xLSTM: Extended Long Short-Term Memory

EECE 571F

Parshan Javanrood and Ziqiao (John) Lin

January 21,2026

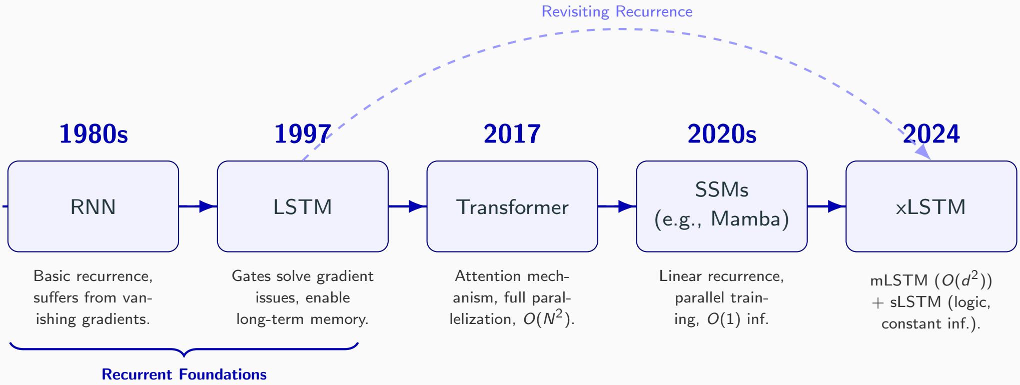
University of British Columbia

Extended Long Short-Term Memory (xLSTM)

By the end of this presentation, you'll know...

- The evolution from **Recurrent Foundations** to the Attention era
- How **sLSTM** uses **exponential gating** and **stabilizers** to overcome the sigmoid bottleneck
- The transition to **mLSTM** with **matrix memory** for enhanced storage capacity
- Why **removing memory mixing** enables Transformer-like **parallel training** via parallel scan
- How xLSTM achieves **constant inference cost** and scales effectively in the LLM era

Evolution of Sequence Models: A Timeline



Standard LSTM: Architecture Overview

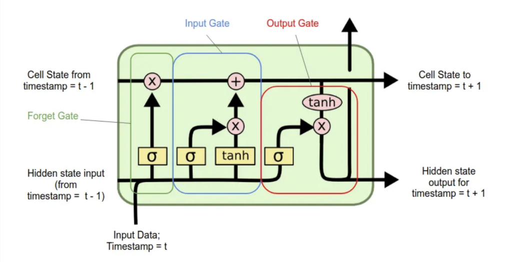


Figure 1: Standard LSTM Cell

Standard Forward Pass

Cell:	$c_t = f_t\, c_{t-1} + i_t\, z_t$	
Hidden:	$h_t = o_t \cdot \tilde{h}_t,$	$\tilde{h}_t = \tanh(c_t)$
Cell Input:	$z_t = \tanh(\tilde{z}_t),$	$\tilde{z}_t = \mathbf{w}_z^\top \mathbf{x}_t + r_z h_{t-1} + b_z$
Input Gate:	$i_t = \sigma(\tilde{i}_t),$	$\tilde{i}_t = \mathbf{w}_i^\top \mathbf{x}_t + r_i h_{t-1} + b_i$
Forget Gate:	$f_t = \sigma(\tilde{f}_t),$	$\tilde{f}_t = \mathbf{w}_f^\top \mathbf{x}_t + r_f h_{t-1} + b_f$
Output Gate:	$o_t = \sigma(\tilde{o}_t),$	$\tilde{o}_t = \mathbf{w}_o^\top \mathbf{x}_t + r_o h_{t-1} + b_o$

Limitation: h_{t-1} dependency prevents parallelization.

Image Credit to: https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

The Shift: From Recurrence to Attention

Limitations of LSTMs vs. The Rise of Attention

Traditional LSTM Constraints

- **Scalar Compression:** Information is forced into a fixed-size vector; $O(d)$ capacity.
- **Saturated Gating:** Sigmoid gates $(0 \to 1)$ inhibit significant memory revision.
- **Sequential Flow:** $O(N)$ dependency prevents GPU parallelization.
- **Retrieval Failure:** Struggles with exact-match "Nearest Neighbor Search."

The Transformer Paradigm

- **Memory as Indexing:** Stores history as Key-Value pairs; no initial compression.
- **Dynamic Routing:** Softmax attention allows "sharp" focus on any token.
- **Full Parallelism:** All tokens interact simultaneously during training.
- **Global Context:** Direct point-to-point connections regardless of distance.

Limitation: Inability to Revise Storage Decisions

The Sigmoid Bottleneck in LSTM Memory Management

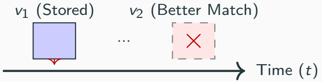
The Core Issue

LSTMs struggle to **overwrite** or update stored values when more relevant information appears later in a sequence.

Case Study: Nearest Neighbor Search

- **Initial Storage:** The model encounters vector v_1 (similar to a reference). The input gate stores it in the cell state.
- **New Information:** A "more similar" vector v_2 appears later in the sequence.
- **Revision Failure:** Due to the **squashing effect** of Sigmoid gates, the LSTM cannot significantly "suppress" v_1 to prioritize v_2.

