# Revisiting Other Language Model Architectures

Presented by: Kellan Duan, Jack Belmont, Kevin Kotzbauer, and Rylan Tang

# Mixtral of Experts
# Jiang et al.

Combining Mixture of Experts with Transformers

# Motivation

Large Dense models such as Mistral 7B or other models scale well but require costly computation at inference.

How can we improve these models?

# Mixtral

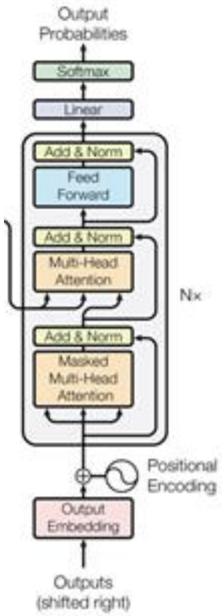
Mixtral 8x7B introduces a sparse mixture of Experts architecture.

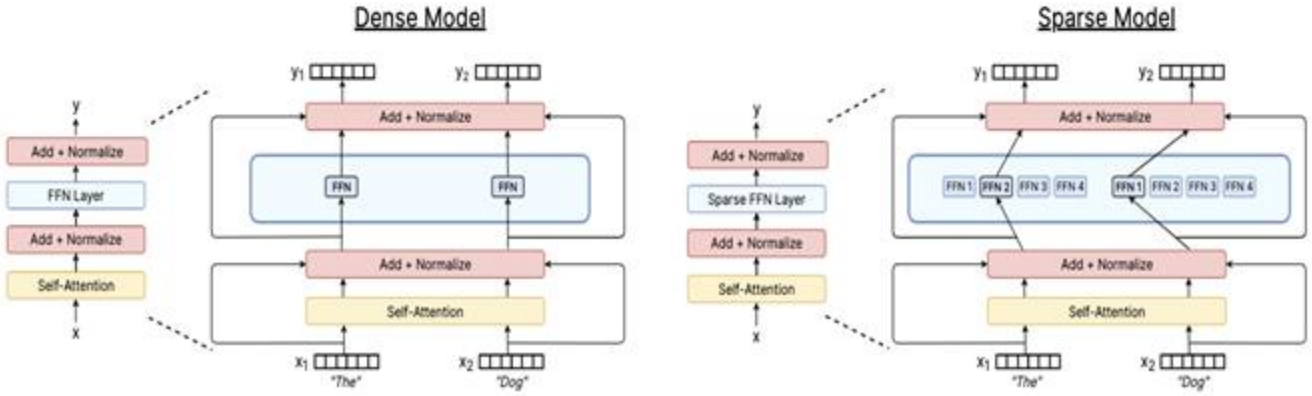
- **Decoder** only model.
- 47B Total parameters, 13B active parameters.
- Better performance than Llama2 70B and GPT3.5 on most benchmarks
- Trained w/32k context window

# More on Mixtral

- Each layer has 8 experts.
- Router selects top 2 experts token, wise.
- Mixtral demonstrates superior capabilities in mathematics, code generation, and tasks that require multilingual understanding, significantly outperforming Llama 2 70B, while being more efficient in doing so.

# Quick Review on Transformers/Relate it to Mixtral

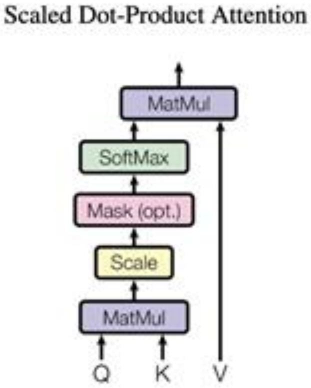


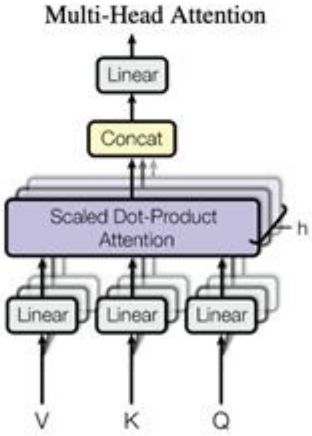



# Brief overview of model architecture

| Parameter | Value |
|---|---|
| `dim` | 4096 |
| `n_layers` | 32 |
| `head_dim` | 128 |
| `hidden_dim` | 14336 |
| `n_heads` | 32 |
| `n_kv_heads` | 8 |
| `context_len` | 32768 |
| `vocab_size` | 32000 |
| `num_experts` | 8 |
| `top_k_experts` | 2 |

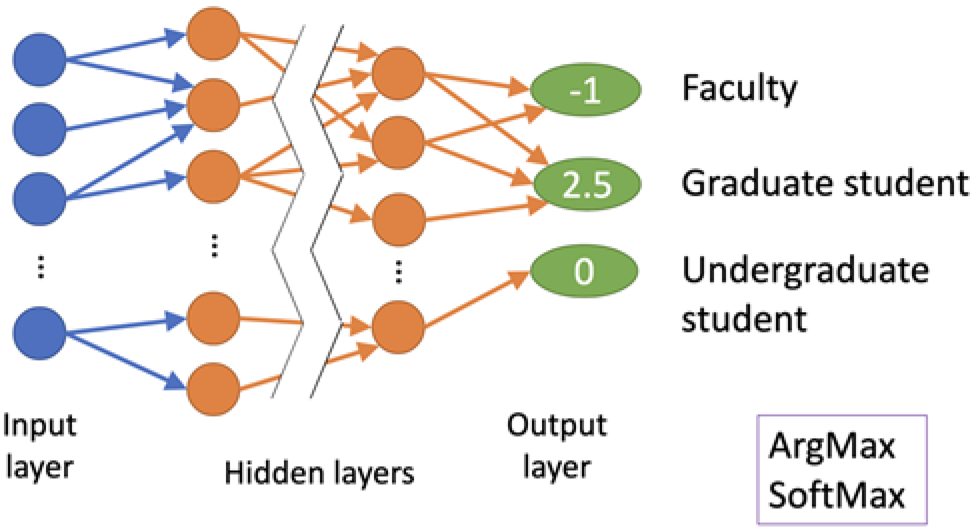
**Table 1: Model architecture.**



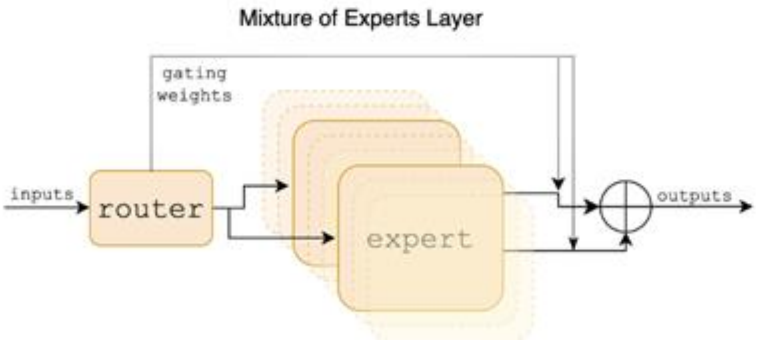



# How do routers work?

- Q/A: How does Mixtral's router learn to select the right expert for each token ?



# Mixture of Experts Layer



## How routers pick Top 2 Experts.

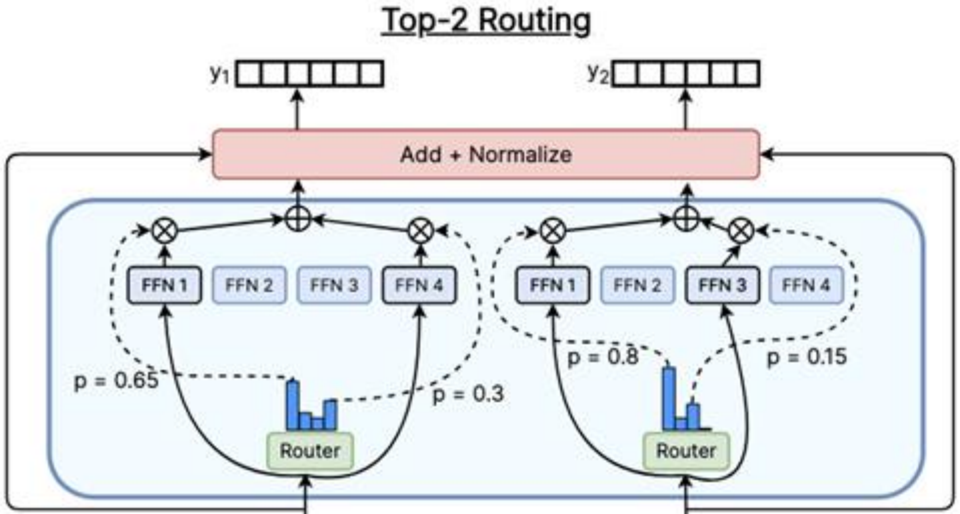
$$\sum_{i=0}^{n-1} G(x)_i \cdot E_i(x).$$

$$\mathrm{Softmax}(\mathrm{TopK}(x \cdot W_g))$$

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}.$$

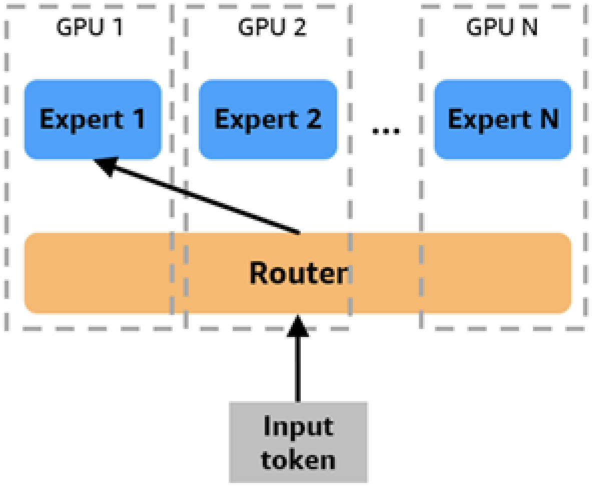
# Top-2 Routing

- Mixtral utilizes Top-2 Routing



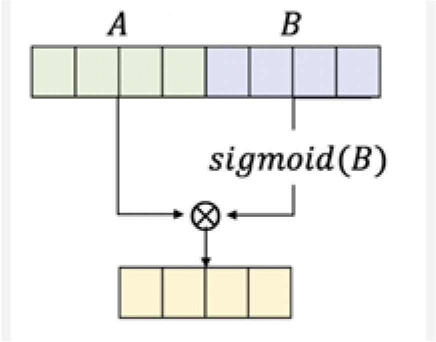
# Expert Parallelism

- Expert Parallelism is a partitioning strategy where the MoE layer can be distributed to multiple GPUs through Model Parallelism Techniques.
- EP introduces challenges in load balancing.



# SwiGLU architecture

$$y = \sum_{i=0}^{n-1} \mathrm{Softmax}(\mathrm{Top2}(x \cdot W_g))_i \cdot \mathrm{SwiGLU}_i(x).$$



# Results-MIxtral on wide range of benchmarks

- Mixtral displays a superior performance in code and mathematics
- More importantly, it does well with fewer parameters (and a lot fewer active parameters)

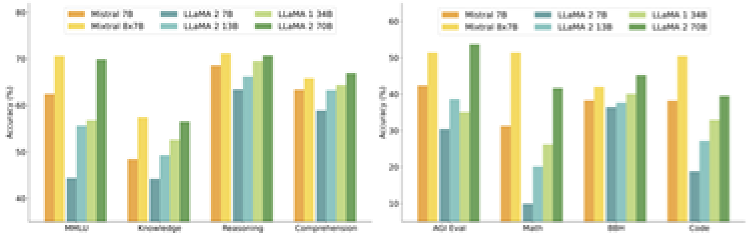



**Figure 2: Performance of Mixtral and different Llama models on a wide range of benchmarks.** All models were re-evaluated on all metrics with our evaluation pipeline for accurate comparison. Mixtral outperforms or matches Llama 2 70B on all benchmarks. In particular, it is vastly superior in mathematics and code generation.

