Model Architecture for Scalability

CSE585 Advanced Scalable Systems for GenAI

Insu Jang, Mosharaf Chowdhury

Sep 5, 2024

SymbioticLab
UNIVERSITY OF MICHIGAN

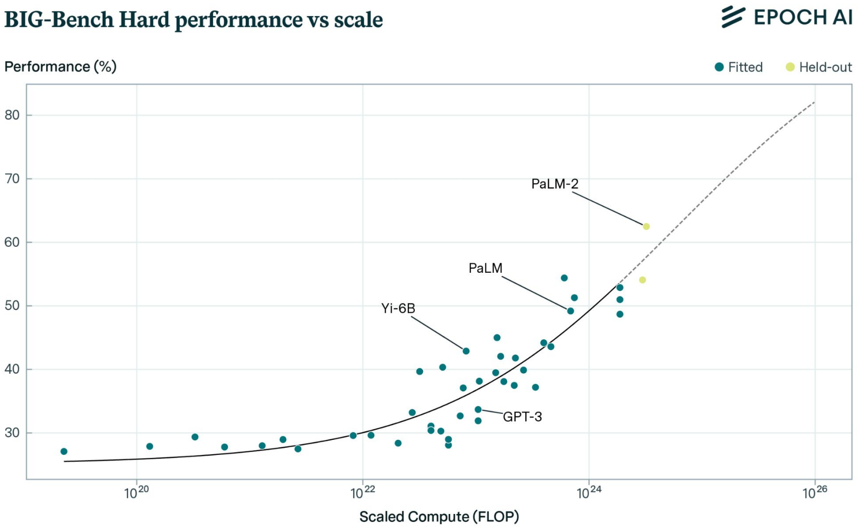
Agenda

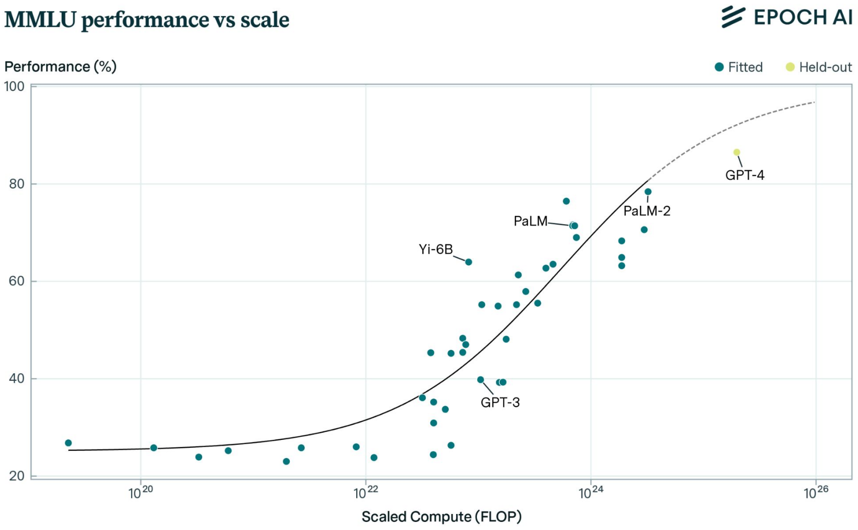
- Mixture of Experts (MoE)
- State Space Models (SSMs)

Mixture of Experts (MoE)

Increasing Model Size

- Model capacity (size) is critical for model performance
- More parameters → more computations → higher cost





https://epochai.org/blog/how-predictable-is-language-model-benchmark-performance

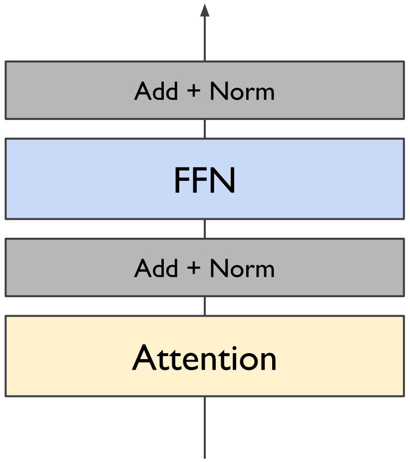
How Can Reduce Compute Cost?

- Traditionally, the network of the model is active for every example
    - However, all parts may not contribute equally

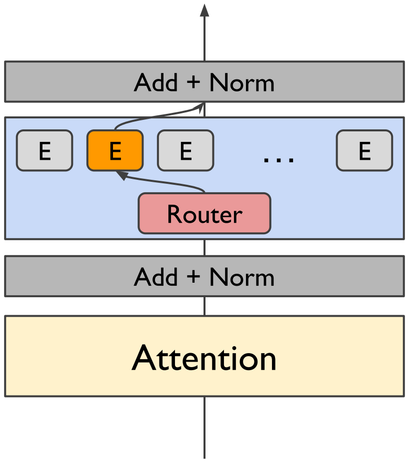
- Conditional Computation
    - Activate parts of the network on a per-example basis
        - Not all parts of the network are used for each example
    - In theory, this is expected to increase model capacity with the same number of parameters

How Can Reduce Compute Cost?

- Activated parameters become experts for data to be trained
- Model becomes a set (mixture) of experts