Nearest Neighbor Search Failure



The cell state is "saturated" by v_1. The model lacks the dynamic range to fully "switch" to v_2.

Limitations of the LSTM

Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key

Value

Prediction

Limitations of the LSTM

Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**
Value	12
Prediction	12

Limitations of the LSTM

Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**	**1.5**
Value	12	17
Prediction	12	12

Limitations of the LSTM

Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**	**1.5**	**4**
Value	12	17	14
Prediction	12	12	14

Limitations of the LSTM

Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**	**1.5**	**4**	**5.5**
Value	12	17	14	8
Prediction	12	12	14	8

Limitations of the LSTM

Nearest Neighbor Search

Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**	**1.5**	**4**	**5.5**	**9**
Value	12	17	14	8	11
Prediction	12	12	14	8	8

Limitations of the LSTM

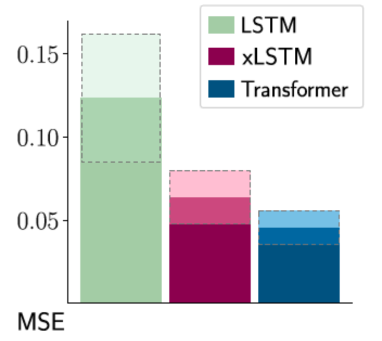
Nearest Neighbor Search Problem:

- **Goal:** "Predict the value of the closest key to the query"

Query: 6

Key	**2**	**1.5**	**4**	**5.5**	**9**
Value	12	17	14	8	11
Prediction	12	12	14	8	8

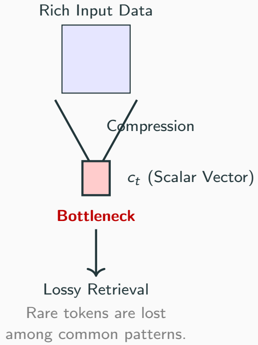
Figure 2: Mean Squared Error



Limitation: Limited Storage Capacity

The Compression Bottleneck of Scalar Cell States

Scalar vs. Matrix Capacity



The Core Issue

Information must be compressed into **scalar cell states** ($c_t \in \mathbb{R}$), which severely limits the fidelity of the stored memory.

Consequence: Rare Token Prediction

- **High Compression:** Multi-dimensional features are forced into a single vector space, leading to "cluttered" memory.
- **Information Loss:** Specific details of less frequent tokens are "washed out" by dominant statistical patterns.
- **Failure:** The model cannot distinguish or recall specific, rare information because the storage capacity ($O(d)$) is insufficient.

Limitation: Lack of Parallelizability

The Sequential Bottleneck of Memory Mixing

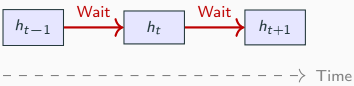
The Core Issue

LSTMs rely on **Memory Mixing** (hidden-to-hidden connections), where the current state h_t strictly depends on the previous state h_{t-1}.

Impact: Sequential Training

- **Recurrence Constraint:** Temporal dependencies force the model to process tokens one-by-one ($O(N)$ complexity).
- **Hardware Inefficiency:** Unlike Transformers, it is impossible to parallelize training across the time dimension.
- **Result:** Significantly slower training speeds on modern GPUs, creating a massive "compute gap" compared to attention-based models.

The Sequential Chain ($O(N)$)



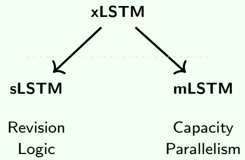
No Parallelism:
Each step must finish before the next begins.

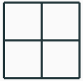
Introducing xLSTM: Extended LSTM

From LSTM to xLSTM: Three Pillars

- **sLSTM (Scalar LSTM):** Introduces **Exponential Gating** to allow high-dynamic memory revision (replaces Sigmoid).
- **mLSTM (Matrix LSTM):** Replaces scalar cells with **Matrix Memory** ($O(d^2)$), drastically increasing storage capacity.
- **Parallelization:** By removing memory mixing in mLSTM, the model enables **Parallel Scan** for hardware efficiency.

The xLSTM Family





Matrix Memory ($d \times d$)

The Result: xLSTM scales like a Transformer while maintaining the constant inference memory of an LSTM.

sLSTM: Re-designing the LSTM Forward Pass

Standard LSTM

$$c_t = f_t \, c_{t-1} + i_t \, z_t \tag{1}$$

$$h_t = o_t \cdot \tilde{h}_t, \; \tilde{h}_t = \tanh(c_t) \tag{2}$$

$$z_t = \tanh(\tilde{z}_t), \;\; \tilde{z}_t = \mathbf{w}_z^T \mathbf{x}_t + r_z h_{t-1} + b_z \tag{3}$$

$$i_t = \sigma(\tilde{i}_t), \;\; \tilde{i}_t = \mathbf{w}_i^T \mathbf{x}_t + r_i h_{t-1} + b_i \tag{4}$$

$$f_t = \sigma(\tilde{f}_t), \;\; \tilde{f}_t = \mathbf{w}_f^T \mathbf{x}_t + r_f h_{t-1} + b_f \tag{5}$$

$$o_t = \sigma(\tilde{o}_t), \;\; \tilde{o}_t = \mathbf{w}_o^T \mathbf{x}_t + r_o h_{t-1} + b_o \tag{6}$$

