xLSTM:
Recurrent Neural Network Architectures for Scalable and Efficient Large Language Models

PhD Defense

Maximilian Beck, maximilianbeck@live.de maxmbeck maxbeck.ai

March 2026, JKU Linz

First Supervisor:	Prof. Dr. Sepp Hochreiter
Second Supervisor:	Prof. Mag. Dr. Günter Klambauer
External Reviewer:	Prof. Dr. Gerhard Neumann
Additional Reviewer:	Prof. Dr. Martina Seidl

Language models are transforming society – but their efficiency matters

Impact Areas:

Education Arts & Media Finance Software development AI research …

In daily lives:

- LLMs as **chatbots**
- LLMs as **personal assistants**

<u>BUT</u> LLMs need a lot of energy:

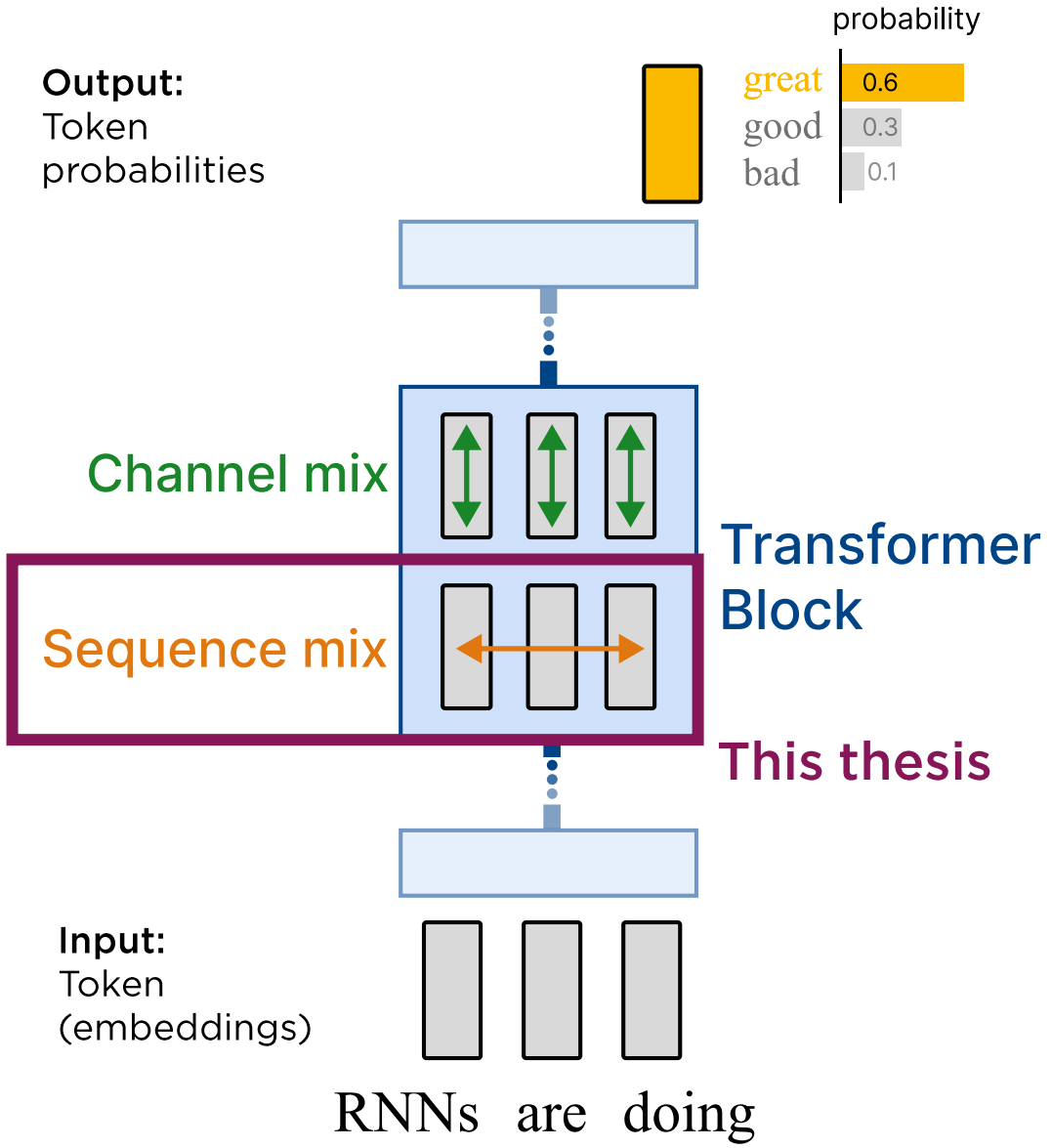
- A **median Gemini text prompt** uses roughly the energy of **~9 seconds of TV** [1,2]
- Voice or image prompts consume even more energy
- Training energy costs are not included [3]

Rapid adoption and increasing autonomy of LLMs makes compute- and energy-efficient architectures essential

[1] Elsworth, Huang, Patterson, et al. Measuring the environmental impact of delivering AI at Google Scale. arXiv, 2508.15734, 2025.
[2] https://cloud.google.com/blog/products/infrastructure/measuring-the-environmental-impact-of-ai-inference/
[3] Rahman and Owen. The training compute of notable AI models has been doubling roughly every six months. 2024. URL https://epoch.ai/data-insights/compute-trend-post-2010.

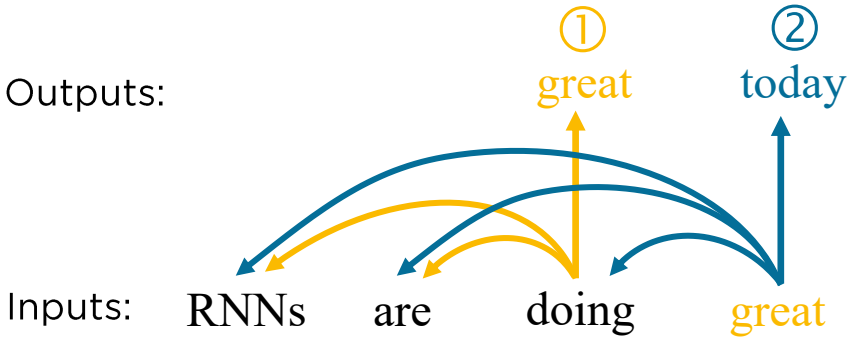
What are LLMs?

- LLMs are neural networks
  based on the Transformer architecture
- LLMs **process text as sequence of tokens**
  (a token represents a short word or a few characters)
- LLMs **predict the probability of the next tokens**
- **Transformers „transform“ token embedding vectors**
  by a stack of Transformer blocks
- Each block consists of **two „transformation“ components**:
  - Channel mix: mixing along the token vector/channel dim
  - Sequence mix: mixing along the sequence dimension



Attention

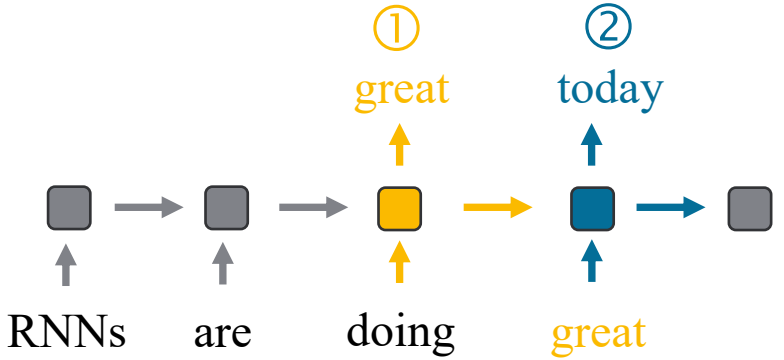
Attention compares every input to all previous inputs



- ✓ Good performance on language
- ✓ Very efficient training due to parallel computation
- ✗ Growing memory with sequence length
- ✗ Quadratic compute cost

vs. traditional RNNs & LSTMs

Recurrent Neural Networks (RNNs) maintain a fixed memory, which is updated for every new input



- ✗ Worse performance on language
- ✗ Inefficient training (not parallelizable)
- ✓ Fixed memory, independent of sequence length
- ✓ Linear compute cost

Research Question & Contributions

Research question: Can we design architectures with Transformer quality and RNN efficiency?