Transformer Layer



Transformer MoE Layer

Model is larger

But compute is the same

	# Params	FLOPS
T5-Base	0.2B	124B
Switch-Base (12 experts)	7B	124B
T5-Large	0.7B	425B
Switch-Large (24 experts)	26B	425B

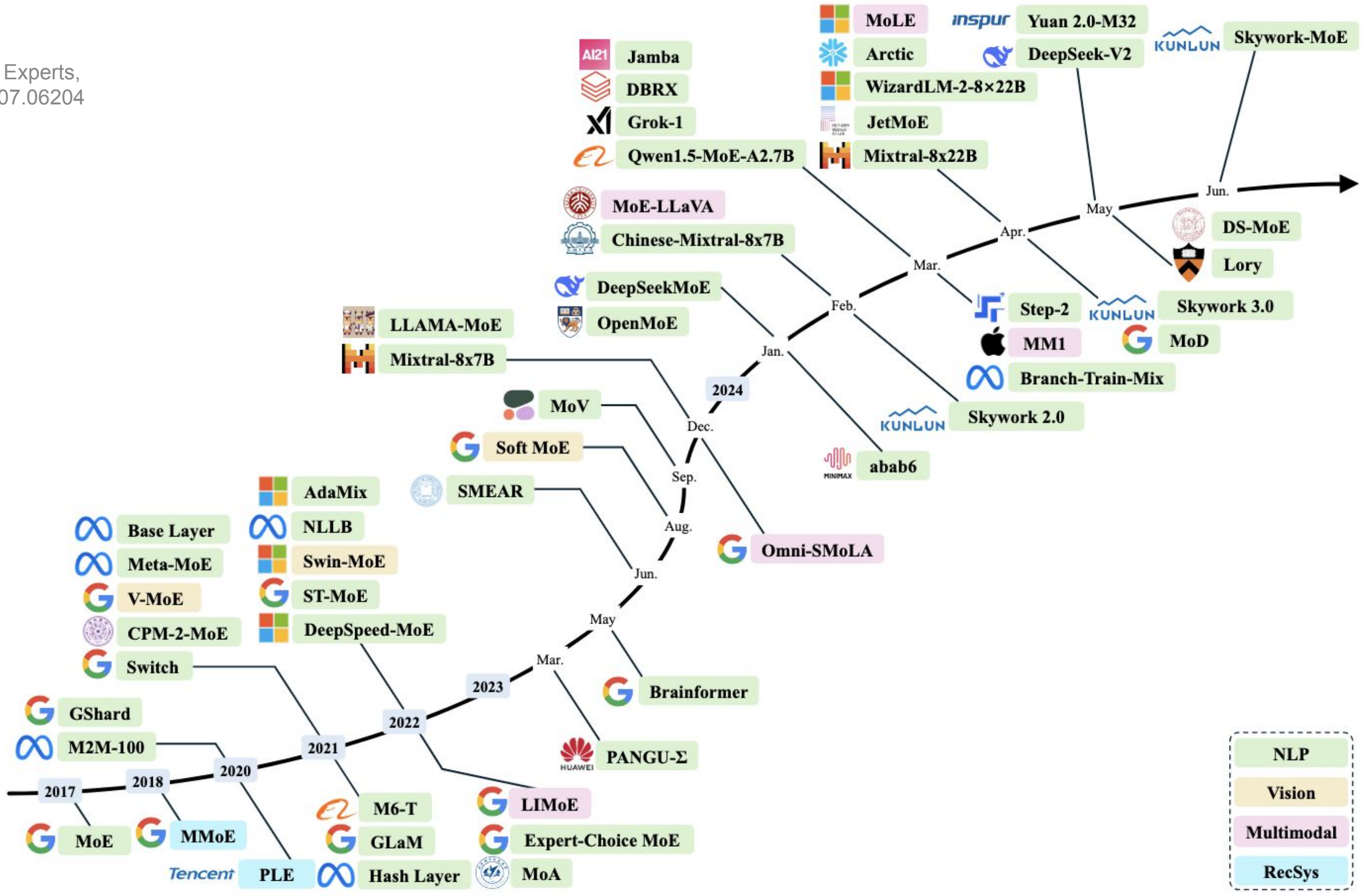


Fig. 1. A chronological overview of several representative mixture-of-experts (MoE) models in recent years. The timeline is primarily structured according to the release dates of the models. MoE models located above the arrow are open-source, while those below the arrow are proprietary and closed-source. MoE models from various domains are marked with distinct colors: Natural Language Processing (NLP) in green, Computer Vision in yellow, Multimodal in pink, and Recommender Systems (RecSys) in cyan.

Challenges in Practice

- Branching problems
  - GPUs are better for data plane than control plane
- Reduced batch sizes for conditionally active network chunks
- Network bandwidth bottleneck
- Model scarcity comes at the cost of model performance
- Existing conditional computation research deals with datasets too small given number of parameters

Sparsely Gated Mixture of Experts

- Uses a Mixture of Experts (MoE) layer containing:
    - Sparse MoE layer with n expert networks (E1…En)
    - A gate network/router G outputs sparse n-dimensional vector

- MoEs replace Feed Forward Network (FFN) in transformer blocks

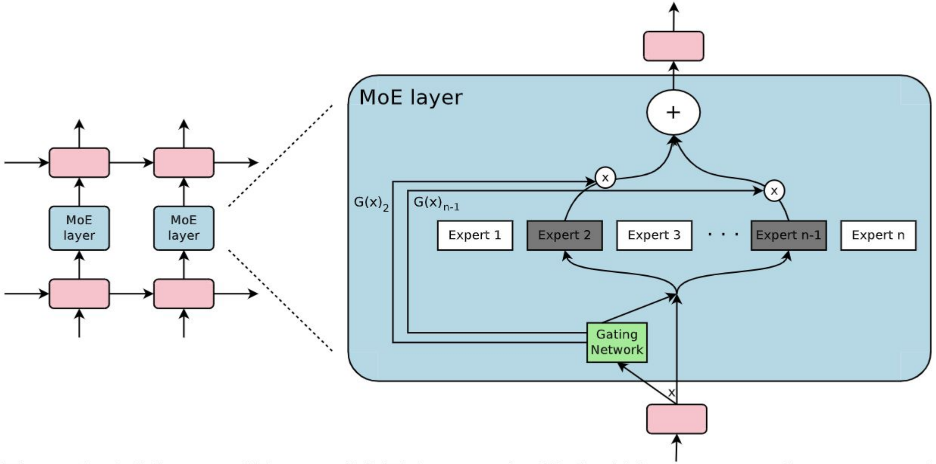


Figure 1: A Mixture of Experts (MoE) layer embedded within a recurrent language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

Aside: Mixture of Experts in Transformers

Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity, https://arxiv.org/abs/2101.03961

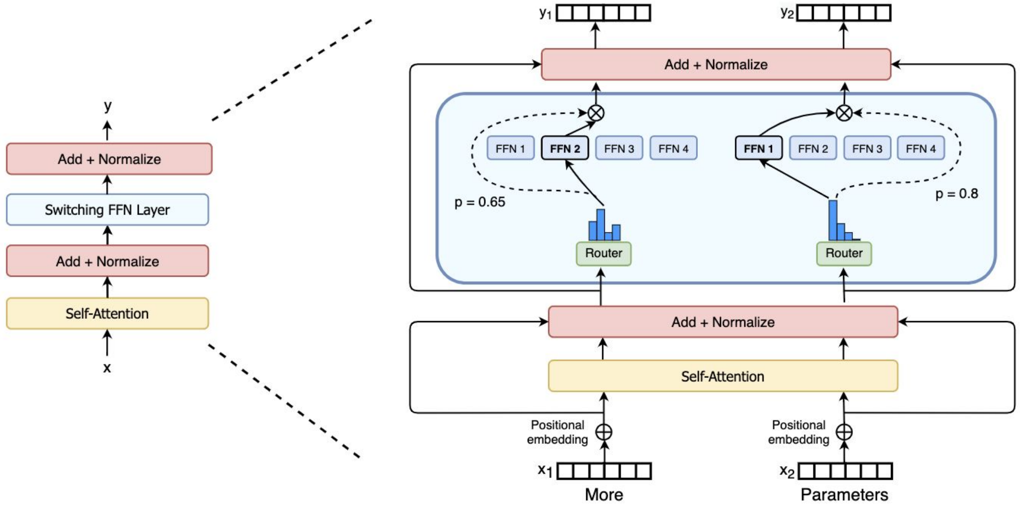


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Gating w/ a Performance Goal

- Dense MoE would use softmax for gating
- Sparse MoE (this paper) proposed <u>noisy</u> <u>top-K</u> gating
    - Top-K to reduce computation
    - Noise for load balancing
- Gating network is trained with the model itself by back propagation

Challenge: Shrinking Batch Size

- Modern GPUs need large batch sizes for computational efficiency
  - Amortizes the overhead of parameter loads and updates
- If the gating network chooses ***k*** out of ***n*** experts for each example, then for a batch of ***b*** examples, each expert receives a much smaller batch of approximately ***kb/n*** examples, which is much smaller than ***b***
-
- They proposed a combination of parallelism techniques
  - We'll see more in the coming weeks

Challenge: Network Bandwidth

- Communication between experts, depending on how/where they are located, can become a bottleneck

- To maintain computational efficiency, *the ratio of an expert's computation to the size of its input and output* **must exceed** *the ratio of computational to network capacity of the computing device*
  - The latter can be 1000:1 for highly parallel devices like GPUs
  - The former is controlled by the size of the hidden layer in this design and can be increase by using more or more complex hidden layers

Challenge: Expert Balancing

- Gating networks tend to favour a few select experts
- Need some constraint to ensure experts are trained/selected somewhat evenly
  - Create additional loss functions for variation in gate values and load
  - Add both loss functions to the model's overall loss function