Limitation: Sigmoid $\in [0, 1]$ cannot amplify signals. tanh limits the dynamic range of memory.

sLSTM (Proposed)

$$c_t = f_t \, c_{t-1} + i_t \, z_t \tag{7}$$

$$\mathbf{n_t} = f_t \, n_{t-1} + i_t \tag{8}$$

$$h_t = o_t \cdot \tilde{h}_t, \; \tilde{h}_t = c_t / n_t \tag{9}$$

$$z_t = \tanh(\tilde{z}_t), \;\; \tilde{z}_t = \mathbf{w}_z^T \mathbf{x}_t + r_z h_{t-1} + b_z \tag{10}$$

$$i_t = \exp(\tilde{i}_t), \;\; \tilde{i}_t = \mathbf{w}_i^T \mathbf{x}_t + r_i h_{t-1} + b_i \tag{11}$$

$$f_t = \exp(\tilde{f}_t) \text{ OR } \sigma(\tilde{f}_t), \;\; \tilde{f}_t = \mathbf{w}_f^T \mathbf{x}_t + r_f h_{t-1} + b_f \tag{12}$$

$$o_t = \sigma(\tilde{o}_t), \;\; \tilde{o}_t = \mathbf{w}_o^T \mathbf{x}_t + r_o h_{t-1} + b_o \tag{13}$$

Innovation: exp allows **amplification**. Normalizer n_t ensures stability via **division**.

sLSTM: Numerical Stability via Stabilizer m_t

Problem: $\exp(\cdot)$ gates $\implies$ c_t, n_t grow exponentially $\implies$ **Overflow**.

The Solution: m_t

Track the **running maximum** of log-gates:

$$m_t = \max(\tilde{f}_t + m_{t-1}, \tilde{i}_t)$$

(Similar to Log-Sum-Exp trick)

$\implies$

Stable Implementation

Rescale gates by m_t:

$$i'_t = \exp(\tilde{i}_t - m_t)$$
$$f'_t = \exp(\tilde{f}_t + m_{t-1} - m_t)$$

Updated States:

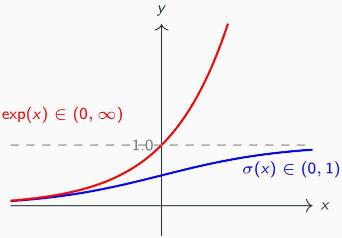
$$c_t = f'_t c_{t-1} + i'_t z_t$$
$$n_t = f'_t n_{t-1} + i'_t$$

Final Output: $h_t = o_t \cdot \frac{c_t}{n_t}$

(Division by n_t normalizes h_t, keeping it bounded.)

sLSTM Innovation: Sigmoid vs. Exponential Gating

Function Comparison



- **Sigmoid**: Squashing mechanism.
- **Exponential**: Amplification mechanism.

Why Switch to Exp?

Gating	Mathematical Logic
Traditional (σ)	Cannot increase state magnitude beyond the previous step.
sLSTM (exp)	Enables **Signal Amplification**, allowing the model to "revise" history.
Stability	Handled by m_t and n_t to prevent the "Exploding Exp" problem.

Key Takeaway: Exponential gating transforms the LSTM from a simple "forget/remember" unit into a powerful "search/update" mechanism.

sLSTM: New Memory Mixing & Headwise Architecture

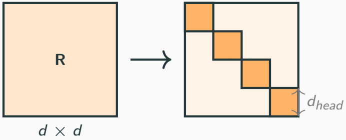
From Scalar to Vector Pre-activations

1. Standard Scalar Recurrence:
$\tilde{z}_t = w_z^\top \mathbf{x}_t + r_z h_{t-1} + b_z$ (Classic mixing)

2. New Vectorized Formulation:
$\tilde{\mathbf{z}}_t = \mathbf{W}_z \mathbf{x}_t + \mathbf{R}_z \mathbf{h}_{t-1} + \mathbf{b}_z$
$\tilde{\mathbf{i}}_t, \tilde{\mathbf{f}}_t, \tilde{\mathbf{o}}_t$ follow the same vector pattern.

- **Memory Mixing**: The $\mathbf{R}_z \mathbf{h}_{t-1}$ term allows the model to correlate all features of the previous state.
- **High Dynamic Range**: Combined with exponential gating, this enables "searching" and "updating" discrete states.

Block-diagonal Weights



Headwise Memory Mixing:
To balance complexity, **R** is restricted to a **block-diagonal structure**.

- Prevents full d^2 compute.
- Each head performs independent internal mixing.