# How does sparse MoE routing affect performance on MMLU and HellaSwag?

- On MMLU, Mixtral obtains a better performance despite a significantly smaller capacity.
- Average expert utilization rate is always 2/8 since there are 2 experts/8 experts being utilized.

| | 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 |

**Table 3: Comparison of Mixtral with Llama 2 70B and GPT-3.5.** Mixtral outperforms or matches Llama 2 70B and GPT-3.5 performance on most metrics.

# Multilingual benchmark Results

- We see that mixtral 8x7B outperforms Llama 2 70B and Llama 1 33B on all 4 languages provided here.

| Model | Active Params | French Arc-c | French HellaS | French MMLU | German Arc-c | German HellaS | German MMLU | Spanish Arc-c | Spanish HellaS | Spanish MMLU | Italian Arc-c | Italian HellaS | Italian MMLU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **LLaMA 1 33B** | 33B | 39.3% | 68.1% | 49.9% | 41.1% | 63.3% | 48.7% | 45.7% | 69.8% | 52.3% | 42.9% | 65.4% | 49.0% |
| **LLaMA 2 70B** | 70B | 49.9% | 72.5% | 64.3% | 47.3% | 68.7% | 64.2% | 50.5% | 74.5% | 66.0% | 49.4% | 70.9% | 65.1% |
| **Mixtral 8x7B** | 13B | **58.2%** | **77.4%** | **70.9%** | **54.3%** | **73.0%** | **71.5%** | **55.4%** | **77.6%** | **72.5%** | **52.8%** | **75.1%** | **70.9%** |

**Table 4: Comparison of Mixtral with Llama on Multilingual Benchmarks.** On ARC Challenge, Hellaswag, and MMLU, Mixtral outperforms Llama 2 70B on 4 languages: French, German, Spanish, and Italian.

# Mixtral does better with LESS active PARAMETERS!

- Active parameters are parameters that are used for processing an individual token in the feed forward network layer in the transformer here during inference.

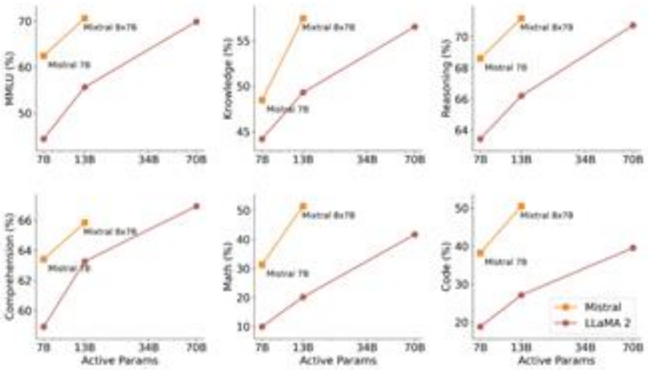
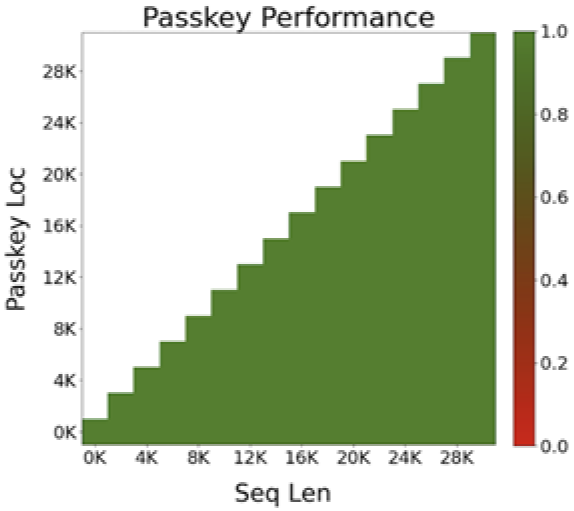



**Figure 3: Results on MMLU, commonsense reasoning, world knowledge and reading comprehension, math and code for Mistral (7B/8x7B) vs Llama 2 (7B/13B/70B).** Mixtral largely outperforms Llama 2 70B on all benchmarks, except on reading comprehension benchmarks while using 5x lower active parameters. It is also vastly superior to Llama 2 70B on code and math.

# Mixtral not affected by location of passkey or length of input sequence.

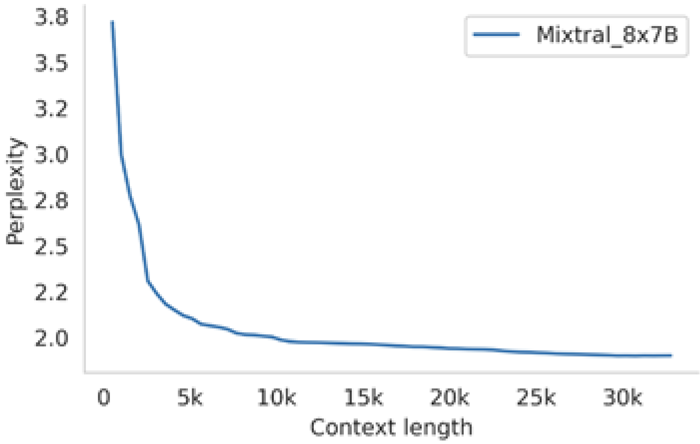
- Passkey retrieval task: Measures the ability of a model to retrieve a passkey inserted randomly in a long prompt.



# A good analogy/comparison for passkey retrieval

# Perplexity decreases as context length increases

- Perplexity predicts how well the Mixtral model predicts the next token in a sequence from dataset.



# Mixtral has less bias and more positive sentiment

- BBQ stands for Bias Benchmark for QA
- BOLD stands for Bias in Open-Ended Language Generation Dataset

| | Llama 2 70B | Mixtral 8x7B |
|---|---|---|
| BBQ accuracy | 51.5% | 56.0% |
| BOLD sentiment score (avg $\pm$ std) | | |
| gender | $0.293 \pm 0.073$ | $0.323 \pm 0.045$ |
| profession | $0.218 \pm 0.073$ | $0.243 \pm 0.087$ |
| religious_ideology | $0.188 \pm 0.133$ | $0.144 \pm 0.089$ |
| political_ideology | $0.149 \pm 0.140$ | $0.186 \pm 0.146$ |
| race | $0.232 \pm 0.049$ | $0.232 \pm 0.052$ |

**Figure 5: Bias Benchmarks.** Compared Llama 2 70B, Mixtral presents less bias (higher accuracy on BBQ, lower std on BOLD) and displays more positive sentiment (higher avg on BOLD).

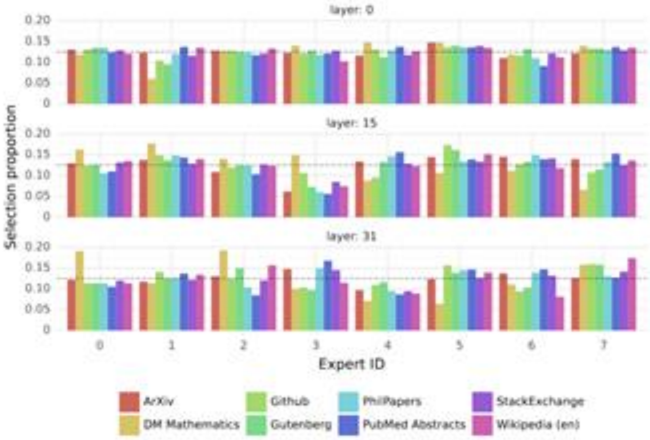
# Mixtral-Instruct

- Mixtral-Instruct is the best open-weights model and it outperforms models including GPT-3.5-Turbo, Gemini Pro, Claude-2.1, Llama 2 70B, etc.
- Apache 2.0 License makes it super easy to access!

| Model | Arena Elo rating | MT-bench (score) | License |
|---|---|---|---|
| GPT-4-Turbo | 1243 | 9.32 | Proprietary |
| GPT-4-0314 | 1192 | 8.96 | Proprietary |
| GPT-4-0613 | 1158 | 9.18 | Proprietary |
| Claude-1 | 1149 | 7.9 | Proprietary |
| Claude-2.0 | 1131 | 8.06 | Proprietary |
| Mixtral-8x7b-Instruct-v0.1 | 1121 | 8.3 | Apache 2.0 |
| Claude-2.1 | 1117 | 8.18 | Proprietary |
| GPT-3.5-Turbo-0613 | 1117 | 8.39 | Proprietary |
| Gemini Pro | 1111 | | Proprietary |
| Claude-Instant-1 | 1110 | 7.85 | Proprietary |
| Tulu-2-DPO-70B | 1110 | 7.89 | AI2 ImpACT Low-risk |
| Yi-34B-Chat | 1110 | | Yi License |
| GPT-3.5-Turbo-0314 | 1105 | 7.94 | Proprietary |
| Llama-2-70b-chat | 1077 | 6.86 | Llama 2 Community |

# Proportion of tokens assigned to each expert on different domains

- Surprisingly, we do not observe obvious patterns in the assignment of experts based on the topic.



# Consecutive tokens are often assigned the same experts.

- You can see the same tokens being assigned to the same experts. Green color region overlaps gray region. Selection of experts are more aligned with syntax than domain.

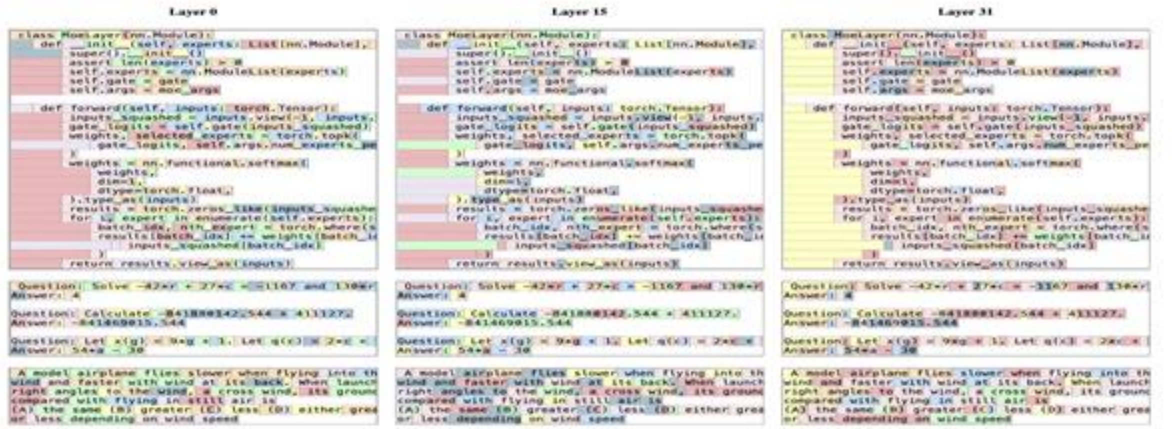
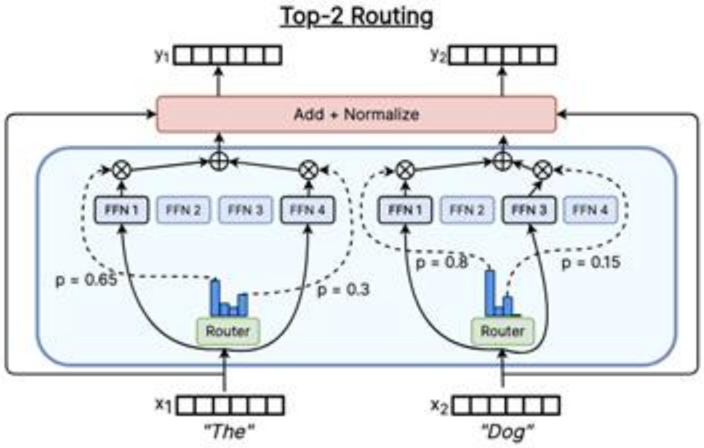



**Figure 8: Text samples where each token is colored with the first expert choice.** The selection of experts appears to be more aligned with the syntax rather than the domain, especially at the initial and final layers.