Sparsity Allows for More Experts, and therefore, Better Performance

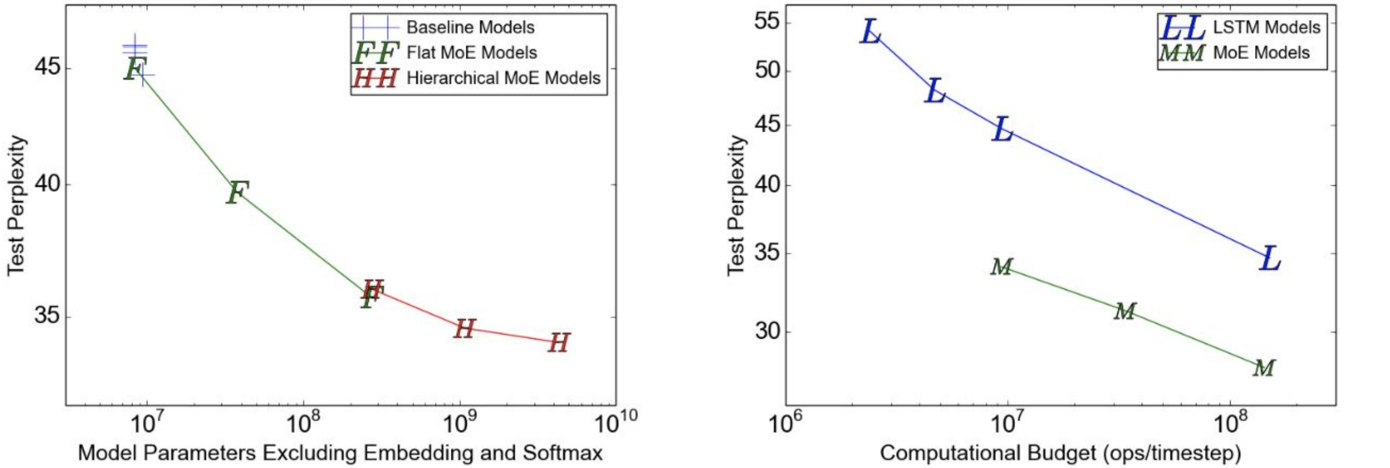


Figure 2: Model comparison on 1-Billion-Word Language-Modeling Benchmark. On the left, we plot test perplexity as a function of model capacity for models with similar computational budgets of approximately 8-million-ops-per-timestep. On the right, we plot test perplexity as a function of computational budget. The top line represents the LSTM models from (Jozefowicz et al., 2016). The bottom line represents 4-billion parameter MoE models with different computational budgets.

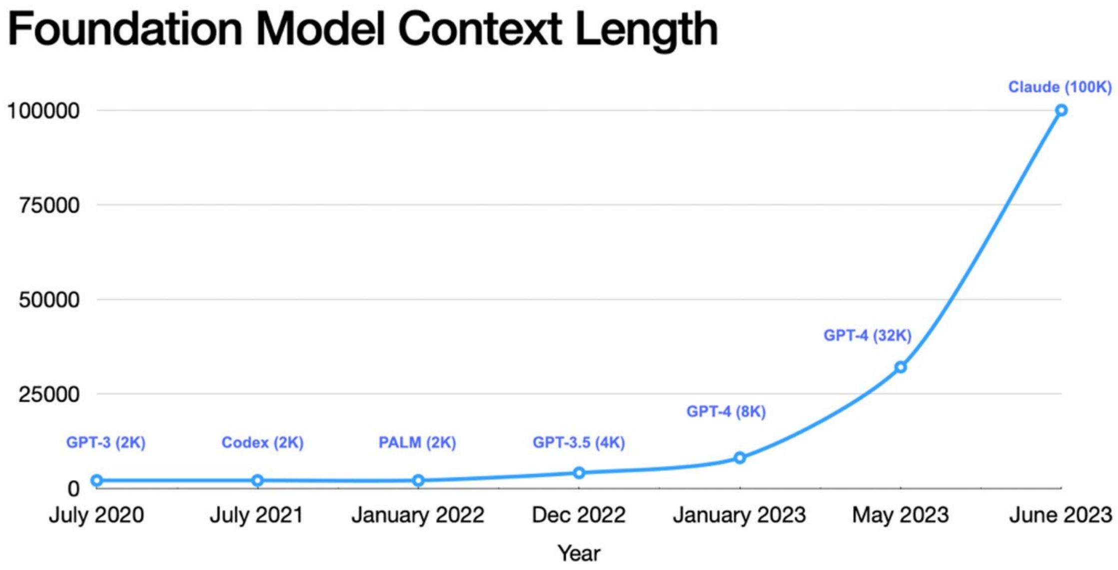
State Space Models (SSMs)

Useful Links to Understand SSM

- A Visual Guide to Mamba and State Space Models
- Introduction to State Space Models (SSM)
- Mamba Explained

- Presentation only covers basics of SSM and Mamba
    - Mamba: Linear-Time Sequence Modeling with Selective State Spaces [COLM'23]

Need for Long-Context Support



- **Chatbot**: need to remember all chat history
- **Multimodal**: all modal presentations are converted to tokens in context

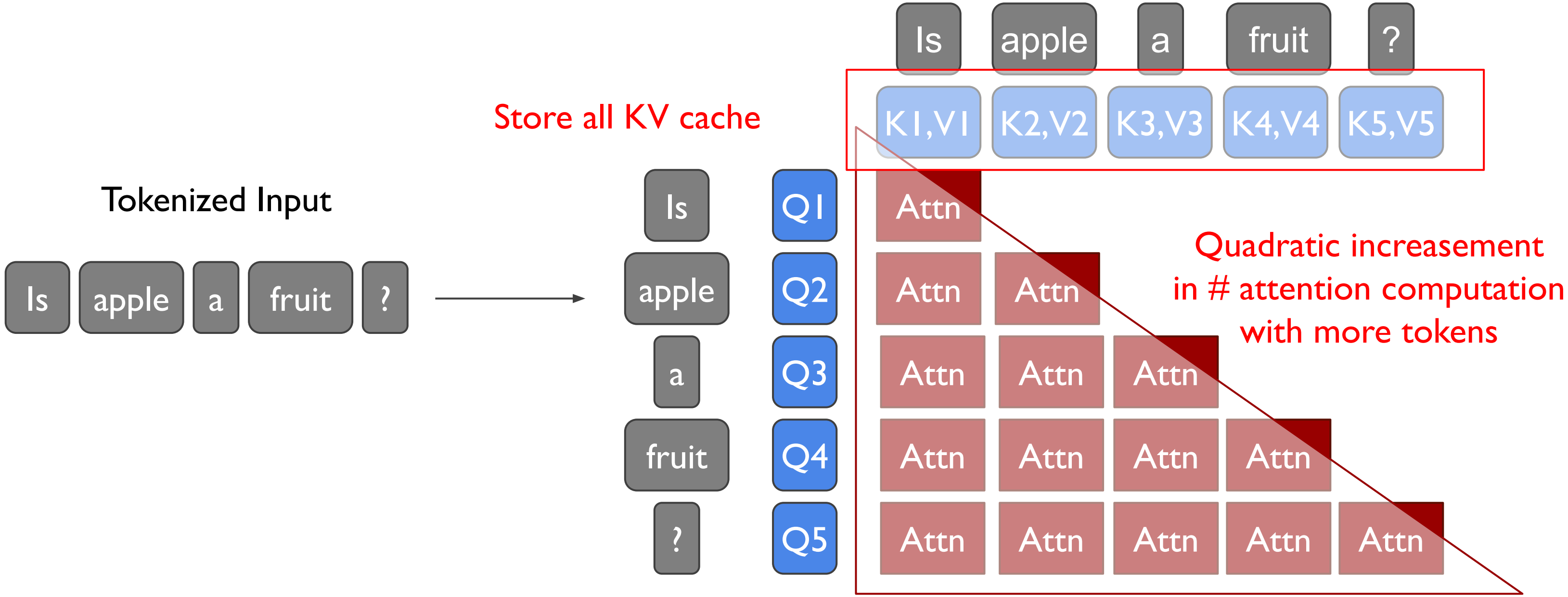
  e.g. A 10 hours of video (1fps) includes 9.9M tokens*

https://cerebras.ai/blog/variable-sequence-length-training-for-long-context-large-language-models/

* Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context

Problems of Transformer Architecture

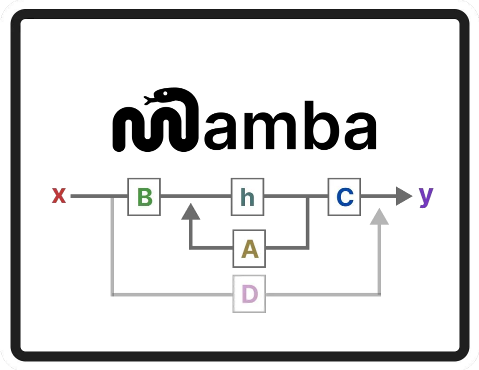
- Quadratic computation increasement with longer context
- Linear memory usage increasement with longer context (in inference)



State Space Model: Alternative to Transformer

- <u>Linear scaling</u> in sequence length (vs quadratic for Transformer)
- <u>Faster inference</u> (5x higher throughput than Transformer, matching quality)

- Implement Mamba language model based on SSM architecture



	Transformer	Mamba
Architecture	Attention-based	SSM-based
Complexity	High	Low
Inference Speed	O(n)	O(1)
Training Speed	$O(n^2)$	O(n)

What is State Space?

