CS 639: Foundation Models
Attention Variations

Fred Sala

University of Wisconsin-Madison

Feb. 24, 2026

Outline

- **Finish up Decoder-Only Models**
  - From last time: GPT family of models, scaling, new ideas and tricks,
- **Attention: Variations**
  - Motivation, basic forms of sparse attention, beyond softmax, hardware-aware implementations
- **Beyond Attention**
  - Basic ideas, S4 sequence model, Mamba model

Outline

- **Finish up Decoder-Only Models**
  - From last time: GPT family of models, scaling, new ideas and tricks,
- **Attention: Variations**
  - Motivation, basic forms of sparse attention, beyond softmax, hardware-aware implementations
- **Beyond Attention**
  - Basic ideas, S4 sequence model, Mamba model

Decoder-Only Models: GPT

Let's get rid of the first part

- 1) **Outputs** in natural language
- 2) Tight alignment with **input**

- Let's handle this acronym as well: **GPT**
  - **G**enerative (i.e., a language model that generates rich content, as opposed to representations or predictions)
    - Unlike BERT!
  - **P**retrained
    - Like BERT!
  - **T**ransformers (also like BERT)

Quick Interlude: Language Models

What's a "language model"?

- Basic idea: use probabilistic models to assign a probability to a sentence W

$$P(W) = P(w_1, w_2, \ldots, w_n) \text{ or } P(w_{\text{next}} | w_1, w_2 \ldots)$$

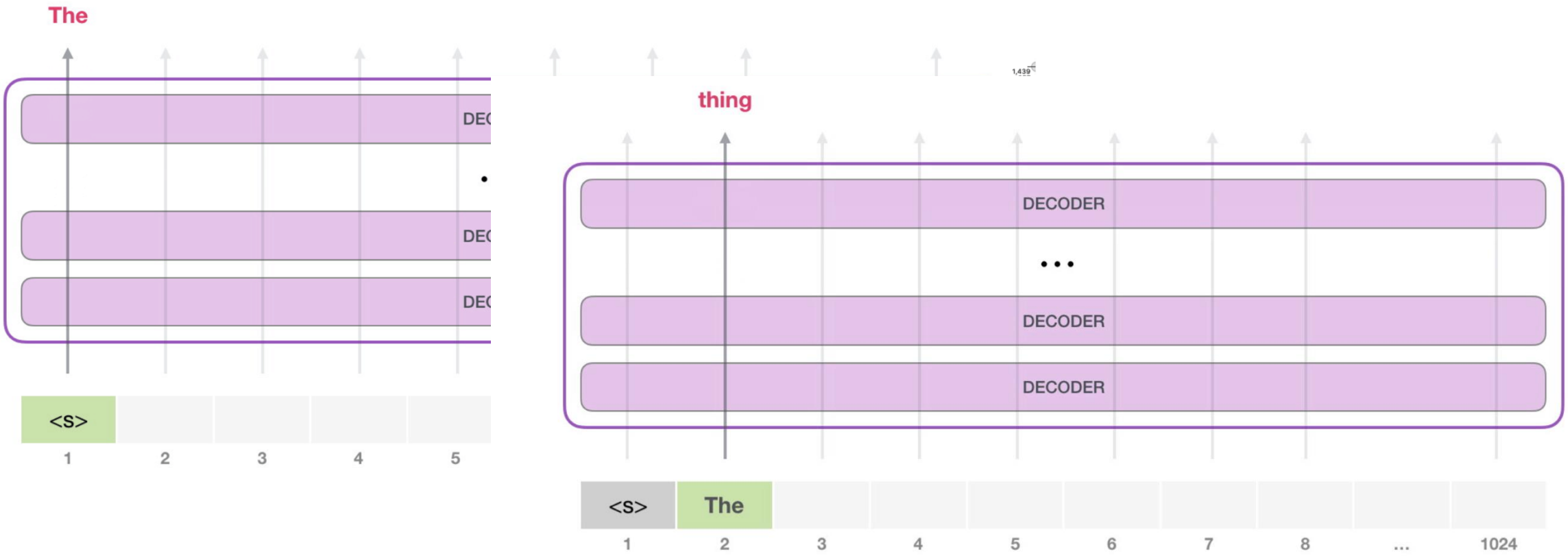
- We mostly care about the latter: gives us a **distribution to sample next word** (or next token)
- GPT models will also do this!
  - But idea's much older: Shannon's example

Zero-order approximation	XFOML RXKHRJFFJUJ ALPWXFWJXYJ FFJEYVJCQSGHYD QPAAMKBZAACIBZLKJQD
First-order approximation	OCRO HLO RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL
Second-order approximation	ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE
Third-order approximation	IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE
First-order word approximation	REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

Decoder-Only Models: GPT

Autoregressive next token prediction mechanism:

- Plug in your current token, get next token
- Once you decode next token, plug it back in



From GPT2 to GPT3

Mainly make things larger!