# Importance of Load Balancing!

- Answer to this question: How does Mixtral ensure balanced usage among experts and prevent over- or under-specialization?



# Possible Mixture Of Experts/Load Balancing Analogy

# Limitations

- Active parameter count reflects compute, but memory cost depends on full 47B sparse params.
- Memory overhead remains high compared to dense 13B parameter models.
- SMOE routing adds overhead.
- Running >1 expert per GPU increases memory traffic.
- Best suited for large batched workloads, small batches may underutilize hardware.

# Takeaways

- Mixtral 8x7B achieves better performance with far lower computations though its sparse mixture of expert design which boosts capacity (more parameters) without increasing per token computations (since only active parameters are utilized) during the feed forward network part of transformer.
- Mixtral 8x7B has significant strengths in math, coding, reasoning, and multilingual tasks and also has good long-context ability.
- Mixtral Instruct also has good performance.

# Transformers are SSMs via Structured State Space Duality (SSD)
# *Tri Dao & Albert Gu*

Unifying Transformers and State-Space Models: SSD shows attention kernels and SSM recurrences are two contraction orders of the same semiseparable operator

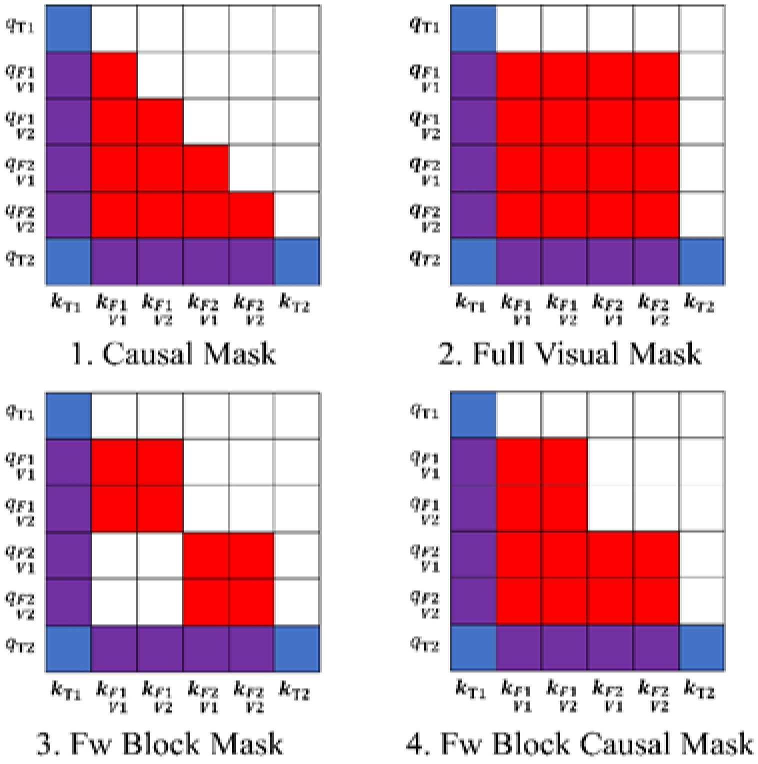
# Motivation

Attention: expressive but $O(T^2)$ compute & memory

SSMs: $O(T)$ long-range modeling, lower memory

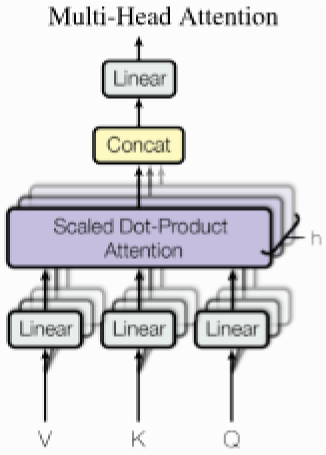
Can we unify them?

Goal: retain Transformer expressivity + SSM scalability



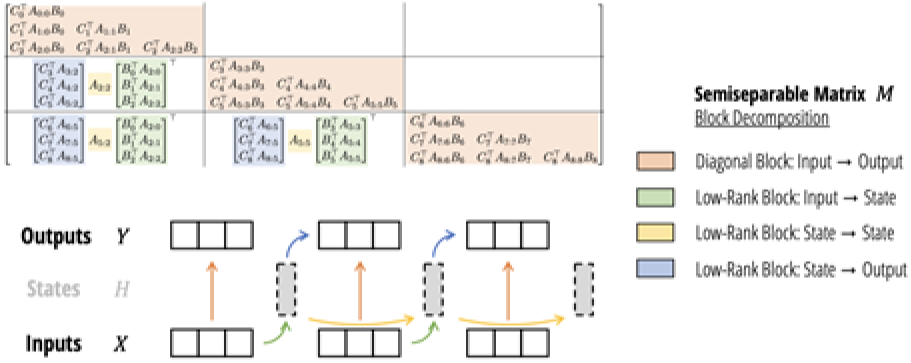
# Multi-Head Attention

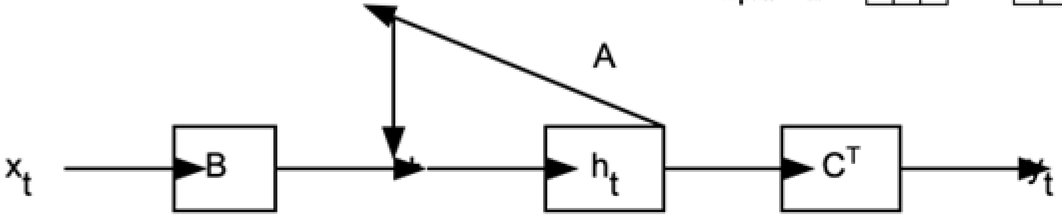
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



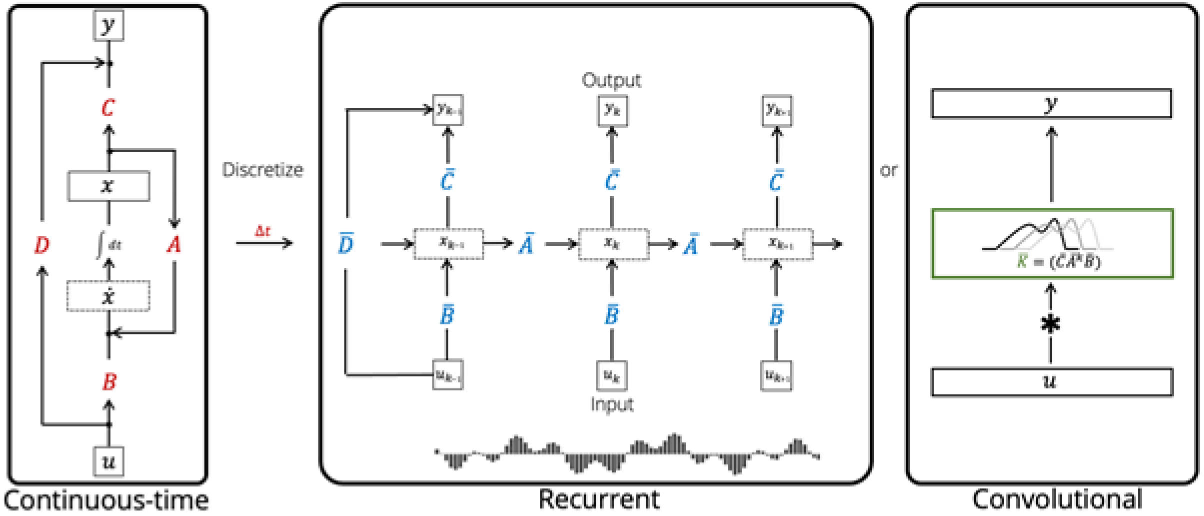
# SSM Overview

State-Space Model recurrence:



# SSM cont.



y
C
x
D
A
B
u
Discretize
Δt
Output
Input
or
y
u
Continuous-time
Recurrent
Convolutional

# Semiseparable Matrices

Key mathematical structure

Definition: lower triangular blocks rank ≤ N

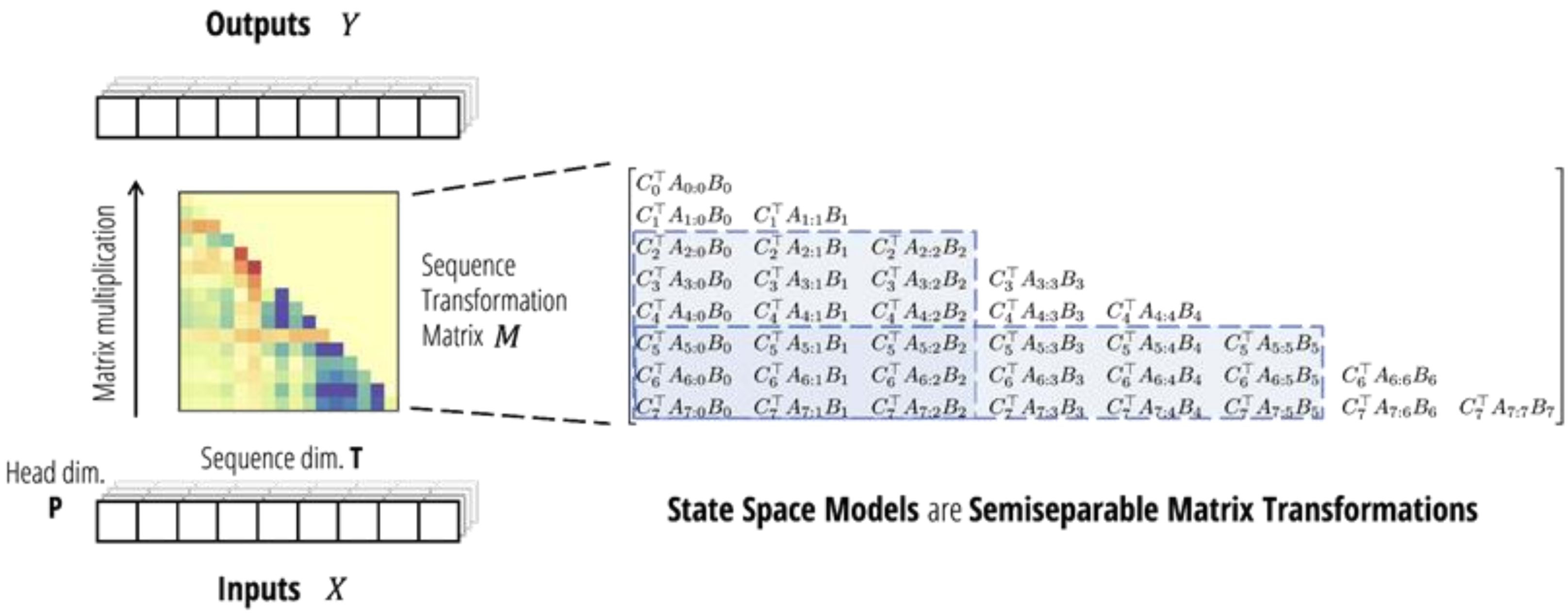
Enables efficient transforms

Semiseparable Matrix

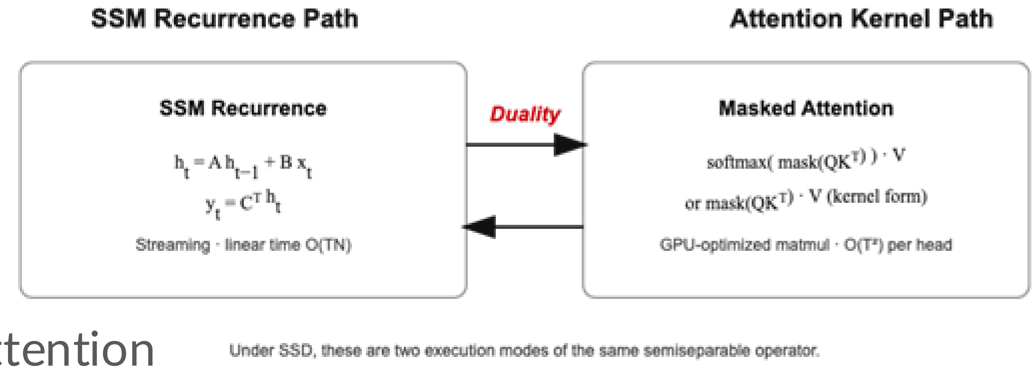
$$\begin{bmatrix} * & 0 & 0 \\ * & * & \\ * & * & * \end{bmatrix}$$

