.13	19.12	20.32	19.20	18.92
dolma_100_subreddits_val_75_pcmasterrace	23.73	22.49	24.02	22.56	22.21
dolma_100_subreddits_val_76_teenagers	18.37	16.35	16.44	15.56	17.02
dolma_100_subreddits_val_77_stopdrinking	21.08	20.02	21.19	20.17	19.98
dolma_100_subreddits_val_78_Overwatch	30.47	28.77	31.13	29.13	28.57
dolma_100_subreddits_val_79_television	23.97	22.63	24.05	22.75	22.49
dolma_100_subreddits_val_80_buildapc	21.55	20.22	21.78	20.29	19.98
dolma_100_subreddits_val_81_askscience	17.25	16.39	17.52	16.34	16.11
dolma_100_subreddits_val_82_programming	23.66	22.61	24.04	22.55	22.24
dolma_100_subreddits_val_83_Guildwars2	32.98	31.17	33.58	31.39	30.91
dolma_100_subreddits_val_84_cars	22.57	21.41	22.73	21.38	21.15
dolma_100_subreddits_val_85_formula1	23.85	22.65	24.09	22.71	22.49
dolma_100_subreddits_val_86_sysadmin	24.23	22.90	24.41	22.96	22.64
dolma_100_subreddits_val_87_hockey	21.46	20.26	21.74	20.37	20.20
dolma_100_subreddits_val_88_india	24.15	22.92	24.42	23.08	22.68

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
dolma_100_subreddits_val_89_SubredditDrama	19.14	18.26	19.63	18.29	18.12
dolma_100_subreddits_val_90_DMAcademy	27.77	26.31	28.38	26.41	26.00
dolma_100_subreddits_val_91_dating_advice	20.18	19.27	20.42	19.40	19.21
dolma_100_subreddits_val_92_Catholicism	19.11	18.22	19.41	18.17	18.03
dolma_100_subreddits_val_93_Drugs	24.50	23.29	24.74	23.32	23.12
dolma_100_subreddits_val_94_trees	23.56	22.38	23.83	22.41	22.25
dolma_100_subreddits_val_95_boardgames	22.69	21.48	23.13	21.61	21.38
dolma_100_subreddits_val_96_Conservative	22.79	21.53	22.97	21.68	21.53
dolma_100_subreddits_val_97_Futurology	23.55	22.36	23.77	22.37	22.17
dolma_100_subreddits_val_98_beyondthebump	21.07	19.89	21.22	20.08	19.83
dolma_100_subreddits_val_99_weddingplanning	20.11	19.01	20.33	19.19	18.96
falcon-refinedweb_val-00000000	15.92	15.46	17.14	15.37	15.22
falcon-refinedweb_val-00000001	18.49	17.91	19.89	17.90	17.71
falcon-refinedweb_val-00000002	18.45	17.90	19.69	17.91	17.68
falcon-refinedweb_val-00000003	16.75	16.23	17.92	16.16	15.89
falcon-refinedweb_val-00000004	16.26	15.66	17.32	15.73	15.41
falcon-refinedweb_val-00000005	15.41	14.96	16.56	14.92	14.74
gab_val-00000000	33.19	30.55	31.57	30.73	30.32
gab_val-00000001	35.64	32.76	33.96	32.80	32.63
gab_val-00000002	34.38	31.68	32.75	31.80	31.65
gab_val-00000003	34.86	32.05	33.26	32.20	32.00
gab_val-00000004	36.20	33.35	34.58	33.42	33.23
gab_val-00000005	33.46	30.82	31.88	31.06	30.72
gab_val-00000006	35.76	32.77	34.26	33.04	32.74
gab_val-00000007	35.54	32.60	33.76	32.78	32.41
gab_val-00000008	35.11	32.03	33.23	32.25	31.86
