University of Idaho

Department of Computer Science

CS 487/587 Adversarial Machine Learning

Dr. Alex Vakanski

Lecture

Mixture of Experts, State Space Models

Introduction to MoE

Mixture of Experts

- *Mixture of Experts (MoE)* is a technique that combines multiple specialized models to handle complex tasks
  - Instead of a single large model, MoE uses an ensemble of "expert" models focused on learning different aspects of a task
  - MoEs are inspired by human teams, where experts collaborate on different aspects of a problem

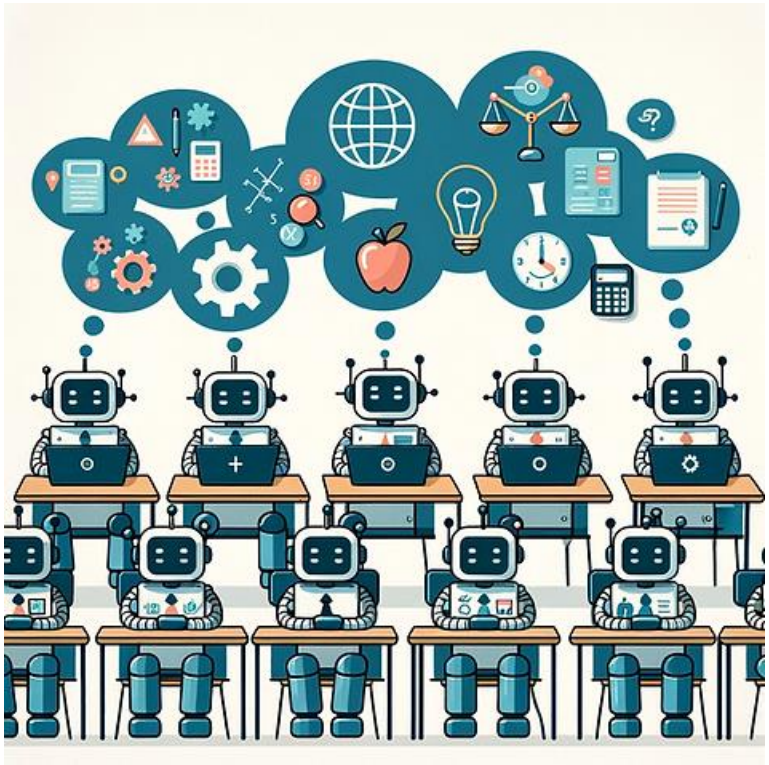


Figure from: https://www.tensorops.ai/post/what-is-mixture-of-experts-llm

Introduction to MoE

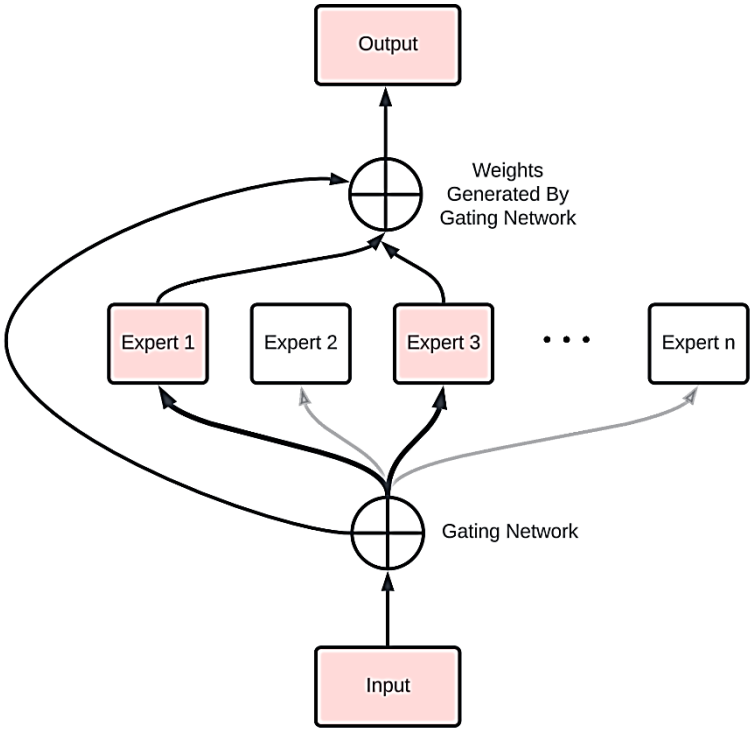
Mixture of Experts

- MoE employ the fact that that an ensemble of weaker ML models specializing in specific tasks can produce more accurate results, similar to traditional ML ensemble methods
  - E.g., Random Forests, Gradient Boosting methods, Bagging ensembles
- Differently from ensemble methods, MoE introduce a new concept of dynamic routing of the input to be processed by different experts

MoE Architecture

Mixture of Experts

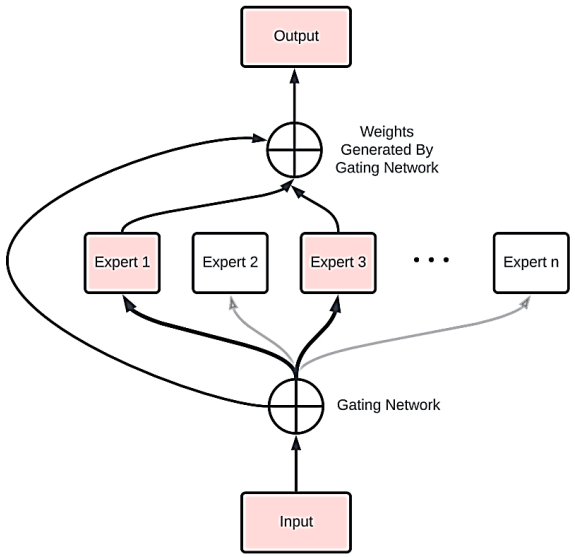
- An MoE system consists of two key components
  - Expert models, which can be neural nets or network layers, trained on different parts of the inputs (tokens)
  - Gating network, which acts as a router and decides which expert(s) should process an input



MoE Working Principles

Mixture of Experts

- The gating network analyzes the inputs, and routes each input to relevant expert(s)
  - In some MoE architectures, each token is sent to more than one expert (e.g., to 2 experts in Mixtral)
  - The experts' outputs are afterward combined through techniques like additive or multiplicative aggregation of the outputs
- During inference, for each input token only the chosen experts are activated, allowing efficient use of computational resources



MoE with Transformers

Mixture of Experts

- In Transformers, *sparse MoE layers* are used instead of dense feed-forward network (FFN) layers
  - I.e., standard FFN layers are replaced with switching FFN layers
  - Each MoE layer has a certain number of experts (e.g., 4), where each expert is an FFN, and each processes specific tokens
  - For example, in the figure, the router sends the token “More” to the second expert (FFN 2), and the token "Parameters” is sent to the first expert (FFN 1)

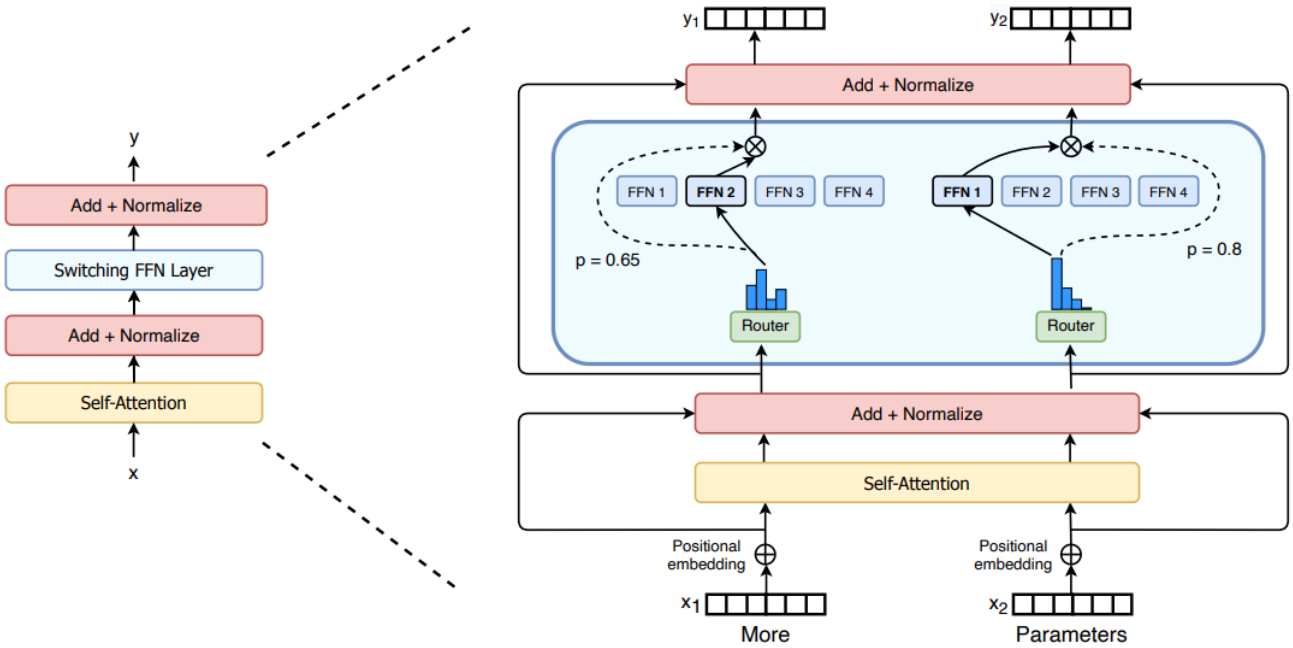
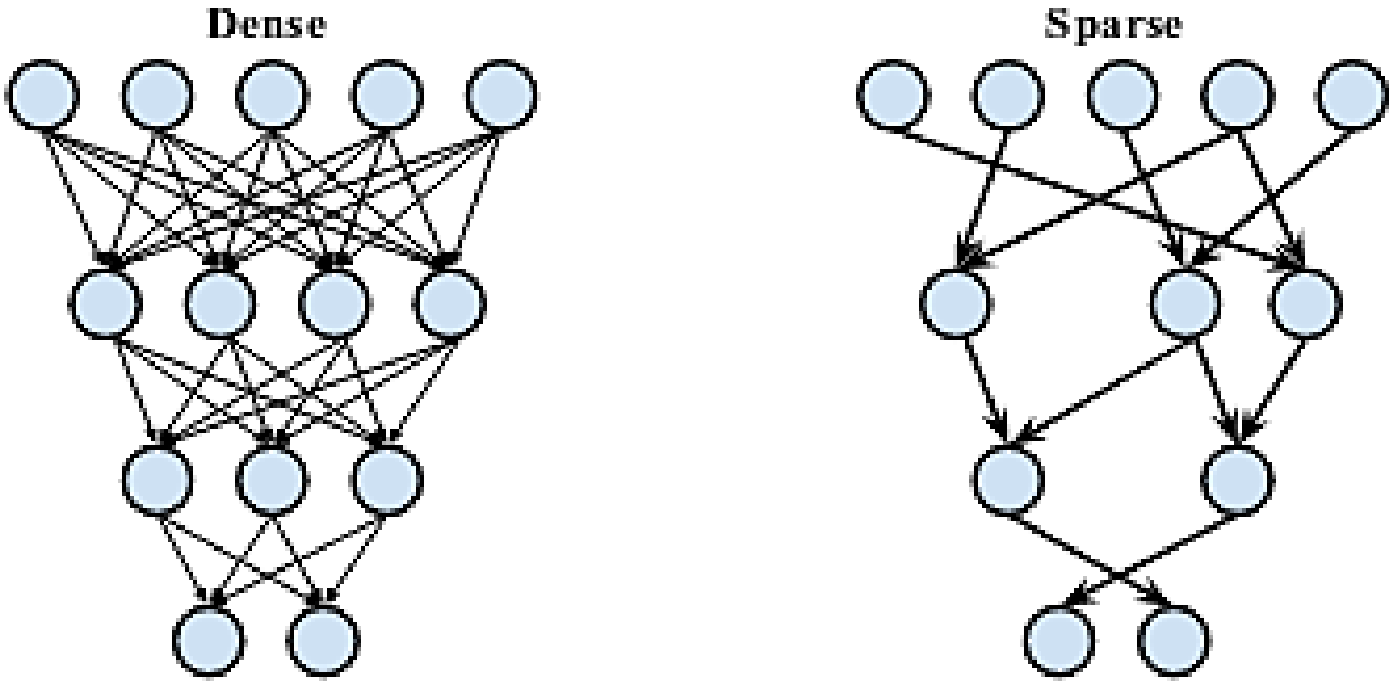


Figure from the Switch Transformers paper: https://arxiv.org/abs/2101.03961

Sparse Blocks

Mixture of Experts

- In *dense models* (e.g., networks with fully-connected layers) all the parameters of the model are used for processing all inputs (e.g., tokens)
- In *sparse models*, the individual inputs are run only through some parts of the model



Sparse Blocks

Mixture of Experts

- Sparsity uses the idea of conditional computation, where parts of the network are active on a per-token basis
  - Therefore, sparsity allows to scale the size of the model without increasing the computation
- In Large Language Models, the size of the models (number of parameters) is directly proportional to the performance: largest models are the best performing
  - Sparsity allows to train very large models with lower computational cost
  - Also, sparsity allows to make inference with a trained sparse model much faster, in comparison to dense models

Routing

Mixture of Experts

- Efficiently routing the tokens is one of the challenges for MoE
  - In recent MoE, the gating network (router) is composed of learned parameters and is trained at the same time as the rest of the network
  - I.e., the gating network is typically a simple neural network with a softmax function, which learns which expert to send each input to