rank( lower blocks ) ≤ N

# Matrices cont.

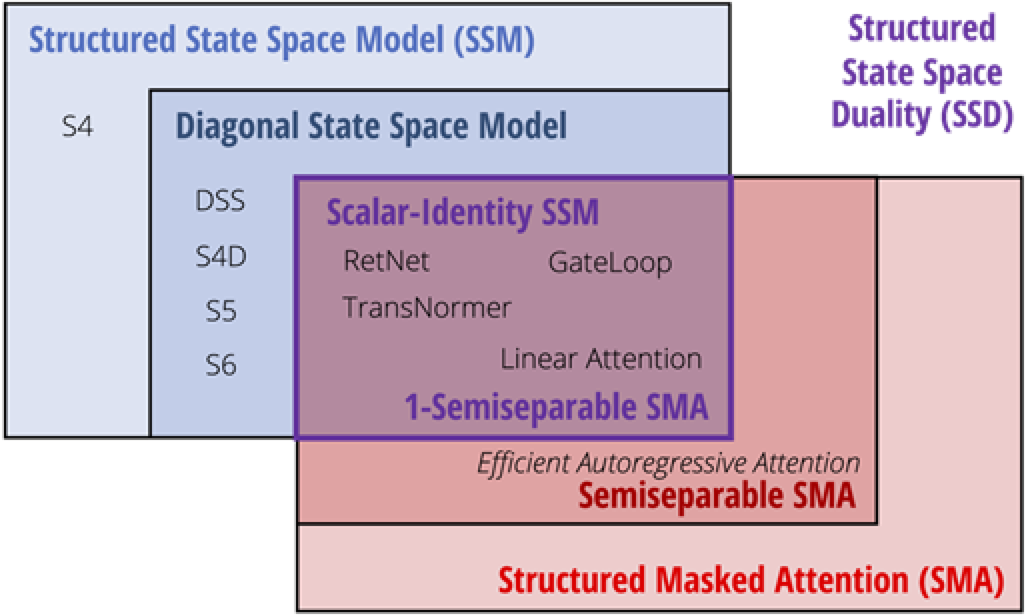


# Dual View




- SSM scan ⇆ Masked Attention
- Two ways to apply the same operator:
- SSM View Attention View
- Recurrent scan Masked structured matrix multiply
- Streaming Batch GPU optimized
- O(TN), O($T^2$) but efficient hardware path

# Dual cont.



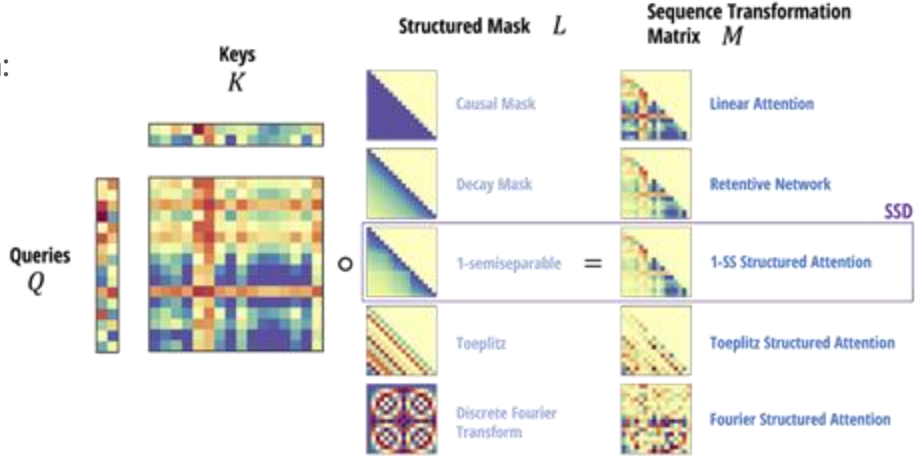
Structured State Space Model (SSM)
S4
Diagonal State Space Model
DSS
S4D
S5
S6
Scalar-Identity SSM
RetNet
GateLoop
TransNormer
Linear Attention
1-Semiseparable SMA
Efficient Autoregressive Attention
Semiseparable SMA
Structured Masked Attention (SMA)
Structured State Space Duality (SSD)

# SSD Core Idea

Switch execution paths based on sequence length:

Short sequences → matmul path (GPU efficient)

Long sequences → scan path (linear time)

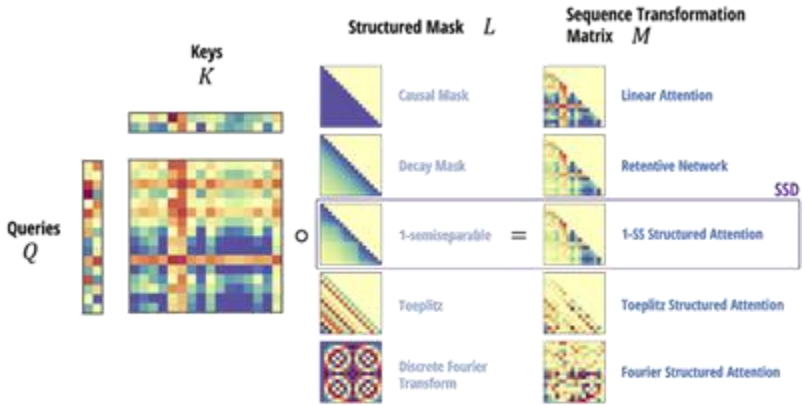


# How SSD Works

Represent attention masks as semiseparable matrices

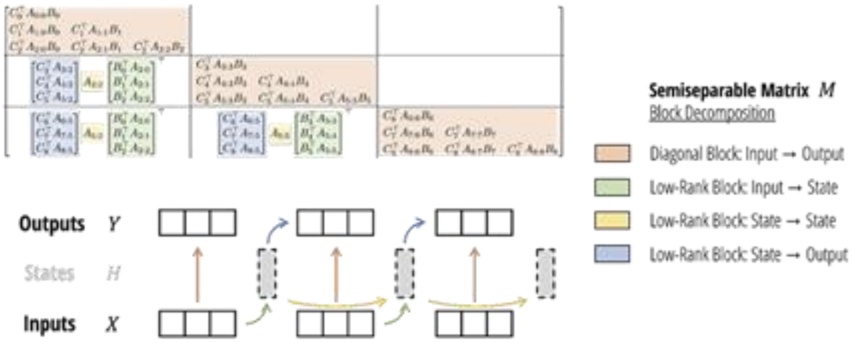
Their convolution-like structure gives a scan implementation

Their matrix form gives matmul implementation

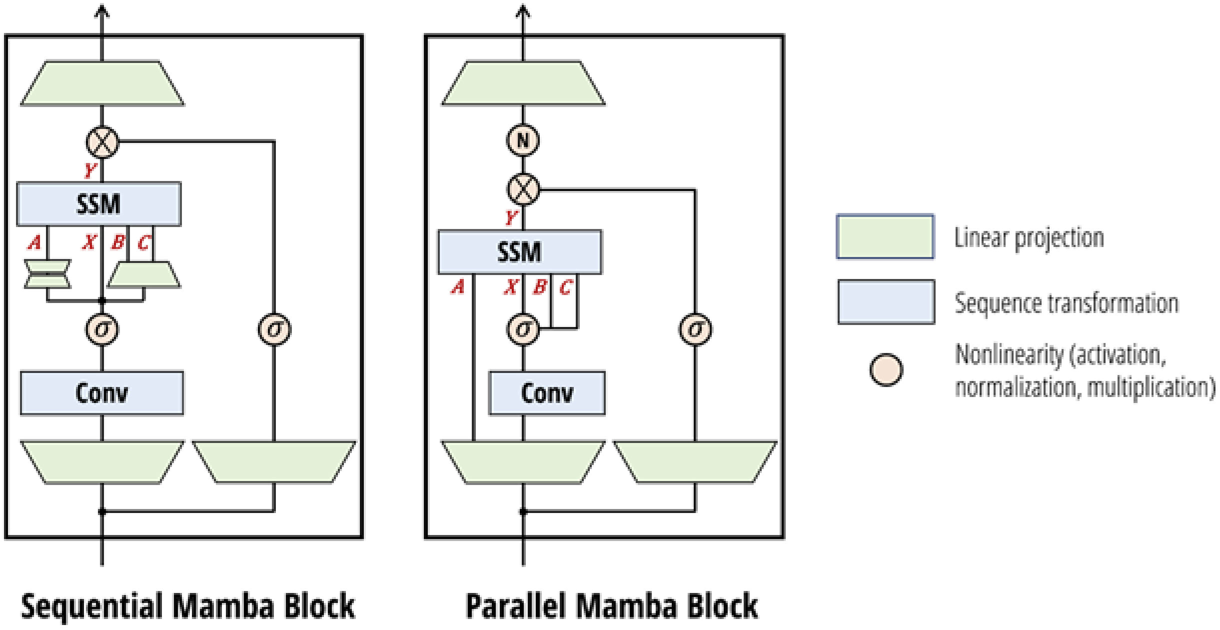


# Mamba-2

- Built on SSD theory:
- Headed SSMs (analog of multi-head attention)
- Grouped values (like grouped linear layers)
- 2–8× faster selective scan operations
- Competitive LM performance vs FlashAttention-2 Transformers



