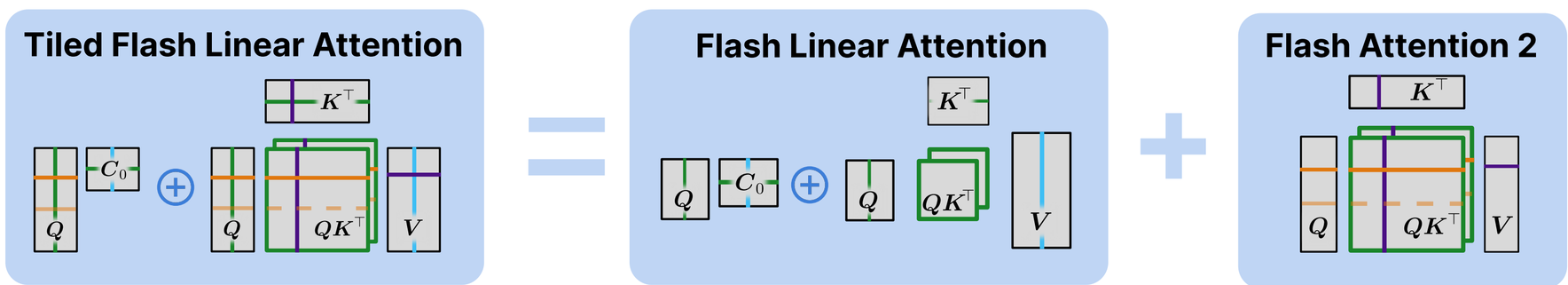
Tiled Flash Linear Attention

More Efficient Linear RNN and xLSTM Kernels

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

1) ELLIS Unit JKU Linz, Institute for Machine Learning 2) NXAI GmbH 3) Now at Google Deepmind

European Laboratory for Learning and Intelligent Systems



Paper

Code

TL;DR: We combine Flash Linear Attention with Flash Attention 2

Motivation

- Gated Linear RNNs become competitive on language modeling: RetNet, Mamba, GLA, xLSTM/mLSTM
- Linear RNNs have a chunkwise formulation which computes intermediate memory states and enables efficient implementations

➡ scales linearly with sequence length

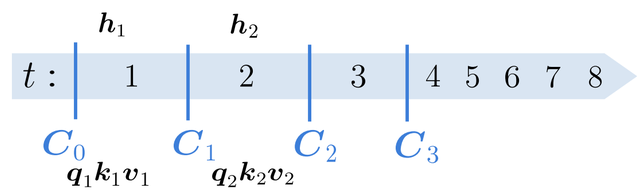
- FLA kernels leverage the chunkwise formulation and are faster than Flash Attention kernels
- **BUT: FLA is limited in the chunksize by available SRAM on the GPU**

➡ **Many memory states are materialized in memory, which causes high memory usage & IO and low arithmetic intensity**

Background: Linear RNN Formulations

Recurrent:

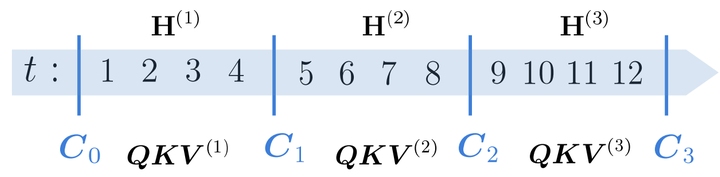
- C state in every time step
- Linear compute



Chunkwise-parallel:

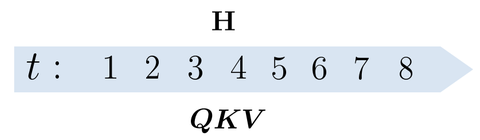
- C state every L-th time step
- Linear compute

➡ between recurrent and parallel



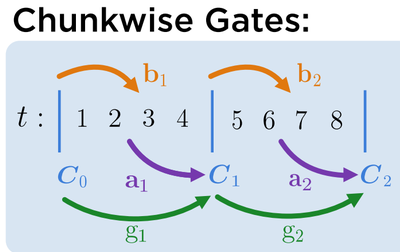
Parallel:

- No C states
- Quadratic compute!