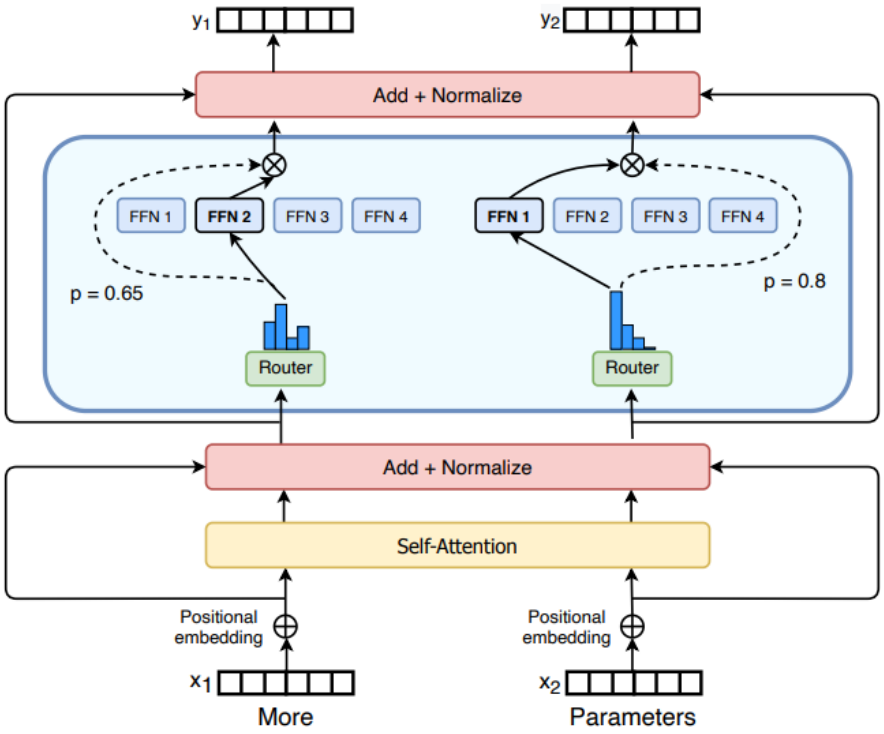


Figure from the Switch Transformers paper: https://arxiv.org/abs/2101.03961

Routing

Mixture of Experts

- In some cases, the gating network may converge to mostly activate the same few experts
  - When all tokens are sent to just 2 or 3 experts, that makes the training inefficient
  - I.e., the load is not properly balanced across all experts
- To mitigate the issue of load balancing, an auxiliary loss can be introduced during model training to encourage giving all experts equal importance
  - The auxiliary loss ensures that all experts receive a roughly equal number of training examples
- In some models (e.g., Mixtral), each token is sent to the top-2 experts to improve load balancing
- There is significant research effort dedicated on efficient load balancing in MoE

Advantages of MoE

Mixture of Experts

- The key advantages of MoE include *faster pretraining* thanks to sparse layers, and *efficient inference* by activating only required experts
  - MoE LLMs are pretrained much faster than dense models with the same number of parameters
    - Model capacity can be defined as the level of complexity that a model is capable of understanding or expressing
    - The size of a model is one of the most important factors for enhanced model capacity, and given a fixed computing budget, training a larger model for fewer steps is preferred than training a smaller model for more steps
    - E.g., Switch Transformers based on MoE achieved a 4x pretraining speed-up over the corresponding dense models
  - Faster inference is very important for end-users
    - Although an MoE might have many parameters, only some of them are used during inference, that results in much faster inference compared to a dense model with the same number of parameters
  - Faster pretraining and inference lead to potential savings in computational cost versus dense models
- Similarly, MoE generate *higher quality outputs* by combining specialized experts

Challenges of MoE

Mixture of Experts

- MoE have *high GPU memory requirements* since all experts needs to be loaded into the memory for both training and inference
  - E.g., loading all experts in Mixtral requires to load 49B parameters in the GPU VRAM memory, which creates challenges for users with low memory GPUs or a single low-end GPU
- *Finetuning* MoE *is more challenging* than dense models, because sparse layers are more prone to overfitting
  - Recent works have shown promising results and introduced strategies for finetuning MoE
- MoE require to carefully balance tradeoffs between training costs and inference efficiency
  - While offering faster inference, MoE require careful management of VRAM and computing resources

GPT-4's MoE

Mixture of Experts

- Based on leaked information about Open AI's *GPT-4* language model, it is implemented using an MoE architecture
- It combines 8 expert models, each with 220B parameters
  - This amounts to about 1.7T parameters, but the actual number of parameters can be somewhere between 1.2T and 1.7T parameters
- MoE allows GPT-4 to achieve superior quality while optimizing computational efficiency during inference
  - MoE also allows to distribute the model across multiple GPUs while routing inputs dynamically

Mixtral: Open Source MoE

Mixture of Experts

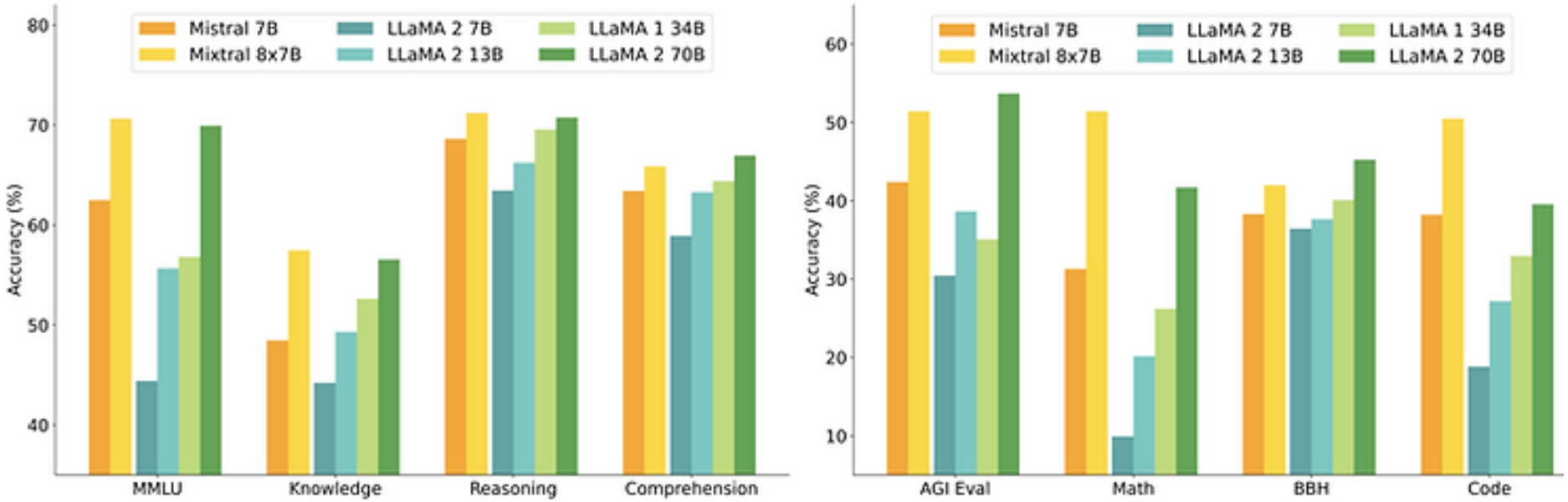
- *Mixtral* is an open-source MoE model with 47B parameters
- It uses 8 expert Mistral-7B models
  - But Mixtral uses only about 12.9B parameters per input token through sparse routing
- This allows Mixtral to outperform much larger models
  - It matches the quality of GPT-3.5 (175B parameters)
  - It is 6X faster than LLaMa-70B parameter

	LLaMA 2 70B	GPT-3.5	Mixtral 8x7B
MMLU (MCQ in 57 subjects)	69.9%	70.0%	**70.6%**
HellaSwag (10-shot)	**87.1%**	85.5%	86.7%
ARC Challenge (25-shot)	85.1%	85.2%	**85.8%**
WinoGrande (5-shot)	**83.2%**	81.6%	81.2%
MBPP (pass@1)	49.8%	52.2%	**60.7%**
GSM-8K (5-shot)	53.6%	57.1%	**58.4%**
MT Bench (for Instruct Models)	6.86	**8.32**	8.30

Mixtral: Open Source MoE

Mixture of Experts

- Mixtral outperforms or matches LlaMA 70B on many benchmarks



Mixtral: Open Source MoE

Mixture of Experts

- It is important to note that although Mixtral uses 8 experts each having the form of a Mistral-7B model, it does not mean that each expert is a separate full network with 7B parameters, as shown in the figure below
  - This design would yield a model with 8x7B = 56B parameters

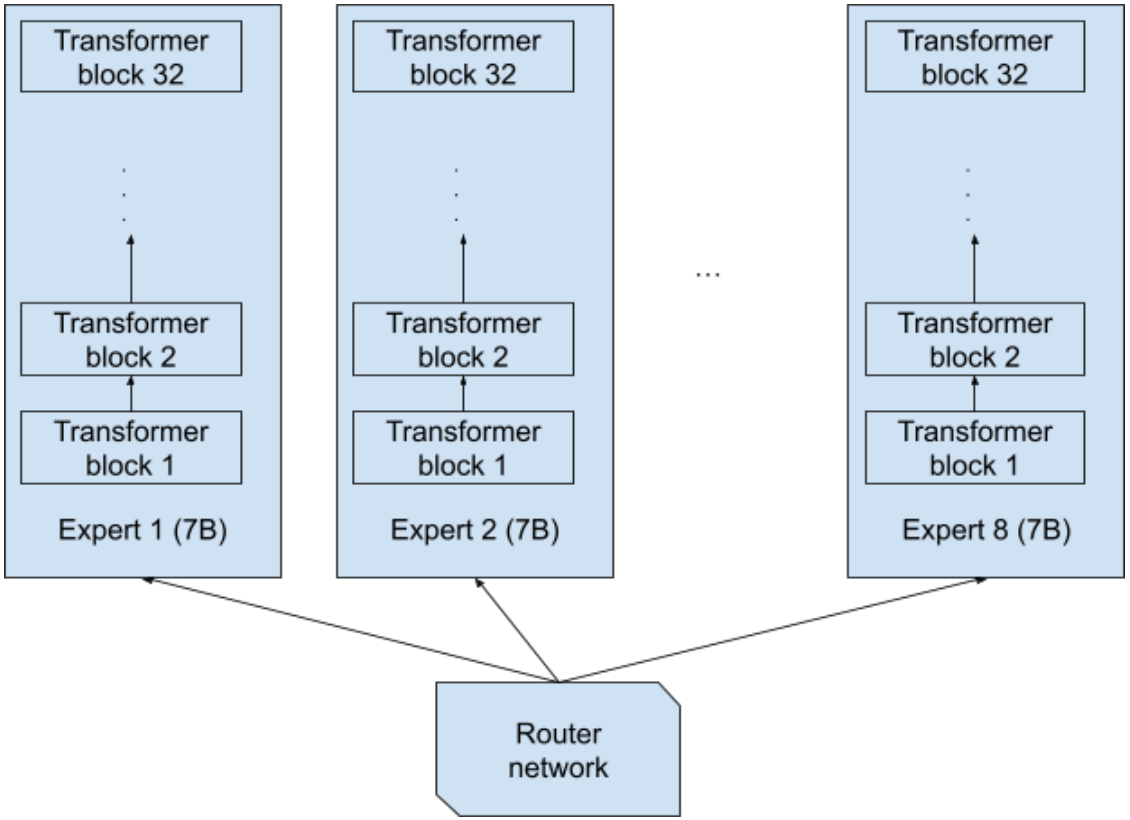


Figure from: https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/

Mixtral: Open Source MoE

Mixture of Experts

- The actual design of Mixtral is depicted in the figure below, where only the FFN layers are replaced with a sparse MoE block with 8 experts, and the remaining layers (including self-attention and normalization layers) are shared by all tokens
  - Therefore, the entire Mixtral model has about 47B parameters, and not 8x7B = 56B parameters, due to the shared layers
  - Each token is routed to two experts and processed by only 12.9B parameters (and not 2x7B = 14B parameter

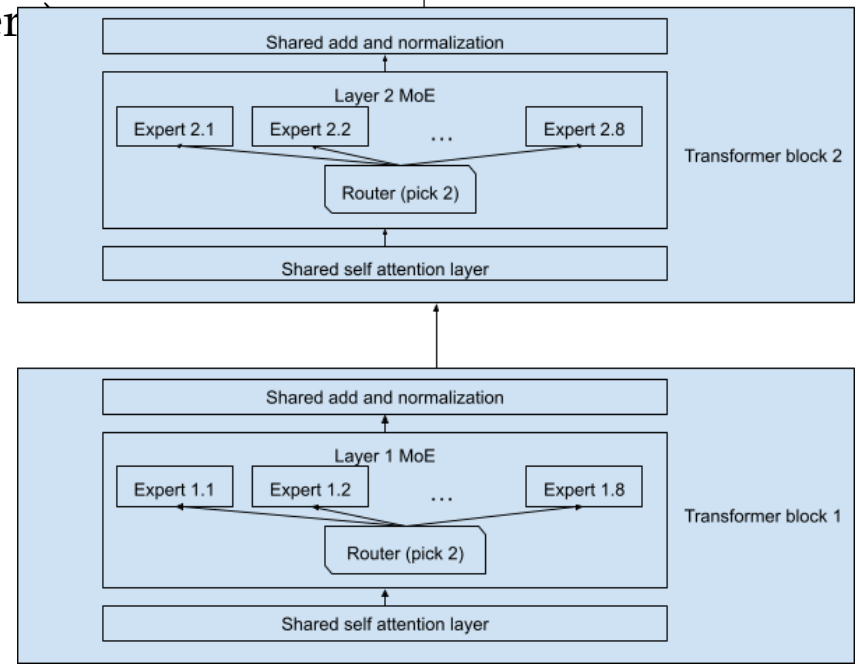


Figure from: https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/

What Does an Expert Learn

Mixture of Experts

- A recent work studied the experts in Mixtral on the MMLU benchmark
  - MMLU includes multiple-choice questions on 57 topics, such as abstract algebra, world religions, professional law, anatomy, astronomy, business ethics, etc.
  - Left figure: in layer 32, for abstract algebra questions, the model makes use of experts 3 and 8 much more than the other experts
  - Right figure, in the area of professional law, in layer 32 the model mostly activates expert 4 while muting experts 3 and 8
  - However, since the tokens are routed through 32 transformer blocks, there are so many expert combinations, and it is difficult to analyze precisely what each expert is learning