Contributions:

1) Novel recurrent architectures: xLSTM family

2) Hardware-aware algorithms: Efficient implementations for xLSTM & linear RNNs

3) Large-scale evaluation: xLSTM 7B & Scaling laws

NeurIPS ’24

xLSTM: Extended Long Short-Term Memory

Maximilian Beck* [1,2,3] Korbinian Pöppel* [1,2,3] Markus Spanring [1]
Andreas Auer [1,2] Oleksandra Prudnikova [1] Michael Kopp
Günter Klambauer [1,2,3] Johannes Brandstetter [1,2,3] Sepp Hochreiter [1,2,3]
*Equal contribution
[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

NeurIPS ’25

Tiled Flash Linear Attention: More Efficient Linear RNN and xLSTM Kernels

Maximilian Beck[1,2] Korbinian Pöppel[1,2] Phillip Lippe[2*] Sepp Hochreiter[1,2]
[1] ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2] NXAI GmbH, Linz, Austria

ICML ’25

xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference

Maximilian Beck [*12] Korbinian Pöppel [*12] Phillip Lippe [*13] Richard Kurle [1] Patrick M. Blies [1]
Günter Klambauer [12] Sebastian Böck [1] Sepp Hochreiter [12]

ICLR ’26

xLSTM Scaling Laws: Competitive Performance with Linear Time-Complexity

Maximilian Beck [1,2] Kajetan Schweighofer [1]
Sebastian Böck [2] Sebastian Lehner [1] Sepp Hochreiter [1,2]

1) Novel recurrent architectures

xLSTM family

NeurIPS ’24

xLSTM: Extended Long Short-Term Memory

Maximilian Beck[*1,2,3] **Korbinian Pöppel**[*1,2,3] **Markus Spanring**[1]
Andreas Auer[1,2] **Oleksandra Prudnikova**[1] **Michael Kopp**
Günter Klambauer[1,2,3] **Johannes Brandstetter**[1,2,3] **Sepp Hochreiter**[1,2,3]
[*]Equal contribution
[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

Our starting point: LSTM

- LSTM overcomes vanishing gradient problem of traditional RNNs through gating [1,2]
- Before the introduction of Transformers in 2016,
  LSTM was one of the main methods for sequence and language modeling

memory update:

$$c_t = \sigma(\tilde{\mathrm{f}}_t)\, c_{t-1} + \sigma(\tilde{\mathrm{i}}_t)\, \tanh(\tilde{z}_t) \quad \text{cell state}$$

$$h_t = \sigma(\tilde{\mathrm{o}}_t)\, \tanh(c_t) \quad \text{hidden state}$$

LSTM

memory states

gates

gates:

$$\tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z \quad \text{cell input}$$

$$\tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \boldsymbol{x}_t + r_\mathrm{i} h_{t-1} + b_\mathrm{i} \quad \text{input gate}$$

$$\tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \boldsymbol{x}_t + r_\mathrm{f} h_{t-1} + b_\mathrm{f} \quad \text{forget gate}$$

$$\tilde{\mathrm{o}}_t = \boldsymbol{w}_\mathrm{o}^\top \boldsymbol{x}_t + r_\mathrm{o} h_{t-1} + b_\mathrm{o} \quad \text{output gate}$$

recurrent weights
(memory mixing)

Note:
Recurrent weights prohibit sequence parallel training.

[1] Hochreiter (1991) Untersuchungen zu dynamischen neuronalen Netzen, Masters thesis
[2] Hochreiter, Schmidhuber (1997) Long Short-Term Memory, Neural Computation

xLSTM: Overcoming the limitations of the LSTM

Initial question:
How far do we get in scaling LSTM to billions of parameters for language modeling?

We found:
Not so far with the original LSTM.

Three main limitations of LSTM:

L1: Inability to revise storage decisions

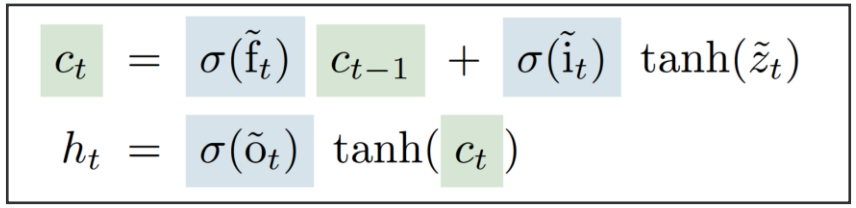
L2: Limited storage capacity

L3: Lack of parallelizability

xLSTM extensions:

1) Exponential Gating
2) Matrix memory
3) Fully parallelizable variant, new memory mixing

Exponential Gating



LSTM

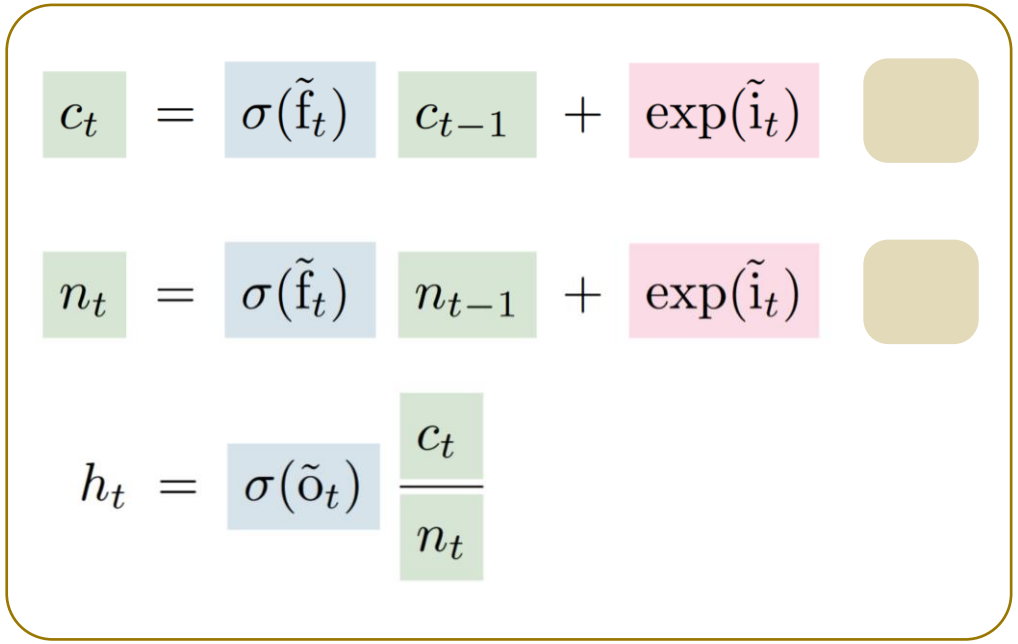
memory states

gates

exponential input gates

xLSTM

(Exponential Gating)



memory cell state

normalizer state ← NEW

hidden state

+ stabilizer state (not shown) ← NEW

xLSTM: Extended Long Short-Term Memory

Two new memory cells with **exponential input gates**:

sLSTM

$$
\begin{aligned}
c_t &= \sigma(\tilde{\mathrm{f}}_t)\, c_{t-1} + \exp(\tilde{\mathrm{i}}_t)\, \tanh(\tilde{z}_t) && c_t \in \mathbb{R} \\
n_t &= \sigma(\tilde{\mathrm{f}}_t)\, n_{t-1} + \exp(\tilde{\mathrm{i}}_t) && n_t \in \mathbb{R} \\
h_t &= \sigma(\tilde{\mathrm{o}}_t)\, \frac{c_t}{n_t} && h_t \in \mathbb{R}
\end{aligned}
$$

scalar cell state

non-linear RNN

more expressive,
i.e. enables state tracking

mLSTM

$$
\begin{aligned}
\boldsymbol{C}_t &= \sigma(\tilde{\mathrm{f}}_t)\, \boldsymbol{C}_{t-1} + \exp(\tilde{\mathrm{i}}_t)\, \boldsymbol{v}_t \boldsymbol{k}_t^\top && \boldsymbol{C}_t \in \mathbb{R}^{d \times d} \\
\boldsymbol{n}_t &= \sigma(\tilde{\mathrm{f}}_t)\, \boldsymbol{n}_{t-1} + \exp(\tilde{\mathrm{i}}_t)\, \boldsymbol{k}_t && \boldsymbol{n}_t \in \mathbb{R}^{d} \\
\boldsymbol{h}_t &= \sigma(\tilde{\mathrm{o}}_t) \odot \frac{\boldsymbol{C}_t\, \boldsymbol{q}_t}{\max\left\{\left|\boldsymbol{n}_t^\top\, \boldsymbol{q}_t\right|, 1\right\}} && \boldsymbol{h}_t \in \mathbb{R}^{d}
\end{aligned}
$$