- State space: a discrete space representing the set of <u>all possible configurations</u> of a system

states

  - A way to mathematically represent a problem by defining a system's possible states
  - States capture all necessary information about the system at a given time to predict its future behavior

Example: a car moving in a straight line

State Space Model Example

Similar to hidden states in LLM!

Example: a car moving in a straight line

Describe how the state of the car (position and velocity) changes over time due to its current state and the input (acceleration)

state variables:

1. $p(t)$ = position of the car at time t
2. $v(t)$ = velocity of the car at time t

State vector $h(t)$

$$h(t) = \begin{bmatrix} p(t) \\ v(t) \end{bmatrix}$$

input: $x(t)$ = an action we take to control the system
(e.g. pressing the accelerator pedal)

Dynamic of the car:

1. $p'(t) = v(t)$: position changes according to velocity
2. $v'(t) = x(t)$: velocity changes according to acceleration

Writing the system of equations as a state space model:

Current state

$$h'(t) = \boxed{A}\, h(t) + \boxed{B}\, x(t)$$

Input

State Space Representation Derivation

Backup

state variables:

1. *p(t)* = position of the car at time *t*
2. *v(t)* = velocity of the car at time *t*

State vector $h(t)$

$$h(t) = \begin{bmatrix} p(t) \\ v(t) \end{bmatrix}$$

Dynamic of the car:

1. *p'(t) = v(t)*: position changes according to velocity
2. *v'(t) = x(t)*: velocity changes according to acceleration

input: *x(t)* = an action we take to control the system (e.g. pressing the accelerator pedal)

State space representation: $$h'(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} h(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} x(t)$$

Derivation:

$$h'(t) = \begin{bmatrix} p'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ x(t) \end{bmatrix}$$

$$\begin{bmatrix} v(t) \\ x(t) \end{bmatrix} = \begin{bmatrix} 0 \cdot p(t) \;+\; 1 \cdot v(t) \\ 0 \cdot p(t) + 0 \cdot v(t) + x(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} x(t)$$

State Space Model

Input
$x(t)$

SSM

State equation

$$h'(t) = Ah(t) + Bx(t)$$

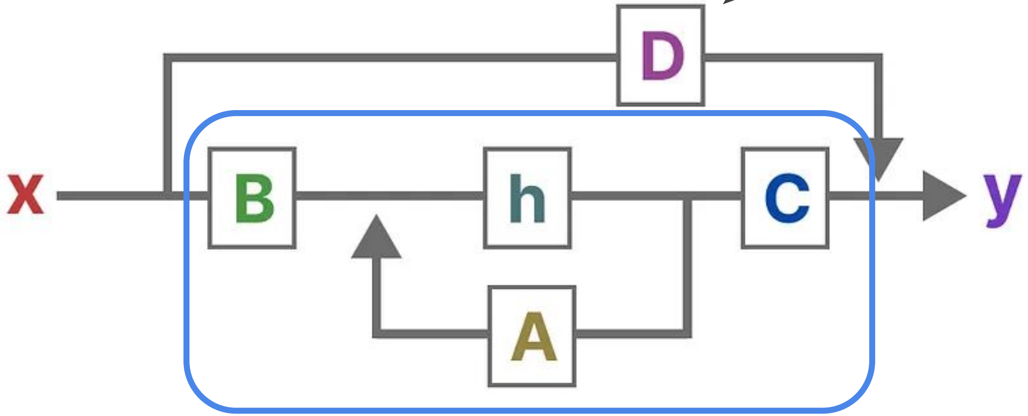
Output equation

$$y(t) = Ch'(t) + Dx(t)$$

Output
$y(t)$

State Space Model

D is similar to a skip-connection



Therefore SSM is often regarded as this part
without skip-connection

State Space Model

2 Background and Overview

2.1 Structured State Space Models

Structured state space sequence models (S4) are a recent class of sequence mo
to RNNs, CNNs, and classical state space models. They are inspired by a p
1-dimensional sequence $x \in \mathbb{R}^{\mathsf{T}} \mapsto y \in \mathbb{R}^{\mathsf{T}}$ through an implicit latent state h

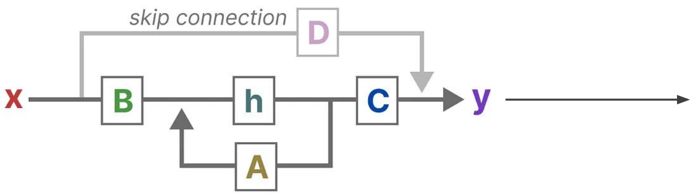
A general discrete form of structured SSMs takes the form of equation (1).

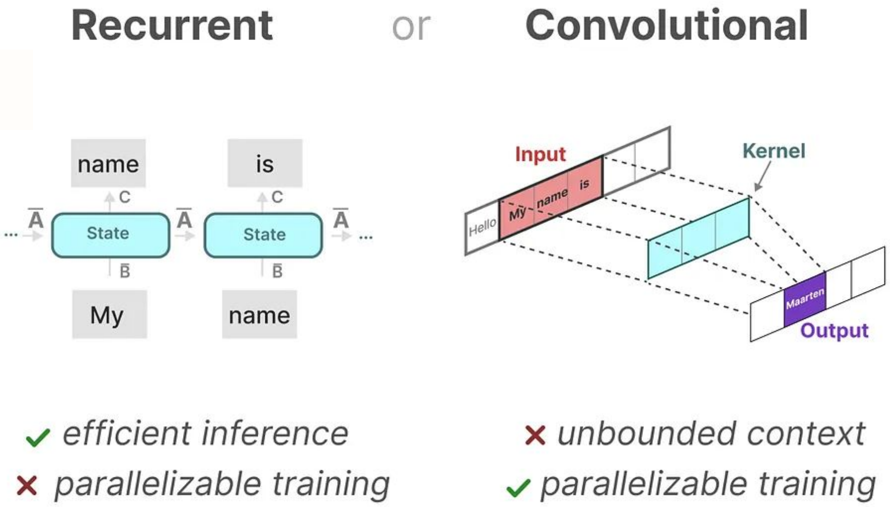
$$h_t = Ah_{t-1} + Bx_t \tag{1a}$$

$$y_t = C^\top h_t \tag{1b}$$

SSM Representations

- Can be represented to two different modes for different purpose
  - Convolutional representation for training efficiency (parallelism)
  - Recurrent representation for inference efficiency (unbounded context)