# M2 Architecture



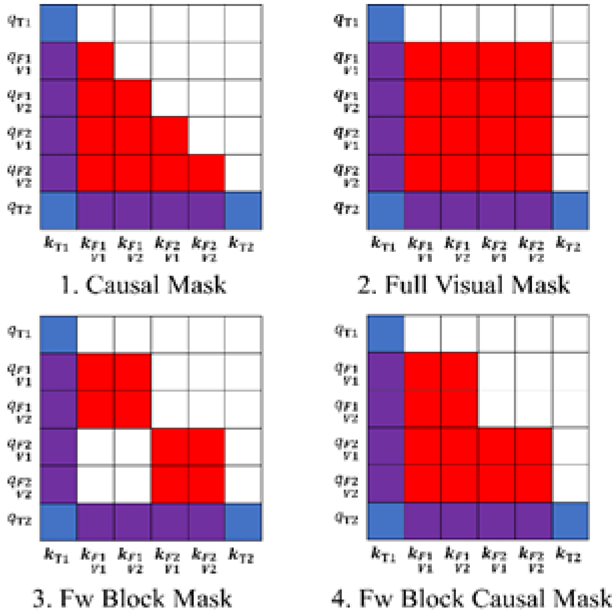
# Empirical Results

Faster training & inference for long sequences

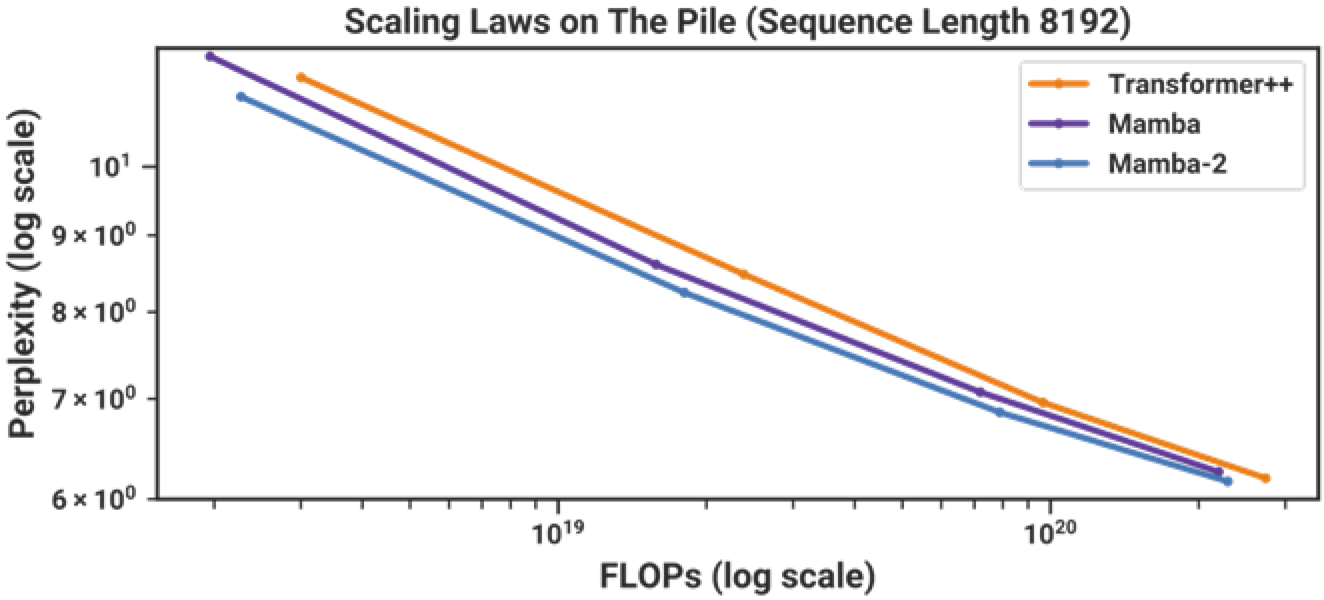
Matches or beats Transformers of similar scale

Scales well with context length

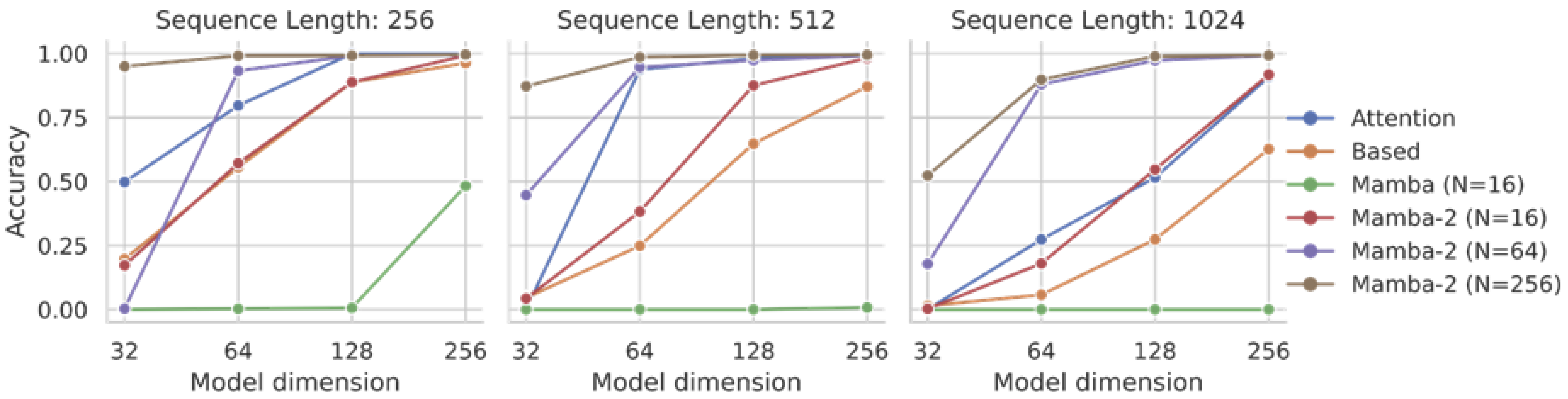
# Structured Attention Masks & Expressivity



# Complexity & Scaling



# M2 cont.


Sequence Length: 256
Sequence Length: 512
Sequence Length: 1024
Accuracy
1.00
0.75
0.50
0.25
0.00
32
64
128
256
Model dimension
Attention
Based
Mamba (N=16)
Mamba-2 (N=16)
Mamba-2 (N=64)
Mamba-2 (N=256)

# Limitations

Structured masks may limit expressivity

Training dynamics differ from Transformers

Hyperparameters maturity still evolving

# Key Contributions

Mathematical unification: Transformers ≈ SSMs

Efficient semiseparable duality

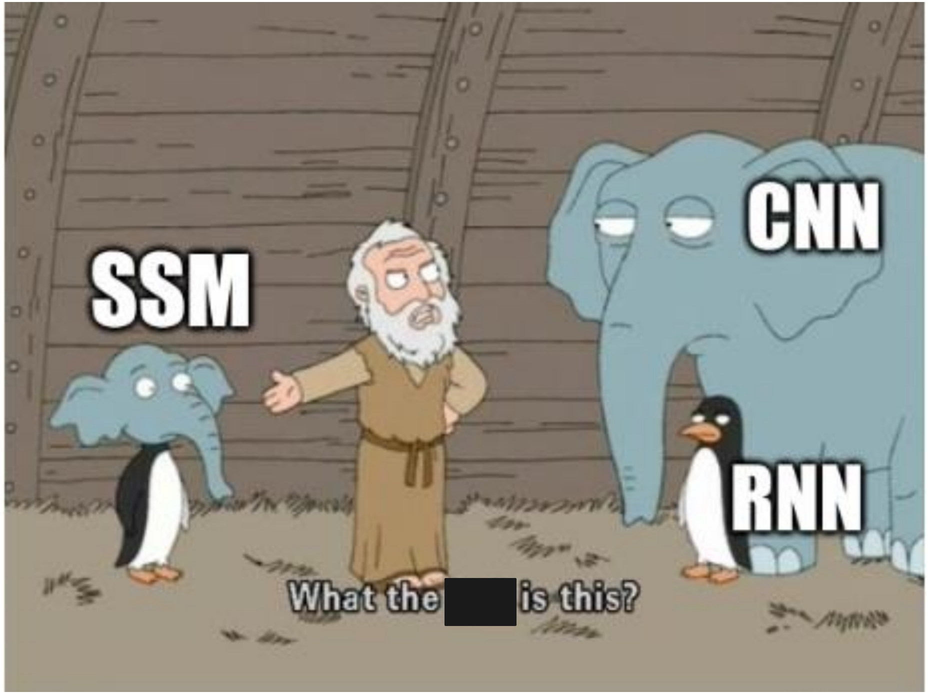
Practical model: Mamba-2

# Summary

SSD bridges SSMs and Transformers

Faster, scalable attention alternative

Mamba-2 shows real-world impact



# RWKV: Reinventing RNNs for the Transformer Era

***Peng et al.***

Presented by Kevin Kotzbauer

# Motivations

State of the art NLPs utilize transformers in their architecture, but have poor  memory and computational complexity

Recurrent Neural Network (RNN) models have linear scaling for memory and computation complexity but underperform transformer models

# Transformer Attention

Attention in transformers uses Q = query, K = key, V = value matrices to attend to previous tokens

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}(QK^\top)V$$

$$\mathrm{Attn}(Q, K, V)_t = \frac{\sum_{i=1}^{T} e^{q_t^\top k_i} \odot v_i}{\sum_{i=1}^{T} e^{q_t^\top k_i}}.$$

# Attention Free Transformer (AFT)

In AFT, replaces Q with W in softmax, where each $w_{t,i}$ is a scalar value representing a learned pair-wise position bias.

Can compute each output token independently, reducing space complexity to O(Nd)

Inspiration for WKV part of RWKV

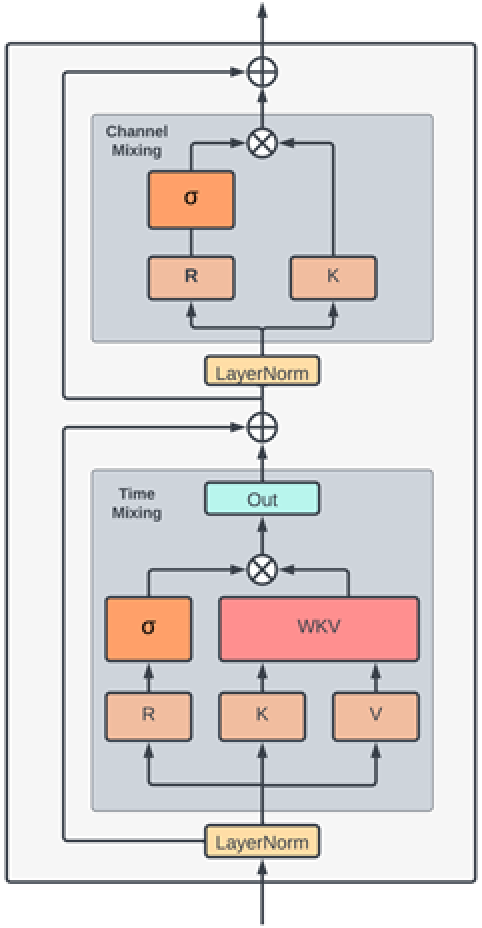
$$\mathrm{Attn}^+(W, K, V)_t = \frac{\sum_{i=1}^{t} e^{w_{t,i}+k_i} \odot v_i}{\sum_{i=1}^{t} e^{w_{t,i}+k_i}}$$

# RWKV: Block Architecture

Separates into Time Mixing and Channel Mixing Sub-Blocks

Time Mixing handles information relating to tokens in different positions interact (attention)