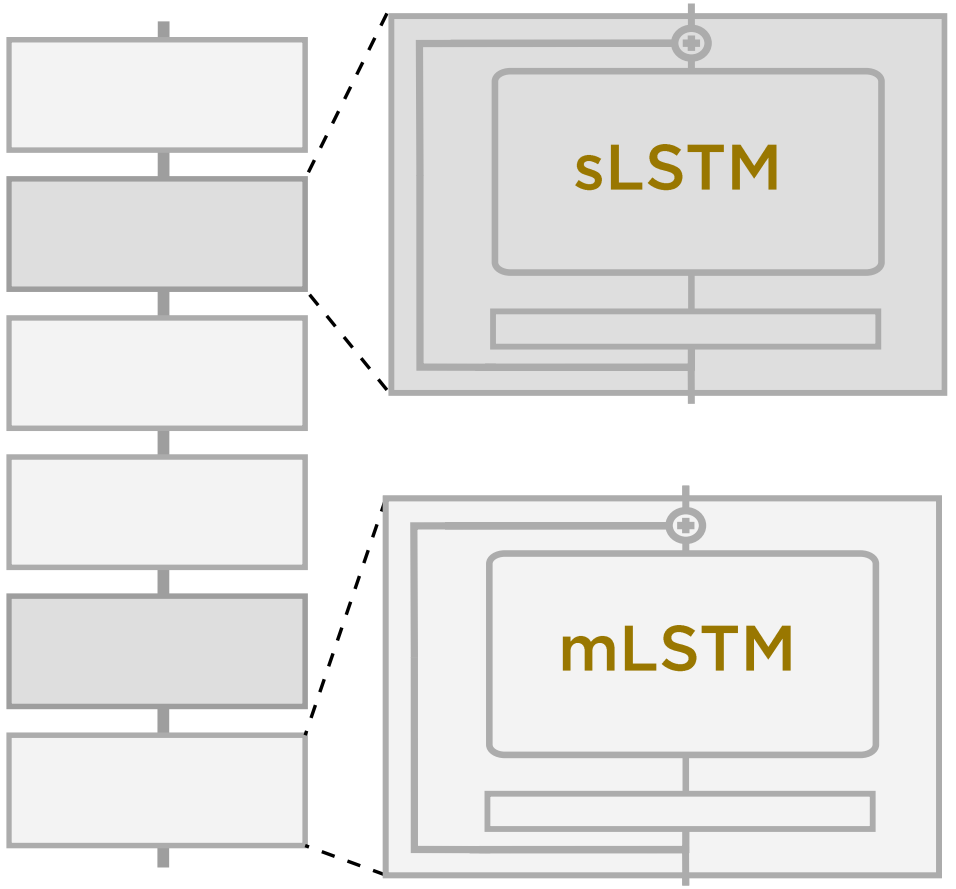
matrix cell state

linear RNN

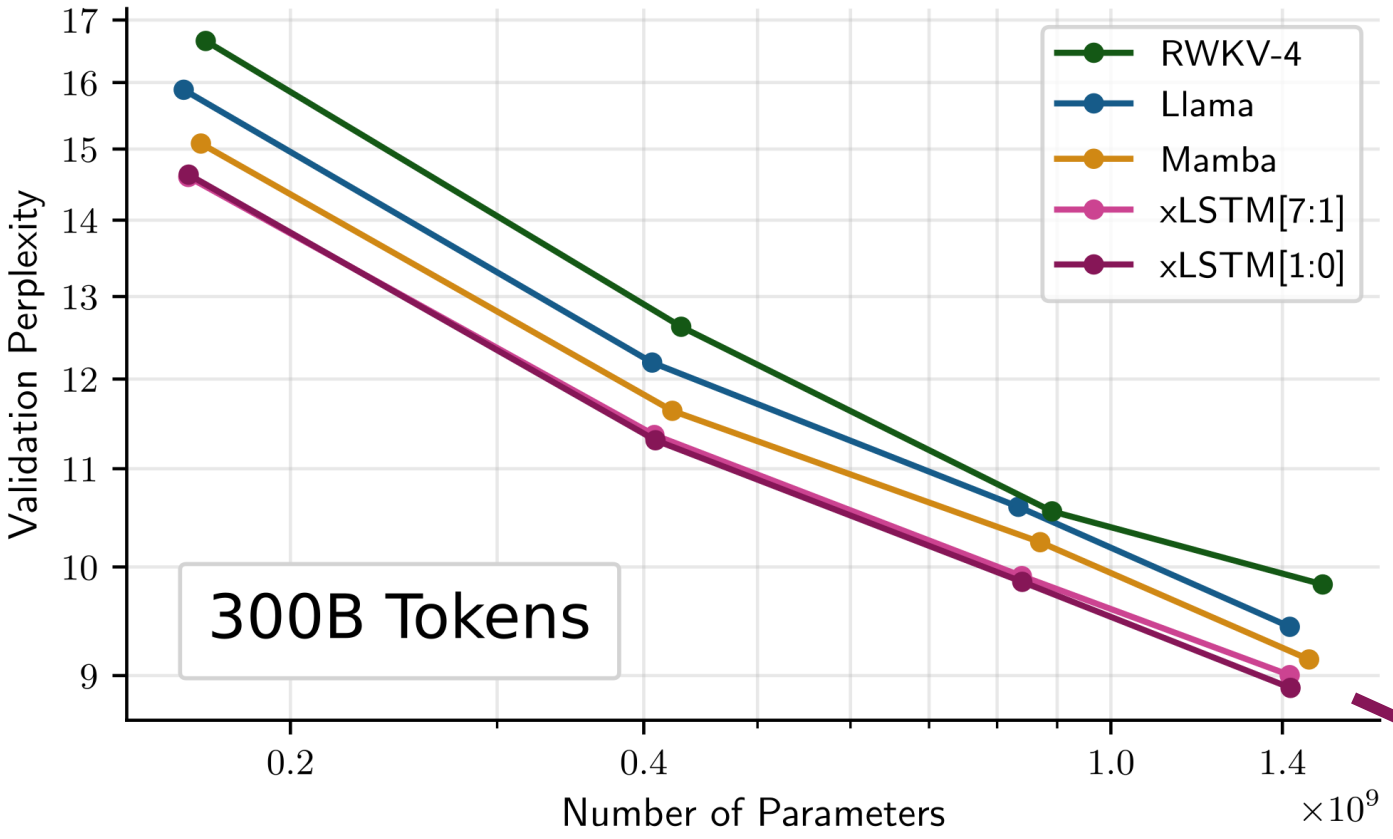
more efficient,
i.e. enables parallel training

xLSTM architecture

- We use Transformer pre-norm blocks
- We stack mLSTM and sLSTM blocks at a certain ratio



Performance in language modeling



What layers to choose?

- sLSTM enhances performance on state tracking tasks, no gain in general language modeling (so far)
- sLSTM better on other tasks
  - Biological and chemical sequence modeling [1]
  - Time-series modeling [2]
- mLSTM performs best on language modeling
  - also used for Vision-LSTM [3]

We focus on mLSTM for the remainder of this talk!

?

Does this trend continue?

xLSTM[7:1]: 7 mLSTM per 1 sLSTM block
xLSTM[1:0]: mLSTM blocks only

[1] Schmidinger, Schneckenreiter, Seidl, ..., Klambauer (2024) Bio-xLSTM: Generative modeling, representation and in-context learning of biological and chemical sequences, NeurIPS
[2] Auer, Podest, Klotz, Böck, Klambauer, Hochreiter (2025) TiRex: Zero-Shot Forecasting Across Long and Short Horizons with Enhanced In-Context Learning, NeurIPS
[3] Alkin, Beck, Pöppel, Hochreiter and Brandstetter (2025) Vision-LSTM: xLSTM as Generic Vision Backbone, ICLR

2) Hardware-aware algorithm design

Efficient implementations for xLSTM & linear RNNs

NeurIPS ’25

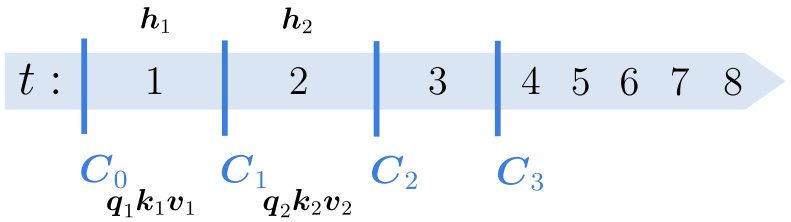
Tiled Flash Linear Attention: More Efficient Linear RNN and xLSTM Kernels