Note: m_t and n_t stability logic still applies to the resulting vector activations to prevent exploding exponents.

mLSTM: Extending to Matrix Memory

sLSTM (Scalar Memory)

$$\text{Cell:}\quad c_t = f_t\, c_{t-1} + i_t\, z_t \tag{7}$$

$$\text{Normalizer:}\quad n_t = f_t\, n_{t-1} + i_t \tag{8}$$

$$\text{Hidden:}\quad h_t = o_t \cdot (\, c_t \,/\, n_t \,) \tag{9}$$

$$\text{Cell input:}\quad z_t = \tanh(\mathbf{w}_z^\top \mathbf{x}_t + \dots) \tag{10}$$

$$\text{Gating:}\quad i_t\,,\, f_t = \exp(\dots) \tag{11-12}$$

$$\text{Output:}\quad o_t = \sigma(\dots) \tag{13}$$

Constraint: Scalar memory has limited capacity. Sequential h_{t-1} prevents parallelization.

mLSTM (Matrix Memory)

$$\mathbf{C}_t = f_t\, \mathbf{C}_{t-1} + i_t\, \mathbf{v}_t \mathbf{k}_t^\top \tag{14}$$

$$\mathbf{n}_t = f_t\, \mathbf{n}_{t-1} + i_t\, \mathbf{k}_t \tag{15}$$

$$\mathbf{h}_t = o_t \odot \tilde{\mathbf{h}}_t,\quad \tilde{\mathbf{h}}_t = \mathbf{C}_t \mathbf{q}_t \,/\, \max\{|\, \mathbf{n}_t^\top \mathbf{q}_t \,|, 1\} \tag{16}$$

$$\mathbf{q_t}, \mathbf{k_t}, \mathbf{v_t} = \mathbf{W}_{q,k,v} \mathbf{x}_t + \mathbf{b}_{q,k,v} \tag{17}$$

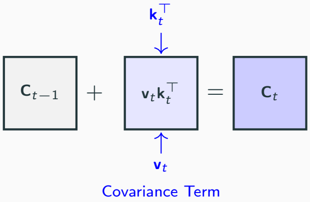
$$i_t\,,\, f_t = \exp(\dots),\quad \tilde{i}_t, \tilde{f}_t = \mathbf{w}_{i,f}^T \mathbf{x}_t + b_{i,f} \tag{18}$$

$$o_t = \sigma(\tilde{\mathbf{o}}_t), \tilde{\mathbf{o}}_t = \mathbf{W}_o \mathbf{x}_t + \mathbf{b}_o) \tag{19}$$

Innovation: Matrix memory $C_t \in \mathbb{R}^{d \times d}$ stores key-value pairs. **Removed hidden-to-hidden recurrent connection** (h_{t-1} to gates) to enable **Parallel Scan**.

mLSTM: The Matrix Memory & Covariance Update

Matrix Update Visualization



- **Capacity**: Compresses history into $\mathbb{R}^{d\times d}$ matrix.
- **Retrieval**: Query $\mathbf{q}_t$ performs a linear projection on $\mathbf{C}_t$.

The Covariance Update Rule

$$\textbf{Update:} \quad \mathbf{C}_t = f_t \, \mathbf{C}_{t-1} + i_t \, \mathbf{v}_t\mathbf{k}_t^\top \tag{14}$$

$$\textbf{Normalizer:} \quad n_t = f_t \, n_{t-1} + i_t \, k_t \tag{15}$$

$E[\mathbf{k}_t], E[\mathbf{v}_t] \approx 0$ (via LayerNorm applied before projecting inputs), $\mathbf{C}_t$ effectively tracks the **Covariance** $E[\mathbf{v}\mathbf{k}^\top]$ between values and keys.

Efficiency: Fixed-size $\mathbb{R}^{d\times d}$ memory $\implies$ $O(1)$ inference memory, unlike Transformers' $O(L)$ KV cache.

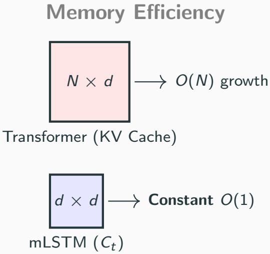
Comparison: Standard LSTM vs. sLSTM vs. mLSTM

Feature	Standard LSTM	sLSTM	mLSTM
Memory	Scalar c_t	Scalar c_t	**Matrix** $\mathbf{C}_t \in \mathbb{R}^{d \times d}$
Gating	Sigmoid (σ)	**Exponential** (exp)	**Exponential** (exp)
Stability	None (Bounded σ)	**Normalizer** n_t, m_t	**Normalizer** $\mathbf{n}_t$
Mixing	Hidden Mixing (h_{t-1})	Hidden Mixing (h_{t-1})	**No Mixing**
Computation	Sequential	Sequential	**Parallelizable**
Inference	$O(1)$ per step	$O(1)$ per step	$O(1)$ per step

Summary of Evolution:

- **Standard** $\rightarrow$ **sLSTM**: Switched to exponential gating for better signal amplification, adding a normalizer for stability.
- **sLSTM** $\rightarrow$ **mLSTM**: Replaced scalar memory with a covariance-based matrix memory and removed hidden-to-hidden dependencies to enable parallel training.

Comparison: mLSTM vs. Transformer



- **Transformer**: Memory scales linearly with sequence length N. Struggles with long sequences.
- **mLSTM**: Compressed history into fixed $d \times d$ matrix. Constant memory regardless of N.

Softmax vs. Linear Attention

Mechanism	Search vs. Summary
Softmax Attn (Transformer)	**Global Search**: Sharp focus on specific tokens via Softmax($\mathbf{QK}^\top$). Precise but $O(N)$ search.
Linear Attn (mLSTM)	**State Tracking**: Updates $\mathbf{C}_t = f_t\mathbf{C}_{t-1} + i_t\mathbf{v}_t\mathbf{k}_t^\top$. Build summary via exp-gating.
Retrieval	**Transformer**: $O(N)$ needle-in-haystack. **mLSTM**: $O(1)$ pattern tracking.