gab_val-00000009	34.13	31.34	32.36	31.50	31.30
m2d2_s2orc_unsplit_val_Art	20.07	19.80	21.88	19.78	19.44
m2d2_s2orc_unsplit_val_Philosophy	14.80	14.82	16.77	14.69	14.47
m2d2_s2orc_unsplit_val_astro-ph	11.70	11.70	13.18	11.52	11.33
m2d2_s2orc_unsplit_val_astro-ph.CO	11.47	11.49	12.90	11.37	11.15
m2d2_s2orc_unsplit_val_astro-ph.EP	12.76	12.73	14.28	12.60	12.45
m2d2_s2orc_unsplit_val_astro-ph.GA	11.70	11.70	13.18	11.52	11.33
m2d2_s2orc_unsplit_val_astro-ph.HE	11.85	11.77	13.29	11.62	11.46
m2d2_s2orc_unsplit_val_astro-ph.IM	15.36	15.33	17.16	15.21	14.92
m2d2_s2orc_unsplit_val_astro-ph.SR	13.08	13.08	14.89	12.86	12.70
m2d2_s2orc_unsplit_val_astro-ph_l1	15.36	15.33	17.16	15.21	14.92
m2d2_s2orc_unsplit_val_atom-ph	12.74	12.84	14.44	12.75	12.53
m2d2_s2orc_unsplit_val_chem-ph	13.20	13.29	15.22	13.14	12.97
m2d2_s2orc_unsplit_val_cond-mat	11.67	11.78	13.37	11.67	11.50
m2d2_s2orc_unsplit_val_cond-mat.dis-nn	12.54	12.67	14.28	12.58	12.38
m2d2_s2orc_unsplit_val_cond-mat.mes-hall	11.24	11.50	13.19	11.30	11.10
m2d2_s2orc_unsplit_val_cond-mat.mtrl-sci	12.19	12.33	14.09	12.18	11.91
m2d2_s2orc_unsplit_val_cond-mat.other	11.87	11.96	13.55	11.83	11.65
m2d2_s2orc_unsplit_val_cond-mat.quant-gas	11.67	11.78	13.37	11.67	11.50
m2d2_s2orc_unsplit_val_cond-mat.soft	12.18	12.23	13.93	12.18	12.02
m2d2_s2orc_unsplit_val_cond-mat.stat-mech	12.03	12.14	13.60	12.08	11.89
m2d2_s2orc_unsplit_val_cond-mat.str-el	10.39	10.50	11.98	10.41	10.22
m2d2_s2orc_unsplit_val_cond-mat.supr-con	11.57	11.66	13.13	11.53	11.30
m2d2_s2orc_unsplit_val_cond-mat_l1	12.54	12.67	14.28	12.58	12.38
m2d2_s2orc_unsplit_val_cs.AI	11.71	12.09	14.20	12.01	11.79
m2d2_s2orc_unsplit_val_cs.AR	13.09	13.36	15.30	13.18	12.99

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
m2d2_s2orc_unsplit_val_cs.CC	8.45	8.81	10.46	8.70	8.54
m2d2_s2orc_unsplit_val_cs.CE	13.21	13.31	15.01	13.18	13.02
m2d2_s2orc_unsplit_val_cs.CG	8.39	8.68	10.12	8.59	8.47
m2d2_s2orc_unsplit_val_cs.CL	14.66	14.75	16.96	14.70	14.47
m2d2_s2orc_unsplit_val_cs.CR	14.63	14.86	16.72	14.74	14.56
m2d2_s2orc_unsplit_val_cs.CV	12.68	12.78	14.38	12.66	12.49
m2d2_s2orc_unsplit_val_cs.CY	16.01	15.93	17.52	15.84	15.67
m2d2_s2orc_unsplit_val_cs.DB	11.86	12.35	14.66	12.27	12.03
m2d2_s2orc_unsplit_val_cs.DC	13.60	14.02	16.20	13.79	13.56
m2d2_s2orc_unsplit_val_cs.DL	14.67	14.83	17.05	14.75	14.50
m2d2_s2orc_unsplit_val_cs.DM	8.11	8.38	9.84	8.27	8.14
m2d2_s2orc_unsplit_val_cs.DS	9.63	9.99	11.76	9.88	9.69
m2d2_s2orc_unsplit_val_cs.ET	14.80	14.95	17.00	14.89	14.67
m2d2_s2orc_unsplit_val_cs.FL	9.51	9.84	11.64	9.74	9.57