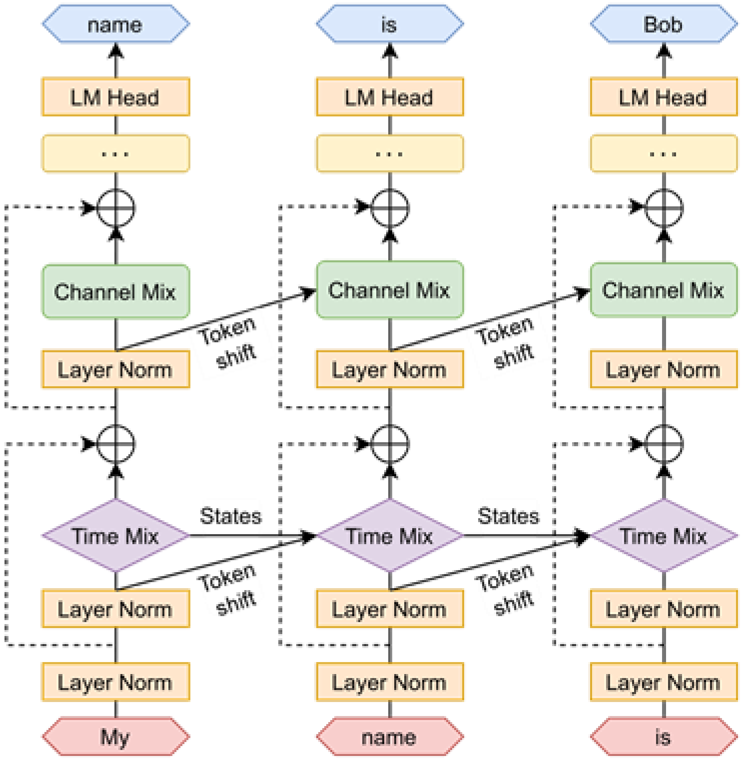
Channel Mixing transforms features dimensions of token into complex, abstract and context-aware representations for output layer



# Token Shift

Channel Shift:
$$r'_t = W'_r \cdot (\mu'_r \odot x_t + (1 - \mu'_r) \odot x_{t-1})$$
$$k'_t = W'_k \cdot (\mu'_k \odot x_t + (1 - \mu'_k) \odot x_{t-1})$$

Time Shift:
$$r_t = W_r \cdot (\mu_r \odot x_t + (1 - \mu_r) \odot x_{t-1}),$$
$$k_t = W_k \cdot (\mu_k \odot x_t + (1 - \mu_k) \odot x_{t-1}),$$
$$v_t = W_v \cdot (\mu_v \odot x_t + (1 - \mu_v) \odot x_{t-1}),$$



# Token Shift

μ is a scalar from [0,1] learned for each R,K,V independently

μ weighs previous token input vs current token, with low values increasing weight of previous tokens

Channel Shift:

$$r'_t = W'_r \cdot (\mu'_r \odot x_t + (1 - \mu'_r) \odot x_{t-1})$$
$$k'_t = W'_k \cdot (\mu'_k \odot x_t + (1 - \mu'_k) \odot x_{t-1})$$

Time Shift:

$$r_t = W_r \cdot (\mu_r \odot x_t + (1 - \mu_r) \odot x_{t-1}),$$
$$k_t = W_k \cdot (\mu_k \odot x_t + (1 - \mu_k) \odot x_{t-1}),$$
$$v_t = W_v \cdot (\mu_v \odot x_t + (1 - \mu_v) \odot x_{t-1}),$$

# WKV Operator

Replace $w_{t,i}$ pairwise position in AFT with a channelwise time decay vector $w_{t,i} = -(t-i)w$

$w \in (R_{\geq 0})^d$ (vector of dimension d, w nonnegative), each channel decays at different rates

Use w decay vector for previous tokens, use learned u vector to weight current token t

$$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}$$

# WKV Operator

$wkv_t$: A single vector representation of hidden dimension d that gets updated at each position t of sequence length

O(d) space complexity to store $wkv_t$ instead of $O(T^2)$ needed for transformer storing $QK^T$ attention matrix

O(Td) for time complexity to update $wkv_t$ over only token length

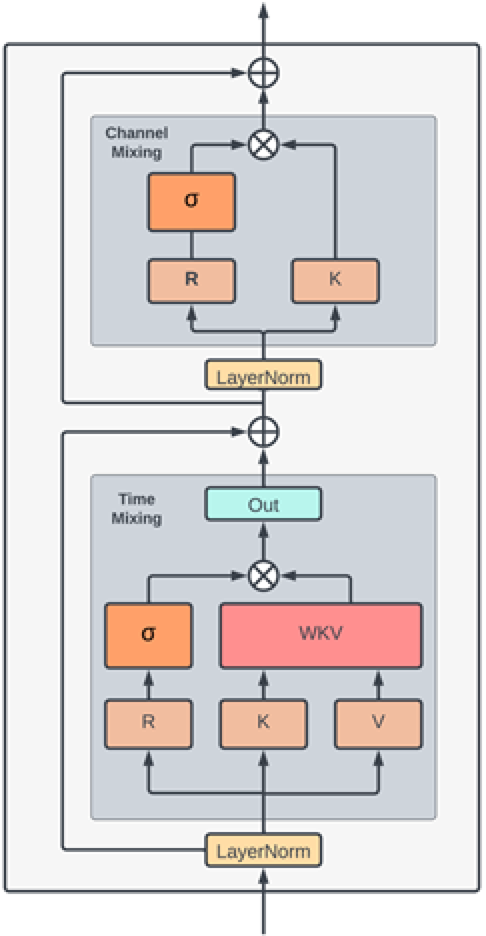
$$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}$$

# Output and Output Gating

Channel output $o'_t$: $$o'_t = \sigma(r'_t) \odot (W'_v \cdot \max(k'_t, 0)^2)$$

Time output $o_t$: $$o_t = W_o \cdot (\sigma(r_t) \odot wkv_t)$$

r: receptance vector, is fed through sigmoid function to act as a gate to control how much each dimension of $wkv_t$ is passed through.



# Parallelization of Training

Time complexity of processing a batch of sequences in a single layer is $O(BTd^2)$ for weight matrices $W_k$, $W_v$, $W_r$, $W_o$

This is the same cost of transformers and can be parallelized

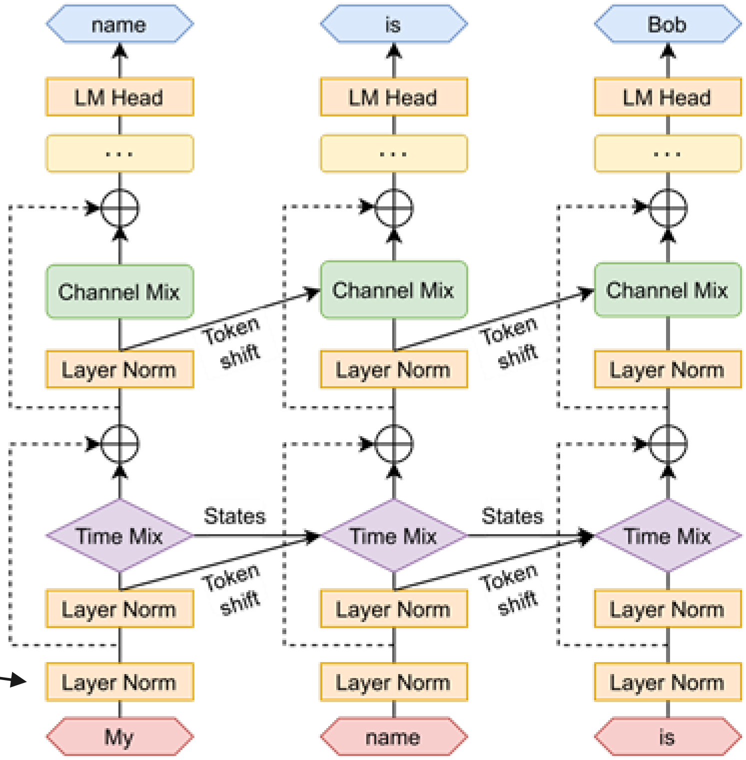
Updating attention scores $wkv_t$ involves a serial scan $O(BTd)$ but matrix multiplication of weight matrices dominates

Overall training cost is the same as transformers

# Small Init Embedding

Embedding matrix undergoes slow changes in initial stages of training

Solution is to initialize the embedding matrix with small values and subsequently applying an additional LayerNorm operation



# Inference Complexity Comparison

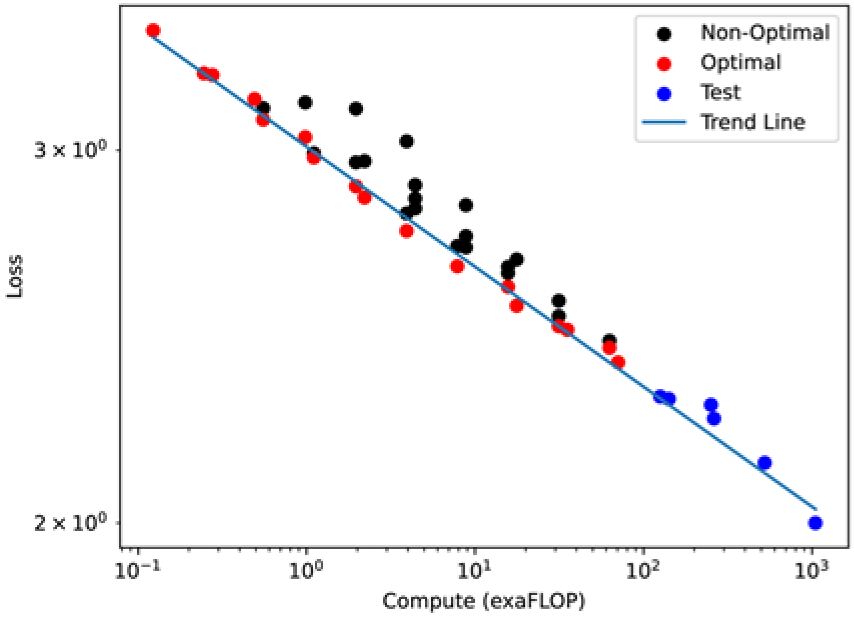
RWKV has linear time and space complexity with regards to number of tokens T and dimensions d

| Model | Time | Space |
|---|---|---|
| **Transformer** | $O(T^2 d)$ | $O(T^2 + Td)$ |
| **Reformer** | $O(T \log Td)$ | $O(T \log T + Td)$ |
| **Performer** | $O(Td^2 \log d)$ | $O(Td \log d + d^2 \log d)$ |
| **Linear Transformers** | $O(Td^2)$ | $O(Td + d^2)$ |
| **AFT-full** | $O(T^2 d)$ | $O(Td)$ |
| **AFT-local** | $O(Tsd)$ | $O(Td)$ |
| **MEGA** | $O(cTd)$ | $O(cd)$ |
| **RWKV (ours)** | $O(\mathbf{Td})$ | $O(\mathbf{d})$ |

# Scaling Law

Log-Log loss is linear, similar to transformer model

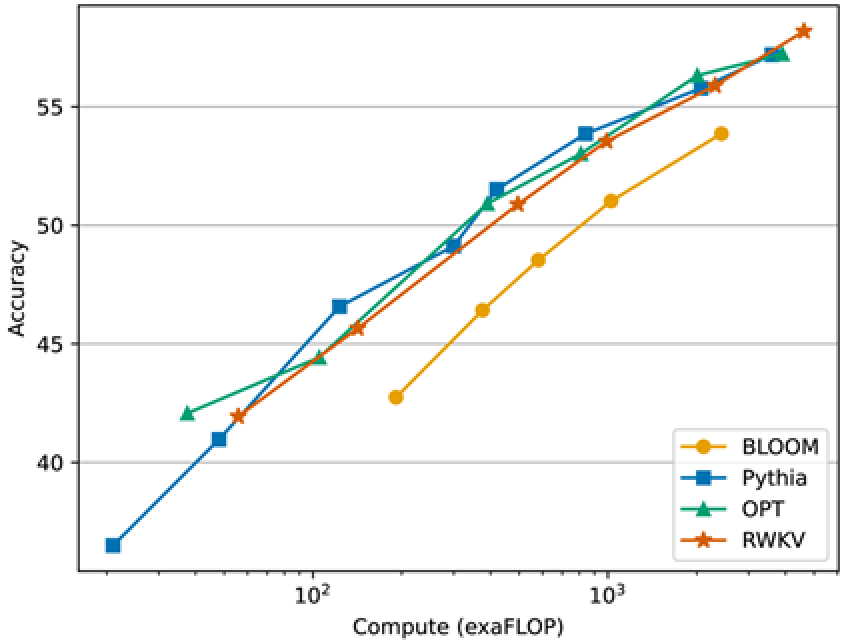
Shows that RWKV model performance can scale with model size, data size, and compute budget