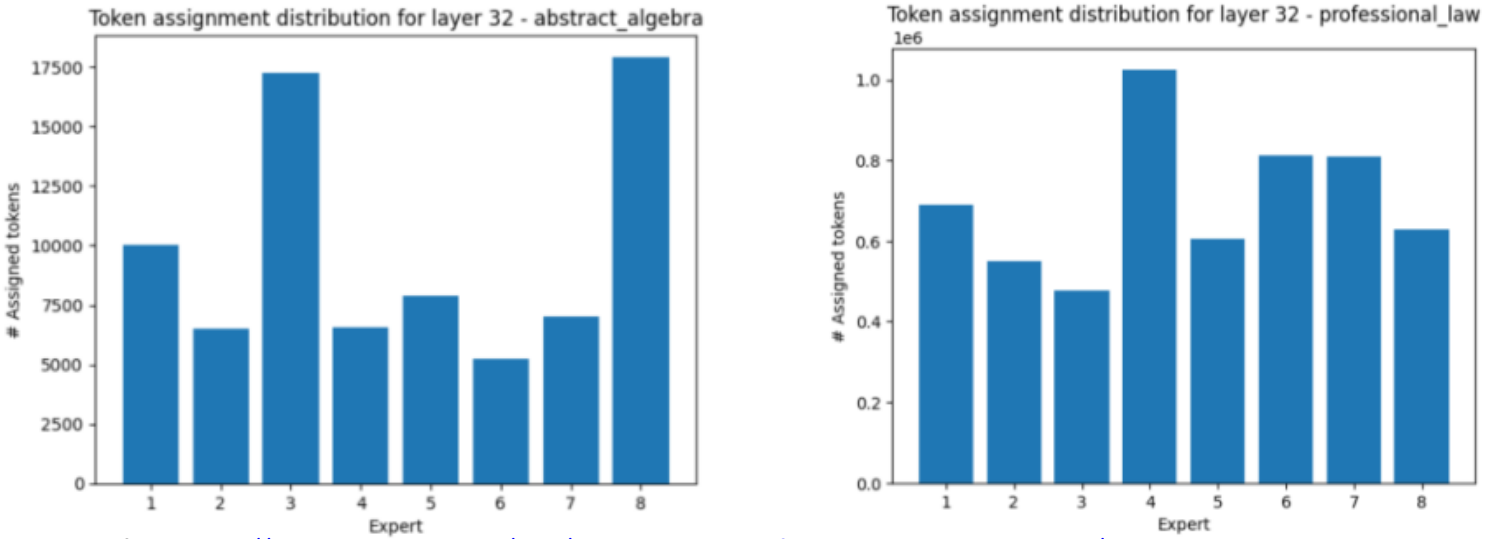


Figure from: https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/

Future Potential of MoE

Mixture of Experts

- MoE offer the potential for building scalable, efficient AI models across domains and applications
- Current open-source MoE LLMs include:
  - Switch Transformers (Google), family of models from 8 to 2048 experts, the largest model has 1.6T parameters
  - NLLB MoE (Meta), an MoE variant of the NLLB translation model
  - OpenMoE, a community effort that has released Llama-based MoEs
  - Mixtral 8x7B (Mistral), 8 experts, 47B parameters

State Space Models

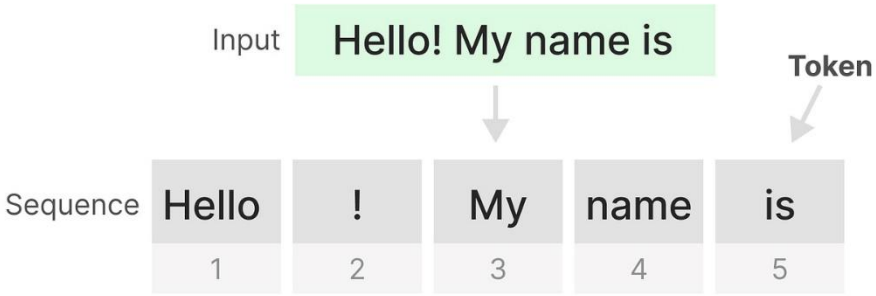
State Space Models

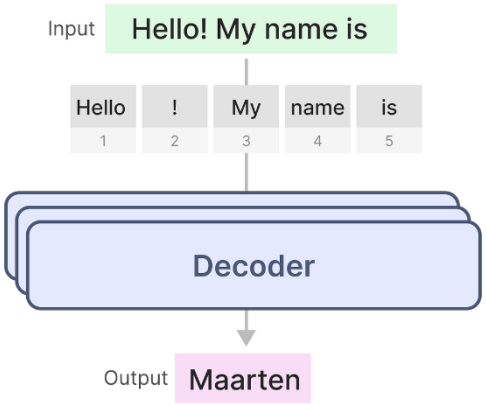
- ***State Space Models*** are a new types of architectures for LLMs
  - This type of architectures don't use the attention mechanism
  - They have potential to address some of the limitations of Transformer Networks, stemming from the attention mechanism
- The concept of State Space Models have been extensively used in control systems and system identification for many decades
- Mamba is a recent LLM that uses State Space Model architecture
  - It demonstrates the potential to replace Transformer LLMs

Transformers

State Space Models

- *Transformers* process textual inputs as sequences of tokens, and derive representations from previous tokens in text





- Attention scores are calculated between all token pairs, based on the relevance of one token for the meaning of another token in a specific context

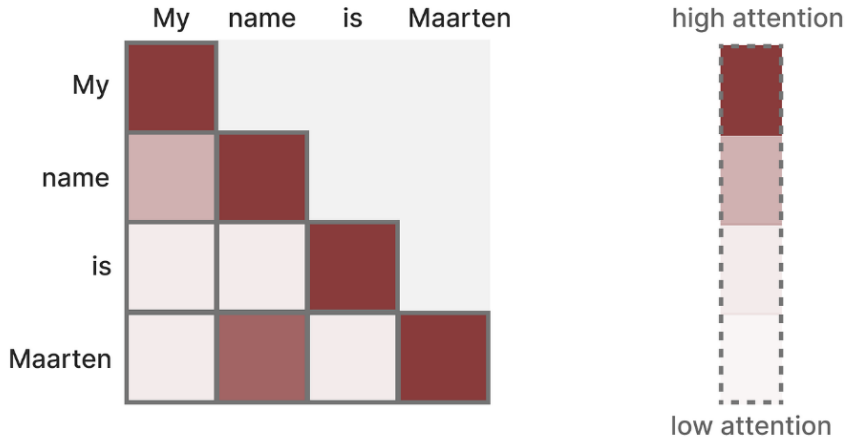


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Transformers

State Space Models

- Transformer architecture allows parallelization, which speeds up training tremendously
  - It is possible to distribute the training data in each batch across multiple GPUs
  - Multi-head attention modules can be processed in parallel
  - The tensors can also be split into smaller sub-tensors and processed in parallel

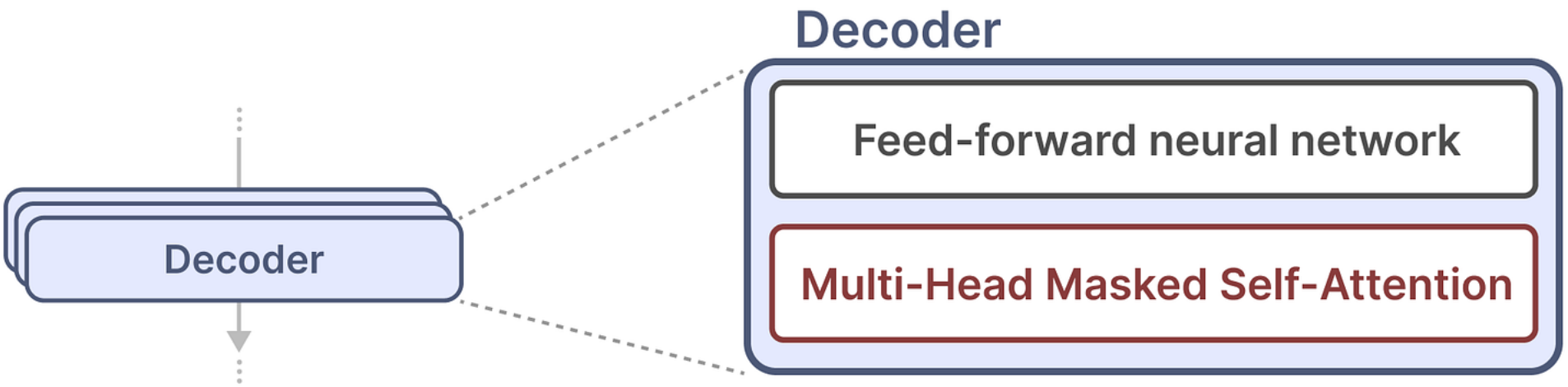


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

The Problem with Transformers

State Space Models

- Despite efficient training, Transformer architecture encounters a bottleneck during inference
- It requires to recalculate the attention scores for the entire sequence, making inference slow and computationally expensive for long sequences
  - A sequence of length L requires L^2 computations (to calculate the attention matrix of size $L \times L$)
  - I.e., there is a quadratic computational cost increase with increasing the length of the context for generating text

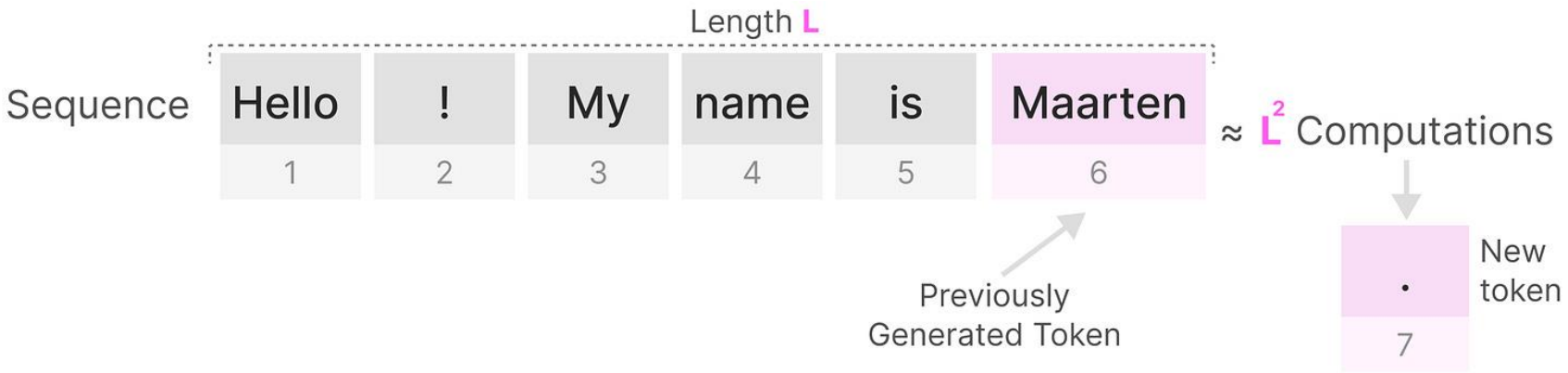


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Pros and Cons of Transformers

State Space Models

- Pros
  - Effective at modeling complex dependencies: every token explicitly attends to all other previous tokens, via attention mechanism
  - Highly parallel training: the core operations are matrix multiplications which can be parallelized across many GPUs
- Cons
  - Quadratic scaling with context length: since every input attends to all prior inputs, the total amount of computation accelerates as the number of tokens increases
  - Autoregressive inference is expensive: the attention scores needs to be recalculated for every new token

RNNs Overview

State Space Models

- *Recurrent Neural Networks* (RNNs) have a looping mechanism (recurrence) that allows to pass information from a previous step to the next
  - At each step, the network takes the input at time step t and a hidden state of the previous time step t-1, to generate the next hidden state and predict the output
  - E.g., to calculate the output y_{t+1}, RNN needs only the input x_{t+1} and the corresponding hidden state
    - RNNs do not need access to all previous inputs for calculating y_{t+1}