m2d2_s2orc_unsplit_val_cs.GL	16.51	16.43	18.18	16.38	16.21
m2d2_s2orc_unsplit_val_cs.GR	13.45	13.60	15.53	13.54	13.29
m2d2_s2orc_unsplit_val_cs.GT	9.25	9.59	11.34	9.49	9.29
m2d2_s2orc_unsplit_val_cs.HC	16.76	16.93	19.08	16.84	16.66
m2d2_s2orc_unsplit_val_cs.IR	13.30	13.46	15.26	13.31	13.21
m2d2_s2orc_unsplit_val_cs.LG	10.39	10.52	12.14	10.44	10.27
m2d2_s2orc_unsplit_val_cs.LO	9.75	10.23	12.50	10.03	9.81
m2d2_s2orc_unsplit_val_cs.MA	11.24	11.65	14.10	11.41	11.19
m2d2_s2orc_unsplit_val_cs.MM	13.12	13.40	15.29	13.25	13.03
m2d2_s2orc_unsplit_val_cs.MS	13.98	14.14	16.27	14.11	13.89
m2d2_s2orc_unsplit_val_cs.NA	10.53	10.80	12.52	10.71	10.47
m2d2_s2orc_unsplit_val_cs.NE	13.76	14.00	16.10	13.89	13.64
m2d2_s2orc_unsplit_val_cs.NI	10.00	10.22	11.61	10.04	9.93
m2d2_s2orc_unsplit_val_cs.OH	15.24	15.43	17.62	15.34	15.10
m2d2_s2orc_unsplit_val_cs.OS	14.61	14.93	17.35	14.80	14.53
m2d2_s2orc_unsplit_val_cs.PF	12.60	12.82	14.71	12.70	12.48
m2d2_s2orc_unsplit_val_cs.PL	15.43	15.74	18.58	15.65	15.40
m2d2_s2orc_unsplit_val_cs.RO	13.04	13.19	14.95	13.12	12.87
m2d2_s2orc_unsplit_val_cs.SC	11.10	11.42	13.33	11.30	11.10
m2d2_s2orc_unsplit_val_cs.SD	13.27	13.42	15.26	13.36	13.13
m2d2_s2orc_unsplit_val_cs.SE	17.72	13.47	15.46	13.40	13.21
m2d2_s2orc_unsplit_val_cs.SI	12.03	12.25	14.03	12.19	11.99
m2d2_s2orc_unsplit_val_cs.SY	11.40	11.79	13.51	11.63	11.39
m2d2_s2orc_unsplit_val_cs_l1	8.39	8.68	10.12	8.59	8.47
m2d2_s2orc_unsplit_val_econ.EM	11.62	11.76	13.73	11.68	11.41
m2d2_s2orc_unsplit_val_econ.TH	9.75	10.16	11.99	9.99	9.88
m2d2_s2orc_unsplit_val_econ_l1	9.75	10.16	11.99	9.99	9.88
m2d2_s2orc_unsplit_val_eess.AS	12.05	12.14	13.88	12.09	11.88
m2d2_s2orc_unsplit_val_eess.IV	13.77	13.89	15.71	13.76	13.54
m2d2_s2orc_unsplit_val_eess.SP	11.29	11.45	12.94	11.28	11.13
m2d2_s2orc_unsplit_val_eess_l1	13.77	13.89	15.71	13.76	13.54
m2d2_s2orc_unsplit_val_gr-qc	12.84	12.99	14.68	12.84	12.71
m2d2_s2orc_unsplit_val_hep-ex	10.47	10.37	11.61	10.13	9.96
m2d2_s2orc_unsplit_val_hep-lat	13.13	13.10	14.57	13.02	12.80
m2d2_s2orc_unsplit_val_hep-ph	11.67	11.81	13.38	11.66	11.45
m2d2_s2orc_unsplit_val_hep-th	11.46	11.49	12.71	11.40	11.24
m2d2_s2orc_unsplit_val_math.AC	7.08	7.37	8.71	7.26	7.13
m2d2_s2orc_unsplit_val_math.AG	8.89	9.27	11.05	9.16	8.95

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
m2d2_s2orc_unsplit_val_math.AP	9.35	9.53	10.90	9.41	9.35
m2d2_s2orc_unsplit_val_math.AT	8.57	8.77	10.16	8.72	8.53
m2d2_s2orc_unsplit_val_math.CA	9.18	9.49	11.01	9.36	9.30
m2d2_s2orc_unsplit_val_math.CO	6.99	7.33	8.69	7.21	7.08
m2d2_s2orc_unsplit_val_math.CT	9.78	10.20	12.04	10.12	9.91