# Evaluations

Performance on average is on par with similar open source transformer research models of similar size

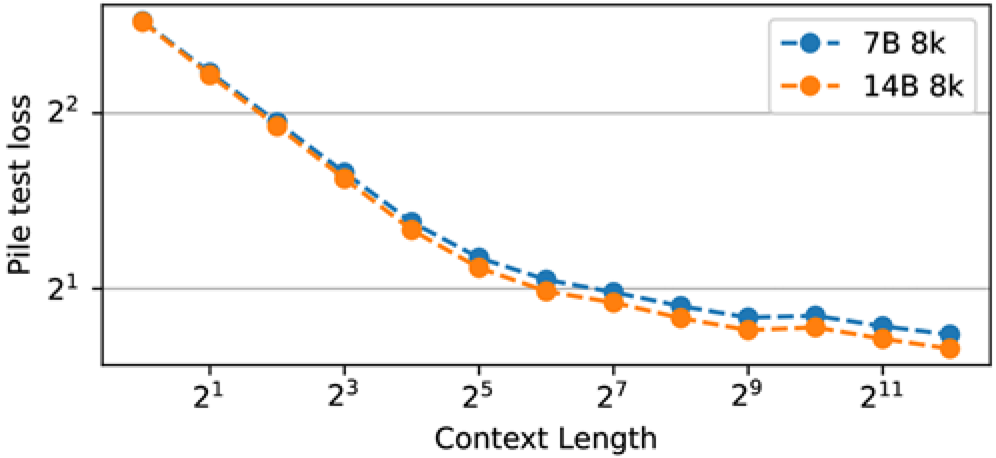
Matched performance of transformers for ARC (Challenge), HellaSwag, LAMBADA (OpenAI), OpenBookQA, Winogrande



# Extended Context Fine Tuning

RNNs can perform poorly in long contexts as they lose information from early tokens due to not keeping memory of all previous tokens through attention

Through training RWKV in batch sizes from 1024-8192 tokens, RWKV is able to improve performance from longer context lengths, making it a viable alternative to transformers

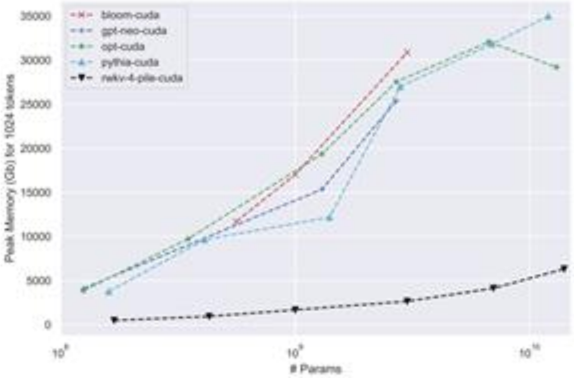


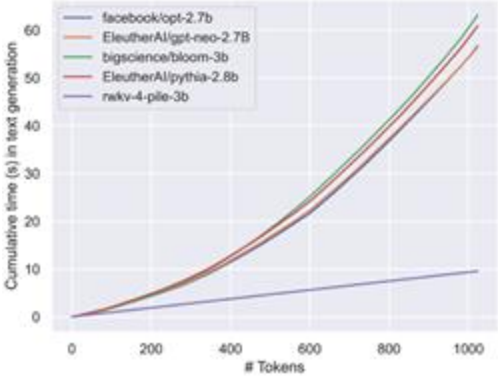
# Large Improvement in Inference Cost

Time to generate tokens during inference is linear for RWKV

Transformer models are quadratic with respect to the number of tokens

Memory performance is improved for RWKV over transformer models



# Long Context Benchmarks

RWKV performs better than transformers and most other models on long context (1000-16000 tokens) except for S4 (Structured State Space Sequence model)

Performance is similar to S4 for natural language and code processing

| MODEL | LISTOPS | TEXT | RETRIEVAL | IMAGE | PATHFINDER | PATH-X | AVG |
|---|---|---|---|---|---|---|---|
| Transformer | 36.37 | 64.27 | 57.46 | 42.44 | 71.40 | ✗ | 53.66 |
| Reformer | 37.27 | 56.10 | 53.40 | 38.07 | 68.50 | ✗ | 50.56 |
| BigBird | 36.05 | 64.02 | 59.29 | 40.83 | 74.87 | ✗ | 54.17 |
| Linear Trans. | 16.13 | 65.90 | 53.09 | 42.34 | 75.30 | ✗ | 50.46 |
| Performer | 18.01 | 65.40 | 53.82 | 42.77 | 77.05 | ✗ | 51.18 |
| FNet | 35.33 | 65.11 | 59.61 | 38.67 | 77.80 | ✗ | 54.42 |
| Nyströmformer | 37.15 | 65.52 | 79.56 | 41.58 | 70.94 | ✗ | 57.46 |
| Luna-256 | 37.25 | 64.57 | 79.29 | 47.38 | 77.72 | ✗ | 59.37 |
| Hrrformer | 39.98 | 65.38 | 76.15 | 50.45 | 72.17 | ✗ | 60.83 |
| S4 | **59.60** | **86.82** | **90.90** | **88.65** | **94.20** | **96.35** | **86.09** |
| RWKV | 55.88 | 86.04 | 88.34 | 70.53 | 58.42 | ✗ | 72.07 |

# Conclusion

RWKV is a new RNN model that combines computation speed improvements of RNNs while maintaining some of the benefits of transformer models

Inference is linear with respect to token length and dimension

RWKV has training computation cost similar to transformers

RWKV can have good performance in long contexts

# Limitations

Compression of prior context information into a single wkv vector can perform worse than transformer models for retrieving precise and trivial details in long contexts

RWKV is very sensitive to prompt engineering, where prompts adjusted (re-ordered) from ones used for GPT to be more suitable for RWKV increased performance from 44.2% to 74.8%

# Hierarchical Reasoning Model

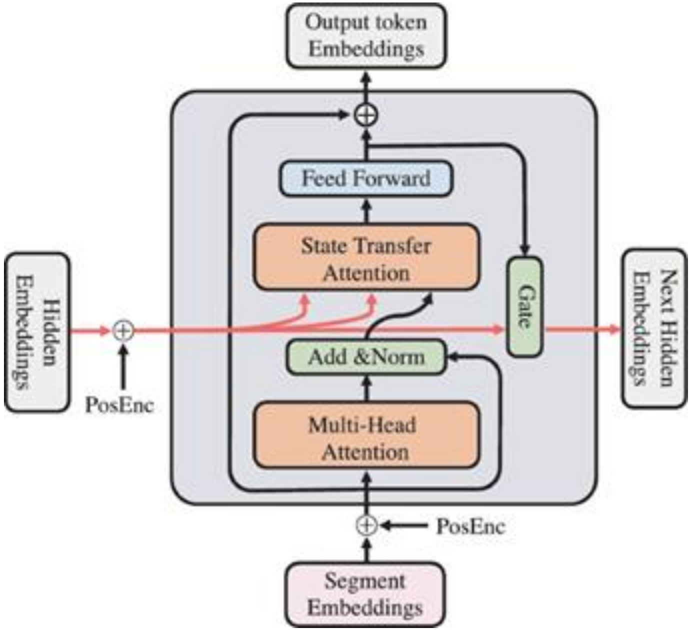
Presented by Rylan Tang

# Motivations

- Chain-of-Thought (CoT) Reasoning Model suffered from high latency and extensive memory usage
- Inspired by human brain
- Save Memory on Current Recurrent Architectures Backpropogation Through Time (BPTT)

# Review: What do we mean by “Recurrent”

- The Reasoning Model employs Recurrent Transformer
- Intermediate Hidden States
- Width (Dimension of Hidden State)
- Depth (# of Layers)
- **Recurrence: data fed back to network input each pass**
- **Prone to Early Convergence**



# HRM Architecture

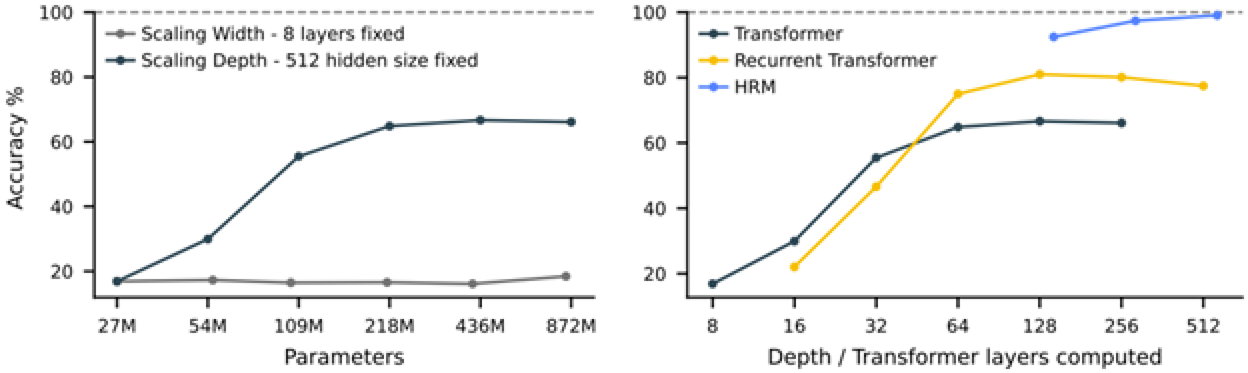
- A high-level (H) module for abstract, deliberate reasoning
  - Internal hidden state at timestep i denoted as z^i_H, with total N high-level cycles
- A low-level (L) module for fast, detailed computations
  - Each nested inside a high-level module
  - Internal hidden state at timestep i denoted as z^i_L, with T low-level cycles each high-level cycle
- Total timestep is N x T
  - A NT-timestep process represents a single forward pass in HRM

# Hierarchical Convergence

- During each cycle, the **L-module** (an RNN) exhibits stable convergence to a local equilibrium
- This equilibrium depends on the high-level state z_H supplied during that cycle
- **After** completing the T steps, the H-module incorporates the sub-computation's outcome (the final state zL) and performs its own update
- **So why NT timesteps each pass?**

# Depth Matters

- NT timesteps enhance model’s depth and reasoning capacity
- Model’s capacity in complex reasoning high correlated with its depth



# HRM Math

$$z_L^i = f_L\left(z_L^{i-1}, z_H^{i-1}, \tilde{x}; \theta_L\right) ,$$

$$z_H^i = \begin{cases} f_H\left(z_H^{i-1}, z_L^{i-1}; \theta_H\right) & \text{if } i \equiv 0 \pmod{T} , \\ z_H^{i-1} & \text{otherwise} \end{cases} .$$

- At each timestep i, the **L-module** updates its state conditioned on
  - Previous L-module state, Z^(i-1)_L
  - Current H-module state, Z^(i-1)_H
    - It's (i-1)th since the current H-module is incomplete yet until all T L-modules within are finished
  - Input Representation, x_tilde
  - Update function: f_L or f_H, updates the corresponding L or H-module to the next hidden state
  - The collection of all parameters such as weights for each update function, \theta
- Since we want to avoid early convergence
  - Each L-module updates upon previous L-module
  - H-modules only update once each time all T low-level modules are finished, based on the **last L-module** in the current H-module
  - During each T-cycle in an H-module, L-modules within achieves local equilibrium
  - When a new H-module starts, it refreshes the context for its L-modules