Core Logic: While Transformers "look back" at raw data, mLSTM maintains a **compressed mental model**, using f_t and i_t to weigh new vs. old correlations.

xLSTM: Residual Block Architecture

- **Goal**: "non-linearly summarize the past in a high-dimensional space to better separate different histories or contexts."
- **The Backbone:** *Pre-LayerNorm Residual*, found in Transformers and SSMs.
- **Anatomy of a Block:** Each residual block ($x_{l+1} = x_l + \text{Block}(x_l)$) contains:
    1. **Layer Normalization:** Applied to input (Pre-LN).
    2. **xLSTM Core:** Either an `sLSTM` or `mLSTM` module.
    3. **Projections:** Up/Down projections integrated directly.
    4. **Residual Connection:** Stabilizes gradient flow.
- **Unified Design:** Enables mixing `sLSTM` and `mLSTM` in the same stack.

xLSTM: Block Variants and Projection Strategies

`sLSTM` **Block**: Post Up-Projection, similar to Transformer.

- **Rationale:** Tracks state logic in lower dimensions before capacity-heavy non-linearities.

`mLSTM` **Block**: Pre Up-Projection, similar to SSM/Mamba.

- **Rationale:** High-dim inputs increase key-value retrieval capacity in matrix memory.

Figure 3: `sLSTM` Block

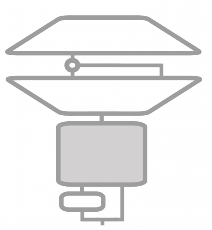
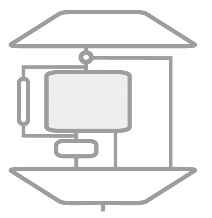


Figure 4: `mLSTM` Block



xLSTM: Stacking Strategy and `xLSTM[a:b]` Notation

Architecture: Stacking blocks, balancing *Parallel Capacity* with *Sequential Reasoning*.

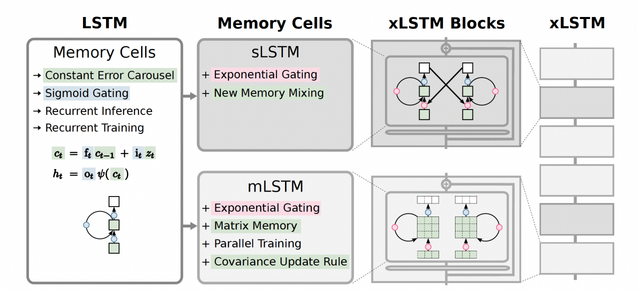


Figure 5: xLSTM Architecture

Notation `xLSTM[a:b]`: Defines the ratio of block types in the residual stack. Number of `mLSTM` blocks **a** vs `sLSTM` blocks **b**

xLSTM: Memory & Complexity

- **Memory:**
  - `mLSTM` **Block**: Massive Matrix Memory $O(d^2)$ capacity, highly parallelizable.
  - `sLSTM` **Block**: Sequential and slow, but offers memory mixing.
- **Algorithmic Complexity:**
  - **Linear Scaling:** $O(N)$ with sequence length (vs. Transformer $O(N^2)$).
  - **Constant Memory:** State is compressed into fixed-size matrices regardless of context length.

$$\text{Cache Size} = O(1) \quad \text{(No growing KV-cache)}$$

Experiments (What they validate)

- Goal: validate two core claims of **xLSTM**:
  1. **State tracking** limitations of LSTMs are fixed via **sLSTM**
  2. **Storage capacity** limitations of LSTMs are fixed via **mLSTM**
- Plus: demonstrate effective **scaling to LLMs** (quality, extrapolation, efficiency)
- Experiments
  1. Synthetic Tasks: Formal Languages & State Tracking
  2. Multi-Query Associative Recall(MQAR) & Nearest Neighbor Search
  3. Large-Scale Language Modeling
  4. Scaling Laws
  5. Performance & Throughput Analysis
- Baselines: LLaMa(Transformer), Mamba(SSM), RWKV(RNN)

Experiment 1. Synthetic Tasks: Formal Languages & State Tracking

Setup

- Tasks from Chomsky hierarchy: Regular, Context-Free, Context-Sensitive
- Logic/state task example: **Parity** (even/odd sum over a sequence)

Measures

- Ability to **track discrete state** over long horizons
- Requires strong recurrent updates and **memory mixing** (state interactions)