m2d2_s2orc_unsplit_val_math.CV	7.81	8.07	9.36	7.99	7.87
m2d2_s2orc_unsplit_val_math.DG	7.96	8.18	9.50	8.08	7.98
m2d2_s2orc_unsplit_val_math.DS	7.88	8.12	9.61	8.08	7.96
m2d2_s2orc_unsplit_val_math.FA	7.71	7.96	9.35	7.88	7.81
m2d2_s2orc_unsplit_val_math.GM	7.85	8.15	9.57	8.07	7.93
m2d2_s2orc_unsplit_val_math.GN	6.27	6.56	7.82	6.45	6.38
m2d2_s2orc_unsplit_val_math.GR	7.39	7.66	9.00	7.51	7.41
m2d2_s2orc_unsplit_val_math.GT	7.47	7.71	9.27	7.62	7.47
m2d2_s2orc_unsplit_val_math.HO	14.52	14.70	16.52	14.51	14.31
m2d2_s2orc_unsplit_val_math.KT	7.54	7.80	9.14	7.70	7.58
m2d2_s2orc_unsplit_val_math.LO	9.84	10.41	12.53	10.13	10.03
m2d2_s2orc_unsplit_val_math.MG	8.25	8.53	9.99	8.42	8.26
m2d2_s2orc_unsplit_val_math.NA	9.85	10.05	11.66	9.95	9.83
m2d2_s2orc_unsplit_val_math.NT	8.26	8.51	9.92	8.43	8.31
m2d2_s2orc_unsplit_val_math.OA	7.21	7.55	9.07	7.47	7.32
m2d2_s2orc_unsplit_val_math.OC	9.70	10.01	11.62	9.85	9.69
m2d2_s2orc_unsplit_val_math.PR	8.91	9.20	10.58	9.04	8.99
m2d2_s2orc_unsplit_val_math.QA	8.09	8.40	9.93	8.28	8.16
m2d2_s2orc_unsplit_val_math.RA	7.18	7.44	8.75	7.39	7.27
m2d2_s2orc_unsplit_val_math.RT	8.39	8.71	10.33	8.65	8.49
m2d2_s2orc_unsplit_val_math.SG	8.63	8.88	10.36	8.76	8.59
m2d2_s2orc_unsplit_val_math.SP	9.39	9.65	11.27	9.52	9.37
m2d2_s2orc_unsplit_val_math_l1	7.81	8.07	9.36	7.99	7.87
m2d2_s2orc_unsplit_val_nlin.AO	11.82	12.01	13.77	11.90	11.75
m2d2_s2orc_unsplit_val_nlin.CD	12.73	12.91	14.88	12.87	12.60
m2d2_s2orc_unsplit_val_nlin.CG	12.43	12.75	14.88	12.61	12.44
m2d2_s2orc_unsplit_val_nlin.PS	11.29	11.44	12.86	11.39	11.22
m2d2_s2orc_unsplit_val_nlin.SI	9.44	9.81	11.28	9.64	9.51
m2d2_s2orc_unsplit_val_nlin_l1	12.43	12.75	14.88	12.61	12.44
m2d2_s2orc_unsplit_val_nucl-ex	13.02	12.94	14.61	12.85	12.63
m2d2_s2orc_unsplit_val_nucl-th	11.65	11.78	13.43	11.68	11.48
m2d2_s2orc_unsplit_val_physics.acc-ph	13.75	14.01	16.17	13.74	13.58
m2d2_s2orc_unsplit_val_physics.ao-ph	13.92	14.04	15.91	13.89	13.68
m2d2_s2orc_unsplit_val_physics.app-ph	13.70	13.81	15.54	13.62	13.43
m2d2_s2orc_unsplit_val_physics.atm-clus	13.00	13.13	15.11	13.00	12.74
m2d2_s2orc_unsplit_val_physics.atom-ph	12.74	12.84	14.44	12.75	12.53
m2d2_s2orc_unsplit_val_physics.bio-ph	13.30	13.42	15.26	13.32	13.08
m2d2_s2orc_unsplit_val_physics.chem-ph	13.20	13.29	15.22	13.14	12.97
m2d2_s2orc_unsplit_val_physics.class-ph	11.01	11.27	12.85	11.12	10.94
m2d2_s2orc_unsplit_val_physics.comp-ph	11.23	11.37	12.88	11.26	11.08
m2d2_s2orc_unsplit_val_physics.data-an	13.18	13.33	14.97	13.25	13.00
m2d2_s2orc_unsplit_val_physics.ed-ph	12.21	12.33	13.88	12.18	12.03
m2d2_s2orc_unsplit_val_physics.flu-dyn	11.81	11.99	13.73	11.81	11.64
