Hardware-aware Algorithms for Sequence Modeling

Tri Dao

https://tridao.me
Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

Find the bug with this code:
```javascript
... for (var i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000)
} ...
```

The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:
```javascript
for (var i = 0; i < 5; i++) {
    const value = i;
    setTimeout(() => console.log(value), 1000)
}
```

Generate Art
(Stable Diffusion – Stability.AI)

Design Drugs
(AlphaFold – DeepMind)

What enabled these advances? What are outstanding problems? How do we approach them?
Scale Brings Quality and Capabilities

Language models explaining jokes

Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

1.3B model: The joke is that if you try 10000 different seed choices, you’ll eventually find one that works, but you’ll be accused of overfitting.

175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm. The punchline, "I guess no good seed goes unpunished," is a play on the phrase "no good deed goes unpunished." In this case, "good seed" refers to a starting point for the random restarts, and the joke implies that even when trying to improve the neural network’s performance, the person is still accused of overfitting.

Scale is more closely tied to advances in ML than ever before
Core Challenge with Scale: Efficiency

Efficiency eases **training**, **deployment**, and facilitates **research**

Efficiency unlocks **new capabilities** (e.g., long context)

Write a 4000 word essay on the best ice cream flavor

11 tokens in prompt
Up to 4,000 tokens in response
This model can only process a maximum of 4,001 tokens in a single request, please reduce your prompt or response length.

Learn more about pricing
Approach to Efficiency: Understanding Algorithms & Systems

**Fundamental algorithms**
- Fast matrix-vector multiply
- Attention mechanism

**Hardware accelerators & distributed systems**
- Block-oriented device
- Asymmetric memory hierarchy
Main Idea: Hardware-aware Algorithms

IO-awareness: reducing reads/writes to GPU memory yields significant speedup

State-space expansion: expand recurrent states in SRAM only to avoid memory cost

FlashAttention: fast and memory-efficient attention algorithm, with no approximation

Mamba: selective state-space model that matches Transformers on language model, with fast inference and up to 1M context

D., Fu, Ermon, Rudra, Ré, NeurIPS 2022
D., 2023

Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio
Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio
Motivation: Modeling Long Sequences

Enable New Capabilities
NLP: Large context required to understand books, plays, codebases.

Close Reality Gap
Computer vision: higher resolution can lead to better, more robust insight.

Open New Areas
Time series, audio, video, medical imaging data naturally modeled as sequences of millions of steps.
Efficiency is the Bottleneck for Modeling Long Sequences with Attention

Context length: how many other elements in the sequence does the current element interact with.

Increasing context length slows down (or stops) training

How to efficiently scale models to longer sequences?
Background: Attention is the Heart of Transformers
Background: Attention Mechanism

\[ O = \text{Softmax}(QK^T)V \]

Attention scales quadratically in sequence length N
Is there a fast, memory-efficient, and exact attention algorithm?

Background: Approximate Attention

Approximate attention: tradeoff quality for speed fewer FLOPs

Our Observation: Attention is Bottlenecked by Memory Reads/Writes

The biggest cost is in moving the bits!
Standard implementation requires repeated R/W from slow GPU memory
Background: GPU Compute Model & Memory Hierarchy

1. Inputs start out in HBM (GPU memory)
2. Data moved to compute units & SRAM for computation
3. Output written back to HBM

Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)
How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges:

(1) Compute softmax normalization without access to full input.

(2) Backward without the large attention matrix from forward.

Approaches:

(1) Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

(2) Recomputation: Don’t store attn. matrix from forward, recompute it in the backward.
Attention Computation Overview

\[ S = QK^T \]

\[ A = \exp(S) \cdot V = \frac{A}{l} \cdot V \]

Softmax row-wise normalization constant

\[ l = \sum_i \exp(S)_i \]

Compute by blocks: easy to split Q, but how do we split K & V?
Tiling – 1\textsuperscript{st} Attempt: Computing Attention by Blocks

Goal:
Load each block from HBM to SRAM & do \textit{local computation}

\[
\begin{align*}
Q & \rightarrow S^{(1)} = Q (K^{(1)})^T \\
S^{(2)} & = Q (K^{(2)})^T \\
A^{(1)} & = \exp(S^{(1)}) \\
A^{(2)} & = \exp(S^{(2)}) \\
\frac{A^{(1)}}{l} \cdot V^{(1)} + \frac{A^{(2)}}{l} \cdot V^{(2)} & = V^{(1)}
\end{align*}
\]

Example: Split K into 2 blocks

\[ l = \sum_i \exp(S^{(1)}_i) + \sum_i \exp(S^{(2)}_i) \]

Challenge: How to compute softmax normalization with just local results?