- Why? **Scaling** produces emergent behaviors… more soon!
- 96 decoder blocks (getting very tall)
- Context size: **2048**
- 175 billion parameters in total (800GB!)

Training dat

GPT-3 training data[1]:9

Dataset	# tokens	Proportion within training
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

https://en.wikipedia.org/wiki/GPT-3

Brown et al ‘20

Open Source: Llama 3.1

Mainly make things larger! Note: multiple model sizes:

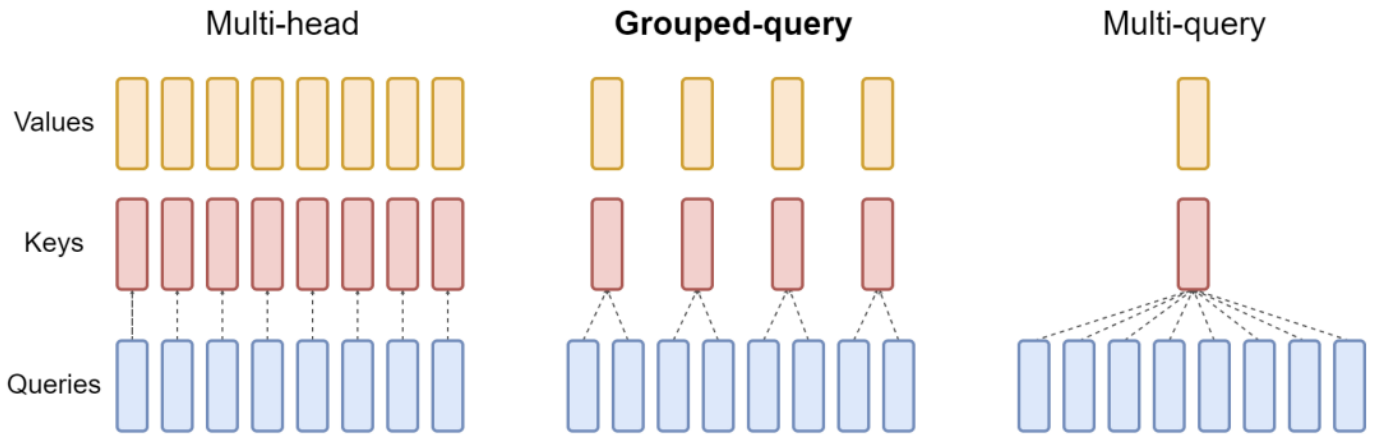
	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

Dubey et al ‘24

Open Source: Llama 3.1

Some improvements for Llama 3.1:

- "We use an attention mask that **prevents self-attention between different documents** within the same sequence"
- "**grouped query attention** (GQA; Ainslie et al. (2023)) with 8 key-value heads to improve inference speed…"



Ainslie et al '23

ɔ et al '21

Open Source: Llama 3.1

Some improvements for Llama 3.1:

- "We use an attention mask that **prevents self-attention between different documents** within the same sequence"
- "**grouped query attention** (GQA; Ainslie et al. (2023)) with 8 key-value heads to improve inference speed…"
- "We use a **vocabulary with 128K tokens.** Our token vocabulary combines 100K tokens from the tiktoken3 tokenizer with 28K additional tokens to better support non-English languages"

Zhao et al '21

Break & Questions

Outline

- **Finish up Decoder-Only Models**
  - From last time: GPT family of models, scaling, new ideas and tricks,
- **Attention: Variations**
  - Motivation, basic forms of sparse attention, beyond softmax, hardware-aware implementations
- **Beyond Attention**
  - Basic ideas, S4 sequence model, Mamba model