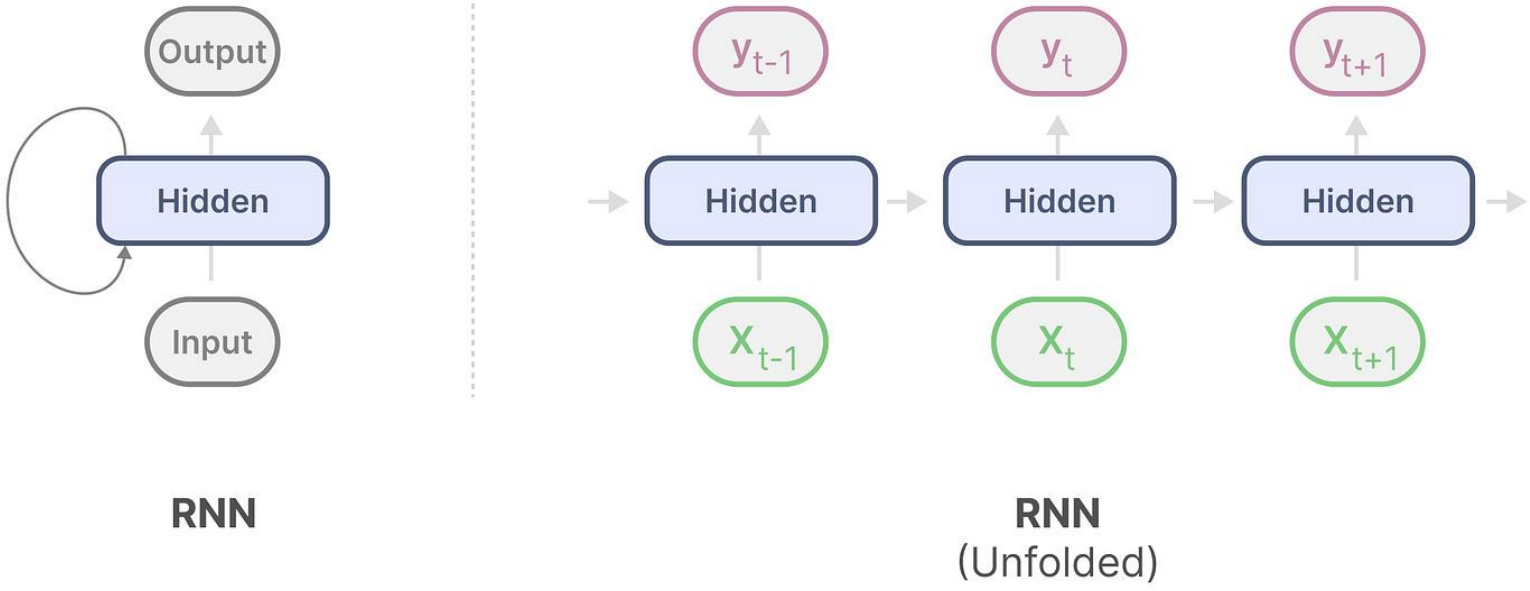


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

RNNs Overview

State Space Models

- RNNs can do inference fast, since they scale linearly with the sequence length
  - When generating the output, the RNN only needs to consider the previous hidden state and current input
    - It does not need to recalculate all previous hidden states, as in Transformer
  - Each hidden state is an aggregation (compressed view) of all previous hidden states

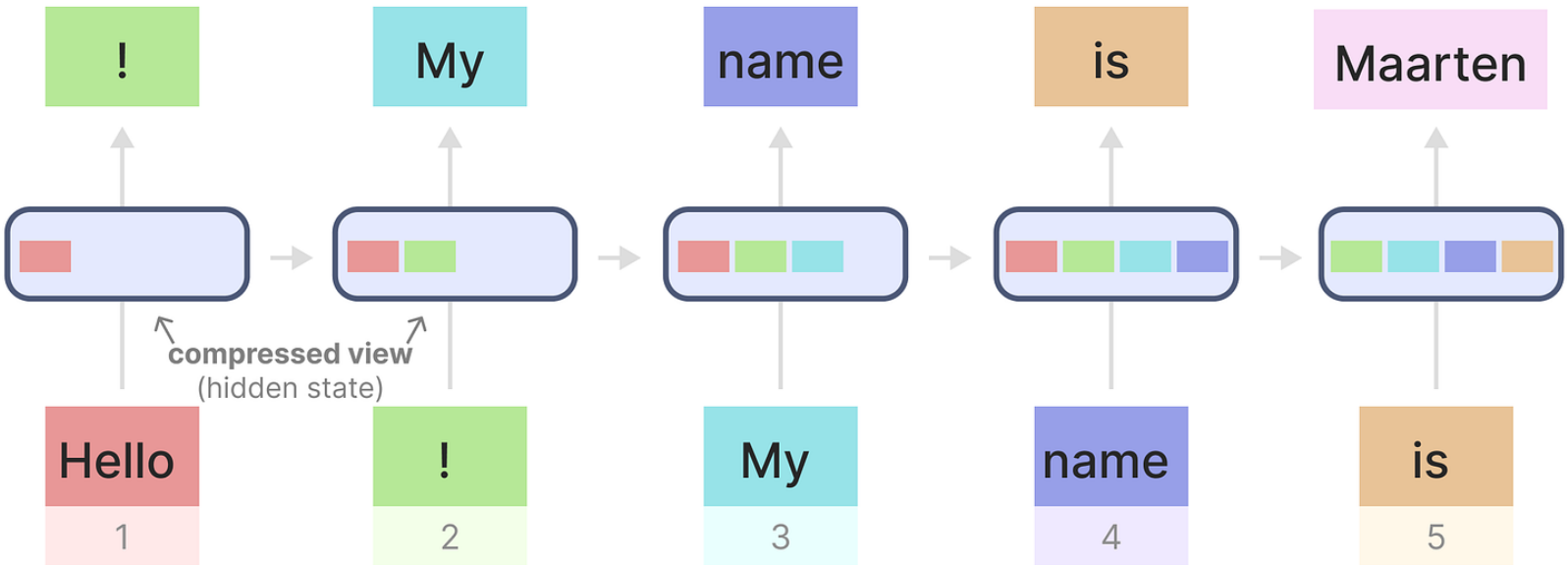
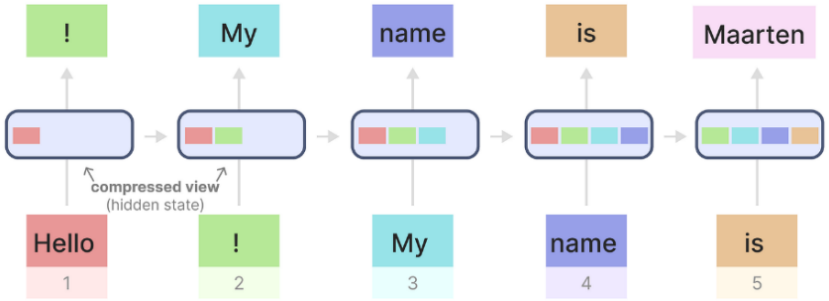


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

The Problem with RNNs

State Space Models

- RNNs suffer from forgetting information over time
  - The last hidden state to produce the name "Maarten" may not contain information about the word "Hello"



- Training cannot be done in parallel since RNNs need to go through each step at a time sequentially
  - GPUs have enormous throughput for parallel computation, but are otherwise very slow for sequential computation

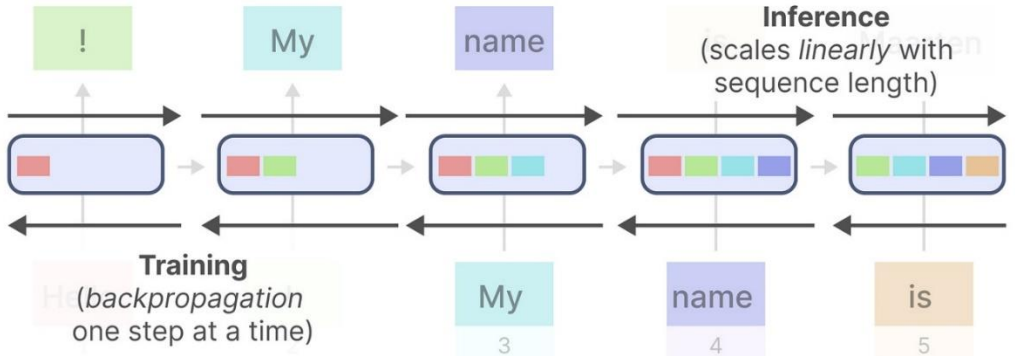


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Pros and Cons of RNNs

State Space Models

- Pros
  - Efficient autoregressive inference: since the hidden state encapsulates all prior inputs, the model only needs to consider a small set of new information for each subsequent input
  - No limits to context length at inference: there is nothing in the formulation of RNNs that constrains the model to a maximal sequence length
- Cons
  - Ineffective modeling of complex sequential dependencies: all prior context must be compressed into a hidden state having a fixed amount of bits
  - Slow training: training requires sequential backpropagation through time, making poor utilization of GPUs

Transformers vs RNNs

State Space Models

- The problems with RNNs and Transformers are completely opposite
  - Potential solution are State Space Models architectures, which allow fast inference similarly to RNNs, but can also use kernels and be parallelized during training similarly to Transformers

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)
RNNs	**Slow...** (not parallelizable)	**Fast!** (scales **linearly** with sequence length)

Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models

State Space Models

- *State Space* represents a system's possible states using the minimum number of variables
  - For instance, to navigate through a maze, the *state space* is the map (space) of all possible locations (states)
  - The "state space representation" is a simplified description of this map, consisting of the current state, possible future states, and the changes required to get to the next state (going right or left)

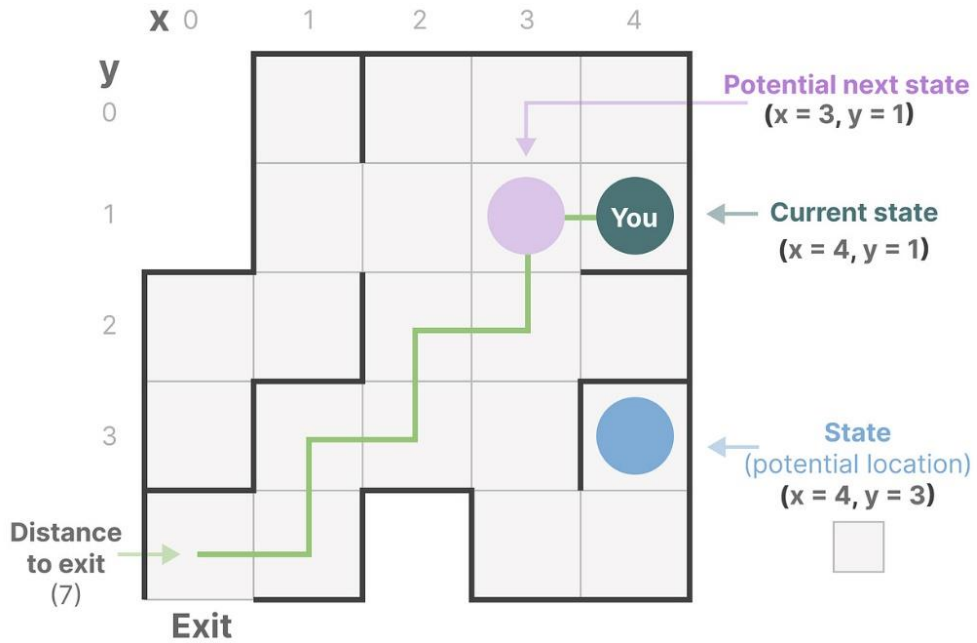


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models

State Space Models

- SSM equations describe state representations and predict future states and outputs based on inputs
  - SSM maps input sequences $x(t)$ to latent state representations $h(t)$ and derive predicted output sequences $y(t)$
  - By solving these equations, SSM can predict the system states from observed data
    - E.g., the location of an object moving in 3D space can be predicted from its state at time t through the two equations

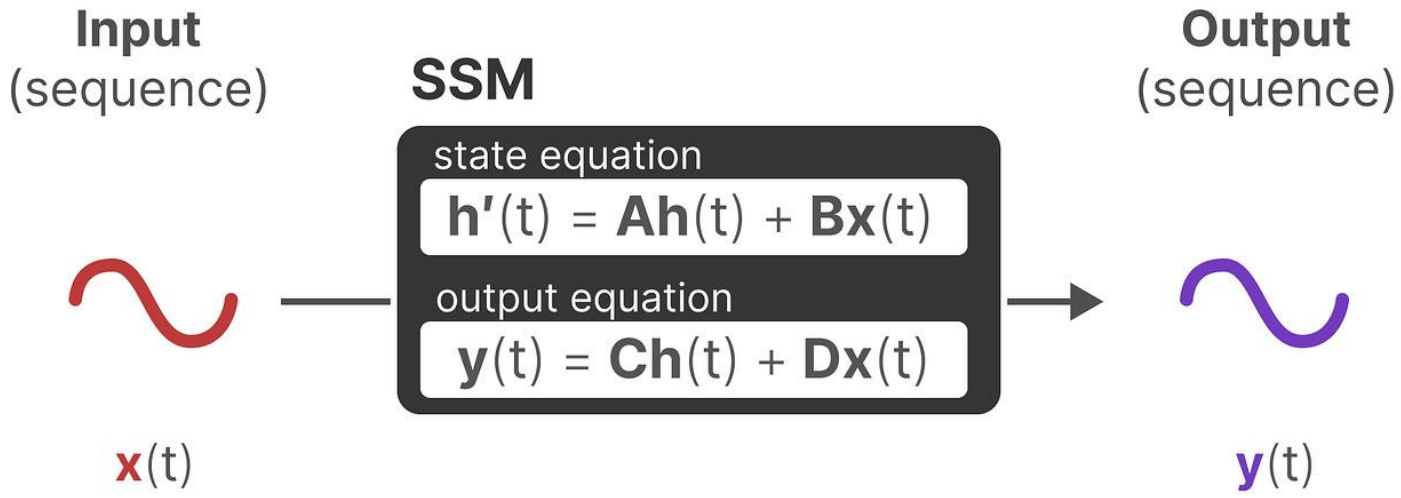


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models

State Space Models

- Components of SSM equations:
  - t represents time
  - $x(t)$ is input to the model, $h(t)$ is latent (hidden) state, and $y(t)$ is output of the model
  - The matrices are: A – state matrix, B – input matrix, C – output matrix, and D – feedforward matrix
    - The matrices comprise the parameters of the SSM, and determine how the state $h(t)$ evolves over the sequence of inputs $x(t)$