Maximilian Beck[1,2] **Korbinian Pöppel**[1,2] **Phillip Lippe**[2*] **Sepp Hochreiter**[1,2]
[1] ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2] NXAI GmbH, Linz, Austria

Chunkwise-parallel combines recurrent and parallel formulations

Recurrent formulation:

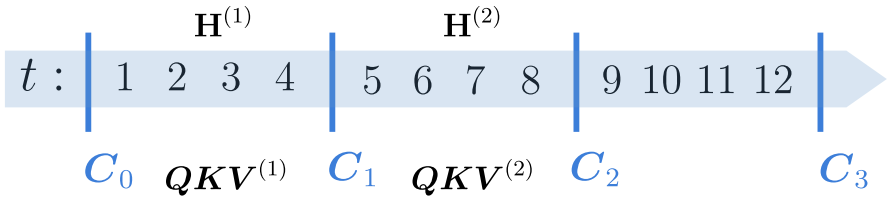
- Materialize C state in every time step
- Linear compute
- Not sequence parallel



$$C_t = \mathrm{f}_t\, C_{t-1} + \mathrm{i}_t\, k_t\, v_t^\top$$
$$h_t = C_t^\top\, q_t$$

Chunkwise-parallel formulation:

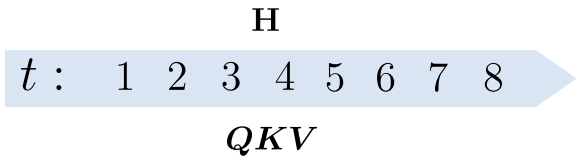
- Materialize C state every L-th time step
- Linear compute
- Sequence parallel



$$C_k = \mathrm{g}_k C_{k-1} + \left(\mathbf{a}_k \odot K^{(k)}\right)^\top V^{(k)}$$
$$\mathbf{H}^{(k)} = \left(\mathbf{b}_k \odot Q^{(k)}\right) C_{k-1} + \left(Q^{(k)} K^{(k)\top} \odot \mathbf{D}_k\right) V^{(k)}$$

Parallel formulation:

- No C states
- Quadratic compute
- Sequence parallel

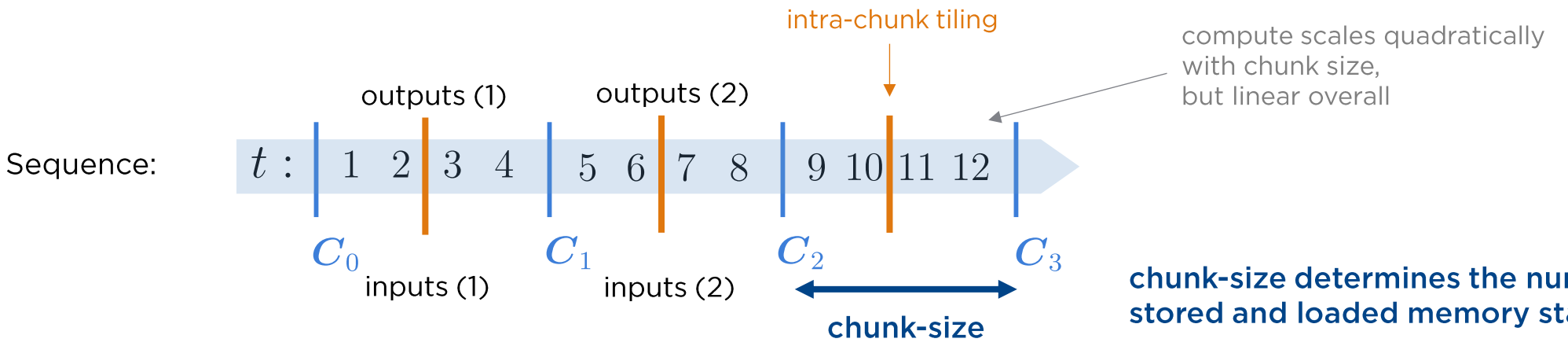


$$\mathbf{H} = \left(QK^\top \odot \mathbf{D}\right) V$$

Hua, Dai, Liu, Le (2022) Transformer quality in linear time, ICML
Sun, Dong, Huang, Ma, Xia, Xue, Wang (2023) Retentive network: A successor to transformer for large language models

Arbitrary large chunk sizes with TFLA

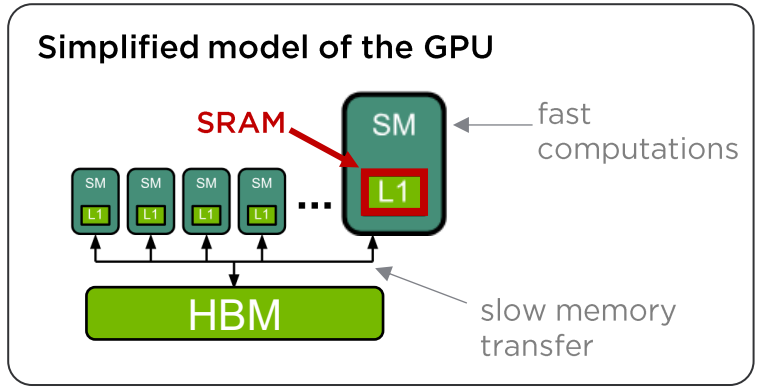
FLA kernels [1] leverage a chunkwise-parallel formulation of linear RNNs:



chunk-size determines the number of stored and loaded memory states

Problem: Chunk size in FLA kernels is limited by physical SRAM size.
(all inputs & outputs per chunk must fit in SRAM)

➡ **We need to load & store many memory states! → Slow and high GPU memory usage!**

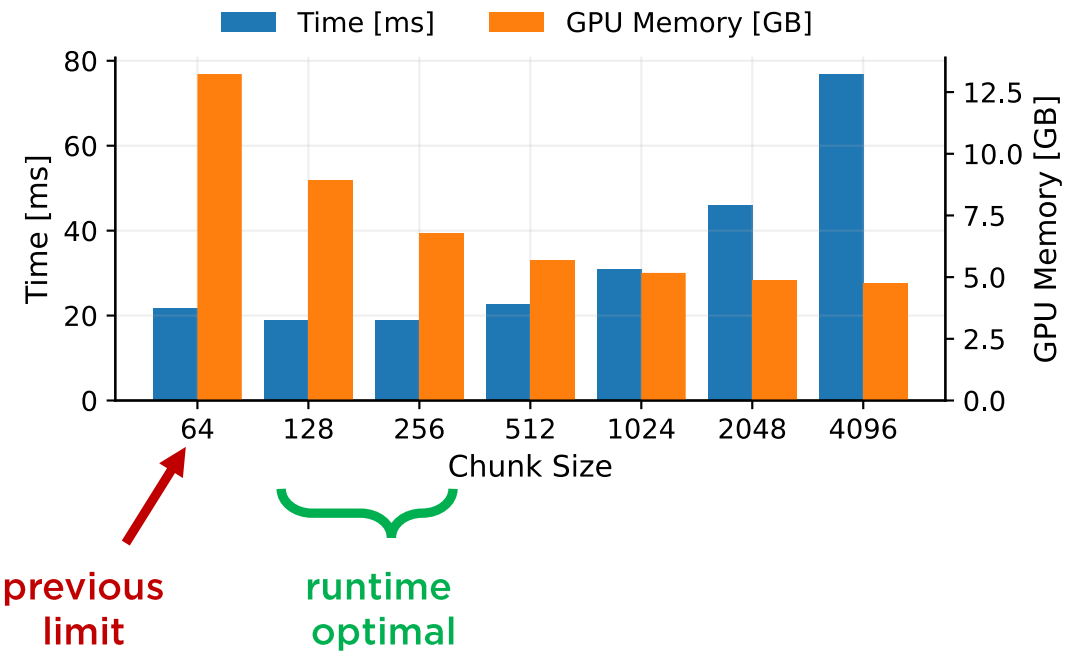


➡ **Solution: TFLA introduces an additional tiling dimension within the chunks!**
(only inputs & outputs per tile must fit in SRAM, use more tiles for larger chunk size)

[1] Yang, Wang, Shen, Panda, Kim (2024) Gated linear attention transformers with hardware-efficient training, ICML