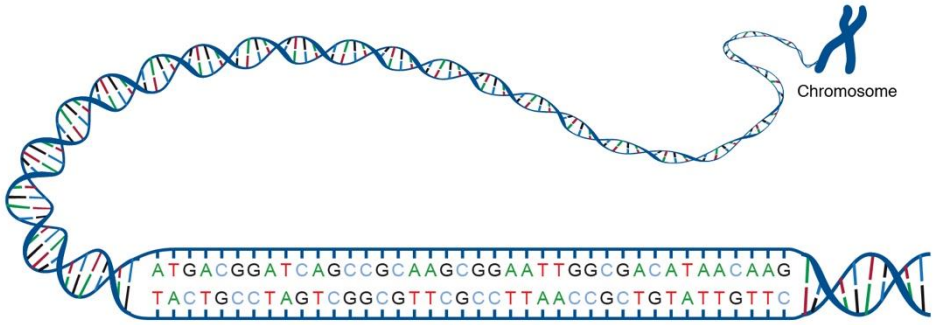
Motivation: Complexity

Back to our computational complexity issues

- Self-attention requires comparing each token to each token
  - Query vs key
- For a sequence of length n, this is n^2 dot products
  - Worse, per-head per-layer!

This is expensive!

- We want to put long objects in our context window
- Documents, code, other naturally long objects (e.g., genetic strings)



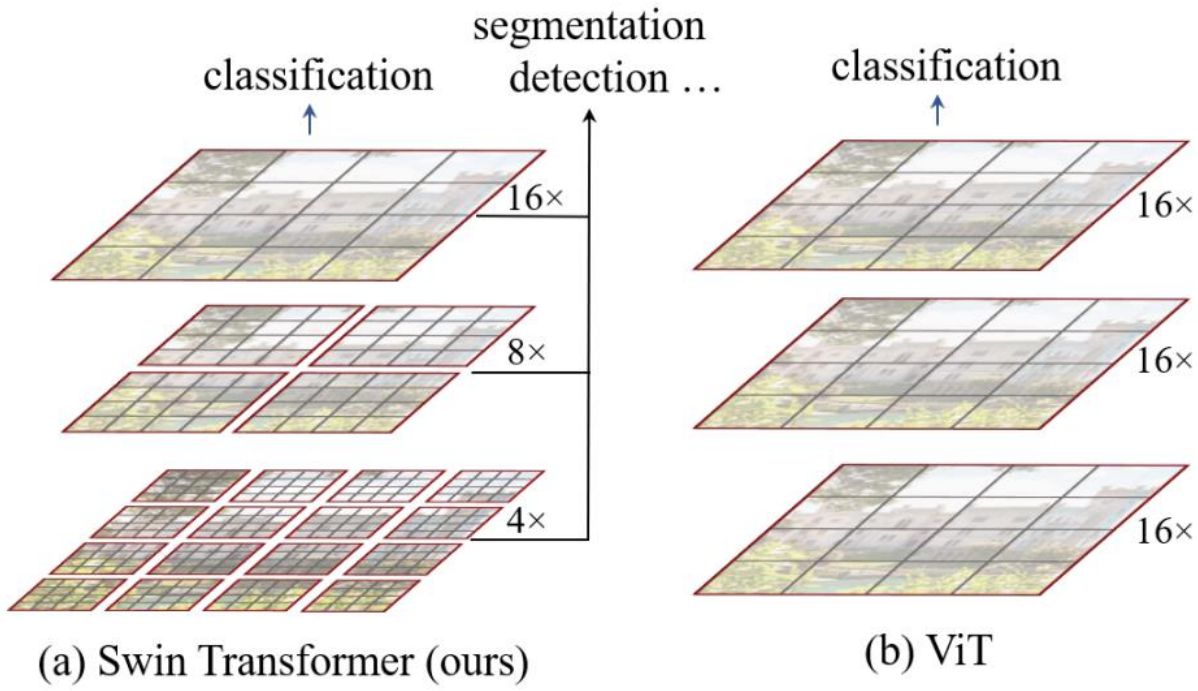
Motivation: Biases

Beyond cost, there are other behaviors we'd seek

- Example: for images, we want *locality*
- We saw this when we built CNNs

Side note: all our Transformers processed text so far,

- Next we'll build them for other data modalities
- Not trivial; so far we relied heavily on aspects of text sequences
- Right: image Transformers