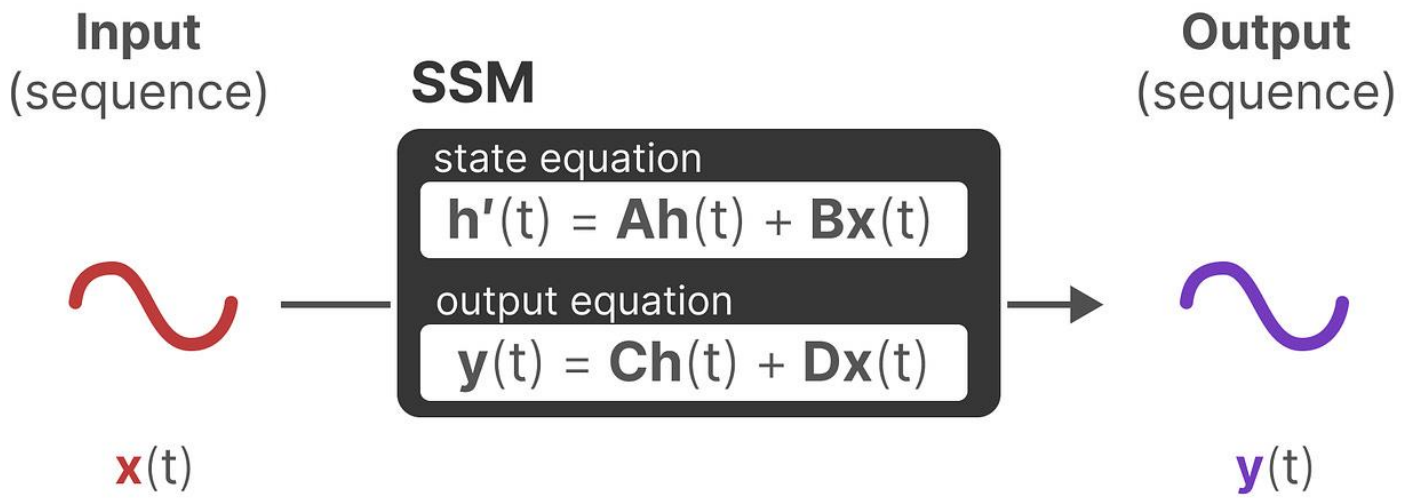
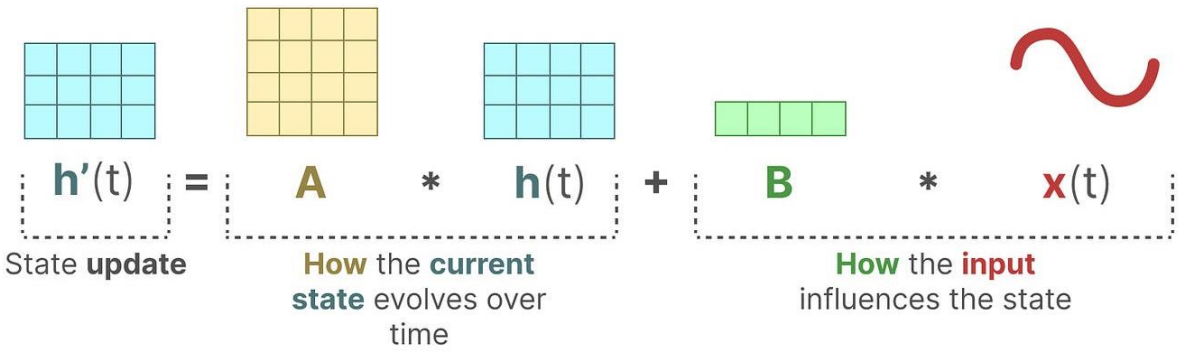


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models

State Space Models

- The differential equation $h'(t) = Ah(t) + Bx(t)$ describes how the state $h(t)$ changes (through matrix A) based on how the input $x(t)$ influences the state (through matrix B)



- The output equation $y(t) = Ch(t) + Dx(t)$ describes how the state $h(t)$ is translated to the output $y(t)$ =(through matrix C) and how the input $x(t)$ influences the output (through matrix D)

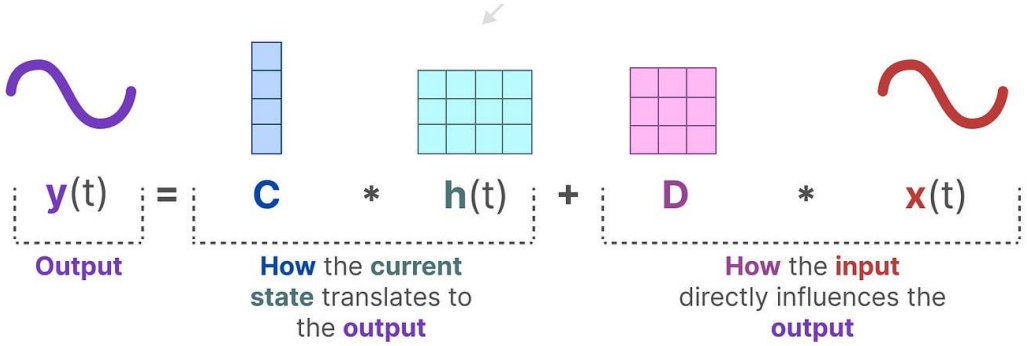


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models

State Space Models

- We can visualize the equations of SSM as in the figure below
  - Since the operation through matrix *D* is similar to a skip-connection, often the focus is only on matrices *A*, *B*, and *C* as the core of SSM

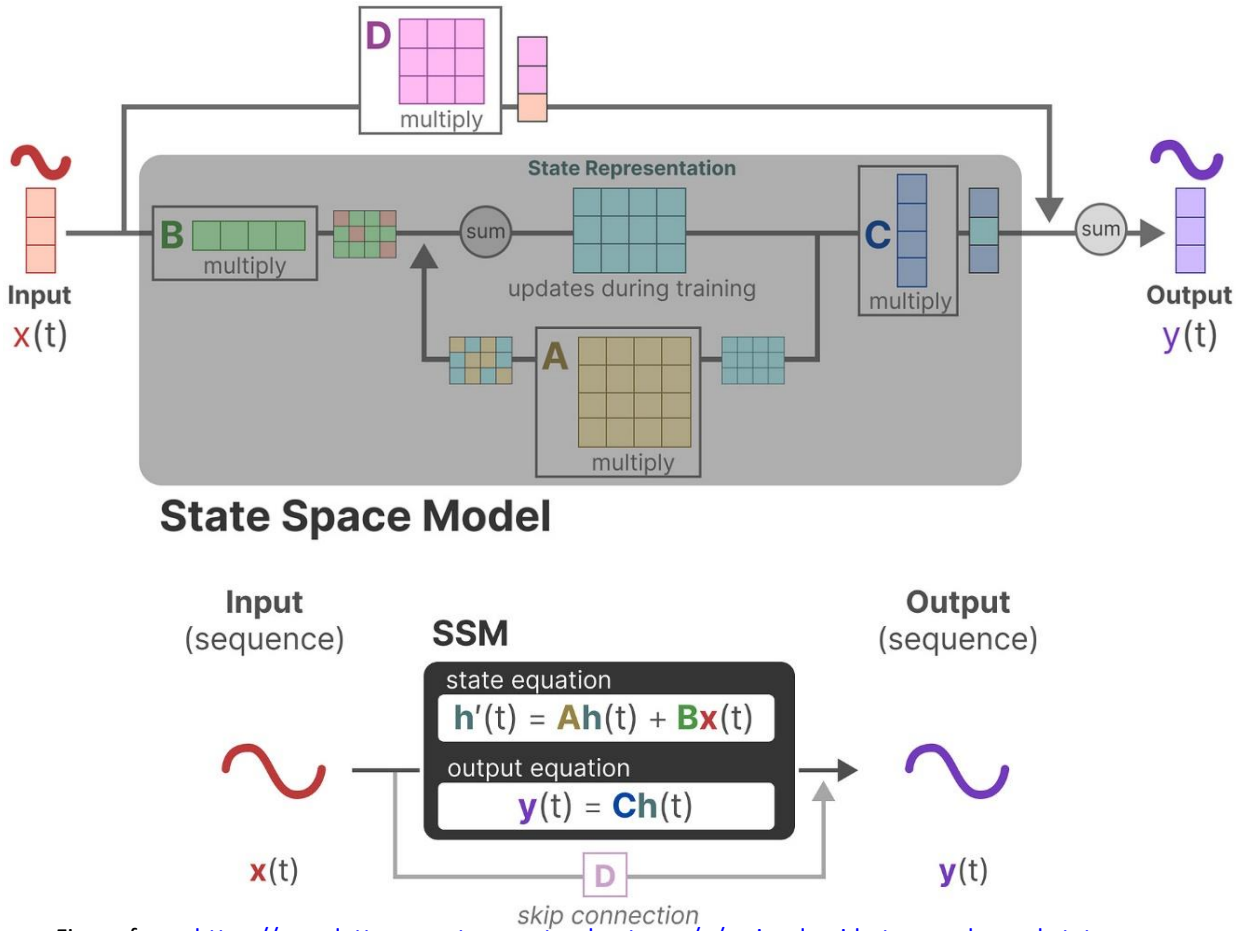


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Discrete State Space Models

State Space Models

- Because LLMs work with sequence inputs rather than continuous inputs, a discretized version of SSM is used in LLM architectures
- For instance, we can approximate a continuous signal with a discrete signal by sampling from it at discrete time moments 0, 1, 2, 3, 4, etc.

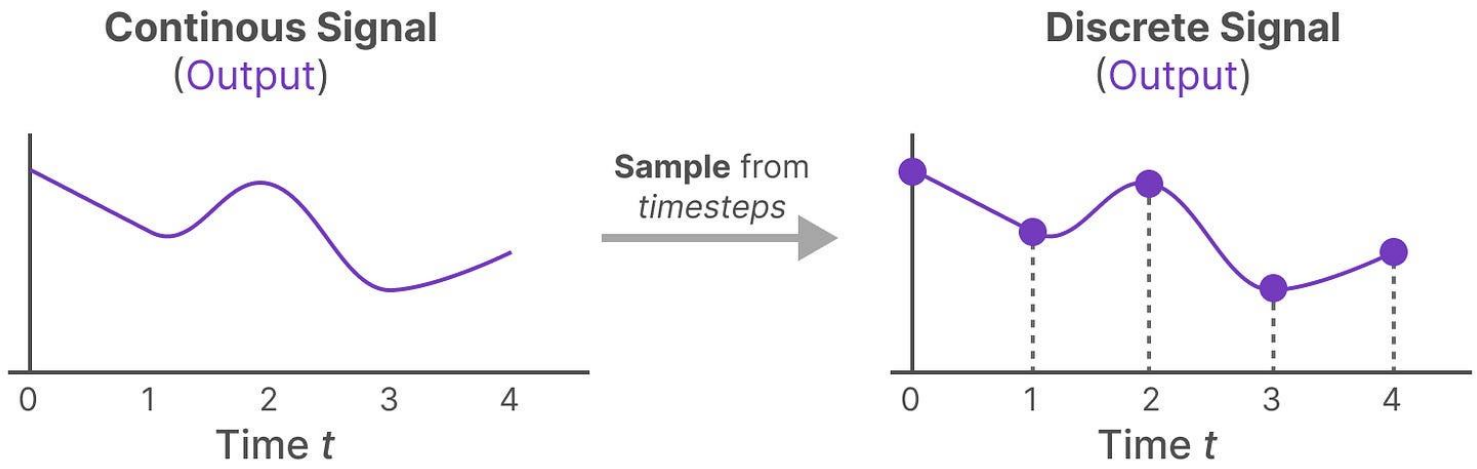


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Discrete State Space Models

State Space Models

- Similarly, we can discretize continuous SSM by approximating the vectors and matrices with discrete forms, as shown below
  - In the discrete SSM equations, the matrices $\bar{A}$ and $\bar{B}$ represent discretized parameters of the model
  - The discrete vectors x, h, and y have the subscript k for representing discretized timesteps, instead of the dependence of time t used to represent continuous vectors

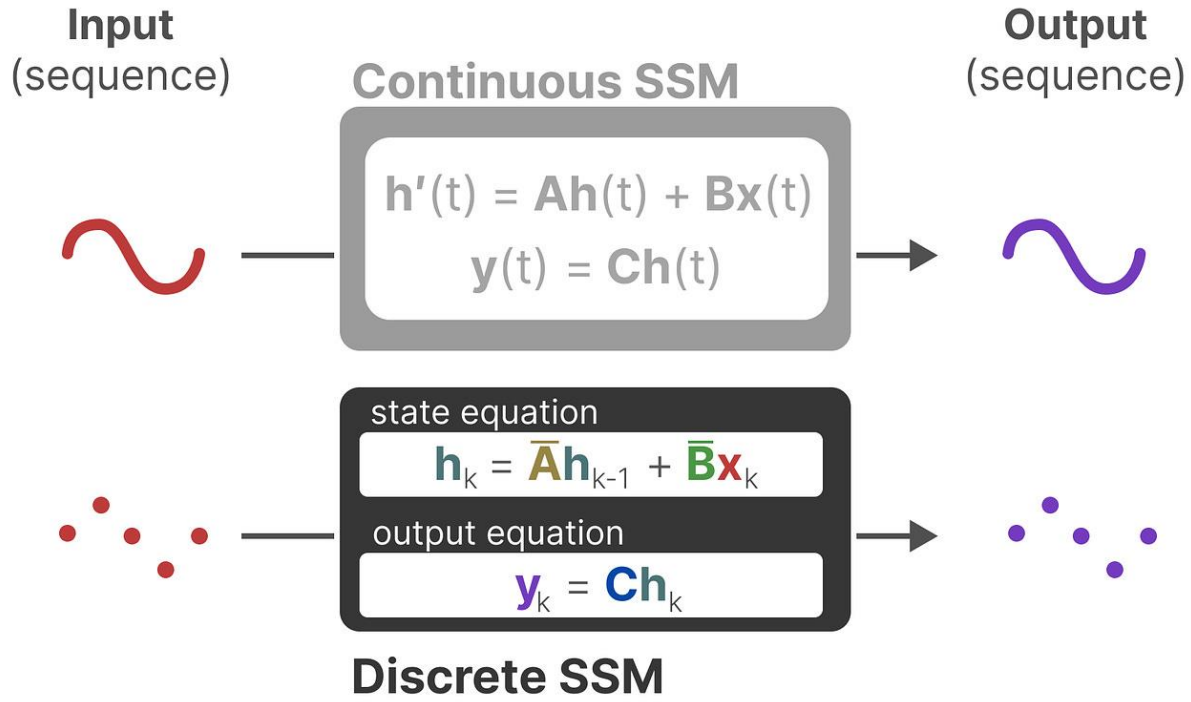


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Recurrent Representation

State Space Models

- Discretized SSM allows for problem formulation in discrete timesteps, forming recurrence similar to RNNs
  - I.e., at each timestep, we calculate how the current input ($\bar{B}\ x_k$) influences the previous state ($\bar{A}\ h_{k-1}$), and then calculate the predicted output (C h_k)