Softmax row-wise normalization constant
**Tiling – 2\textsuperscript{nd} Attempt: Computing Attention by Blocks, with Softmax Rescaling**

**Goal:**
Load each block from HBM to SRAM & do \textit{local computation}

\begin{align*}
  Q &\quad \rightarrow \quad (K^{(1)})^T \quad \rightarrow \quad S^{(1)} = Q (K^{(1)})^T \quad \rightarrow \quad A^{(1)} = \exp(S^{(1)}) \\
  &\quad \rightarrow \quad S^{(2)} = Q (K^{(2)})^T \quad \rightarrow \quad A^{(2)} = \exp(S^{(2)}) \\
  &\quad \rightarrow \quad l^{(1)} = \sum_i \exp(S^{(1)})_i \quad l^{(2)} = l^{(1)} + \sum_i \exp(S^{(2)})_i \\
\end{align*}

Output we want:
\begin{align*}
  l &= \sum_i \exp(S^{(1)})_i + \sum_i \exp(S^{(2)})_i \\
  O &= \frac{A^{(1)}}{l} \cdot V^{(1)} + \frac{A^{(2)}}{l} \cdot V^{(2)} 
\end{align*}

\begin{align*}
  &\quad \rightarrow \quad \text{Output} \\
  &\quad \rightarrow \quad O^{(1)} = \frac{A^{(1)}}{l^{(1)}} \cdot V^{(1)} \\
  &\quad \rightarrow \quad O^{(2)} = \frac{l^{(2)}}{l^{(2)}} \cdot O^{(1)} + \frac{A^{(2)}}{l^{(2)}} \cdot V^{(2)} \\
\end{align*}

Wrong denominator 🙄
Local computation
Rescaling to correct denominator

Tiling + Rescaling allows \textit{local computation} in SRAM, without writing to HBM, and get the \textit{right answer}!
Tiling

Decomposing large softmax into smaller ones by scaling.

1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output with respect to that block.
3. Update output in HBM by scaling.

Animation credit: Francisco Massa
Recomputation (Backward Pass)

By storing softmax normalization from forward (size N), quickly recompute attention in the backward from inputs in SRAM.

<table>
<thead>
<tr>
<th></th>
<th>Standard</th>
<th>FlashAttention</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFLOPs</td>
<td>66.6</td>
<td>75.2 (↑13%)</td>
</tr>
<tr>
<td>HBM reads/writes (GB)</td>
<td>40.3</td>
<td>4.4 (↓9x)</td>
</tr>
<tr>
<td>Runtime (ms)</td>
<td>41.7</td>
<td>7.3 (↓6x)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
S &= Q K^T \\
A &= \exp(S) \cdot V \\
\ell &= \sum_i \exp(S)_i
\end{align*}
\]

Output

FlashAttention speeds up backward pass even with increased FLOPs.
FlashAttention: 2-4x speedup, 10-20x memory reduction

2-4x speedup — with no approximation!
10-20x memory reduction — memory linear in sequence length
GPT3: Faster Training, Longer Context, Better Model

FlashAttention speeds up GPT-3 training by 2x, increase context length by 4x, improving model quality

FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning

Key ideas:
- Reduce non-matmul FLOPs
- Parallelize over seqlen dimension to improve occupancy
- Better work partitioning between warps to reduce communication

Upshot: **2x** faster wallclock, can train models with 2x context length for the same cost
Optimizing FlashAttention for H100 GPU
Ganesh Bikshandi and Jay Shah

New hardware features on H100:
- `wgmma` instruction: higher matmul throughput
- `TMA`: faster loading from global memory <-> shared memory
- `FP8`: lower precision, higher throughput

Upshot: **1.2-2.5x** speed up by using new features
Flash-Decoding: Faster Decoding for Long Context Inference
Tri Dao, Daniel Haziza, Francisco Massa, Grigory Sizov

Decoding IO bottleneck: all about loading KV cache as fast as possible

Previous methods:
- Parallelizes across blocks of queries, batch size, and heads only
- Does not to occupy the entire GPU during decoding → slow KV cache loading.