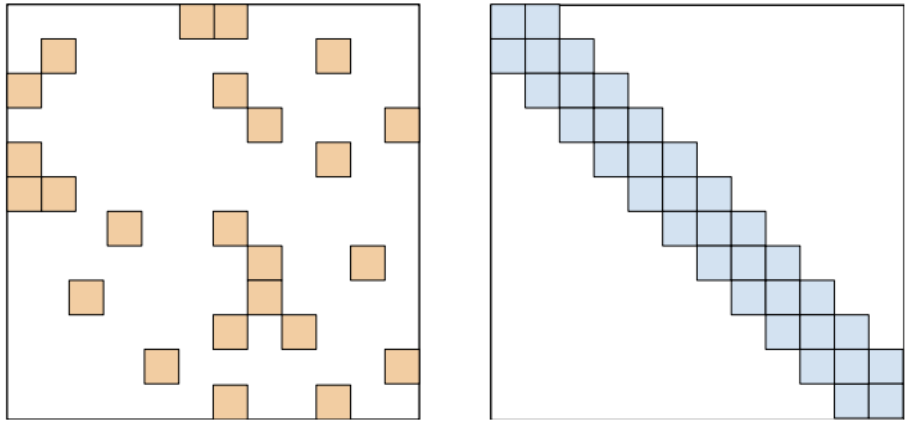
Sparse Attention Mechanisms

Sparsity is a basic way to reduce complexity

- ***Sparse vector***: contains less than O(*n*) non-zeroes
- ***Sparse matrix***: contains less than O(*n* x *m*) non-zeroes

We can do this with attention too!

- To start with, we'll do fewer comparisons, but leave everything else unchanged
- I.e., we'll keep softmax
  - And come back to this later



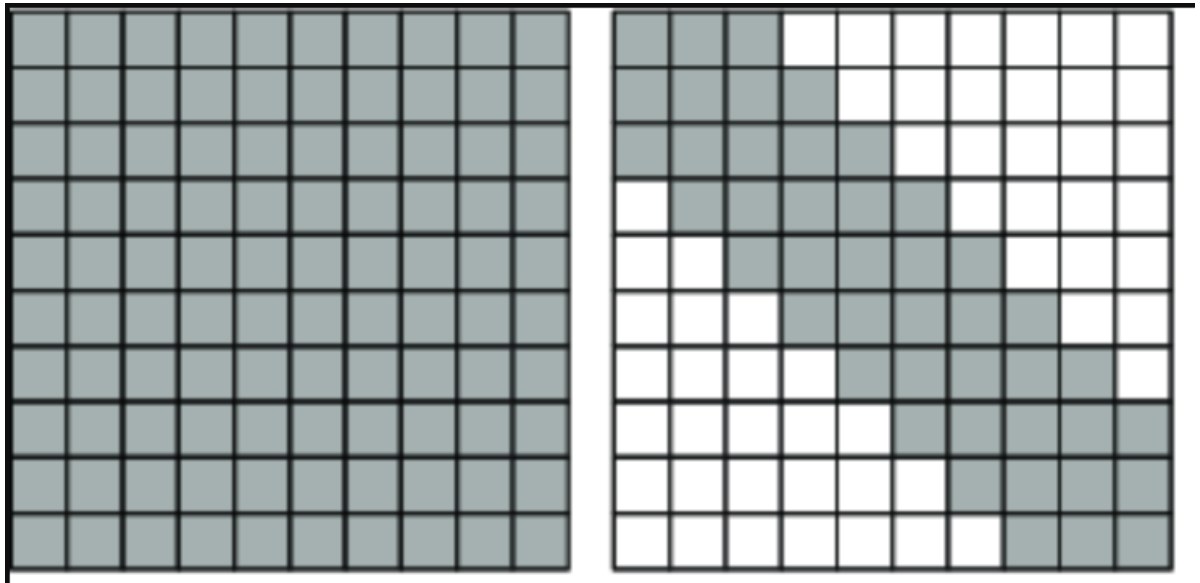
(a) Random attention (b) Window attention

Zaheer at al

Variation 1: Sliding Window

Instead of full attention, only use a **window** around the token we are currently examining