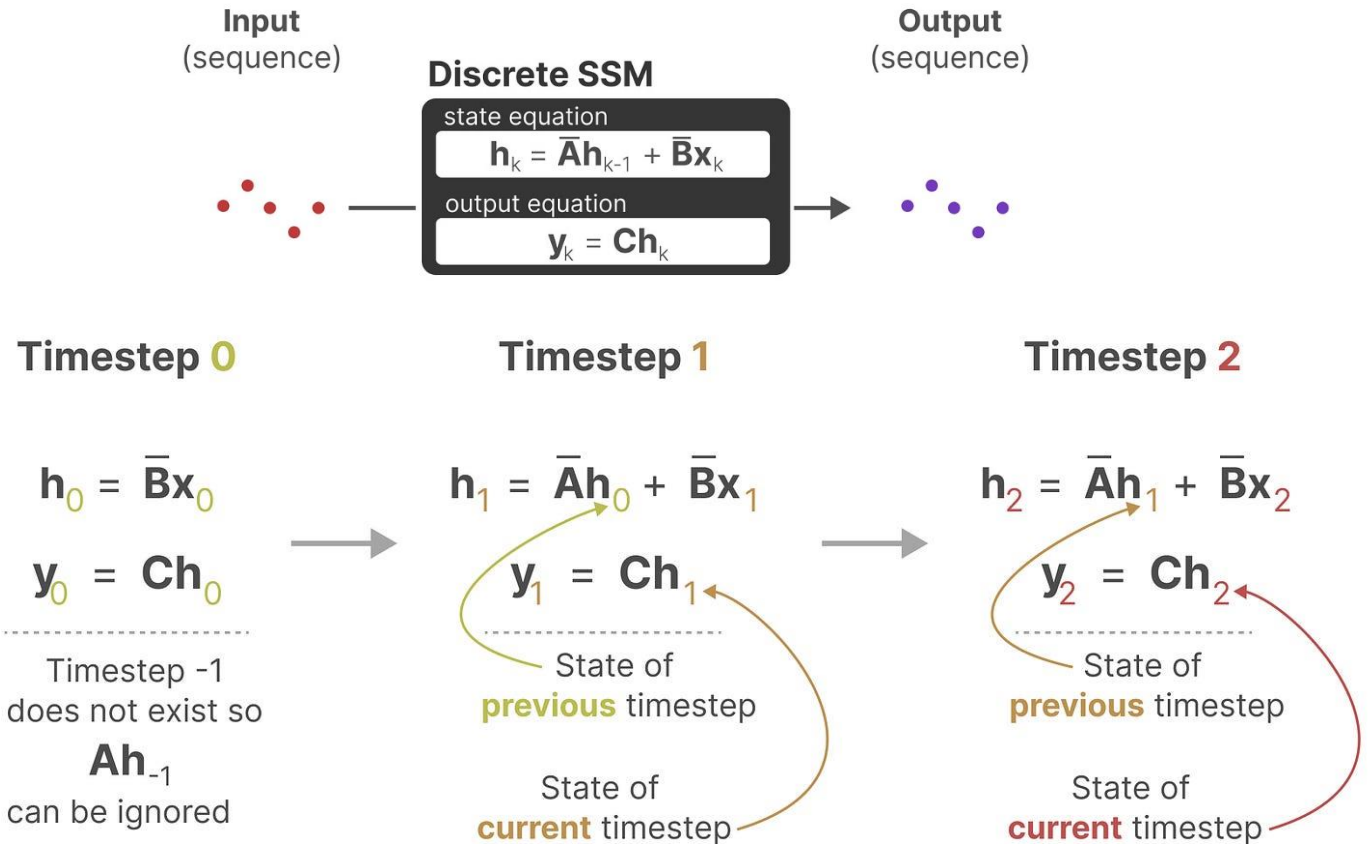


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Recurrent Representation

State Space Models

- Therefore, the representation of discrete SSM can be depicted with a graph of an RNN

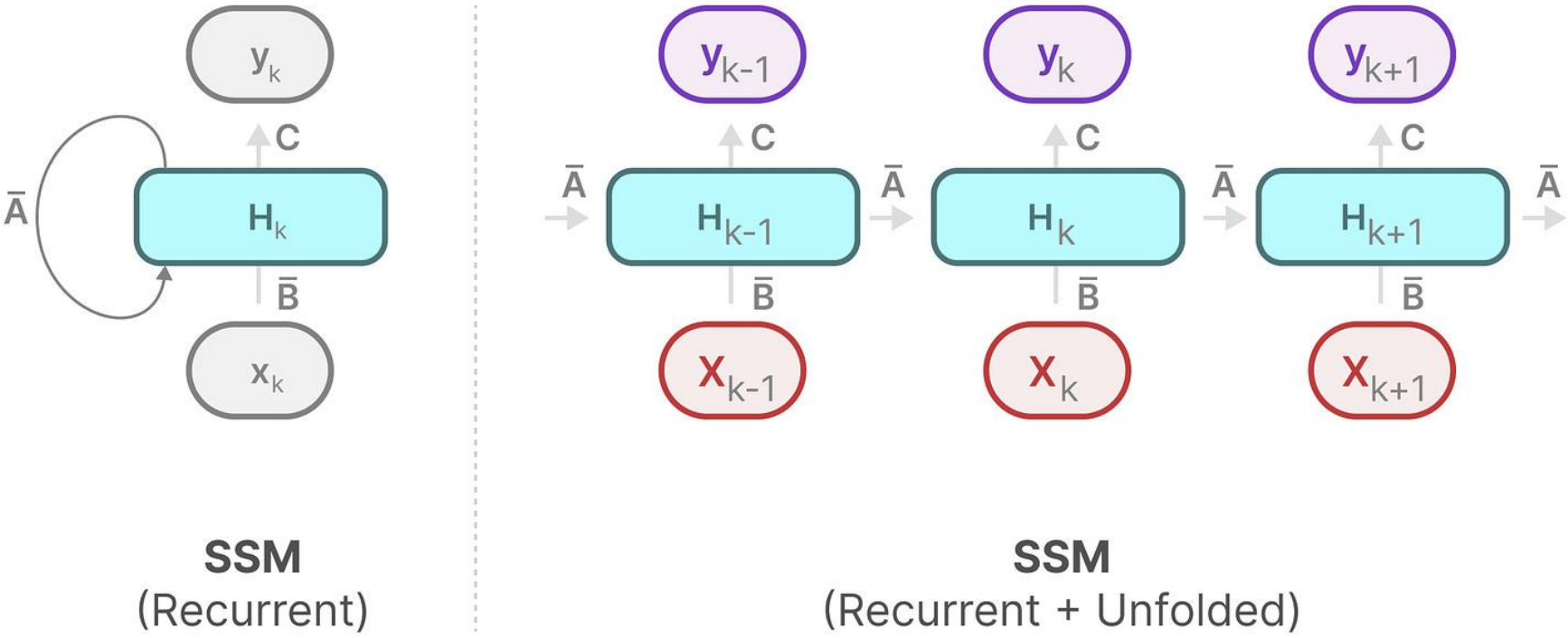


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Convolutional Representation

State Space Models

- Another representation that we can use for discrete SSM is that of convolutions
  - For instance, the figure shows convolutional filters (kernels) in Convolutional NNs applied for generating features for image classification

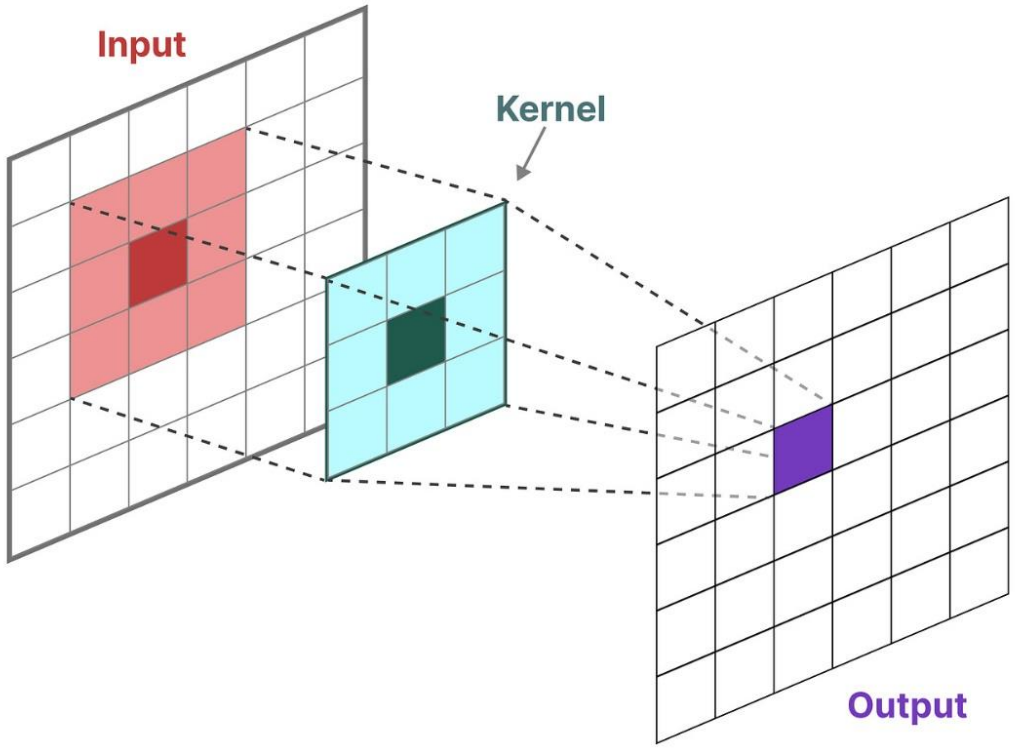


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Convolutional Representation

State Space Models

- For sequences of text which are 1-dimenisional, we can consider applying a 1-dimensional convolutional filter (kernel)
  - The 1-d kernel slides over the input sequence of tokens and it is used to calculate the output token over each position of the input sequence

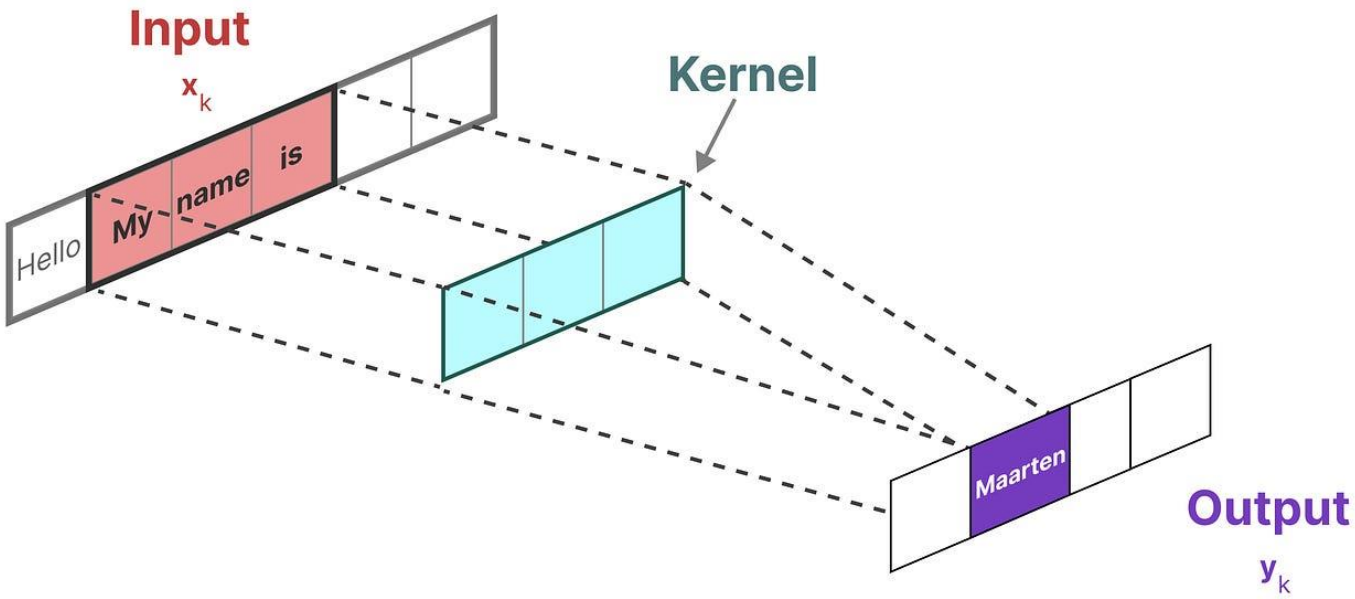


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Convolutional Representation