Recurrent SSM Representation

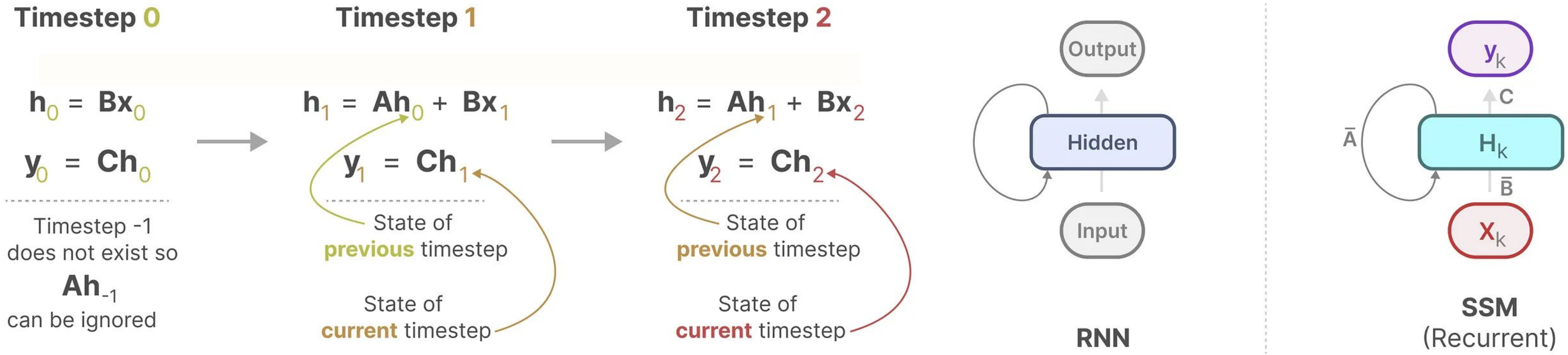
SSM

State equation

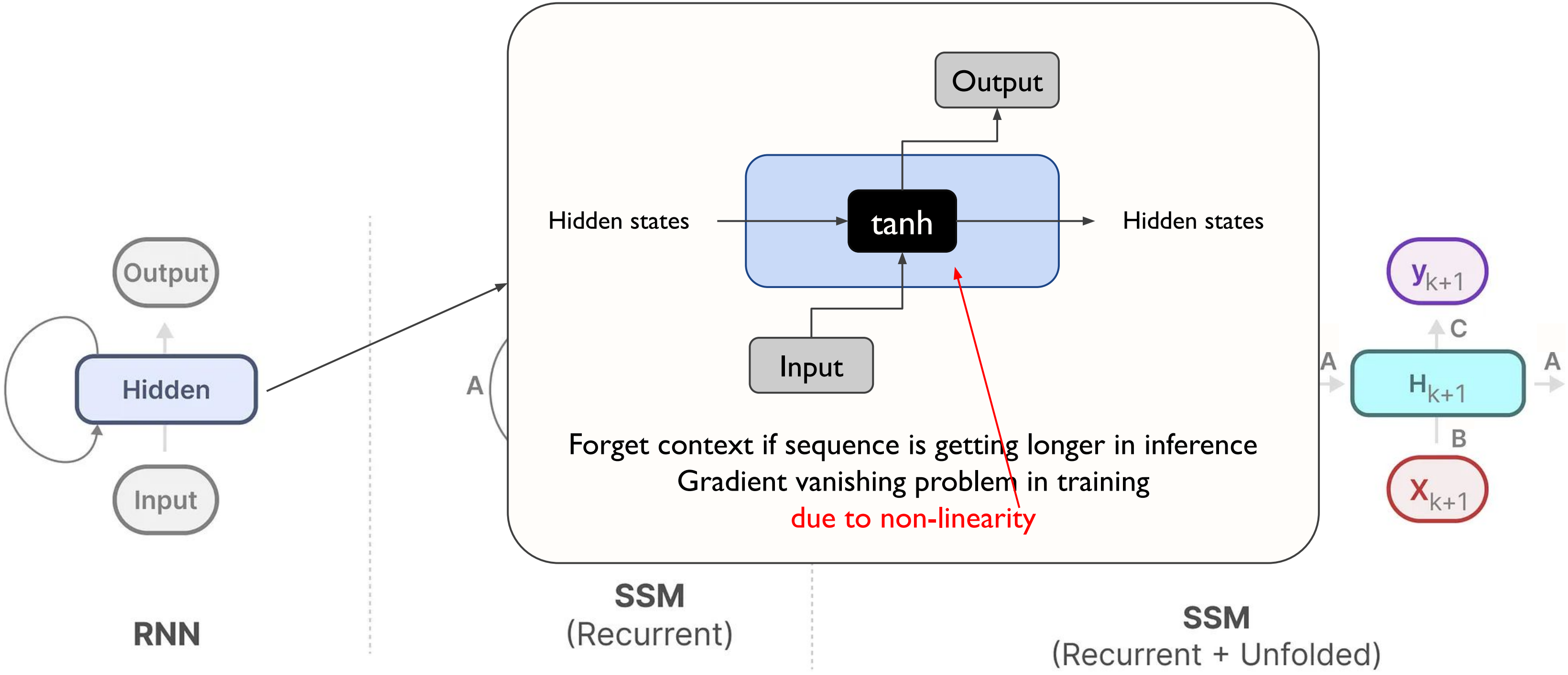
$$h'(t) = Ah(t) + Bx(t)$$

Output equation

$$y(t) = Ch'(t) + Dx(t)$$



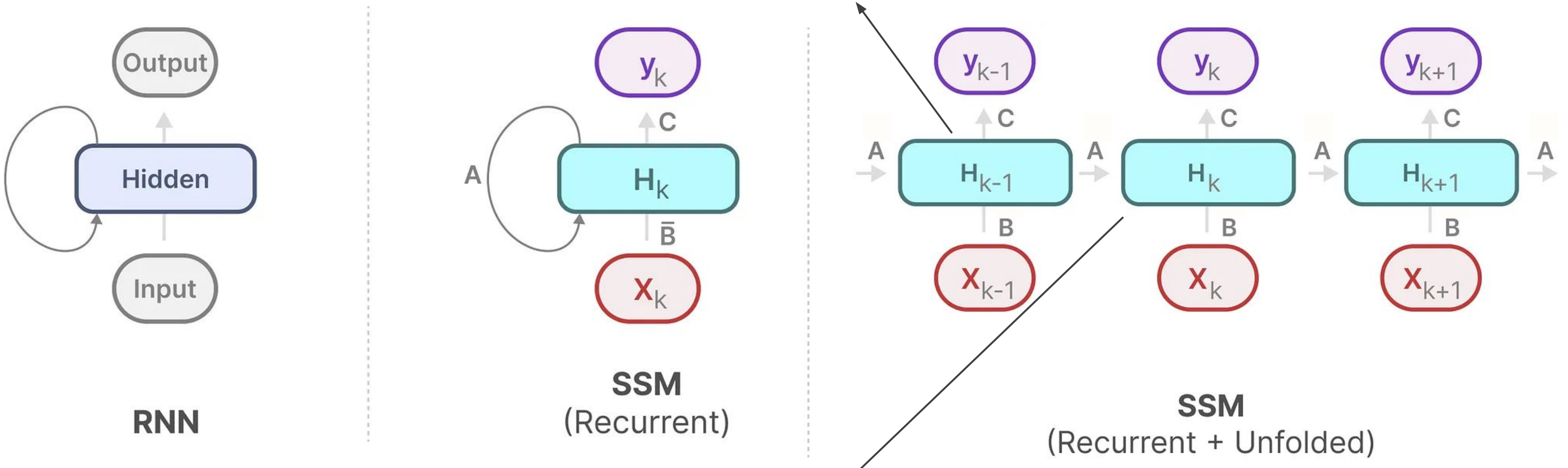
Recurrent SSM Representation



Recurrent SSM Representation

Linear-recurrence of SSM maintains unbounded context

$$h'(t) = Ah(t) + Bx(t)$$

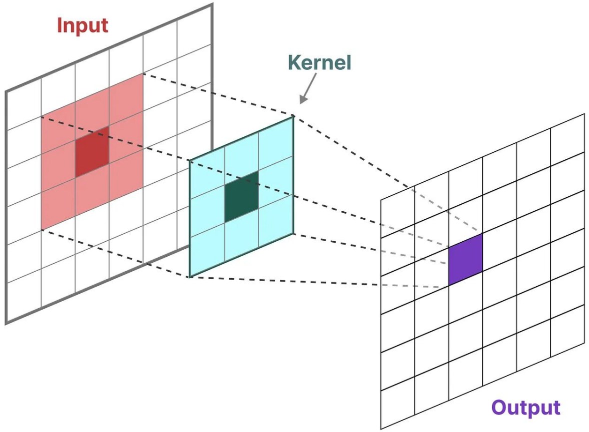


Structural state management allows inference
with static amount of computation and memory even for longer context