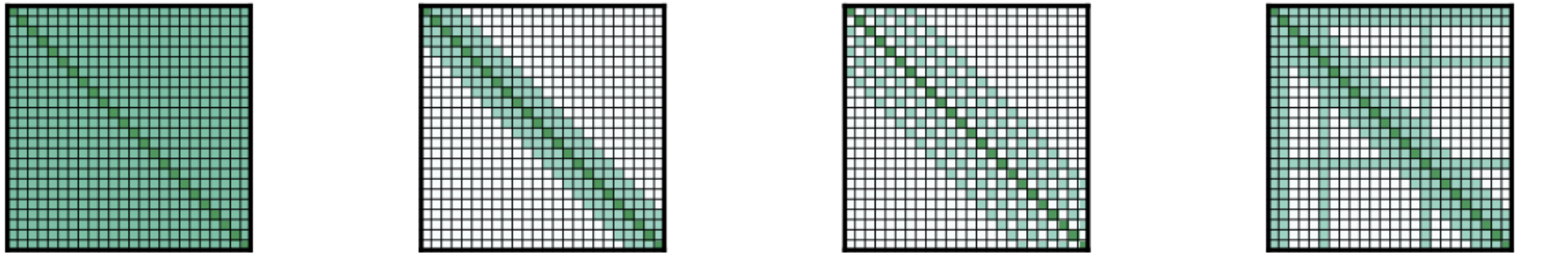
- Basically, relies on locality
- Not very different from the CNNs we built earlier
- Complexity: *sequence length* x *size of window*
- **Downside**: can miss some important information



Variation 2: Local + Global

Basic idea: Sliding window may forget some important aspects, so we need something more.

- Add “global” tokens that get compared to all other tokens being attended to (and vice-versa)
- Idea introduced in ***LongFormer*** (Beltagy et al ‘20)
- Note: we don’t use too many



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Variation 3: Block Attention

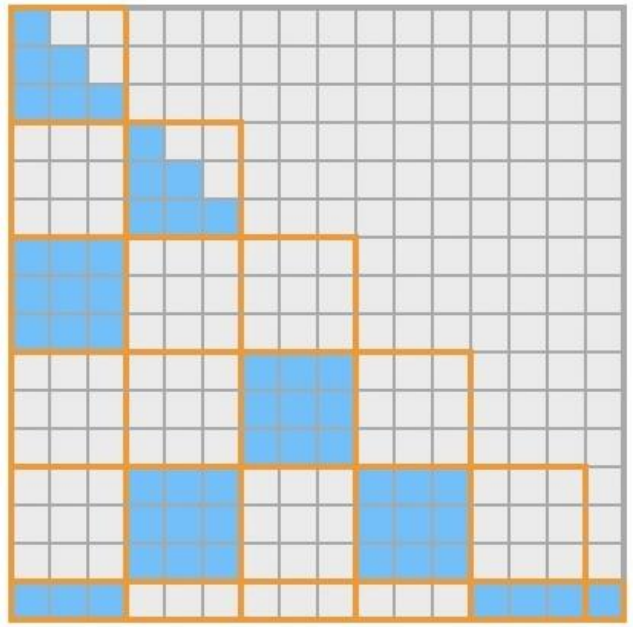
Instead of sparsity in terms of entries, sparsity in terms of blocks

- I.e., blocks generally contain information
- Attend to the whole thing

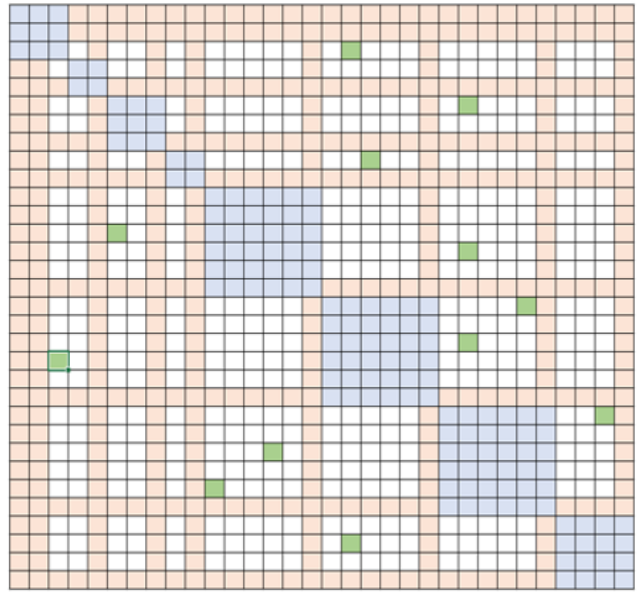
- Can also be combined with other approaches.

Ex: block + global + random

This is another design problem!