State Space Models

- The kernel can be derived from the SSM formulation as the product $\overline{CA^iB}$

$$\text{kernel} \longrightarrow \overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{\overline{k}}\overline{\mathbf{B}}, \ldots)$$

$$\mathbf{y} = \mathbf{x} * \overline{\mathbf{K}}$$

output input kernel

- In practice, we can use the SSM kernel to slide over each set of input tokens and calculate the output, similar to applying 1-d convolution

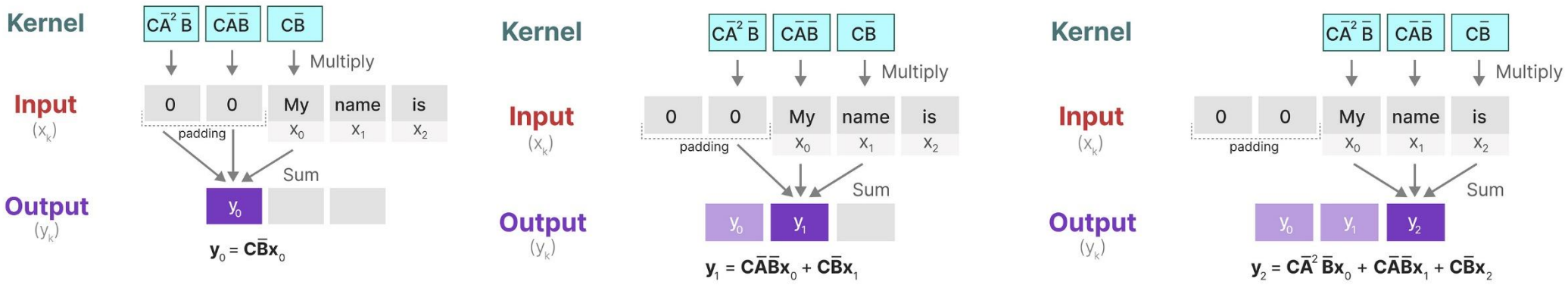


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

The Three Representations of SSM

State Space Models

- The representations of SSM as recurrent and convolutional systems have different sets of advantages and disadvantages
  - Interestingly, the different representations of SSM allow to have efficient inference with the recurrent SSM and parallelizable training with the convolutional SSM

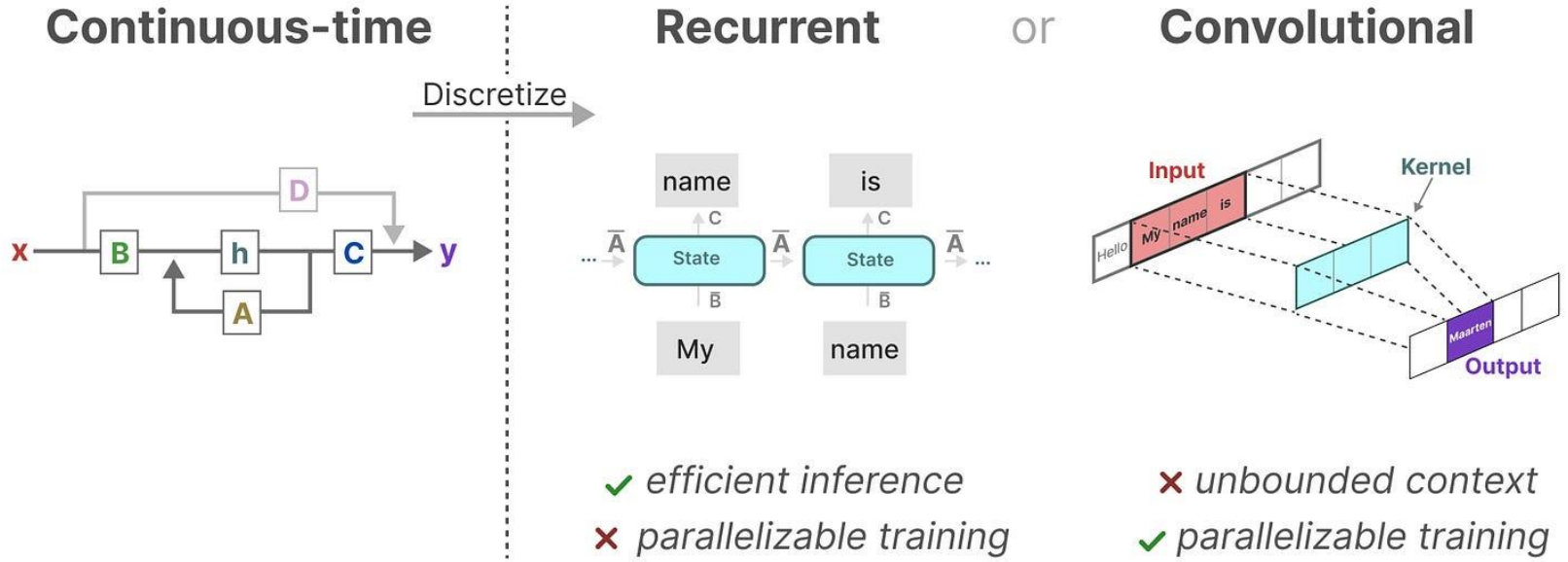


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

The Three Representations

State Space Models

- With these different representations of SSM:
  - During training the model uses the convolutional representation over the entire sequences of tokens for parallelized training
  - During inference the model uses the recurrent representation to make next token predictions for each new token at a time

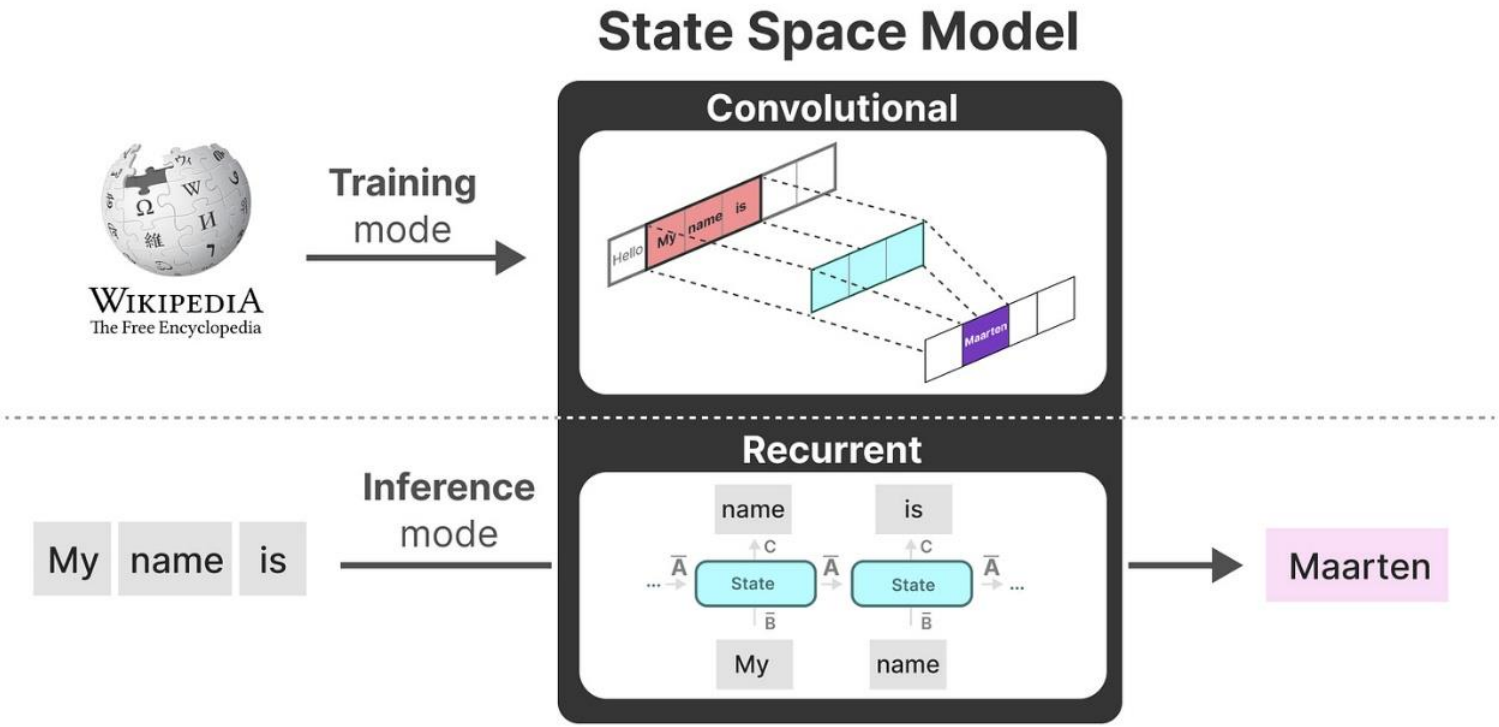


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

S4 Architecture

State Space Models

- Despite the promises of SSM for text modeling, there are challenges for handling long sequences and capturing long-range dependencies
- *Structured State Space for Sequences* architecture (a.k.a. *S4*) proposed a novel solution for handling dependencies in long sequences
  - S4 parameterizes the matrix A as a diagonal matrix, initialized with approximations of the HiPPO (High-order Polynomial Project Operators) matrix
    - Imposing a HiPPO-based structure on the matrix A (hence the name "structured" in S4) enables to memorize previous tokens, by tracking the coefficients of a Legendre polynomial
    - By using a HiPPO matrix structure, S4 architecture allows for handling long sequences and storing memory efficiently

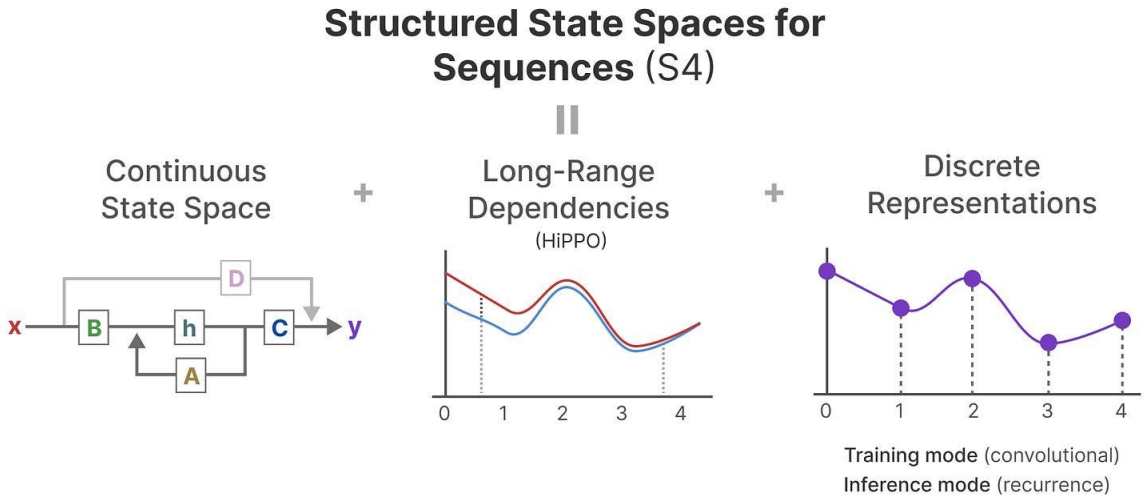


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba

State Space Models

- *Mamba* builds upon the S4 architecture, and introduced two important modifications to S4
  1. Selective scan SSM parameters, which allows the model to filter irrelevant information
  2. Efficient hardware implementation, that allows for efficient GPU storage of intermediate results
- Importantly, Mamba demonstrated strong empirical results
- Mamba architecture is also referred to as a Selective Scan SSM (or S6) since it extends the S4 model for computations of the SSM matrices with the selective scan algorithm

Mamba -Departure from LTI

State Space Models

- Departure from Linear Time Invariance (LTI) in S4
  - Matrices A, B, and C in S4 are time-invariant, meaning that they are the same for every token (i.e., across all time steps in a sequence)
    - These matrices are fixed, regardless of the input
    - As a result, the model treats all tokens equally, and it cannot perform content-aware reasoning
  - Conversely, content-aware reasoning is easy for Transformers, because the attention scores are dynamically calculated based on the input sequence
- Mamba introduces selective scan algorithm to dynamically calculate matrices A, B, and C as time-varying matrices that change for every input token
  - This allows to selectively choose what information to keep in the hidden state, and what information to ignore
  - The matrices learn which input tokens are the most important at each step, hence the name selective scan

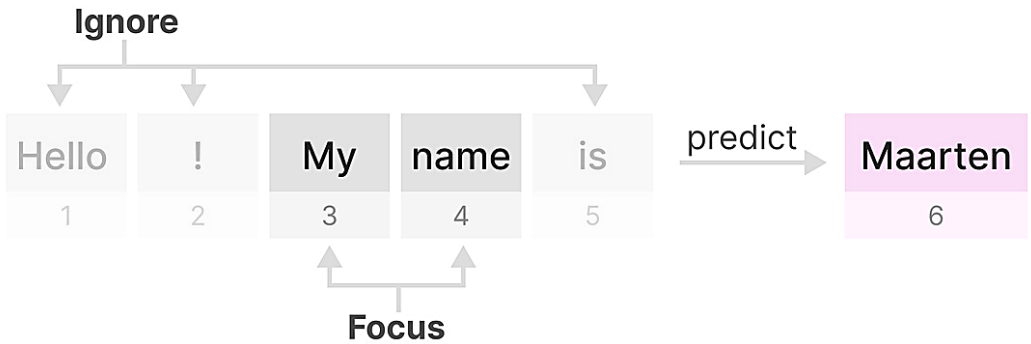


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba – Hardware-aware Resource Management