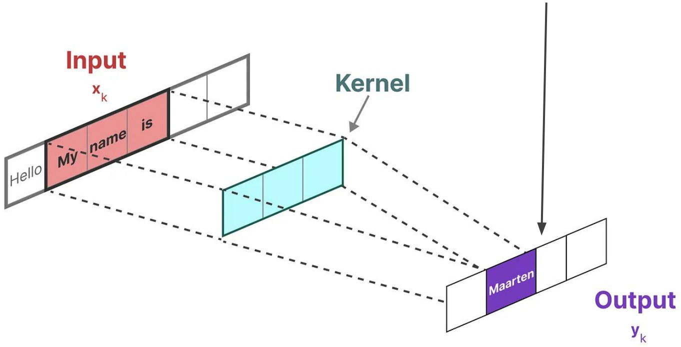
Convolutional SSM Representation

- Multi-head, Multi-contract, and Grouped-contract SSM

Computing each output is independent, thus highly parallelizable



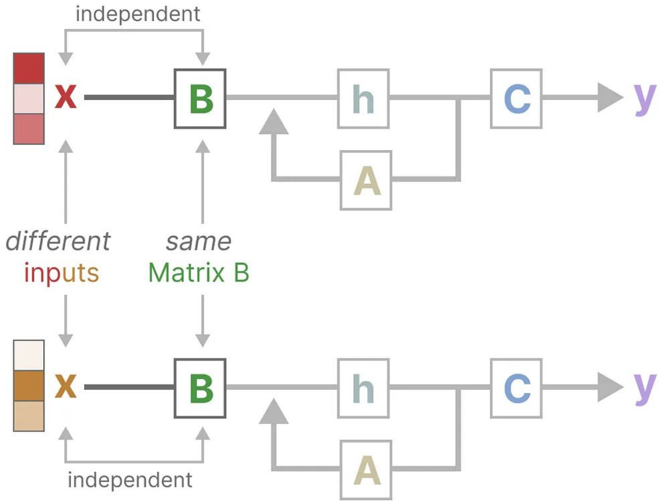
In CNN for images, 2D kernel is used to derive features



1D kernel is used for SSM based LLM

SSMs Still Perform Poorly in Language Modeling

- Lack the ability of focus or ignore particular inputs
- Matrices (A, B, C) are *time-invariant* and constant for every token
  → SSM cannot perform content-aware reasoning



Linear Time Invariant (LTI) SSM

State equation

$$h_t = A h_{t-1} + B x_t$$

Invariant over time

Output equation

$$y_t = C h_t$$

Selective State Space Model

- *Selectively propagate* or forget information

Linear Time Invariant (LTI) SSM

State equation

$$h_t = Ah_{t-1} + Bx_t$$

Output equation

$$y_t = Ch_t$$

Selective SSM

State equation

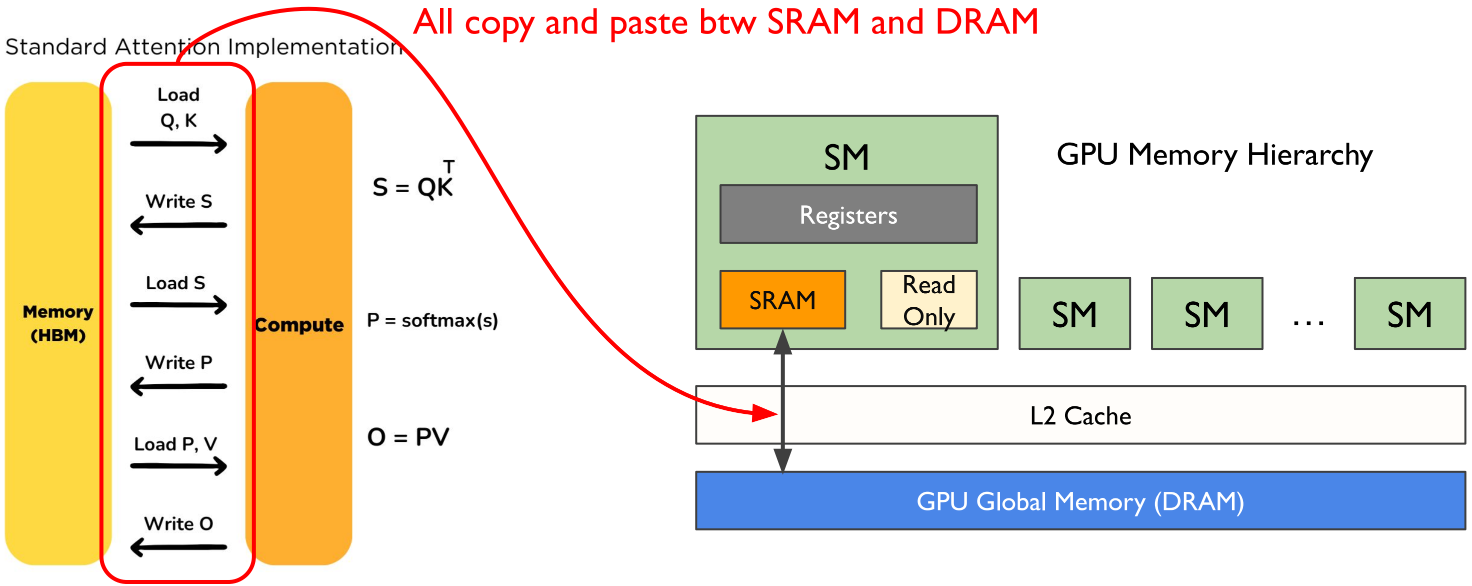
$$h_t = A_t h_{t-1} + B_t x_t$$

Output equation

$$y_t = C_t h_t$$

SSM is Hardware-Aware

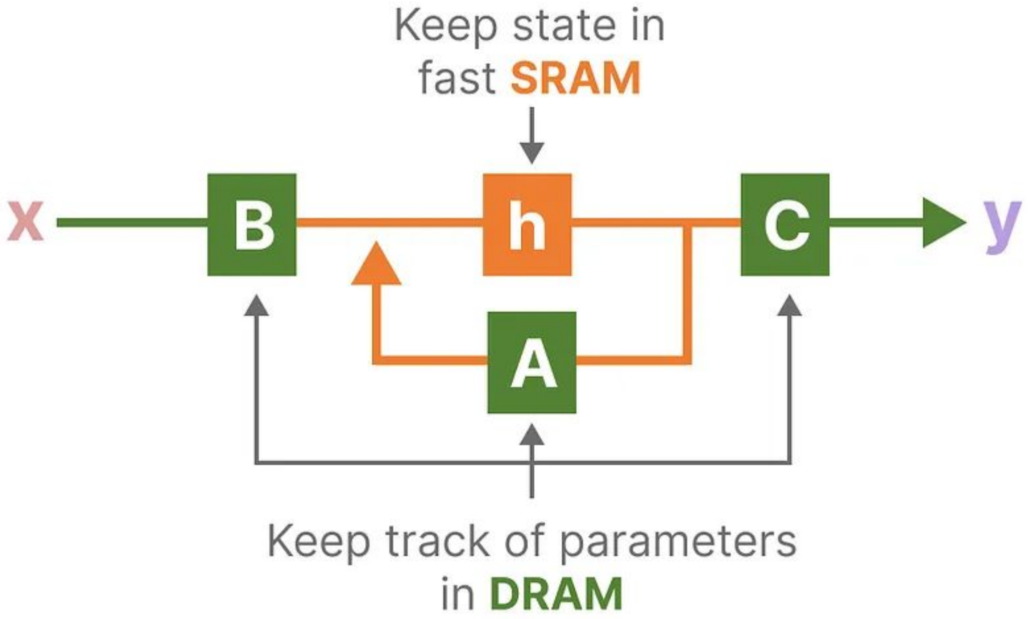
- Designed with hardware architecture in mind