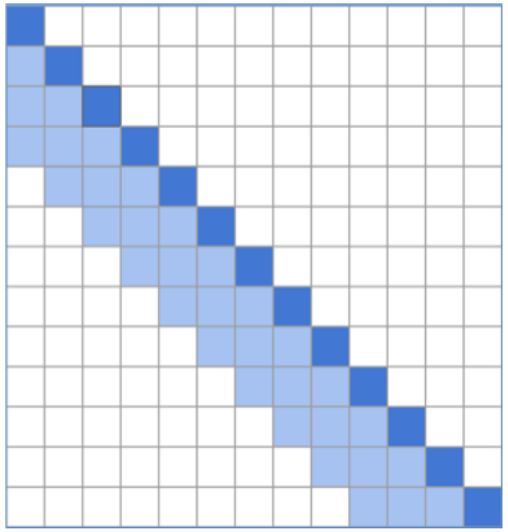
Variable Sparsity Structure



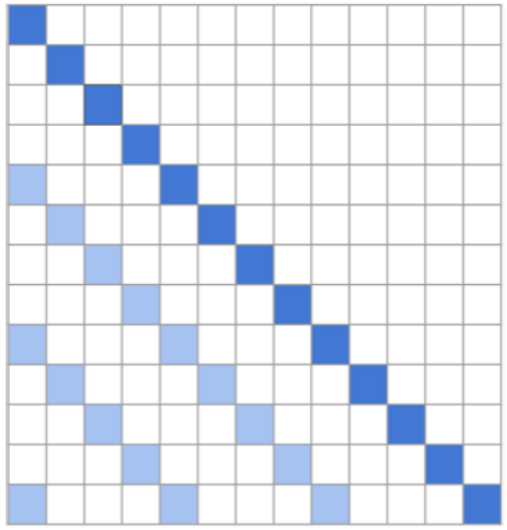
Variation 4: Learned Attention

Likewise, we can *learn* where the positions are that we should attend to

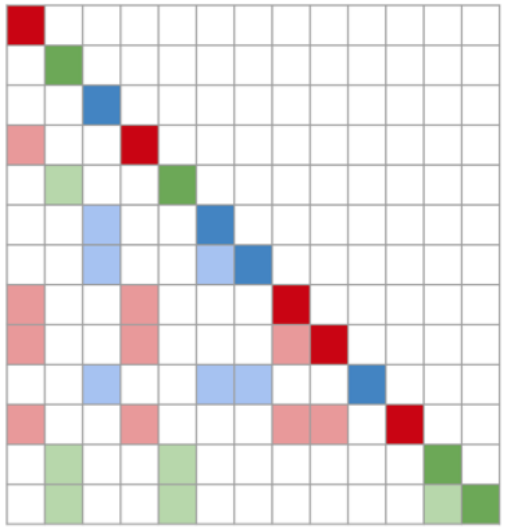
- Classic example: Routing Transformer (Roy et al '20)
- How? ***Online k-means clustering*** to tell you where to look



(a) Local attention



(b) Strided attention



(c) Routing attention

Beyond Sparsity

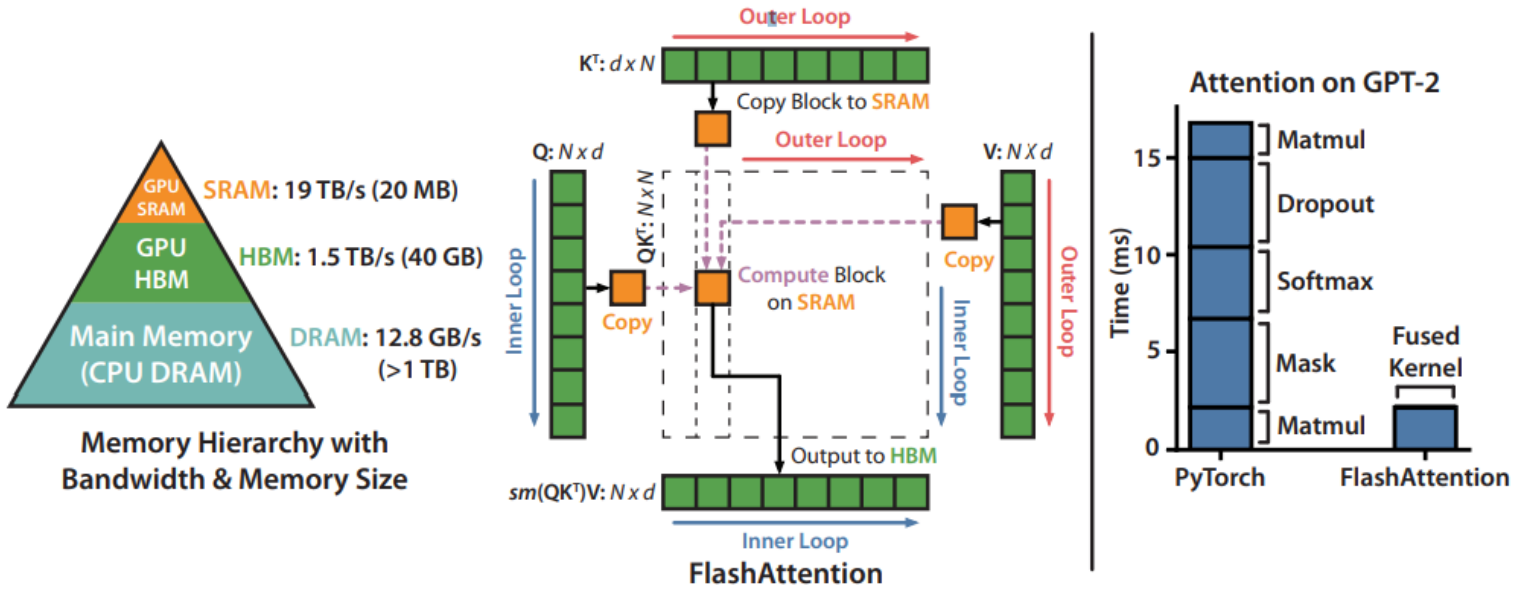
We can also change other fundamental aspects of attention,