Chunkwise-parallel formulation in 4 parts

Chunkwise Gates:



Inter-Chunk Recurrence:

$$C_k = g_k\, C_{k-1} + \left(a_k \odot K^{(k)}\right)^\top V^{(k)}$$
$$H^{(k)}_{\text{inter}} = \left(b_k \odot Q^{(k)}\right) C_{k-1}$$

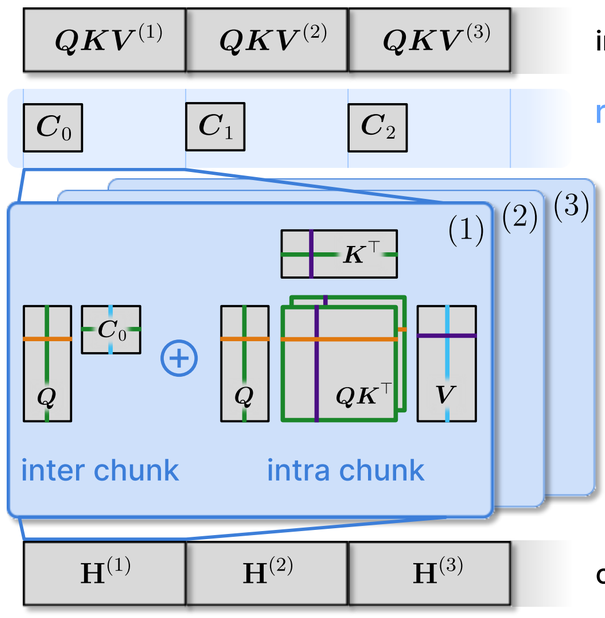
Intra-Chunk Parallel:

$$H^{(k)}_{\text{intra}} = \left(Q^{(k)}\, K^{(k)\top} \odot D^{(k)}\right) V^{(k)}$$

Output Combination:

$$H^{(k)} = H^{(k)}_{\text{inter}} + H^{(k)}_{\text{intra}}$$

TFLA Kernels Overview



- TFLA divides the inuts into chunks of size L (1st level of of sequence parallelism)
- The recurrent kernel materializes the (first) memory state for each chunk
- TFLA parallelizes along the chunk dimension L (2nd level of sequence paralellism)

➡ **Tiling along the chunk dimension L enables arbitrary large chunk sizes!**

TFLA Forward Pass Tiling

Simplified form of the parallel intra-chunk forward pass for chunk k:

$$\underset{(L_{hq}\times d_{hv})}{H^{(k)}} = \underset{(L_{hq}\times d_{qk})}{Q^{(k)}}\ \underset{(d_{qk}\times d_{hv})}{C_{k-1}} + \left(\underset{(L_{hq}\times d_{qk})}{Q^{(k)}}\ \underset{(d_{qk}\times L_{kv})}{K^{(k)\top}}\right)\ \underset{(L_{kv}\times d_{hv})}{V^{(k)}}$$

	chunk size dimensions		head dimensions	
We **parallelize along**	L_{hq}	hidden-query	d_{hv}	hidden-value
We **loop over**	L_{kv}	key-value	d_{qk}	query-key

We apply TFLA to 2 mLSTM variants

mLSTMexp

$$m_t = \max\left\{\log\sigma(\tilde{f}_t) + m_{t-1},\ \tilde{i}_t\right\}$$
$$C_t = f_t\, C_{t-1} + \exp(\tilde{i}_t - m_t)\, k_t\, v_t^\top$$
$$n_t = f_t\, n_{t-1} + \exp(\tilde{i}_t - m_t)\, k_t$$
$$\tilde{h}_t = \frac{C_t^\top q_t}{\max\left\{\left|n_t^\top q_t\right|, \exp(-m_t)\right\}}$$
$$h_t = o_t \odot \text{NORM}(\tilde{h}_t)$$

- **Exponential input gate**
- **Additional max and normalizer state** for stabilization
- Tested at 7B parameter scales