Experiment 1. Synthetic Tasks: Formal Languages & State Tracking

	Context Sensitive		Deterministic Context Free		Regular					
	Bucket Sort	Missing Duplicate	Mod Arithmetic (w Brackets)	Solve Equation	Cycle Nav	Even Pairs	Mod Arithmetic (w/o Brackets)	Parity	Majority	Majority Count
Llama	0.92 ± 0.02	0.08 ± 0.0	0.02 ± 0.0	0.02 ± 0.0	0.04 ± 0.01	1.0 ± 0.0	0.03 ± 0.0	0.03 ± 0.01	0.37 ± 0.01	0.13 ± 0.0
Mamba	0.69 ± 0.0	0.15 ± 0.0	0.04 ± 0.01	0.05 ± 0.02	0.86 ± 0.04	1.0 ± 0.0	0.05 ± 0.02	0.13 ± 0.02	0.69 ± 0.01	0.45 ± 0.03
Retention	0.13 ± 0.01	0.03 ± 0.0	0.03 ± 0.0	0.03 ± 0.0	0.05 ± 0.01	0.51 ± 0.07	0.04 ± 0.0	0.05 ± 0.01	0.36 ± 0.0	0.12 ± 0.01
Hyena	0.3 ± 0.02	0.06 ± 0.02	0.05 ± 0.0	0.02 ± 0.0	0.06 ± 0.01	0.93 ± 0.07	0.04 ± 0.0	0.04 ± 0.0	0.36 ± 0.01	0.18 ± 0.02
RWKV-4	0.54 ± 0.0	0.21 ± 0.01	0.06 ± 0.0	0.07 ± 0.0	0.13 ± 0.0	1.0 ± 0.0	0.07 ± 0.0	0.06 ± 0.0	0.63 ± 0.0	0.13 ± 0.0
RWKV-5	0.49 ± 0.04	0.15 ± 0.01	0.08 ± 0.0	0.08 ± 0.0	0.26 ± 0.05	1.0 ± 0.0	0.15 ± 0.02	0.06 ± 0.03	0.73 ± 0.01	0.34 ± 0.03
RWKV-6	0.96 ± 0.0	0.23 ± 0.06	0.09 ± 0.01	0.09 ± 0.02	0.31 ± 0.14	1.0 ± 0.0	0.16 ± 0.0	0.22 ± 0.12	0.76 ± 0.01	0.24 ± 0.01
LSTM (Block)	0.99 ± 0.0	0.15 ± 0.0	0.76 ± 0.0	0.5 ± 0.05	0.97 ± 0.03	1.0 ± 0.0	0.91 ± 0.09	1.0 ± 0.0	0.58 ± 0.02	0.27 ± 0.0
LSTM	0.94 ± 0.01	0.2 ± 0.0	0.72 ± 0.04	0.38 ± 0.05	0.93 ± 0.07	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.82 ± 0.02	0.33 ± 0.0
xLSTM[0:1]	0.84 ± 0.08	0.23 ± 0.01	0.57 ± 0.09	0.55 ± 0.09	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.75 ± 0.02	0.22 ± 0.0
xLSTM[1:0]	0.97 ± 0.0	0.33 ± 0.22	0.03 ± 0.0	0.03 ± 0.01	0.86 ± 0.01	1.0 ± 0.0	0.04 ± 0.0	0.04 ± 0.01	0.74 ± 0.01	0.46 ± 0.0
xLSTM[1:1]	0.7 ± 0.21	0.2 ± 0.01	0.15 ± 0.06	0.24 ± 0.04	0.8 ± 0.03	1.0 ± 0.0	0.6 ± 0.4	1.0 ± 0.0	0.64 ± 0.04	0.5 ± 0.0

Figure 6: Test of xLSTM's exponential gating with memory mixing

Experiment 1. Synthetic Tasks: Formal Languages & State Tracking

Results

- Strongly outperforms **Transformers** and **SSMs** (e.g., Mamba)
- Transformers/Mamba often fail on hard tracking (e.g., Parity: accuracy < 0.5)

Significance

- sLSTM (scalar LSTM + exponential gating) excels at **discrete state tracking/logic**
- Highlights a weakness of many modern "linear" sequence models on stateful tasks

Experiment 2. MQAR & Nearest Neighbor Search (Associative Recall)

Setup

- “Needle-in-a-haystack”: store many **key-value** pairs in a sequence
- Later: retrieve correct value for a queried key (multi-query)

Measures

- **Associative memory capacity**
- Ability to **revise** stored info (update value when better evidence arrives)

Experiment 2. MQAR & Nearest Neighbor Search (Associative Recall)

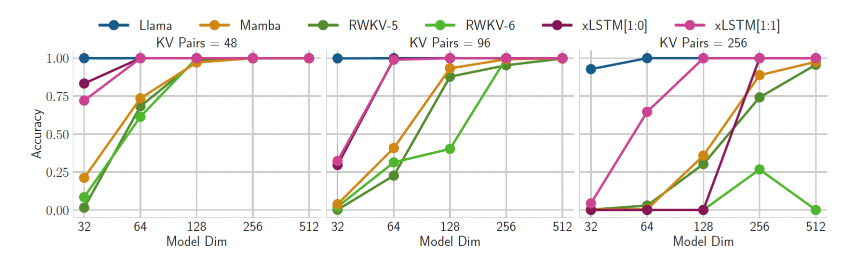


Figure 7: Test of memory capacities of different models at the Multi-Query Associative Recall task with context length 2048.

Experiment 2. MQAR & Nearest Neighbor Search (Associative Recall)

Results

- **mLSTM** performs comparably to **Transformers**
- Outperforms traditional LSTMs and some SSM baselines
- Matrix Memory stores far more information than scalar LSTM memory

Significance

- Validates that **Matrix Memory** fixes LSTM **storage capacity** limits
- mLSTM behaves like a **recurrent key-value mechanism**

Experiment 3. Large-Scale Language Modeling (SlimPajama)

Setup

- Train on **SlimPajama** (cleaned RedPajama; 300B tokens)
- Model sizes: **125M** up to **1.3B** parameters

Measures

- **Validation perplexity** for next token prediction and on downstream tasks that measure common sense reasoning.