- A big one is softmax → precludes any way to really parallelize the operation
- One solution: "**linear attention**". Gets rid of softmax and factorizes in a clever way (Katharopoulos et al '20)
  - **Intuition**: the query/key interactions are nonlinear with softmax, but if we get rid of it, we can compute K^TV once, then multiply by each query as we go.

Hardware Considerations?

So far we dealt with **computational complexity**… but we should also think about practical implementation issues

- GPUs: A little bit of fast memory, lots of slower memory
- Avoid using slow memory when possible?

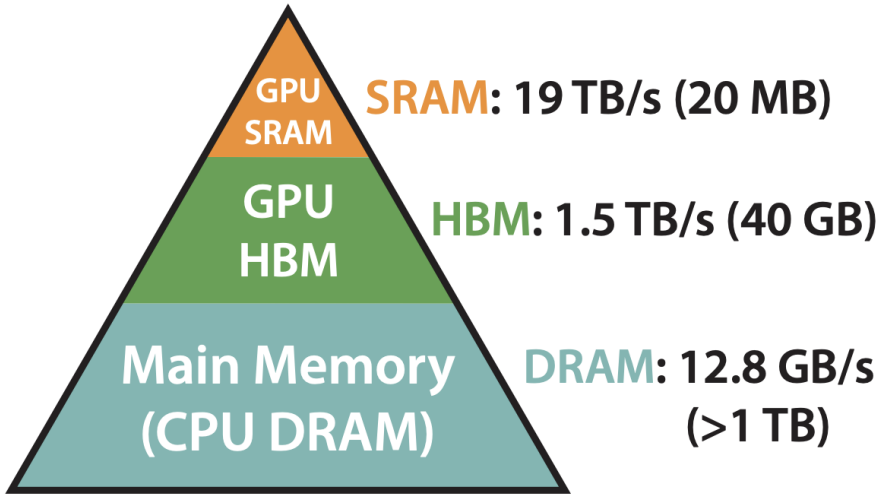


Dao et al ‘22

Exact, Hardware-Aware Attention:

Idea for FlashAttention

- Different kinds of GPU memory



Memory Hierarchy with Bandwidth & Memory Size

- Fast: on-chip SRAM
  - But very little of this: 192KB for each of ~100 processors for an A100 (20MB)
- Slow(er): HBM
  - But lots: 40-80GB for an A100

- **Goal**: use fast as much as possible, avoid moving to HBM

Flash Attention: **Basic Idea**

Will use two tricks for higher efficiency

- Tiling and re-computing.

First, recall standard attention

- Will use HBM memory repeatedly
  - Lots of reads and writes:

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Flash Attention: **Tiling**

Will use two tricks for higher efficiency

- Tiling and re-computing.

How do we avoid writing and reading from HBM?

- A: don't load the whole thing, use custom **tiling** and save the pieces (small). Standard version

$$m(x) := \max_i \; x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \text{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

- Tiling version: two components (can extend)

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Flash Attention: **Recomputing**

Will use two tricks for higher efficiency

- Tiling and re-computing.