Flash-Decoding:
- Faster loading: parallelize KV cache over seqlen dim
- Separate reduction step to combine results

Upshot: **2-8x** faster end-to-end generation on CodeLlama 34B with context 32k-100k.

Animation credit: Daniel Haziza
Summary – FlashAttention

FlashAttention: fast and memory-efficient algorithm for exact attention

Key algorithmic ideas: tiling, recomputation

Upshot: faster training, better models with longer sequences

Code: https://github.com/Dao-AILab/flash-attention
Attention is bottlenecked by memory reads/writes
Tiling and recomputation to reduce IOs
Applications: faster Transformers, better Transformers with long context

Structured State Space Models (S4)
Selection Mechanism
Applications: language modeling, DNA, audio

Slides credit: Albert Gu (CMU)
Deep Sequence Model

- CNN (ResNet)
- Transformer
- SSNN
Recurrent Neural Networks (RNN)

- Natural autoregressive (causal) model
- Slow training on accelerators and poor optimization (vanishing gradients)

Sequential
Attention (Transformers)

- Strong performance, parallelizable
- Quadratic-time training, linear-time inference (in the length of the sequence)
Selective State Spaces

- **Efficiency**: parallelizable training + fast inference
- **Performance**: matches Transformers on LM
- **Long Context**: improves up to million-length sequences
State Space Models (SSM)

\[ h'(t) = Ah(t) + Bx(t) \]
\[ y(t) = Ch(t) + Dx(t) \]

Outline

• Structured State Space Models (S4)

• Selective State Space Models (Mamba)

• Applications
Structured State Space Models (S4)

Modeling Sequences with Structured State Spaces

Deep learning model related to SSMs, RNNs, CNNs
\[ h'(t) = Ah(t) + Bx(t) \]
\[ y(t) = Ch(t) + Dx(t) \]
\[ h'(t) = Ah(t) + Bx(t) \]
\[ y(t) = Ch(t) + Dx(t) \]
\[ h'(t) = Ah(t) + Bx(t) \]

\[ y(t) = Ch(t) + Dx(t) \]
\[ h'(t) = Ah(t) + Bx(t) \]
\[ y(t) = Ch(t) + Dx(t) \]
SSMs: Continuous Representation

\[ \dot{h} = Ah + Bx \]

\[ y = Ch + Dx \]

Continuous Representation

\[ \dot{h} = \bar{A}h + \bar{B}x \]

\[ y = \bar{C}h + \bar{D}x \]

Recurrent Representation

\[ y = \bar{K} * x \]

Convolutional Representation

Operates on signals and sequences
SSM: Recurrent Representation

\[
\dot{h} = Ah + Bx \\
y = Ch + Dx
\]

Continuous Representation

\[
\dot{h} = \tilde{A}h + \tilde{B}x \\
y = \tilde{C}h + \tilde{D}x
\]

Recurrent Representation

\[
y = \bar{K} * x
\]

Convolutional Representation

Efficient autoregressive computation
Computing SSMs Recurrently

\[ h'(t) = Ah(t) + Bx(t) \]
SSM: Convolutional Representation

Efficient parallelizable computation
Computing SSMs Convolutionally

Output can be computed without computing state
Computing SSMs Convolutionally

\[ y(t) = x(t) \ast K(t) \]
Computing SSMs Convolutionally

\[ y(t) = x(t) \ast K(t) \]

Parallelizable + nearly-linear computation
Computing SSMs Convolutionally

\[ y(t) = x(t) \ast K(t) \]

Generalizes convolutional neural networks (CNN)
Linear Time Invariant (LTI)

Parameters are constant (invariant) through time

\[ h'(t) = A h(t) + B x(t) \]
\[ y(t) = C h(t) + D x(t) \]

Can use LTI SSM to refer to any model that is a:

- **Linear recurrence** (e.g. LRU)
- **Global convolution** (e.g. Hyena)