# Approximation Gradient

- It will demand O(T) memory if we backpropogate through each intermediate low-level hidden states
- **One-step** approximation of the HRM gradient, using the gradient of the last state of each module and treating other states as constant
- **Output head → final state of the H-module → final state of the L-module → input embedding**

# Approximation Gradient

$$z_L^\star = f_L(z_L^\star, z_H^{k-1}, \tilde{x}; \theta_L) \; .$$

$$z_H^k = f_H(z_H^{k-1}, z_L^\star; \theta_H)$$

$$z_H^\star = \mathcal{F}(z_H^\star; \tilde{x}, \theta) \quad \text{where } \theta = (\theta_I, \theta_L)$$

- Why could we skip intermediate steps?
- Notations
  - z*L, the local fixed point (the point of convergence) of L-modules inside an H-module
  - z*H, the fixed point of H-modules
  - f_L or f_H, the update function of each L or H-module to next hidden state
  - F, **a more compact representation of f_H**, where the up-to-date L-module state z*L is written into updated params, \theta_L
- Because of its recurrent nature, the L-module update function is related to the current state
  - That results in a gradient involving **Jacobian Matrix**

$$\frac{\partial z_H^\star}{\partial \theta} = \left(I - J_{\mathcal{F}}\big|_{z_H^\star}\right)^{-1} \frac{\partial \mathcal{F}}{\partial \theta}\bigg|_{z_H^\star} \; .$$

# Approximation Gradient Ctd.

- Jacobian Matrix J could be eliminated by Neumann Series

$$\frac{\partial z_H^*}{\partial \theta} = \left(I - J_{\mathcal{F}}\big|_{z_H^*}\right)^{-1} \frac{\partial \mathcal{F}}{\partial \theta}\bigg|_{z_H^*} \quad (I - J_{\mathcal{F}})^{-1} = I + J_{\mathcal{F}} + J_{\mathcal{F}}^2 + J_{\mathcal{F}}^3 + \dots \quad \frac{\partial z_H^*}{\partial \theta_H} \approx \frac{\partial f_H}{\partial \theta_H}, \quad \frac{\partial z_H^*}{\partial \theta_L} \approx \frac{\partial f_H}{\partial z_L^*} \cdot \frac{\partial z_L^*}{\partial \theta_L}, \quad \frac{\partial z_H^*}{\partial \theta_I} \approx \frac{\partial f_H}{\partial z_L^*} \cdot \frac{\partial z_L^*}{\partial \theta_I} .$$

- That eliminates the self-recurrence part and makes the calculation of fixed-point hidden state z*H, z*L's gradient on parameters easier
- Same skipping works for L-modules

$$\frac{\partial z_L^*}{\partial \theta_L} \approx \frac{\partial f_L}{\partial \theta_L}, \quad \frac{\partial z_L^*}{\partial \theta_I} \approx \frac{\partial f_L}{\partial \theta_I} .$$

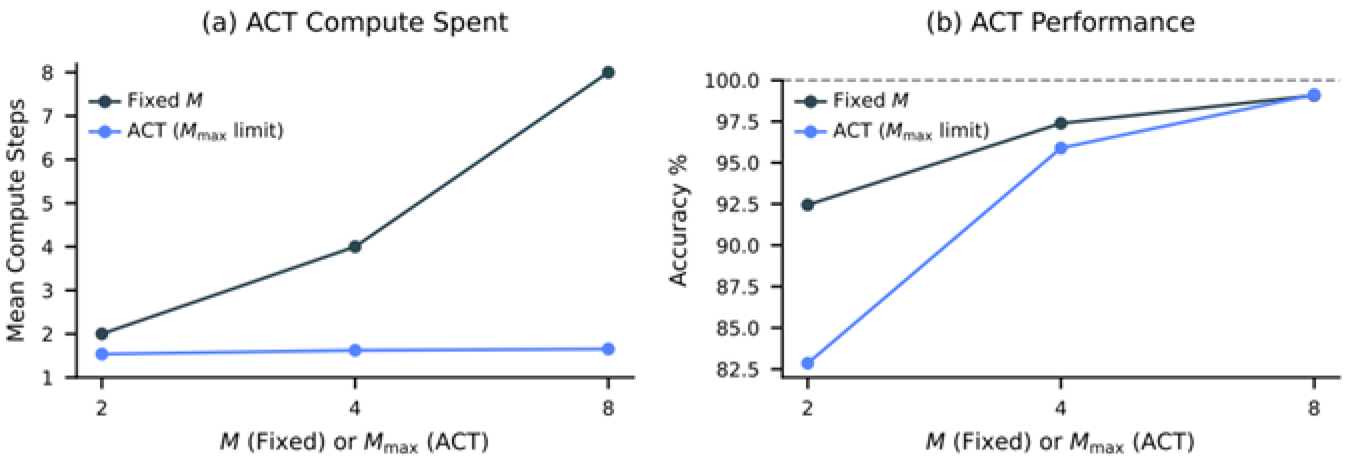
# Deep Supervision

- Each NT-timestep forward pass is called a *segment*
- We say we go through *m* segments before termination
    - During training, we **detach** the m-th segment from previous segments, preventing the gradients from **flowing backward to previous segments**

# Adaptive Computational Time (ACT)

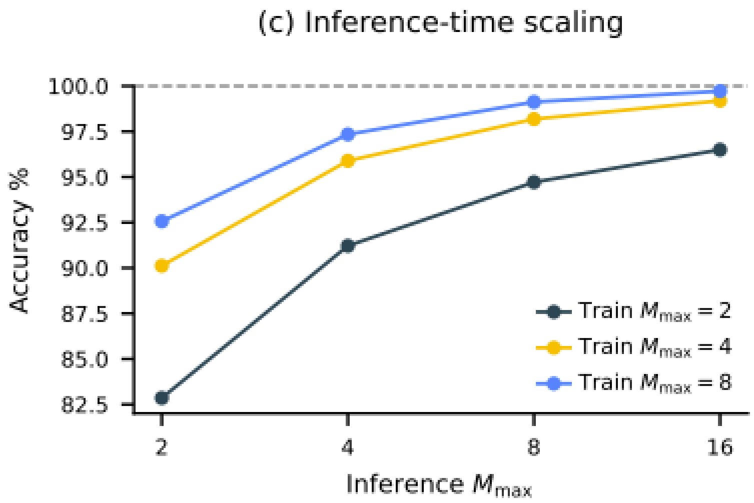
- Motivations: human brains think “fast and slow”
- **Q-Learning Algorithm** is used to determine adaptively the number of segments m, choosing between “halt” and “continue” actions
- We need to compute Q-Value for each complete segment $$\hat{Q}^m = \sigma(\theta_Q^\top z_H^{mNT})$$
- Define fixed M_max, max # of segments
- Define M_min: with probability ε, it is sampled uniformly from the set {2, · · · , M_max} (to encourage longer thinking), and with probability 1−ε, it is set to 1
- Selects “Halt” when
    - the segment count surpasses M_max
    - OR (Q_halt exceeds the estimated continue value Q_continue AND the segment count surpasses M_min)

# ACT v.s. Fixed Segments



# Inference-Time Scaling

- Scaling on depth enhances inference accuracy
- Especially for long-term complex reasoning like Sudoku
- M_max = # of segments = # of NT-timestep passes in HRM
- **Increasing M_max directly** is more effective than increasing N or T for individual modules



# Architectural Details

- Sequence-to-sequence encoding
- Softmax output heads
- **Encoder-only** Transformer Blocks
- PostNorm with weights initialized via truncated LeCun Normal initialization

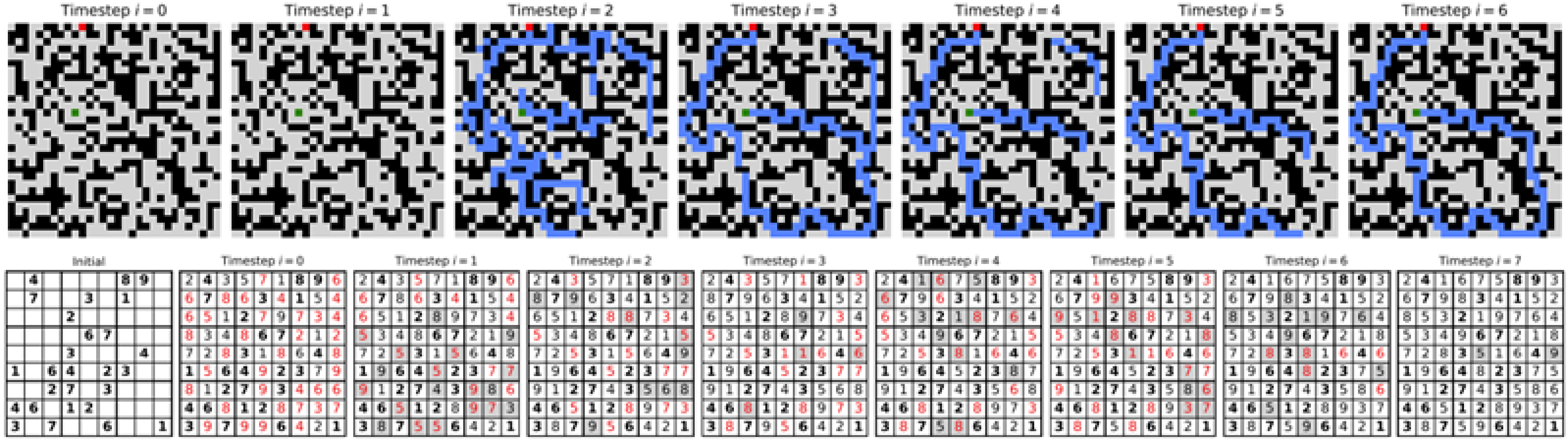
# Performance

- Using only **1,000** input-output examples, without pre-training or CoT supervision, HRM learns to solve problems that are intractable for even the most advanced LLMs
    - In the Abstraction and Reasoning Corpus (ARC) AGI Challenge 27,28,29 - a benchmark of inductive reasoning - HRM, trained from scratch with only the official dataset (~1000 examples), with only **27M** parameters and a 30x30 grid context (900 tokens), achieves a performance of **40.3%**
- which substantially surpasses leading CoT-based models like o3-mini-high (34.5%) and Claude 3.7 8K context (21.2%), despite their considerably larger parameter sizes and context lengths

# Underlying Reasoning Algorithms

- How does machine make reasoning under HRM?
- We can’t give a definitive answer but could observe its intermediate steps
- Maze: more like breadth-first, looking for several routes first and then eliminating them
- Sudoku: mainly depth-first with backtracking

# Visualization of HRM Reasoning



# References


- A. Bahadir Akdemir. *Mamba: An SSM Method for Efficient and Powerful Sequence Modeling.* Medium, 2024.
- T. Dao. *Mamba-2 Part I: The Model.* 2024. https://tridao.me/blog/2024/mamba2-part1-model/
- T. Dao. *Mamba-2 Part II: The Theory.* 2024. https://tridao.me/blog/2024/mamba2-part2-theory/
- Hazy Research. *S4: Structured State Space Models.* 2022. https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3
- UvA Deep Learning Course. *Transformers and Multi-Head Attention Notebook.* 2023. https://uvadlc-notebooks.readthedocs.io/
- ResearchGate Figure: *Variations of Attention Masks.* https://www.researchgate.net/figure/Variations-of-Attention-Masks_fig3_383791939
- Bócachancla Blog. *Understanding Mamba.* 2024.