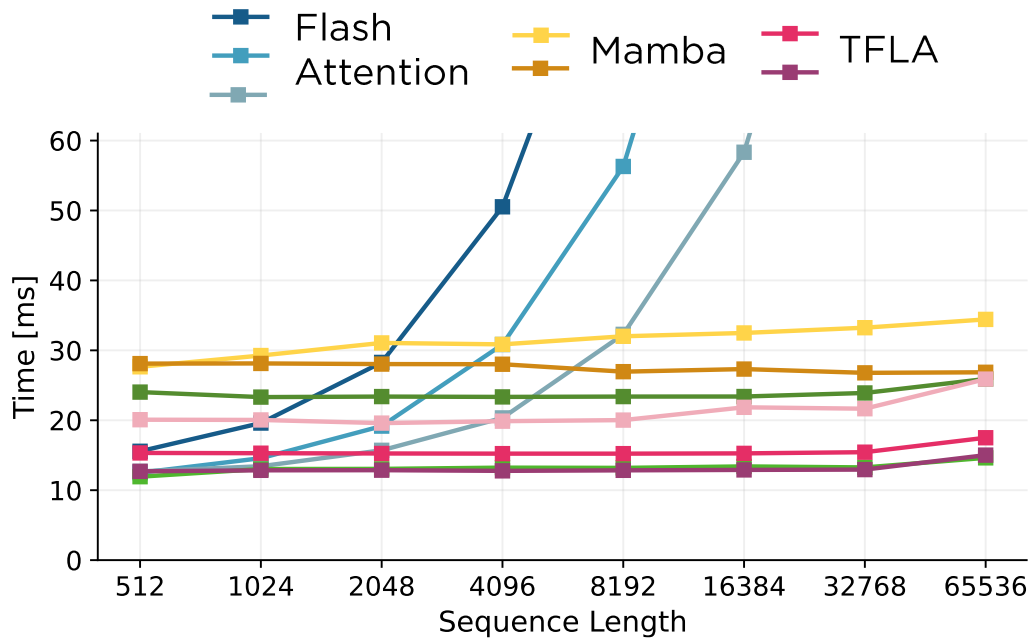
Results

Trade-off between memory & runtime



&

State of the art training kernel runtimes



Training kernel runtime

➡ **TFLA kernels are faster than FlashAttention & 2x faster than Mamba**

seq len 8192
embedding dim 4096
head dim 512
batch size 8

3) Large-scale evaluation

xLSTM 7B & Scaling laws

ICML ’25

xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference

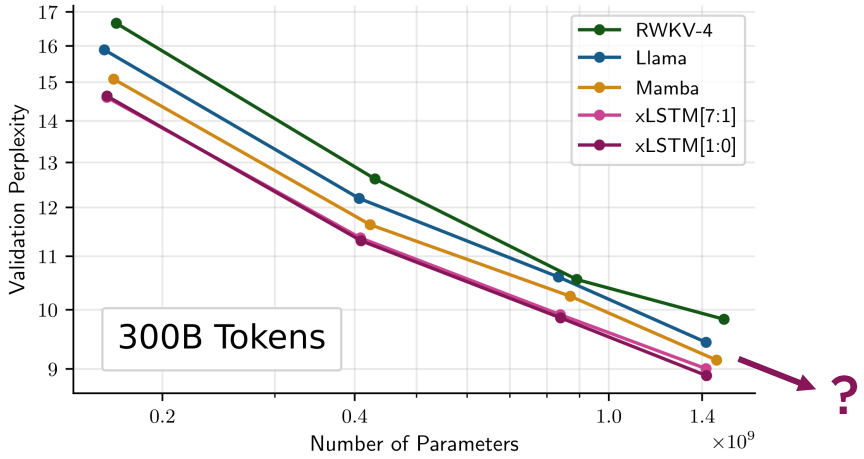
Maximilian Beck [*1 2] Korbinian Pöppel [*1 2] Phillip Lippe [*1 3] Richard Kurle [1] Patrick M. Blies [1]
Günter Klambauer [1 2] Sebastian Böck [1] Sepp Hochreiter [1 2]

ICLR ’26

xLSTM Scaling Laws: Competitive Performance with Linear Time-Complexity

Maximilian Beck [1,2] Kajetan Schweighofer [1]
Sebastian Böck [2] Sebastian Lehner [1] Sepp Hochreiter [1,2]

Recall: So far we trained xLSTM models up to 1.4B parameters



Does this trend continue?

How to scale xLSTM to 7B parameters and beyond?

Naive scaling of xLSTM:

- Training instabilities
- Very low training speeds

With xLSTM 7B we optimize the xLSTM architecture for...

Stable convergence

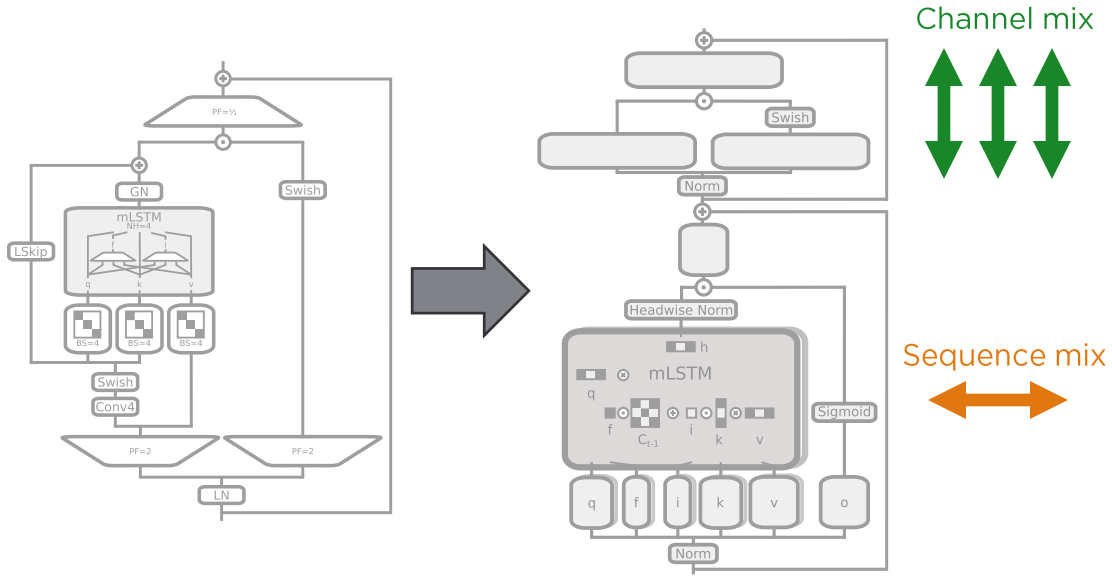
- Gate soft-capping $\text{softcap}_a(\boldsymbol{x}) = a \cdot \tanh(\boldsymbol{x}/a)$
- Pre-Norm with RMS norm instead of Layernorm
- Negative input gate bias initialization

&

High efficiency during training and inference

- Fast & efficient kernels ✓
- Block architecture optimizations

Optimized block architecture: 2-3x speedup



Channel mix

Sequence mix

„Mamba-style“ „Transformer-style“

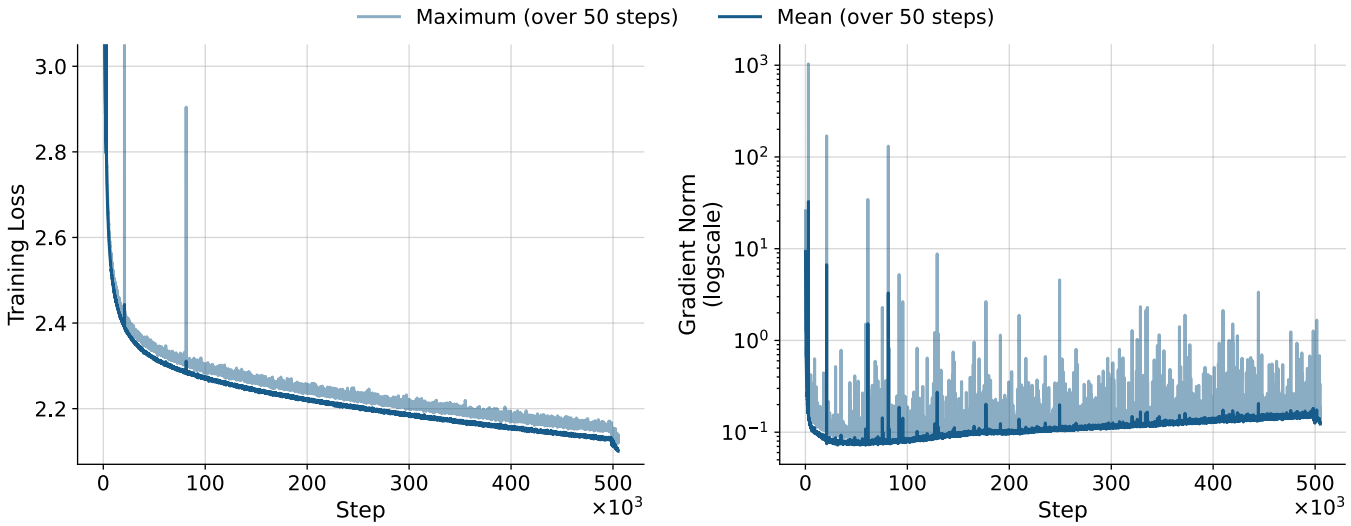
Model		Throughput ↑ 1K tokens/sec	Speedup ↑	ppl ↓	Δ ppl
160M	Previous	76.20		20.43	
	Ours	225.99	×2.97	21.34	+0.91
400M	Previous	28.13		15.26	
	Ours	102.40	×3.64	15.74	+0.48
1.4B	Previous	10.57		12.46	
	Ours	37.03	×3.50	12.68	+0.22
7B	Previous	3.46		-	
	Ours	9.15	× 2.64	-	