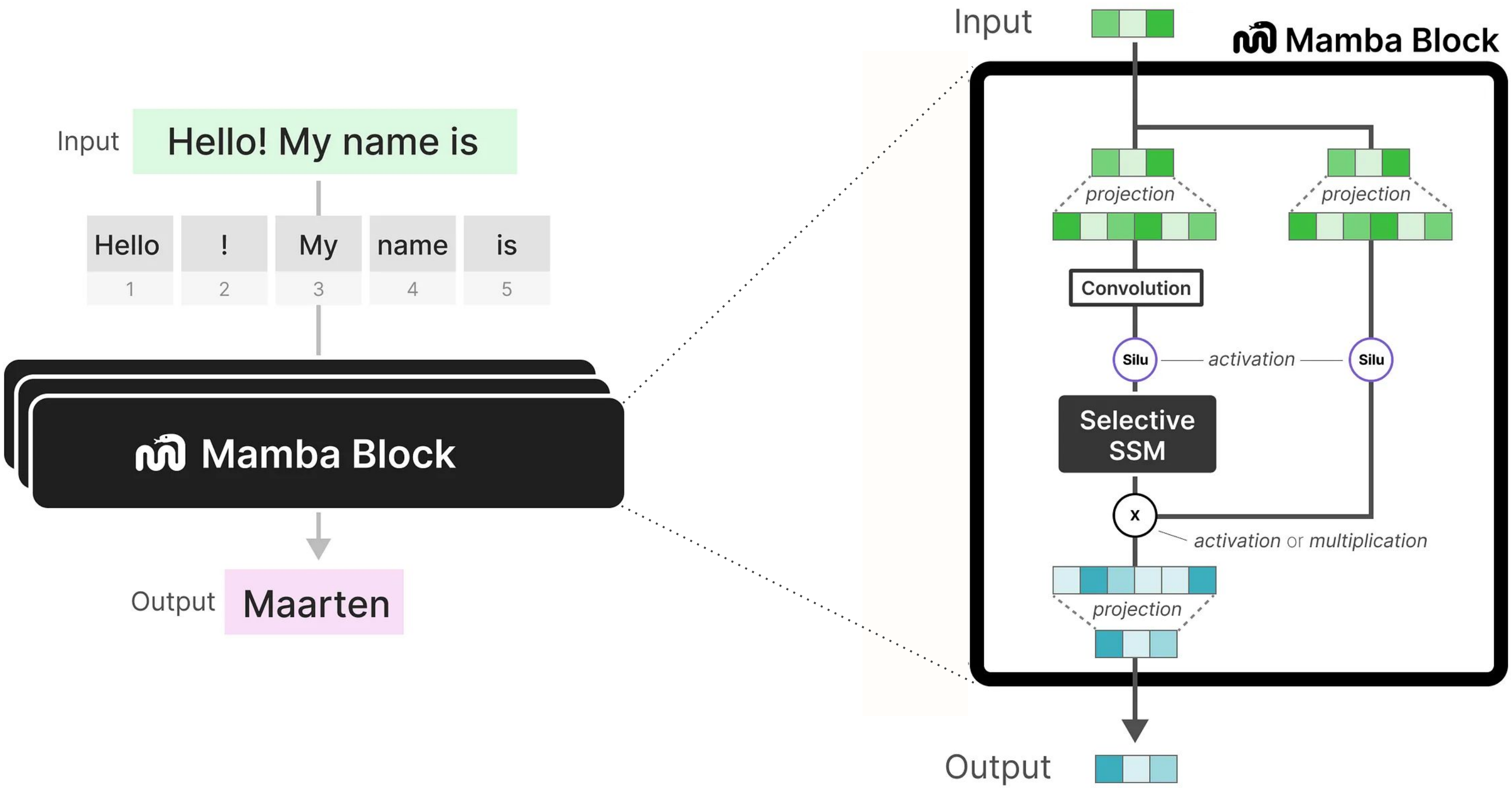
https://huggingface.co/docs/text-generation-inference/conceptual/flash_attention

SSM is Hardware-Aware

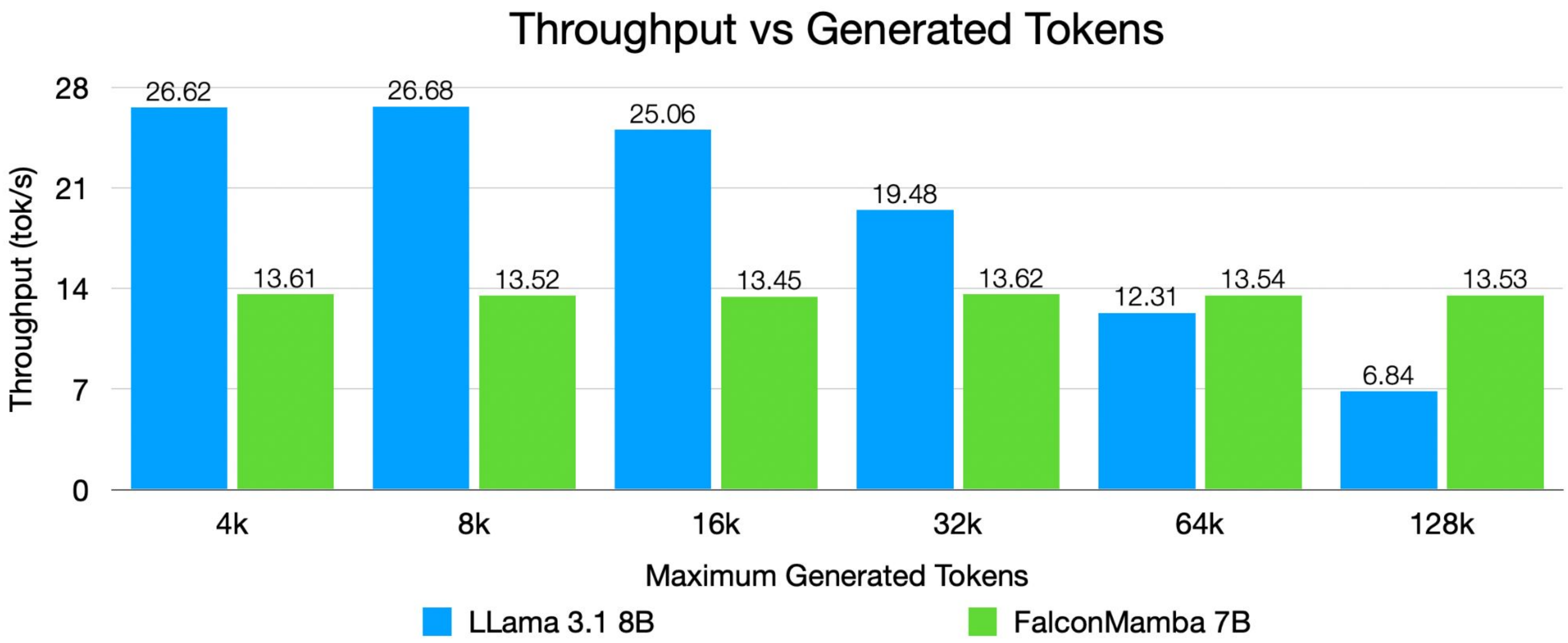
- Designed with hardware architecture in mind