Experiment 3. Large-Scale Language Modeling (SlimPajama)

	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

Figure 8: Validation set perplexity at next token prediction and on downstream tasks

Experiment 3. Large-Scale Language Modeling (SlimPajama)

Results

- xLSTM achieves **lower perplexity** than **Mamba** and **RWKV** across sizes
- Competitive with **LLaMA** (Transformer), matching or slightly beating it

Significance

- Core “LLM-era” evidence: xLSTM **scales effectively**
- Unlike classic LSTMs that saturate, xLSTM shows robust scaling behavior

Experiment 4. Scaling Laws

Setup

- Train xLSTM models from **125M to 1.3B parameters**
- Dataset: **SlimPajama** (300B tokens)
- Baselines: **Mamba**, **RWKV**, **LLaMA (Transformer)**

Measures

- **Validation Perplexity** (next-token prediction quality) as model size and compute increases

Experiment 4. Scaling Laws

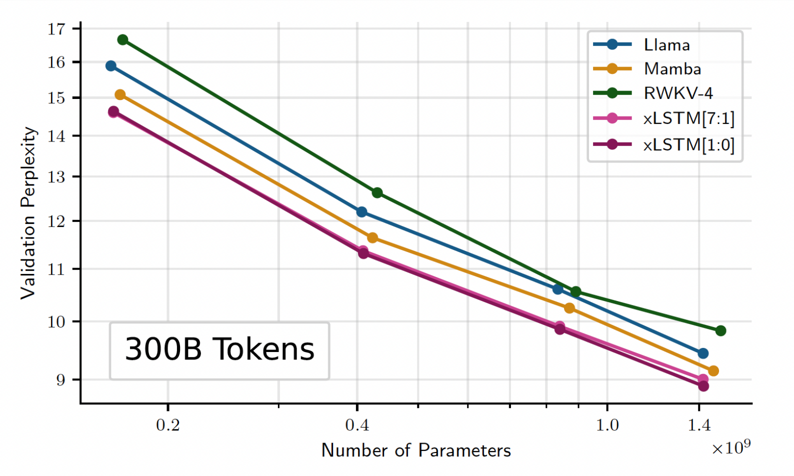


Figure 9: Scaling laws. Next token prediction perplexity on the SlimPajama validation set

Experiment 4. Scaling Laws: Measures & Goal

Results

- xLSTM outperforms **Mamba** and **RWKV** at all tested scales
- Lies on the **Pareto frontier** (best quality for a given compute budget)

Significance

- Shows xLSTM **does not saturate** like traditional LSTMs
- Demonstrates **Transformer-level scaling behavior**
- Makes xLSTM viable for large-scale LLM architecture

Experiment 5. Performance & Throughput Analysis

Setup

- Compare inference speed latency and throughput

Measures

- **Computational efficiency** and practicality at scale

Experiment 5. Performance & Throughput Analysis

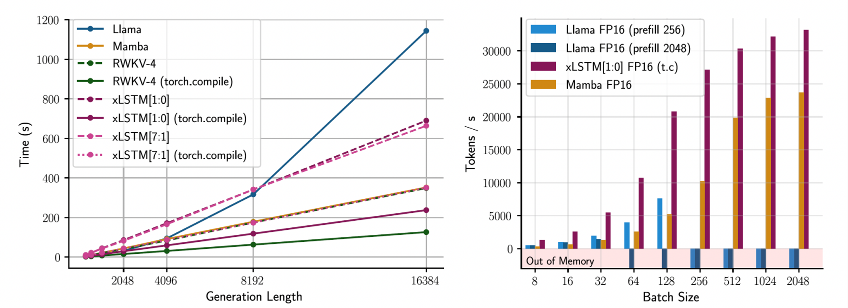


Figure 10: Inference Generative Speed. **Left**: Generation times, **Right**: Token throughput

Experiment 5. Performance & Throughput Analysis

Inference

- mLSTM: **linear generation time** $O(N)$
- No growing KV cache with sequence length $\Rightarrow$ higher throughput/batch sizes

Significance

- Efficient constant-memory inference

Overall Conclusion from Section 4

- **sLSTM**: strong discrete **state tracking** and logic capability
- **mLSTM**: high **associative memory capacity** (Transformer-like retrieval)
- xLSTM: competitive **LLM perplexity**, adhering to the **Scaling Law**, and strong **throughput**

Limitations and Conclusion

Current Limitations

Parallelizability **sLSTM** retains hidden-to-hidden recurrent connections, preventing a fully parallel implementation. Even with CUDA optimizations, it remains <**2x slower** than mLSTM.

CUDA Efficiency **mLSTM** suffers from unoptimized CUDA kernels, making it **4x slower** than *FlashAttention* implementations currently.

Memory Cap While Matrix Memory (d^2) is independent of sequence length (N), extreme increases in N may eventually overload memory. *Note: Not an issue for contexts up to 16k tokens.*

Research Conclusion

Can xLSTM overcome the limitations of standard LSTMs?

"At least as far as current technologies like Transformers or State Space Models."