2-3x speedup in training throughput at slight trade-off in validation PPL

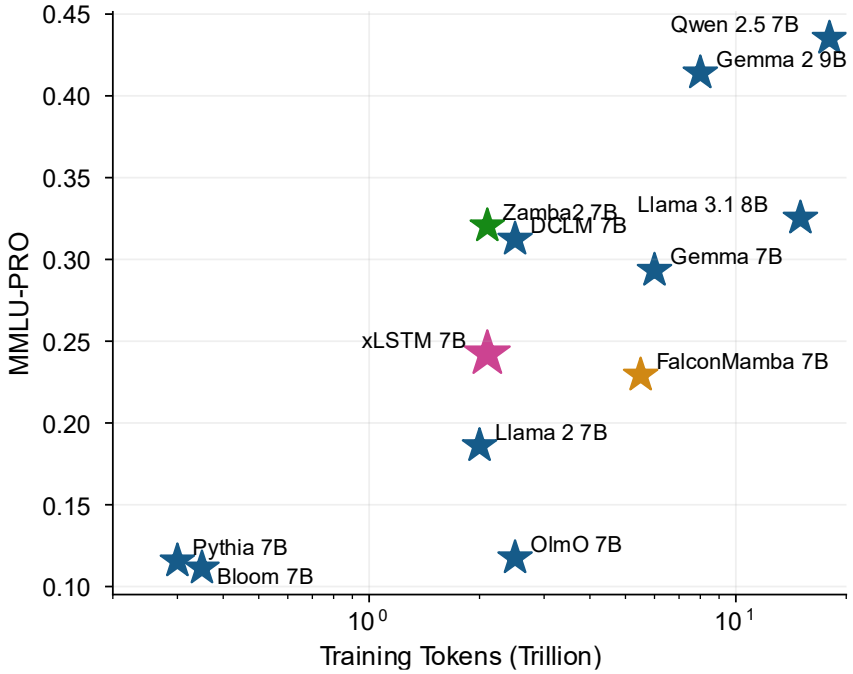
- PPL difference gets smaller for larger models
- Could be mitigated by a few more training steps

[1] Gu, Dao (2024) Mamba: Linear-Time Sequence Modeling with Selective State Spaces, COLM

xLSTM 7B is stable and performs comparably on downstream tasks



xLSTM 7B was pre-trained with 8k context window
on 2.3T tokens from DCLM
on 256 H100 GPUs using only FSDP within 12 days (293h).

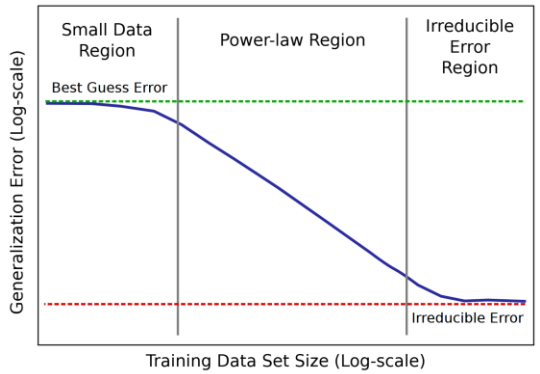


xLSTM 7B is on par with comparable models on downstream tasks

The xLSTM 7B architecture is the basis for our scaling law analysis

xLSTM scaling laws

Neural Scaling Laws

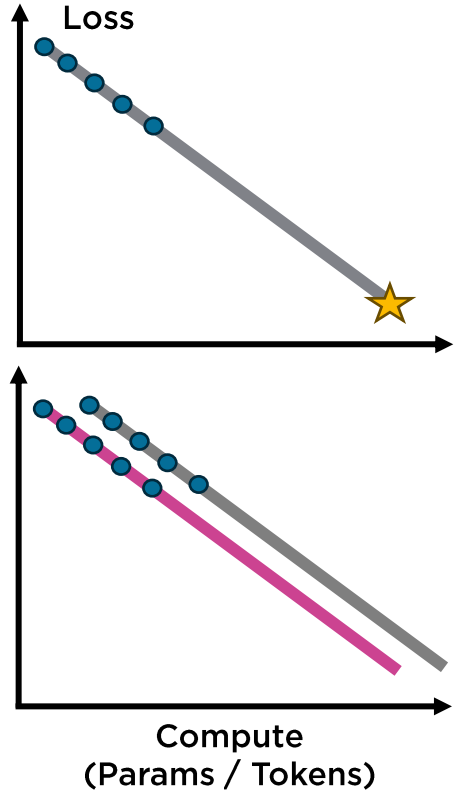


- Empirical relation between loss and compute, model or dataset size
- These relations follow power law trends,
  and often hold over many orders of magnitude [1,2]

Significant practical implications:

- Predict the outcome of & derisk very large training runs
- Compare model classes
- Allow for compute-optimal allocation of model parameters N & training tokens D [2]

$$N^*(H), D^*(H) = \underset{N,D \text{ s.t. } C(N,D)=H}{\operatorname{argmin}} L(N, D)$$



Note: The common assumption for the FLOPs $C = 6ND$ for Transformer scaling laws is not sufficient when comparing model classes and different contexts

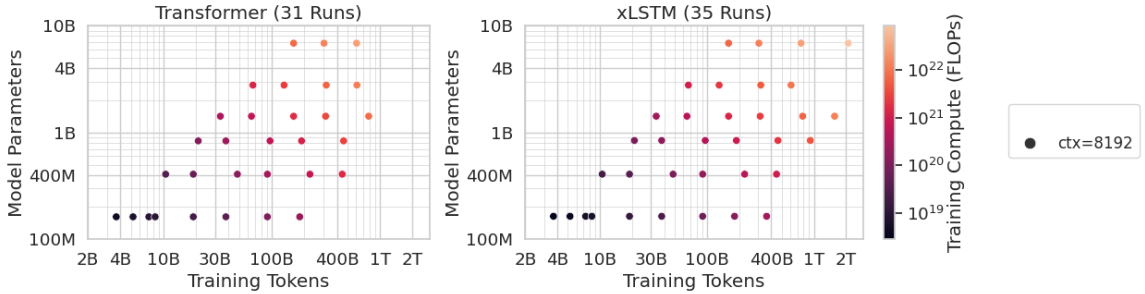
- It only counts feedforward layer FLOPs, ignores attention FLOPs
- We provide a more faithful FLOP calculation

[1] Hestness, Narang, Ardalani, ..., Zhou (2017) Deep Learning Scaling is Predictable, Empirically, ArXiv
[2] Kaplan, McCandlish, Henighan, ..., Amodei (2020) Scaling Laws for Neural Language Models, ArXiv
[3] Hoffmann, Borgeaud, Mensch, ..., Sifre (2022) Training Compute-Optimal Large Language Models, ArXiv