Mamba: SSM based Neural Net Architecture



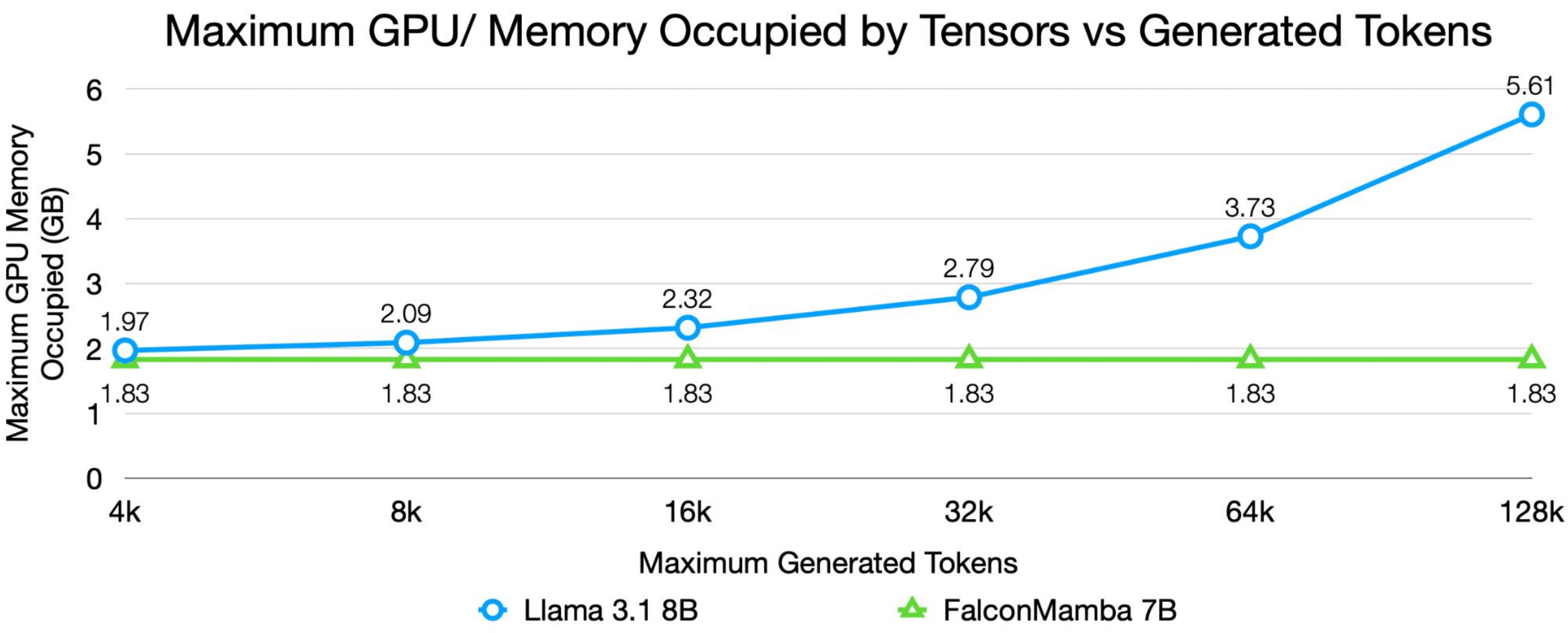
FalconMamba: Attention-Free LLM



Batch size: 1. Used one H100 80GB GPU

https://huggingface.co/blog/falconmamba

FalconMamba: Attention-Free LLM



Batch size: 1. Used one H100 80GB GPU
https://huggingface.co/blog/falconmamba

Lessons Learned

- Title is not all you need
  - Transformers are SSMs: allow studies for transformer to SSM
  - What it really says: propose some parallelisms to SSM based NN

Lessons Learned

- Even without model architecture background, you can still do systems research. A model is a set of operations

```
from transformers.models.llama import LlamaForCausalLM, LlamaTokenizerFast

tokenizer = LlamaTokenizerFast.from_pretrained("meta-llama/Meta-Llama-3.1-8B")
model =
LlamaForCausalLM.from_pretrained("meta-llama/Meta-Llama-3.1-8B")

input_ids = tokenizer("Hey how are you doing?", return_tensors="pt")["input_ids"]
out = model.generate(input_ids, max_new_tokens=10)
print(tokenizer.batch_decode(out))
['<|begin_of_text|>Hey how are you doing? I was wondering if you could help me
with something']
```

```
from transformers.models.mamba import MambaForCausalLM
from transformers import AutoTokenizerFast

tokenizer = AutoTokenizerFast.from_pretrained("state-spaces/mamba-2.8b-hf")
model = MambaForCausalLM.from_pretrained("state-spaces/mamba-2.8b-hf")

input_ids = tokenizer("Hey how are you doing?", return_tensors="pt")["input_ids"]
out = model.generate(input_ids, max_new_tokens=10)
print(tokenizer.batch_decode(out))
["Hey how are you doing?\n\nI'm doing great.\n\nI"]
```

Lessons Learned

main

transformers / src / transformers / models / mamba / modeling_mamba.py

```python
class MambaMixer(nn.Module):
    """
    Compute ∆, A, B, C, and D the state space parameters and compute the `contextualized_states`.
    A, D are input independent (see Mamba paper [1] Section 3.5.2 "Interpretation of A" for why A isn't selective)
    ∆, B, C are input-dependent (this is a key difference between Mamba and the linear time invariant S4,
    and is why Mamba is called **selective** state spaces)
    """

      def cuda_kernels_forward(

            # 2. Convolution sequence transformation
            conv_weights = self.conv1d.weight.view(self.conv1d.weight.size(0), self.conv1d.weight.size(2))
            if cache_params is not None and cache_position[0] > 0:
                hidden_states = causal_conv1d_update(
                    hidden_states.squeeze(-1),
                    cache_params.conv_states[self.layer_idx],
                    conv_weights,
                    self.conv1d.bias,
                    self.activation,
                )
                hidden_states = hidden_states.unsqueeze(-1)
            else:
                if cache_params is not None:
                    conv_states = nn.functional.pad(
                        hidden_states, (self.conv_kernel_size - hidden_states.shape[-1], 0)
                    )
                    cache_params.update_conv_state(self.layer_idx, conv_states, cache_position)
                hidden_states = causal_conv1d_fn(
                    hidden_states, conv_weights, self.conv1d.bias, activation=self.activation
                )

            if attention_mask is not None:
```

Transformers are SSMs

- Implement theoretical connections between SSMs and attentions
- Studies for Transformers can also be applied to SSM architecture!

Transformer Attention Architecture	Equivalent Pattern in SSM
Multi-head Attention (MHA)	Multi-head SSM (MHS)
Multi-query Attention (MQA)	Multi-Contract SSM (MCS)
Grouped-query Attention (GQA)	Grouped-Contract SSM (GCS)
Multi-key Attention (MKA)	Multi-expand SSM (MES)
Multi-value Attention (MVA)	Multi-input SSM (MIS)

Theoretically possible, haven't seen any model adopting these architectures

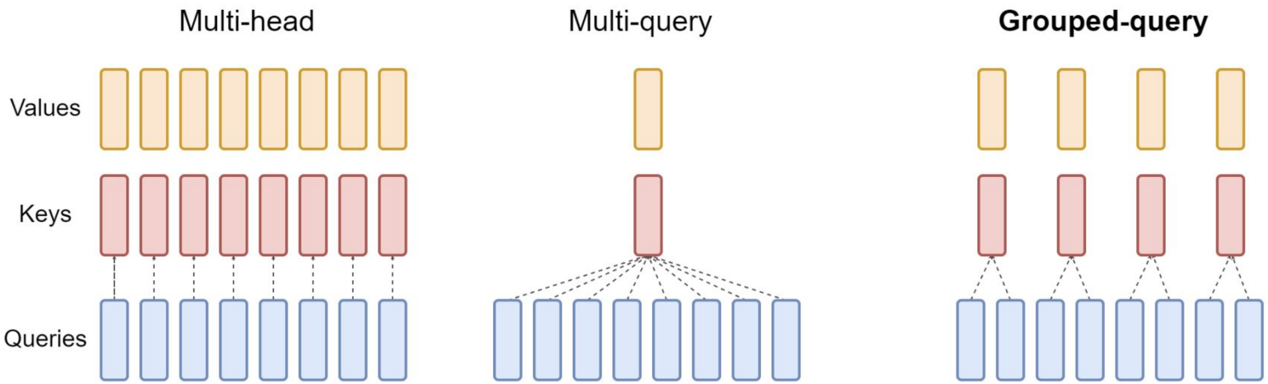
MHA, MQA, and GQA in Transformer

- Multi-head attention, multi-query attention, and grouped-query attention

  Original attention architecture

  Nearly all modern LLMs use this
  Llama3, Phi, Gemma, GPT4o, Claude, etc

- MQA and GQA are introduced by Google Research [EMNLP'23]



Equivalent SSM Architecture

- Multi-head, Multi-contract, and Grouped-contract SSM

System Optimizations for SSMs

- Tensor parallelism and sequence parallelism
- Methods to parallelize the model with multiple accelerators

- Will not cover for SSMs here
- Transformers parallelism will be covered in Sep 17~19