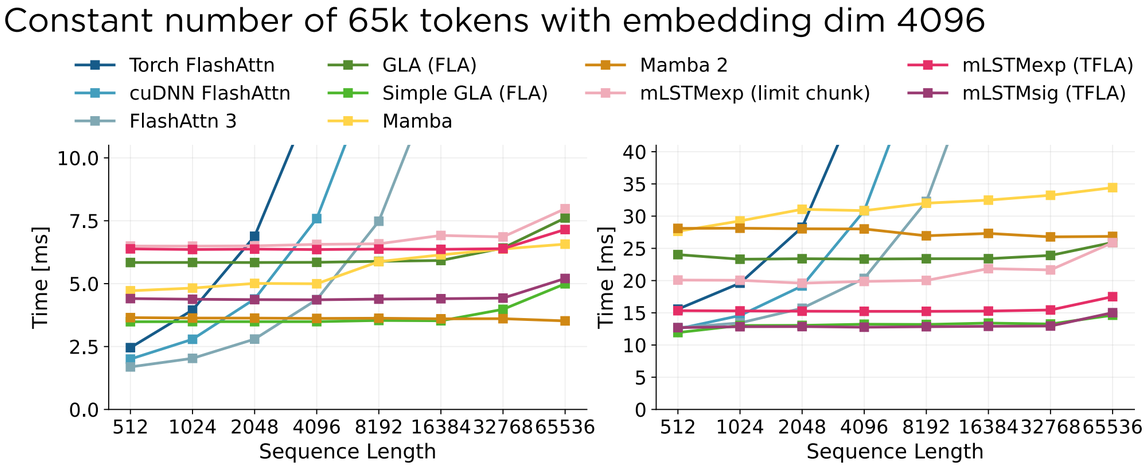
mLSTMsig

$$C_t = f_t\, C_{t-1} + \sigma(\tilde{i}_t)\, k_t\, v_t^\top$$
$$\tilde{h}_t = C_t^\top q_t$$
$$h_t = o_t \odot \text{NORM}(\tilde{h}_t)$$

- **Sigmoid input gate**
- **No max and normalizer state**
- Less FLOPS & faster kernels
- Equal performance up to 1.3B parameters

Benchmark Results

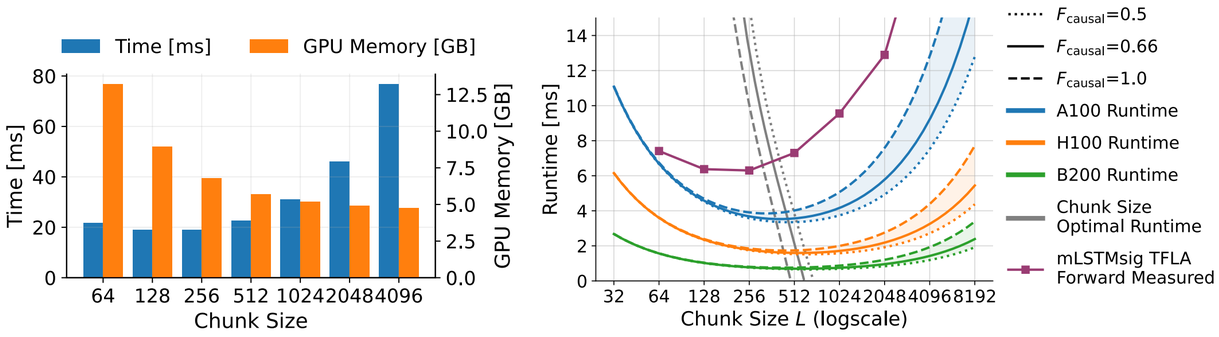
Constant number of 65k tokens with embedding dim 4096



Inference | Training

➡ **TFLA mLSTM kernels are faster than FA3 & 2x faster than Mamba 2**

Chunk size is more than a kernel parameter



Runtime vs. Memory Trade-off | **Theoretical Runtime**

The chunk size...

...enables trade-off between memory & runtime

...interpolates the FLOPs between recurrent and parallel formulation

...determines the arithmetic intensity (i.e. compute vs. memory bound)