State Space Models

- Another novelty that Mamba introduces is hardware-aware parallelism and optimized memory usage by controlling the data transfer between the highly-efficient SRAM memory and less-efficient DRAM memory in GPUs
  - Copying information between SRAM and DRAM is often a bottleneck in GPUs
  - The hardware-aware component focuses on how to store the latent state *h* in the SRAM memory as the most efficient part of memory
  - The A, B, and C matrices are kept in the DRAM memory, so that the cost of moving data doesn't induce a large computational bottleneck

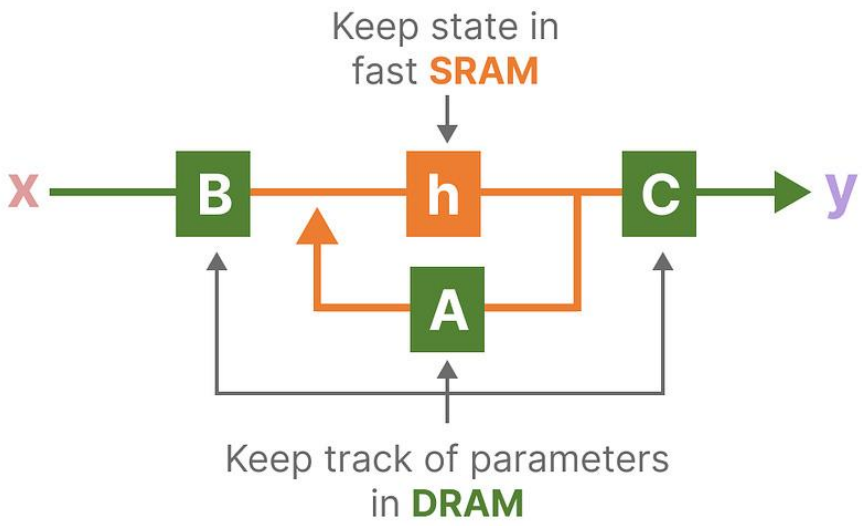


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba Architecture

State Space Models

- Mamba implements Selective SSM as a block, where multiple Mamba blocks can be stack to create a large model
  - The block applies linear (dense) layers to input embedding vectors, followed by convolution, Selective scan layer, and linear layer
  - SiLU (Sigmoid Linear Unit) activation are used, which for input x output $x\sigma(x)$, where $\sigma(x)$ is sigmoid function
  - The output is projected though RMS Norm and Linear layer with Softmax
    - RMS Norm is Root Mean Square layer normalization layer
    - The RMS Norm layer replaces Layer Normalization layers used in Transformers

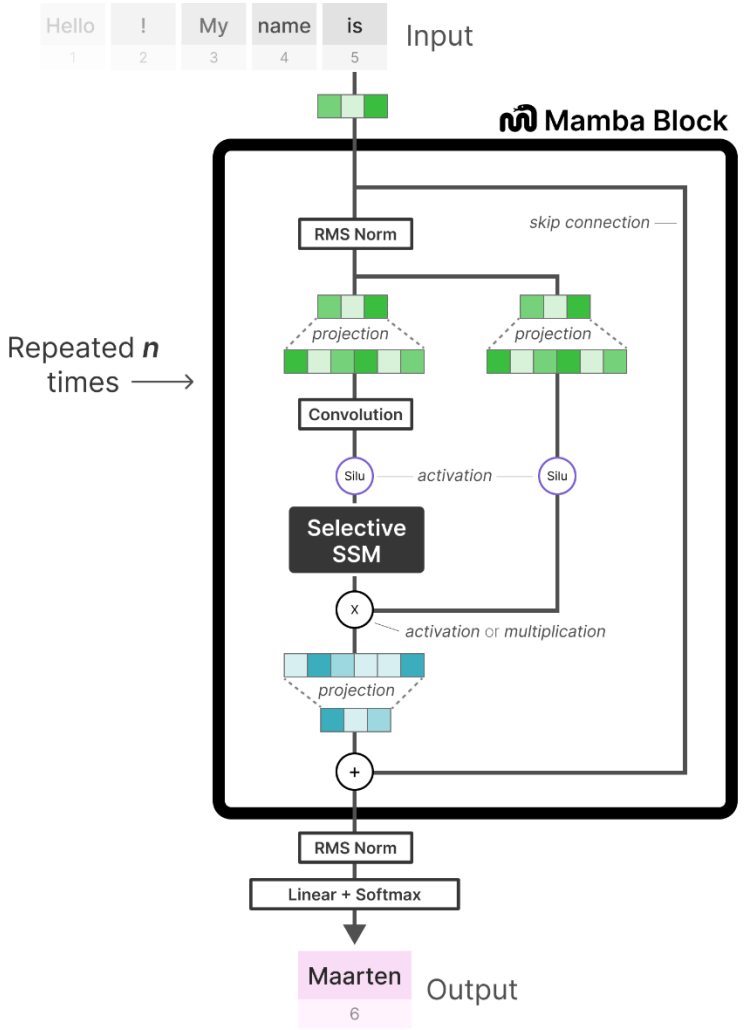
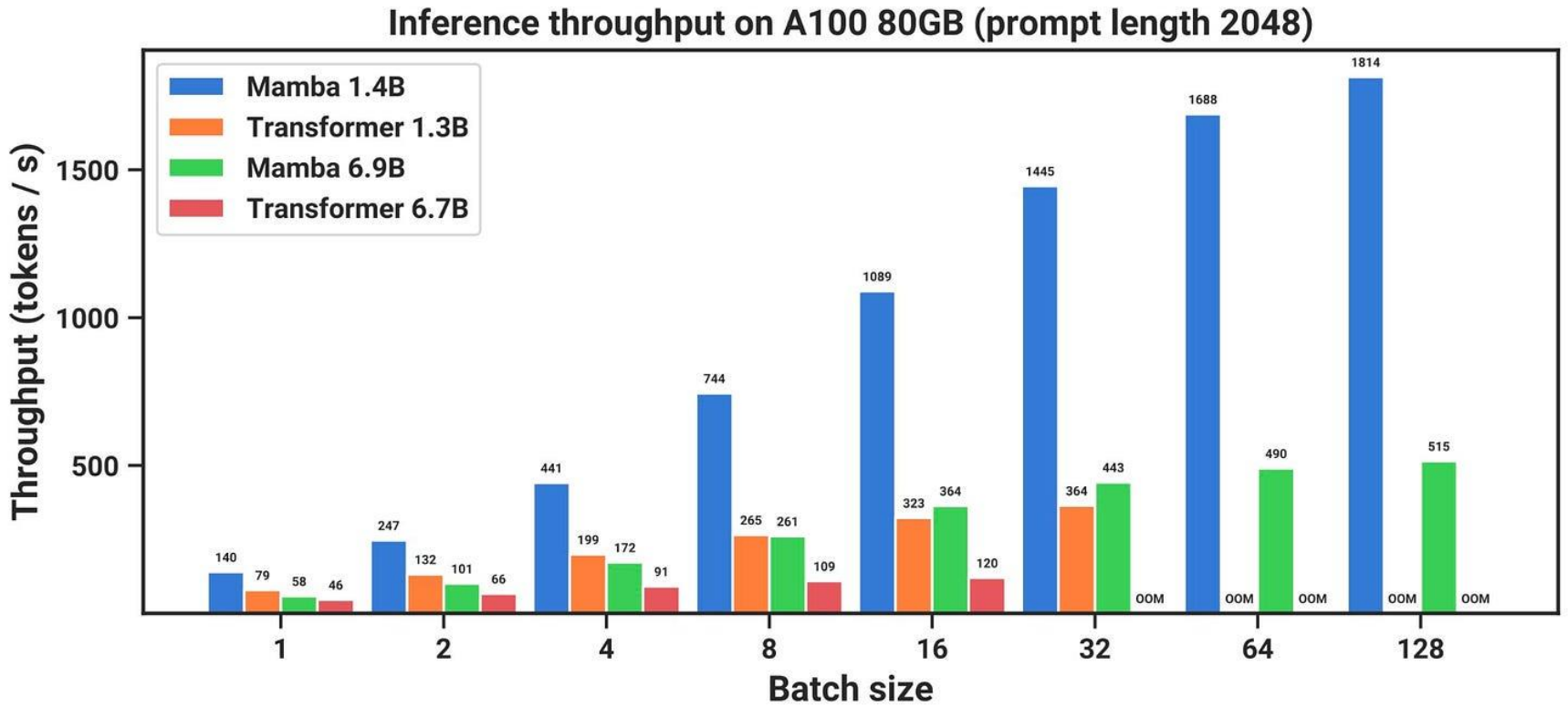


Figure from: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba Results

State Space Models

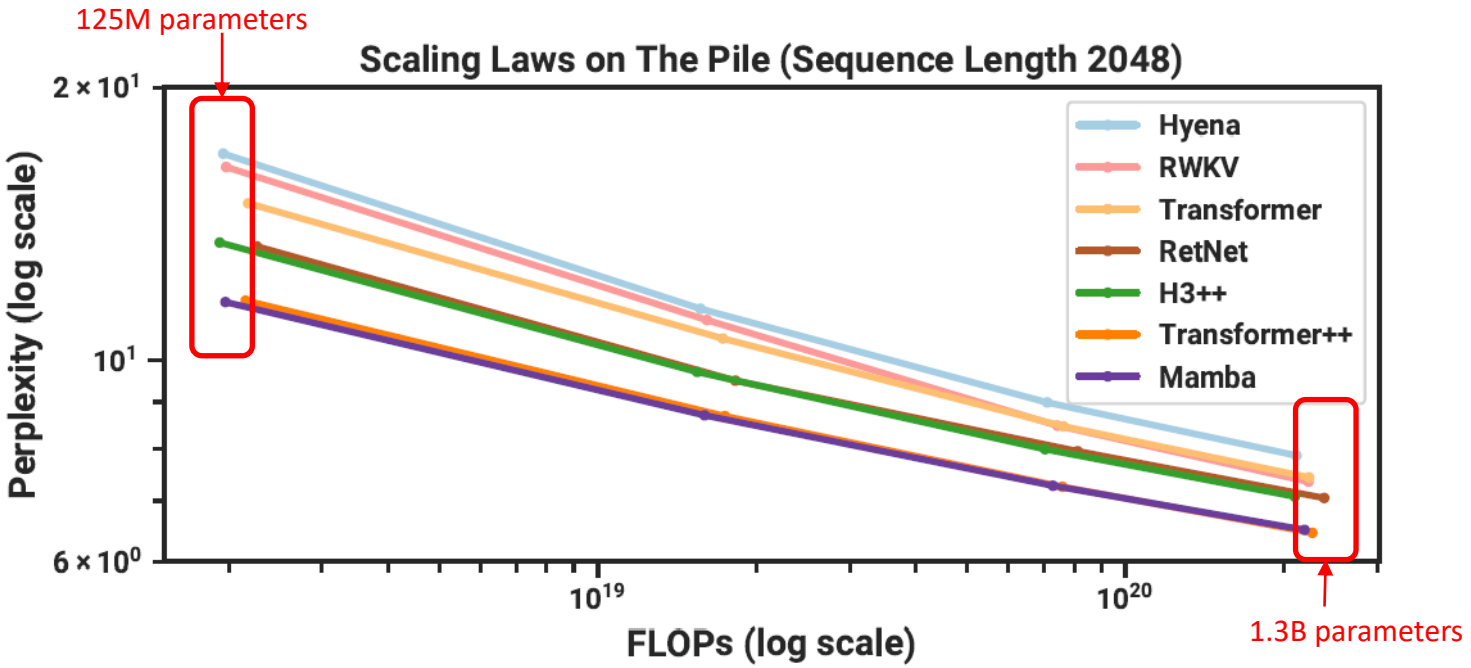
- The figure below shows a comparison of generated tokens with Mamba and corresponding Transformers
  - Mamba offers significant speed up at inference, and it achieves 4-5x higher inference throughput than a Transformer of similar size
  - Inference was performed with Mamba on one million token sequences



Mamba Results

State Space Models

- The graph below shows the perplexity of different LLMs as a function of the required computation (FLOPs) for model training
  - Recall that lower perplexity is preferred
  - The size of the models is changed from 125M to 1.3B parameters
  - Mamba uses less computational resources, in comparison to other LLMs architectures, including the original Transformer and Transformer++ (based on PaLM and LLaMA)



Mamba Results

State Space Models

- Mamba outperformed models of similar size on language, audio, and genomics data
  - It is the first non-Transformer architecture that achieved performance at the level of Transformers architectures
- One limitation of this study is that Mamba was implemented with 3B and 6.9B parameters
  - Large Mamba models with size that correspond to GPT-4, Claude, Gemini were not implemented
  - It is not clear if the performance by Mamba architecture can match the Transformers at the size of 100+ billion parameters
- Another open question is regarding the properties of Mamba models regarding finetuning, quantization, instruction tuning, RLHF, etc.

References

1. Miguel Carreira Neves, "LLM Mixture of Experts Explained," available at https://www.tensorops.ai/post/what-is-mixture-of-experts-llm
2. Omar Sanseviero, et al., "Mixture of Experts Explained," available at https://huggingface.co/blog/moe
3. Kyle Kranen and Vinh Nguyen, "Applying Mixture of Experts in LLM Architectures," available at https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/
4. Maarten Grootendorst, "A Visual Guide to Mamba and State Space Models," available at https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state
5. James Chen, "Mamba No. 5 (A Little Bit of …)," available at https://jameschen.io/jekyll/update/2024/02/12/mamba.html
6. Nathan Lambert, "State-space LLMs: Do we need Attention?," available at https://www.interconnects.ai/p/llms-beyond-attention