Great for “continuous” domains (audio, images) but not for text
Outline

• Structured State Space Models (S4)

• Selective State Space Models (Mamba)

• Applications
Motivation: Tradeoffs of the State

Tradeoffs of sequence models can be understood through examining their autoregressive state

RNN  Convolution  Neural ODEs  Attention
Motivation: Tradeoffs of the State

State = **fixed-sized vector** (compression)

✓ Efficient: Constant-time inference, linear-time training

✗ Poor performance on information-dense modalities (language)
Motivation: Tradeoffs of the State

State = cache of entire history (no compression)

✓ Strong performance: Models all connections, long-range dependencies

✗ Inefficient: Linear-time inference, quadratic-time training
Motivation: Tradeoffs of the State

No state compression

Performance ↑
Efficiency ↓

Strong state compression

Efficiency ↑
Performance ↓

\[ \dot{h} = Ah + Bx \]
\[ y = Ch + Dx \]

Continuous Representation
Selection Mechanism

Algorithm 1 SSM (S4)

**Input:** $x : (B, L, D)$  
**Output:** $y : (B, L, D)$

1. $A : (D, N) \leftarrow \text{Parameter}$  
   $\triangleright$ Represents structured $N \times N$ matrix

2. $B : (D, N) \leftarrow \text{Parameter}$
3. $C : (D, N) \leftarrow \text{Parameter}$
4. $\Delta : (D) \leftarrow \tau_\Delta(\text{Parameter})$
5. $A, B : (D, N) \leftarrow \text{discretize}(\Delta, A, B)$
6. $y \leftarrow \text{SSM}(A, B, C)(x)$  
   $\triangleright$ Time-invariant: recurrence or convolution

7. return $y$

Algorithm 2 SSM + Selection (S6)

**Input:** $x : (B, L, D)$  
**Output:** $y : (B, L, D)$

1. $A : (D, N) \leftarrow \text{Parameter}$  
   $\triangleright$ Represents structured $N \times N$ matrix

2. $B : (B, L, N) \leftarrow s_B(x)$
3. $C : (B, L, N) \leftarrow s_C(x)$
4. $\Delta : (B, L, D) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5. $A, B : (B, L, D, N) \leftarrow \text{discretize}(\Delta, A, B)$
6. $y \leftarrow \text{SSM}(A, B, C)(x)$  
   $\triangleright$ Time-varying: recurrence (scan) only

7. return $y$

S4 with selectivity and computed with a scan
Selection Mechanism

Same 1D $\rightarrow$ 1D map, but parameters depend on input
Selection Mechanism

But wait – LTI models were necessary for efficiency
Can't compute large state, must use convolution
Idea: Only materialize the expanded state in more efficient levels of the memory hierarchy
Mamba: A Simplified SSM Architecture

H3  Gated MLP  Mamba
Outline

• Structured State Space Models (S4)

• Selective State Space Models (Mamba)

• Applications
Language Modeling – Scaling Laws

Transformer: GPT-3 model + training recipe
Language Modeling – Scaling Laws

H3, Hyena, RWKV, RetNet: Recent SSMs for LM
Transformer++: Llama model + training recipe
Language Modeling – Scaling Laws

Mamba: First attention-free model to compete with strong modern Transformer models
# Language Modeling – Zero-shot Evals