m2d2_s2orc_unsplit_val_physics.gen-ph	14.15	14.39	16.76	14.18	14.03
m2d2_s2orc_unsplit_val_physics.geo-ph	14.75	14.86	16.81	14.71	14.57
m2d2_s2orc_unsplit_val_physics.hist-ph	15.57	15.43	16.97	15.40	15.18
m2d2_s2orc_unsplit_val_physics.ins-det	14.01	14.16	16.14	14.07	13.79

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
m2d2_s2orc_unsplit_val_physics.med-ph	14.34	14.46	16.50	14.29	14.09
m2d2_s2orc_unsplit_val_physics.optics	12.74	12.94	14.64	12.80	12.54
m2d2_s2orc_unsplit_val_physics.plasm-ph	13.65	13.81	15.77	13.69	13.44
m2d2_s2orc_unsplit_val_physics.pop-ph	13.80	13.67	15.17	13.60	13.41
m2d2_s2orc_unsplit_val_physics.soc-ph	12.79	12.97	14.80	12.83	12.66
m2d2_s2orc_unsplit_val_physics.space-ph	13.00	13.09	14.77	12.94	12.76
m2d2_s2orc_unsplit_val_physics_l1	15.57	15.43	16.97	15.40	15.18
m2d2_s2orc_unsplit_val_plasm-ph	13.65	13.81	15.77	13.69	13.44
m2d2_s2orc_unsplit_val_q-bio	13.69	13.87	15.75	13.75	13.50
m2d2_s2orc_unsplit_val_q-bio.BM	13.28	13.52	15.72	13.41	13.19
m2d2_s2orc_unsplit_val_q-bio.CB	12.06	12.34	14.21	12.19	11.97
m2d2_s2orc_unsplit_val_q-bio.GN	13.21	11.40	12.74	11.32	11.16
m2d2_s2orc_unsplit_val_q-bio.MN	11.96	11.95	13.36	11.90	11.70
m2d2_s2orc_unsplit_val_q-bio.NC	13.69	13.87	15.75	13.75	13.50
m2d2_s2orc_unsplit_val_q-bio.OT	14.90	14.94	17.16	14.92	14.73
m2d2_s2orc_unsplit_val_q-bio.PE	12.57	12.71	14.62	12.69	12.41
m2d2_s2orc_unsplit_val_q-bio.QM	12.49	12.69	14.44	12.56	12.40
m2d2_s2orc_unsplit_val_q-bio.SC	13.68	13.85	15.60	13.75	13.53
m2d2_s2orc_unsplit_val_q-bio.TO	13.49	13.53	15.32	13.48	13.33
m2d2_s2orc_unsplit_val_q-bio_l1	13.69	13.87	15.75	13.75	13.50
m2d2_s2orc_unsplit_val_q-fin.CP	11.37	11.61	13.36	11.41	11.28
m2d2_s2orc_unsplit_val_q-fin.EC	11.72	11.89	13.77	11.77	11.63
m2d2_s2orc_unsplit_val_q-fin.GN	13.79	13.91	15.73	13.83	13.61
m2d2_s2orc_unsplit_val_q-fin.MF	9.91	10.21	11.92	10.04	9.90
m2d2_s2orc_unsplit_val_q-fin.PM	11.00	11.31	13.14	11.14	10.94
m2d2_s2orc_unsplit_val_q-fin.PR	15.87	9.25	10.37	9.20	9.03
m2d2_s2orc_unsplit_val_q-fin.RM	11.35	11.49	13.08	11.41	11.22
m2d2_s2orc_unsplit_val_q-fin.ST	12.43	12.46	14.18	12.43	12.26
m2d2_s2orc_unsplit_val_q-fin.TR	12.79	13.14	15.32	12.89	12.74
m2d2_s2orc_unsplit_val_q-fin_l1	13.79	13.91	15.73	13.83	13.61
m2d2_s2orc_unsplit_val_quant-ph	11.18	11.44	13.18	11.32	11.11
m2d2_s2orc_unsplit_val_stat.AP	13.37	13.56	15.52	13.42	13.15
m2d2_s2orc_unsplit_val_stat.CO	13.07	12.56	14.42	12.46	12.24
m2d2_s2orc_unsplit_val_stat.ME	11.09	11.26	12.91	11.11	10.87
m2d2_s2orc_unsplit_val_stat.ML	11.13	11.39	13.29	11.23	11.06