Outline of our scaling law study

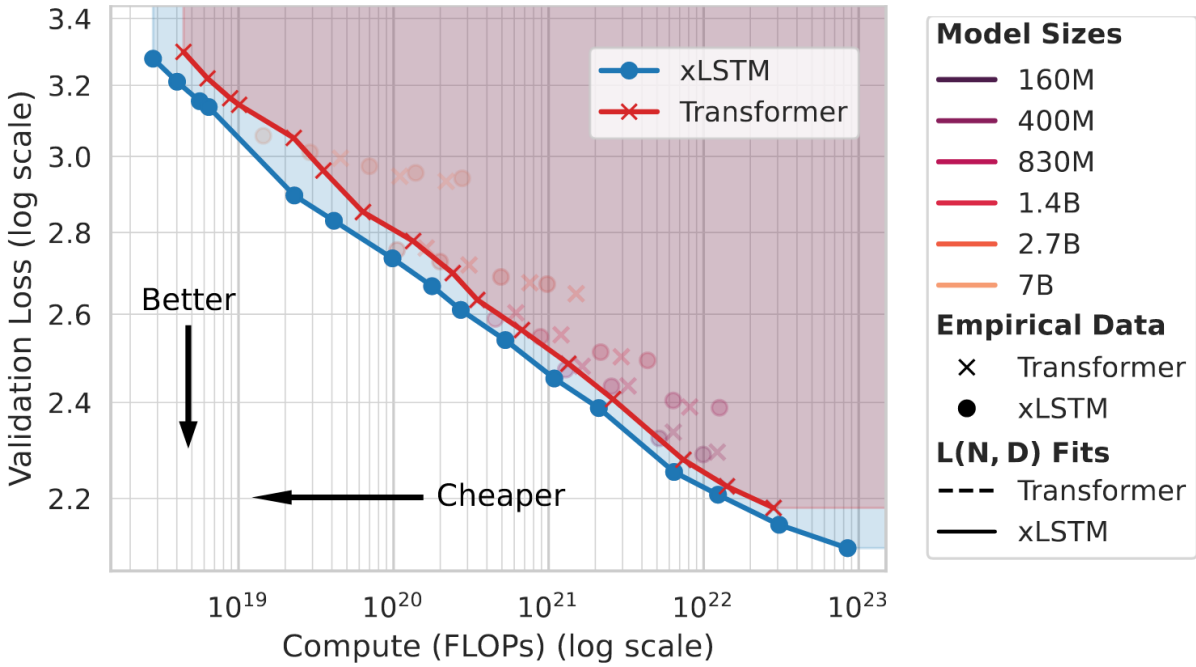
Token/param

Fix model parameters,
vary training tokens



We study:

- **Loss vs. compute**
- Overtraining regime,
  i.e. performance with higher than compute-optimal token/param ratios



➡ **xLSTMs are Pareto-dominant to Transformers in terms of loss vs. compute**

* All models are trained on the DCLM dataset
* For Transformer models we use the Llama 2 architecture
* We only consider scaling in terms of loss (no downstream tasks yet)

Outline of our scaling law study

Token/param

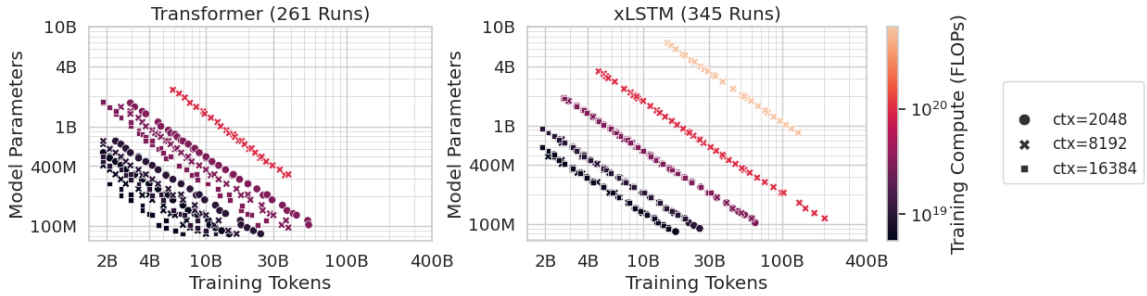
Fix model parameters,
vary training tokens

We study:

- **Loss vs. compute**
- Overtraining regime,
  i.e. performance with higher than compute-optimal token/param ratios

IsoFLOP

Fix compute budget,
vary model parameters & training tokens



We study:

- Compute-optimal model sizes
- Context length dependence
  of the compute-optimal model size

Main results of our IsoFLOP experiments:

➡ **xLSTM attains lower loss for the same compute budget**

➡ **Compute-optimal xLSTM model size is larger**

➡ **Compute-optimal xLSTM model size remains stable across context lengths**

Outline of our scaling law study

Token/param

Fix model parameters,
vary training tokens

We study:

- **Loss vs. compute**
- Overtraining regime,
  i.e. performance with higher than compute-optimal token/param ratios

IsoFLOP

Fix compute budget,
vary model parameters & training tokens

We study:

- Compute-optimal model sizes
- Context length dependence
  of the compute-optimal model size

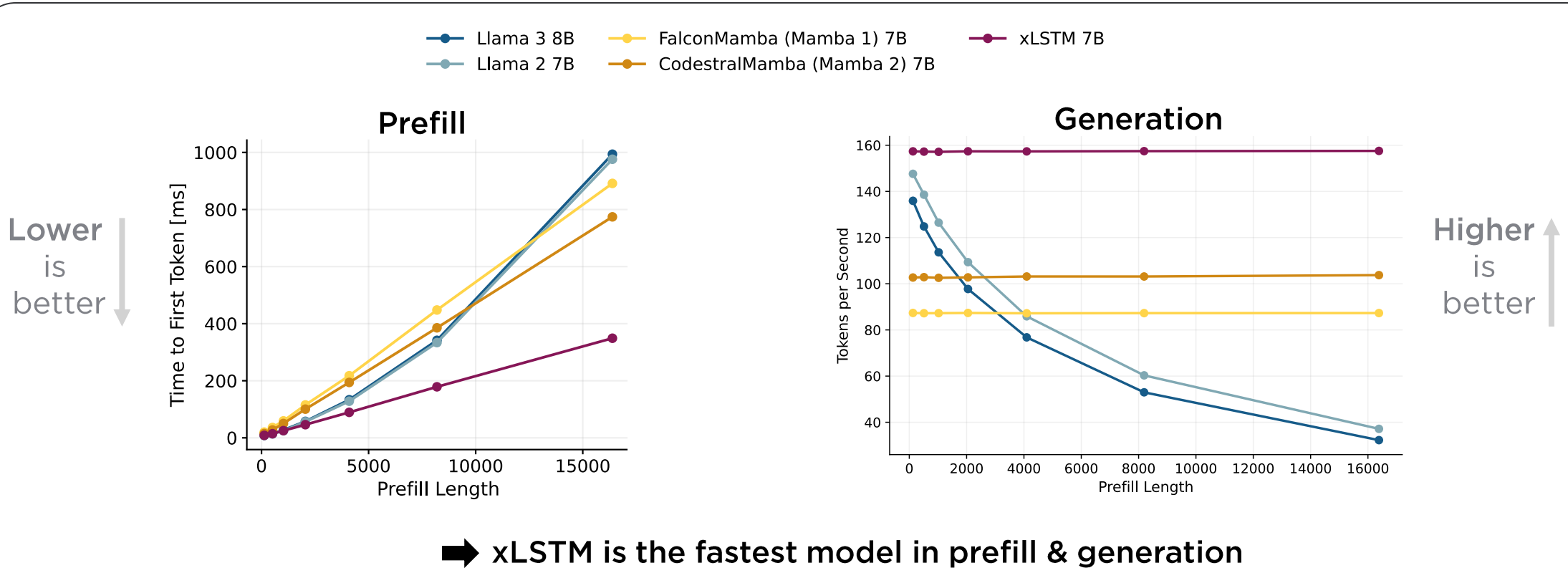
Inference speed

We compare:

- **Prefill speed** in time to first token (TTFT)
- **Generation speed** in step time or tokens/s

Inference speed of xLSTM 7B

- Two inference stages:
  - **Prefill:** build up KV-Cache / cell state over input sequence → TTFT
  - **Generation:** predict next tokens autoregressively → tokens per second
- We measure inference speeds with **batch size 1** (i.e. single user setting)
  - CUDA Graph optimized Huggingface implementations