<table>
<thead>
<tr>
<th>Model</th>
<th>Token.</th>
<th>PILE PPL ↓</th>
<th>LAMBADA ACC ↑</th>
<th>LAMBADA ACC ↑</th>
<th>HELLASWAG ACC ↑</th>
<th>PIQA ACC ↑</th>
<th>ARC-E ACC ↑</th>
<th>ARC-C ACC ↑</th>
<th>WINOGRANDE ACC ↑</th>
<th>AVERAGE ACC ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid H3-130M</td>
<td>GPT2</td>
<td>89.48</td>
<td>25.77</td>
<td>31.7</td>
<td>64.2</td>
<td>44.4</td>
<td>24.2</td>
<td>50.6</td>
<td>40.1</td>
<td></td>
</tr>
<tr>
<td>Pythia-160M</td>
<td>NeoX</td>
<td>38.10</td>
<td>33.0</td>
<td>30.2</td>
<td>61.4</td>
<td>43.2</td>
<td>24.1</td>
<td>51.9</td>
<td>40.6</td>
<td></td>
</tr>
<tr>
<td>Mamba-130M</td>
<td>NeoX</td>
<td>10.56</td>
<td>16.07</td>
<td>44.3</td>
<td>35.3</td>
<td>64.5</td>
<td>48.0</td>
<td>24.3</td>
<td>51.9</td>
<td>44.7</td>
</tr>
<tr>
<td>Hybrid H3-360M</td>
<td>GPT2</td>
<td>12.58</td>
<td>48.0</td>
<td>41.5</td>
<td>68.1</td>
<td>51.4</td>
<td>24.7</td>
<td>54.1</td>
<td>48.0</td>
<td></td>
</tr>
<tr>
<td>Pythia-410M</td>
<td>NeoX</td>
<td>10.84</td>
<td>51.4</td>
<td>40.6</td>
<td>66.9</td>
<td>52.1</td>
<td>24.6</td>
<td>53.8</td>
<td>48.2</td>
<td></td>
</tr>
<tr>
<td>Mamba-370M</td>
<td>NeoX</td>
<td>8.28</td>
<td>8.14</td>
<td>55.6</td>
<td>46.5</td>
<td>69.5</td>
<td>55.1</td>
<td>28.0</td>
<td>55.3</td>
<td>50.0</td>
</tr>
<tr>
<td>Pythia-1B</td>
<td>NeoX</td>
<td>7.82</td>
<td>7.92</td>
<td>56.1</td>
<td>47.2</td>
<td>70.7</td>
<td>57.0</td>
<td>27.1</td>
<td>53.5</td>
<td>51.9</td>
</tr>
<tr>
<td>Mamba-790M</td>
<td>NeoX</td>
<td>7.33</td>
<td>6.02</td>
<td>62.7</td>
<td>55.1</td>
<td>72.1</td>
<td>61.2</td>
<td>29.5</td>
<td>56.1</td>
<td>57.1</td>
</tr>
<tr>
<td>GPT-Neo 1.3B</td>
<td>GPT2</td>
<td>7.50</td>
<td>57.2</td>
<td>48.9</td>
<td>71.1</td>
<td>56.2</td>
<td>25.9</td>
<td>54.9</td>
<td>52.4</td>
<td></td>
</tr>
<tr>
<td>Hybrid H3-1.3B</td>
<td>GPT2</td>
<td>11.25</td>
<td>49.6</td>
<td>52.6</td>
<td>71.3</td>
<td>59.2</td>
<td>28.1</td>
<td>56.9</td>
<td>53.0</td>
<td></td>
</tr>
<tr>
<td>OPT-1.3B</td>
<td>OPT</td>
<td>6.64</td>
<td>58.0</td>
<td>53.7</td>
<td>72.4</td>
<td>56.7</td>
<td>29.6</td>
<td>59.5</td>
<td>55.0</td>
<td></td>
</tr>
<tr>
<td>Pythia-1.4B</td>
<td>NeoX</td>
<td>7.51</td>
<td>6.08</td>
<td>61.7</td>
<td>52.1</td>
<td>71.0</td>
<td>60.5</td>
<td>28.5</td>
<td>57.2</td>
<td>55.2</td>
</tr>
<tr>
<td>RWKV-1.5B</td>
<td>NeoX</td>
<td>7.70</td>
<td>7.04</td>
<td>56.4</td>
<td>52.5</td>
<td>72.4</td>
<td>60.5</td>
<td>29.4</td>
<td>54.6</td>
<td>54.3</td>
</tr>
<tr>
<td>Mamba-1.4B</td>
<td>NeoX</td>
<td>6.80</td>
<td>5.04</td>
<td>64.9</td>
<td>59.1</td>
<td>74.2</td>
<td>65.5</td>
<td>32.8</td>
<td>61.5</td>
<td>59.7</td>
</tr>
<tr>
<td>GPT-Neo 2.7B</td>
<td>GPT2</td>
<td>5.63</td>
<td>62.2</td>
<td>55.8</td>
<td>72.1</td>
<td>61.1</td>
<td>30.2</td>
<td>57.6</td>
<td>56.5</td>
<td></td>
</tr>
<tr>
<td>Hybrid H3-2.7B</td>
<td>GPT2</td>
<td>7.92</td>
<td>55.7</td>
<td>59.7</td>