m2d2_s2orc_unsplit_val_stat.OT	11.31	11.55	13.28	11.45	11.24
m2d2_s2orc_unsplit_val_stat_l1	13.07	12.56	14.42	12.46	12.24
m2d2_s2orc_unsplit_val_supr-con	11.57	11.66	13.13	11.53	11.30
m2d2_wikipedia_unsplit_val_Culture_and_the_	12.30	11.90	12.82	11.78	11.66
m2d2_wikipedia_unsplit_val_Culture_and_the_	12.13	11.74	12.82	11.63	11.48
m2d2_wikipedia_unsplit_val_Culture_and_the_	14.06	13.86	15.17	13.79	13.57
m2d2_wikipedia_unsplit_val_Culture_and_the_	12.16	11.80	12.74	11.79	11.55
m2d2_wikipedia_unsplit_val_Culture_and_the_	11.75	11.25	12.03	11.17	11.03
m2d2_wikipedia_unsplit_val_Culture_and_the_	10.01	9.63	10.36	9.58	9.54
m2d2_wikipedia_unsplit_val_Culture_and_the_	12.13	11.85	12.83	11.73	11.58
m2d2_wikipedia_unsplit_val_Culture_and_the_	12.36	12.09	13.05	11.99	11.87
m2d2_wikipedia_unsplit_val_General_referece	11.80	11.46	12.43	11.46	11.30
m2d2_wikipedia_unsplit_val_General_referece_	10.52	10.20	10.96	10.12	9.99
m2d2_wikipedia_unsplit_val_General_referece_	11.80	11.46	12.43	11.46	11.30
m2d2_wikipedia_unsplit_val_Health_and_fitnes	10.75	10.47	11.14	10.37	10.30
m2d2_wikipedia_unsplit_val_Health_and_fitnes	9.64	9.29	9.95	9.27	9.16
m2d2_wikipedia_unsplit_val_Health_and_fitnes	10.10	9.80	10.43	9.71	9.56

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
m2d2_wikipedia_unsplit_val_Health_and_fitnes	9.14	8.83	9.59	8.63	8.54
m2d2_wikipedia_unsplit_val_Health_and_fitnes	8.91	8.68	9.40	8.61	8.47
m2d2_wikipedia_unsplit_val_Health_and_fitnes	10.75	10.47	11.14	10.37	10.30
m2d2_wikipedia_unsplit_val_Health_and_fitnes	12.91	12.49	13.61	12.42	12.28
m2d2_wikipedia_unsplit_val_History_and_even	13.65	13.29	14.48	13.20	13.00
m2d2_wikipedia_unsplit_val_History_and_even	11.77	11.44	12.36	11.36	11.26
m2d2_wikipedia_unsplit_val_History_and_even	12.78	12.41	13.46	12.37	12.12
m2d2_wikipedia_unsplit_val_History_and_even	12.36	11.88	12.87	11.79	11.64
m2d2_wikipedia_unsplit_val_Human_activites	12.43	12.03	12.98	11.95	11.81
m2d2_wikipedia_unsplit_val_Human_activites_	12.43	12.03	12.98	11.95	11.81
m2d2_wikipedia_unsplit_val_Human_activites_	12.47	12.05	13.12	12.00	11.82
m2d2_wikipedia_unsplit_val_Mathematics_and_	12.90	12.51	13.79	12.48	12.29
m2d2_wikipedia_unsplit_val_Mathematics_and_	8.24	8.26	9.37	8.28	8.06
m2d2_wikipedia_unsplit_val_Mathematics_and_	13.21	12.87	13.90	12.85	12.67
m2d2_wikipedia_unsplit_val_Mathematics_and_	12.90	12.51	13.79	12.48	12.29
m2d2_wikipedia_unsplit_val_Natural_and_phys	9.19	8.22	8.81	7.97	7.96
m2d2_wikipedia_unsplit_val_Natural_and_phys	10.97	10.70	11.53	10.64	10.51
m2d2_wikipedia_unsplit_val_Natural_and_phys	11.69	11.36	12.28	11.22	11.05