Future (Current) Work

- **xLSTM Scaling Laws: Competitive Performance with Linear Time-Complexity** *(October 2025)*
  Shows that xLSTM scales effectively to large models while maintaining linear time complexity, offering a more efficient alternative to Transformers for large-scale training.
- **xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference** *(March 2025)*
  This work releases and evaluates a 7-billion parameter xLSTM model, demonstrating it matches the performance of leading Transformer LLMs (like Llama) of the same size.
- **Vision-LSTM: xLSTM as Generic Vision Backbone** *(June 2024)*
  xLSTM architecture for computer vision tasks, introducing a "Vision-LSTM" (ViL) backbone that processes image patches as sequences.

Our Review

- **Strengths**
  - Clear Problem Description and Design Rationale
  - Comprehensive Experiments
  - Competitive Large-Scale Results
  - **Overall** the paper delivers what authors promise
- **Ideas for Future Improvements**
  - Training efficiency and lack of parallelizability
  - Matrix Memory Computational Cost Underexplored
    - $O(d^2)$ matrix operations per timestep, FLOPs/token experiments
  - Memory Saturation Risk Not Fully Addressed
    - Authors acknowledge that matrix memory may saturate as sequence length grows but only test up to 16k context. Does this scale with the number of parameters/compute?
  - model is relatively small (from 250M to 1.3B). Not sure the performance once the model grows big (like 70B+)

Thank you!

Questions?

References

[1] **Beck, M., et al. (2024).** *xLSTM: Extended Long Short-Term Memory.* arXiv preprint arXiv:2405.04517.

[2] **Hochreiter, S., & Schmidhuber, J. (1997).** *Long Short-Term Memory.* Neural Computation, 9(8), 1735-1780.

[3] **Vaswani, A., et al. (2017).** *Attention Is All You Need.* Advances in Neural Information Processing Systems (NeurIPS).

[4] **Gu, A., & Dao, T. (2023).** *Mamba: Linear-Time Sequence Modeling with Selective State Spaces.* arXiv preprint arXiv:2312.00752.

[5] **Katharopoulos, A., et al. (2020).** *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention.* ICML.

Note: Full bibliography and additional technical details can be found in the xLSTM technical report (Beck et al., 2024).

Appendix A: Mixing Mechanisms

sLSTM: Memory Mixing

- Good for State Tracking.
- Past info mixed into current gates:

$$z_t = w_z^T x_t + \mathbf{r_z h_{t-1}} + b_z$$

- $\mathbf{r_z h_{t-1}}$: Learned weight matrix mixing past history.

mLSTM: No Mixing

- Good for Capacity & Speed.
- Gate calculation independent of previous hidden state:

$$i_t = w_i^T x_t + b_i$$

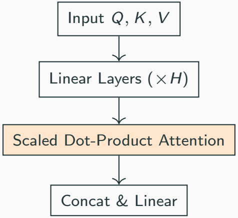
- **No** $r_i h_{t-1}$ term.

Appendix B: mLSTM Structure: Gates & Transformer Analogies

Component	mLSTM Formulation	Transformer Analogy
Key ($\mathbf{k}_t$)	$\frac{1}{\sqrt{d}}\mathbf{W}_k\mathbf{x}_t + \mathbf{b}_k$	Memory Address / Routing
Query ($\mathbf{q}_t$)	$\mathbf{W}_q\mathbf{x}_t + \mathbf{b}_q$	Retrieval Signal
Value ($\mathbf{v}_t$)	$\mathbf{W}_v\mathbf{x}_t + \mathbf{b}_v$	Content Information
Gate Type	**Activation**	**Function / Effect**
Input Gate (i_t)	$\exp(\tilde{i}_t)$	"Write" to Matrix Memory
Forget Gate (f_t)	$\exp(\tilde{f}_t)$	Decay old correlation

Linear Attention Connection: In mLSTM, the exponential gates act as a dynamic normalization mechanism. When the forget gate $f_t = 1$ and the input gate is active, the matrix memory $\mathbf{C}_t = \sum i_\tau \mathbf{v}_\tau \mathbf{k}_\tau^\top$ mathematically mimics **Linear Attention**.

Appendix: Transformer Multi-Head Structure



Key Concept: Instead of a single attention function, we perform H **parallel projections**.

Benefit for sLSTM:

- Mimics Transformer's capacity.
- Stable training via head-wise normalization.
- Efficient GPU utilization.

Appendix: From Single-head to Multi-head sLSTM

Why Multi-head?

- **Feature Diversity:** Each head tracks different aspects of the sequence.
- **Parallelism:** Independent heads allow for efficient sub-dimension processing.
- **Scaling:** Similar to Transformer's MHSA, it enhances capacity.

Note: Total hidden dimension d is split into H heads, each with $d_h = d/H$.

Feature	**Standard LSTM**	**sLSTM (Multi-head**
Gates	Single i, f, o	$H \times (i, f, o)$
Cell State	Scalar/Vector c_t	Parallel heads $\{c_t^{(1)} \dots c$
Dependency	Sequential h_{t-1}	Head-wise parallel
Normalization	None	Exponential (exp) gat

Head Concatenation

$$h_t = \text{Concat}(h_t^{(1)}, h_t^{(2)}, \dots, h_t^{(H)})W^O$$