<td>73.3</td>
<td>65.6</td>
<td>32.3</td>
<td>61.4</td>
<td>58.0</td>
<td></td>
</tr>
<tr>
<td>OPT-2.7B</td>
<td>OPT</td>
<td>5.12</td>
<td>63.6</td>
<td>60.6</td>
<td>74.8</td>
<td>60.8</td>
<td>31.3</td>
<td>61.0</td>
<td>58.7</td>
<td></td>
</tr>
<tr>
<td>Pythia-2.8B</td>
<td>NeoX</td>
<td>6.73</td>
<td>5.04</td>
<td>64.7</td>
<td>59.3</td>
<td>74.0</td>
<td>64.1</td>
<td>32.9</td>
<td>59.7</td>
<td>59.1</td>
</tr>
<tr>
<td>RWKV-3B</td>
<td>NeoX</td>
<td>7.00</td>
<td>5.24</td>
<td>63.9</td>
<td>59.6</td>
<td>73.7</td>
<td>67.8</td>
<td>33.1</td>
<td>59.6</td>
<td>59.6</td>
</tr>
<tr>
<td>Mamba-2.8B</td>
<td>NeoX</td>
<td>6.22</td>
<td>4.23</td>
<td>69.2</td>
<td>66.1</td>
<td>75.2</td>
<td>69.7</td>
<td>36.3</td>
<td>63.5</td>
<td>63.3</td>
</tr>
<tr>
<td>GPT-J-6B</td>
<td>GPT2</td>
<td>–</td>
<td>68.3</td>
<td>66.3</td>
<td>75.4</td>
<td>67.0</td>
<td>36.6</td>
<td>64.1</td>
<td>63.0</td>
<td></td>
</tr>
<tr>
<td>OPT-6.7B</td>
<td>OPT</td>
<td>–</td>
<td>67.7</td>
<td>67.2</td>
<td>76.3</td>
<td>65.6</td>
<td>34.9</td>
<td>65.5</td>
<td>62.9</td>
<td></td>
</tr>
<tr>
<td>Pythia-6.9B</td>
<td>NeoX</td>
<td>6.51</td>
<td>4.45</td>
<td>67.1</td>
<td>64.0</td>
<td>75.2</td>
<td>67.3</td>
<td>35.5</td>
<td>61.3</td>
<td>61.7</td>
</tr>
<tr>
<td>RWKV-7.4B</td>
<td>NeoX</td>
<td>6.31</td>
<td>4.38</td>
<td>67.2</td>
<td>65.5</td>
<td>76.1</td>
<td>67.8</td>
<td>37.5</td>
<td>61.0</td>
<td>62.5</td>
</tr>
</tbody>
</table>

Mamba matches/beats Transformers of similar size.
DNA Pretraining

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>T</td>
</tr>
</tbody>
</table>

Task
Next-token (base pair) pretraining for DNA

Challenge
Can have extremely long-range interactions

Towards genomics foundation models
DNA Scaling Laws – Context Length

Unlike LTI – better scaling with context length
Audio Modeling – Pretraining

Improved perplexity up to 1M sequences (1min audio)
Summary – Mamba

Match or beat strongest Transformer architecture on language

Key algorithmic ideas: selection mechanism, hardware-aware state expansion

Upshot: better models with linear (instead of quadratic) scaling in sequence length

Code: https://github.com/state-spaces/mamba/
Implications for Foundation Models

LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

Extensive work (and speculation) on how statistical modeling assumptions might lead to downstream properties!
Implications for Foundation Models

LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

What if the architecture is the root of these phenomena?
Implications for Foundation Models

LLMs/FMs have many mysterious properties and affordances

...but what is an LLM?

What if the architecture is the root of these phenomena?
Implications for Foundation Models

Scenario 1: SSMs work as well as Transformer downstream

✓ The next dominant architecture?

Scenario 2: SSMs are missing some downstream capabilities

✓ Deeper understanding of FMs