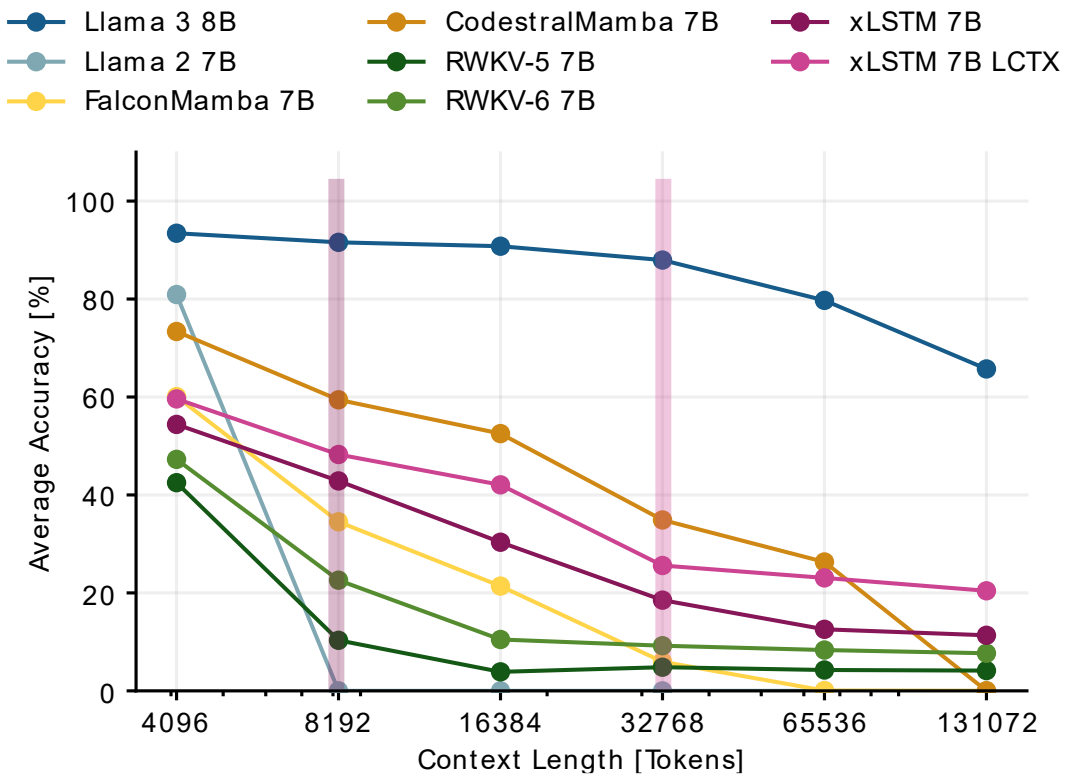


➡ **xLSTM is the fastest model in prefill & generation**

Why doesn’t everybody train xLSTMs or other linear RNNs?

Limitations, Future Work & Conclusion

Long context performance of xLSTM lags behind Transformers



Average Accuracy on RULER Benchmark

- xLSTM 7B trained on ctx len 8192
- xLSTM has good „out-of-the-box“ extrapolation capabilities
  - even for 131k, where Mamba breaks
- **xLSTM 7B LCTX:**
  - Continued pre-training for 50k steps with ctx len 32k improves long context performance

But:

xLSTM lags behind the performance of Transformers (Llama 3)

Reason:

The fixed state size limits the memory capacity

There exists a trade-off:

Memory state size vs. long ctx performance vs. inference speed

Future work

- **Push the memory size vs. efficiency frontier**
  - Larger memory size through new (hybrid) memory designs
  - Better use of memory via improved gating techniques

- **Scale xLSTMs and integrate with Mixture-of-Experts**
  - Train larger xLSTM models
  - Combine xLSTMs with sparse expert routing

- **Strengthen the xLSTM ecosystem**
  - First-class support in major inference engines (e.g. vLLM)
  - Develop xLSTM-based reasoning models

- **Explore other applications and modalities for xLSTM (e.g. robotics, vision-language, VLAs)**

Summary

This thesis has contributed across the full stack of LLM development:

1) Recurrent LLM architecture design:

xLSTM: a scalable recurrent architecture for LLMs

- Matches Transformer language modeling quality
- Constant memory and linear compute scaling with sequence length

Architecture → xLSTM

2) Hardware-aware algorithm design:

TFLA mLSTM kernels for efficient large-scale training

- Enable 7B+ parameter xLSTM models
- Memory vs. runtime trade-off & state-of-the-art kernel speed

Systems → TFLA kernels

3) Large scale LLM training & evaluation:

xLSTM 7B & empirical scaling analysis of xLSTMs

- Favorable performance in terms of loss compared to Transformers
- Significantly faster prefill and generation

Empirical scaling → xLSTM 7B & scaling laws

Conclusion

Research question: Can we design architectures with Transformer quality and RNN efficiency?

Yes. xLSTM demonstrates that modernized recurrent architectures can serve as competitive and efficient LLM backbones.

Thank you!

Huge thanks to all my co-authors and everyone who supported me throughout this PhD journey ☺

Special thanks to Sepp & NXAI

NeurIPS ’24

xLSTM: Extended Long Short-Term Memory

Maximilian Beck* [1,2,3] Korbinian Pöppel* [1,2,3] Markus Spanring [1]
Andreas Auer [1,2] Oleksandra Prudnikova [1] Michael Kopp
Günter Klambauer [1,2,3] Johannes Brandstetter [1,2,3] Sepp Hochreiter [1,2,3]
*Equal contribution
[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

NeurIPS ’25

Tiled Flash Linear Attention: More Efficient Linear RNN and xLSTM Kernels

Maximilian Beck[1,2] Korbinian Pöppel[1,2] Phillip Lippe[2*] Sepp Hochreiter[1,2]
[1] ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2] NXAI GmbH, Linz, Austria

ICML ’25

xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference

Maximilian Beck[*1 2] Korbinian Pöppel[*1 2] Phillip Lippe[*1 3] Richard Kurle[1] Patrick M. Blies[1]
Günter Klambauer[1 2] Sebastian Böck[1] Sepp Hochreiter[1 2]

ICLR ’26

xLSTM Scaling Laws: Competitive Performance with Linear Time-Complexity

Maximilian Beck [1,2] Kajetan Schweighofer [1]
Sebastian Böck [2] Sebastian Lehner [1] Sepp Hochreiter [1,2]

Maximilian Beck, maximilianbeck@live.de maxmbeck maxbeck.ai

Summary of this thesis

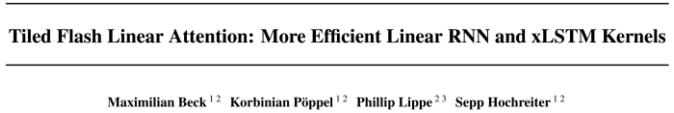
Contribution 1: Enhance LSTMs for competitive Language Modeling Performance


xLSTM: Extended Long Short-Term Memory

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

NeurIPS 2024

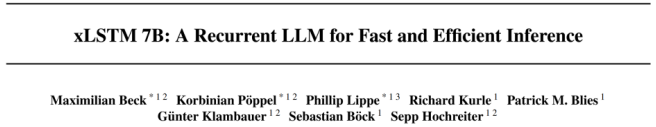
Contribution 2: Introduce a new kernel algorithm for efficient training of RNNs


Tiled Flash Linear Attention: More Efficient Linear RNN and xLSTM Kernels

Maximilian Beck [1 2] Korbinian Pöppel [1 2] Phillip Lippe [2 3] Sepp Hochreiter [1 2]

NeurIPS 2025

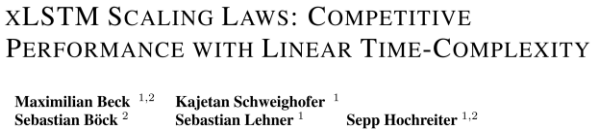
Contribution 3: Build an xLSTM for large scale training and for inference


xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference

Maximilian Beck* [1 2] Korbinian Pöppel* [1 2] Phillip Lippe* [1 3] Richard Kurle [1] Patrick M. Blies [1]
Günter Klambauer [1 2] Sebastian Böck [1] Sepp Hochreiter [1 2]

ICML 2025

Contribution 4: Scaling Laws for xLSTM


xLSTM Scaling Laws: Competitive Performance with Linear Time-Complexity

Maximilian Beck [1,2] Kajetan Schweighofer [1]
Sebastian Böck [2] Sebastian Lehner [1] Sepp Hochreiter [1,2]

ICLR 2026

Date: March 12th 2026

(Co-)authored papers:

- Towards a Neural Debugger for Python (Submitted to ICML 2026)
- Short window attention enables long-term memorization (ICLR 2026)
- FlashRNN: Optimizing Traditional RNNs on Modern Hardware (ICLR 2025)
- Vision-LSTM: xLSTM as Generic Vision Backbone (ICLR 2025)
- A Large Recurrent Action Model: xLSTM enables Fast Inference for Robotics Tasks (ICML 2025)
- Addressing Parameter Choice Issues in Unsupervised Domain Adaptation by Aggregation (ICLR 2023)
- Few-Shot Learning by Dimensionality Reduction in Gradient Space (CoLLAs 2022)

Impact:

- 1142 citations
- xLSTM GitHub repository has 2.1k stars, TFLA GitHub repository has 87 stars
- 13 invited talks about my research
- xLSTM is developed further by NXAI, a spin-off company of JKU

- Part of ELLIS PhD Programm
- Research Internship at Meta FAIR in Paris from May to October 2025