m2d2_wikipedia_unsplit_val_Natural_and_phys	10.43	10.11	10.95	10.00	9.82
m2d2_wikipedia_unsplit_val_Natural_and_phys	11.48	11.09	11.93	10.98	10.90
m2d2_wikipedia_unsplit_val_Philosophy_and_t	11.83	11.72	13.04	11.60	11.45
m2d2_wikipedia_unsplit_val_Philosophy_and_t	12.00	11.61	12.66	11.57	11.43
m2d2_wikipedia_unsplit_val_Philosophy_and_t	10.94	10.61	11.34	10.56	10.42
m2d2_wikipedia_unsplit_val_Religion_and_beli	12.81	12.45	13.44	12.38	12.19
m2d2_wikipedia_unsplit_val_Religion_and_beli	11.11	10.80	11.66	10.71	10.58
m2d2_wikipedia_unsplit_val_Religion_and_beli	11.46	11.06	11.86	10.95	10.85
m2d2_wikipedia_unsplit_val_Religion_and_beli	12.38	12.03	12.94	11.91	11.79
m2d2_wikipedia_unsplit_val_Society_and_socia	10.53	10.24	11.03	10.16	10.05
m2d2_wikipedia_unsplit_val_Society_and_socia	10.47	10.16	10.95	10.14	10.04
m2d2_wikipedia_unsplit_val_Society_and_socia	12.48	12.13	13.02	12.07	11.93
m2d2_wikipedia_unsplit_val_Technology_and_	8.51	8.18	8.66	7.93	7.88
m2d2_wikipedia_unsplit_val_Technology_and_	12.45	12.07	13.00	12.03	11.88
m2d2_wikipedia_unsplit_val_Technology_and_	13.62	13.23	14.56	13.18	12.97
m2d2_wikipedia_unsplit_val_Technology_and_	13.00	12.72	13.87	12.64	12.43
m2d2_wikipedia_unsplit_val_Technology_and_	14.34	13.90	15.20	13.94	13.73
manosphere_meta_sep_val_avfm	19.42	19.27	21.88	19.64	19.18
manosphere_meta_sep_val_incels	11.26	12.18	21.40	11.51	11.29
manosphere_meta_sep_val_mgtow	24.83	24.27	27.50	24.12	23.80
manosphere_meta_sep_val_pua_forum	24.22	23.85	26.52	23.86	23.52
manosphere_meta_sep_val_red_pill_talk	34.59	33.90	37.26	33.90	33.27
manosphere_meta_sep_val_reddit	20.63	19.78	21.10	19.94	19.58
manosphere_meta_sep_val_rooshv	22.46	22.17	24.78	22.01	21.69
manosphere_meta_sep_val_the_attraction	20.85	20.57	23.17	20.57	20.20
mc4_val-00000000	8.35	8.41	10.02	8.23	8.15
mc4_val-00000001	12.17	11.97	13.58	11.74	11.64
mc4_val-00000002	9.96	10.06	11.96	9.86	9.67
mc4_val-00000003	11.38	11.29	12.77	11.12	11.00
mc4_val-00000004	11.96	11.64	13.03	11.50	11.35
ptb_val	15.92	16.65	19.37	16.00	15.92
redpajama_val_arxiv	5.15	5.28	5.78	5.12	5.09
redpajama_val_books	12.91	12.71	13.60	12.61	12.50
redpajama_val_c4	13.01	12.51	13.55	12.49	12.27

Dataset	Llama	Mamba	RWKV-4	xLSTM[7:1]	xLSTM[1:0]
redpajama_val_commoncrawl	10.90	10.56	11.70	10.52	10.35
redpajama_val_github	1.66	1.66	1.75	1.65	1.64
redpajama_val_stackexchange	3.73	3.72	4.03	3.68	3.63
redpajama_val_wikipedia	4.64	4.38	4.68	4.35	4.29
twitterAAE_HELM_fixed_val_AA	346.98	302.79	310.30	301.65	289.97
twitterAAE_HELM_fixed_val_white	118.62	107.34	109.13	107.65	105.13
wikitext_103_val	11.74	11.76	13.73	11.32	11.41