How do we avoid writing and reading from HBM?

- A: don't load the whole thing, use custom **tiling** and save the pieces

"Tiling enables us to implement our algorithm in one CUDA kernel, loading input from HBM, performing all the computation steps (matrix multiply, softmax, optionally masking and dropout, matrix multiply), then write the result back to HBM (masking and dropout in Appendix B). This avoids repeatedly reading and writing of inputs and outputs from and to HBM."

Don't we need to store full S, P for backwards pass, anyway?

- A: **No!** Can recompute on the fly S, P on the fly

Flash Attention: **Tradeoffs?**

Will use two tricks for higher efficiency

- Tiling and re-computing.

What's the tradeoff?

- Using tiling and computing/re-computing things normally trades off **memory consumption** for **speed**

- **But…** by reducing memory consumption, we can stick to fast memory only
  - And this makes us **much faster**
  - So **no tradeoff** at all (except for needing custom CUDA kernels ☺)

Flash Attention: **Tradeoffs?**

Will use two tricks for higher efficiency

- Tiling and re-computing.

Results:

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Break & Questions

Outline

- **Finish up Decoder-Only Models**
  - From last time: GPT family of models, scaling, new ideas and tricks,
- **Attention: Variations**
  - Motivation, basic forms of sparse attention, beyond softmax, hardware-aware implementations
- **Beyond Attention**
  - Basic ideas, S4 sequence model, Mamba model

Attention Alternatives?

Another approach is to get rid of attention and its quadratic cost altogether. Many new alternatives!

- Sometimes called **sub-quadratic** models.
- We'll briefly study a few.
- Step 1: let's get inspired by something RNN-like (well, fully linear for now). Borrow from continuous models:

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$
$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

State-Space Model

Step 1: let's get inspired by something RNN-like (well, fully linear for now). Borrow from continuous models:

State Input

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$

Output → $$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

- Can ignore the "D" (think of this as a skip connection).
- Inputs, outputs are 1-D, state is higher dimensional.

State-Space Model: Discrete Form

Step 2: let's make this a discrete function

State Input

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$

Output → $$y_k = \overline{\boldsymbol{C}} x_k$$

- Ignored D
- Can create approximations of A,B,C through discretizing.
- Looks a lot like an RNN! (or, a linear version of one)

State-Space Model: Convolutional Form

Step 3: let's unroll the recursion

$$x_0 = \overline{B}u_0 \qquad x_1 = \overline{AB}u_0 + \overline{B}u_1 \qquad x_2 = \overline{A}^2\overline{B}u_0 + \overline{AB}u_1 + \overline{B}u_2$$

$$y_0 = \overline{CB}u_0 \qquad y_1 = \overline{CAB}u_0 + \overline{CB}u_1 \qquad y_2 = \overline{CA}^2\overline{B}u_0 + \overline{CAB}u_1 + \overline{CB}u_2$$

$$y_k = \overline{CA}^k\overline{B}u_0 + \overline{CA}^{k-1}\overline{B}u_1 + \cdots + \overline{CAB}u_{k-1} + \overline{CB}u_k$$

- In general, $y = \overline{K} * u.$

- This is a **convolution**!

State-Space Model: Convolutional Form

Step 3: let's unroll the recursion

- Convolution

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

- But a weird one. It's a very **long** convolution.
  - Kernel as long as the input sequence (say, L).
  - Naively, is this better than attention?
  - Let's do **something else** instead.

Interlude: Time & Frequency Domains

Back to Signals and Systems class,

- Convolution in the time-domain is element-wise multiplication in the frequency domain
- So low-complexity.
- But, need to convert to frequency domain
- Solution: **FFT.** O(L log L) (and also for iFFT, to invert back).
- So, can compute fast and use during training!

$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

Back to SSM Picture

Back to the formula

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

- Just directly making all of these trainable parameters doesn't work so well.
  - Similar issues as in RNNs: stuff blowing up
  - Instead, various models propose approaches

S4 (Structured State Space Models) Gu et al' 22

- Build A with a special fixed transition matrix that is good at memorization
- Couple with a particular parametrization to get the discretization.

Using SSMs as Layers

Back to the formula

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

S4 (Structured State Space Models) Gu et al' 22

- Special A state transition matrix
- Special parametrization/choice of trainable parameters

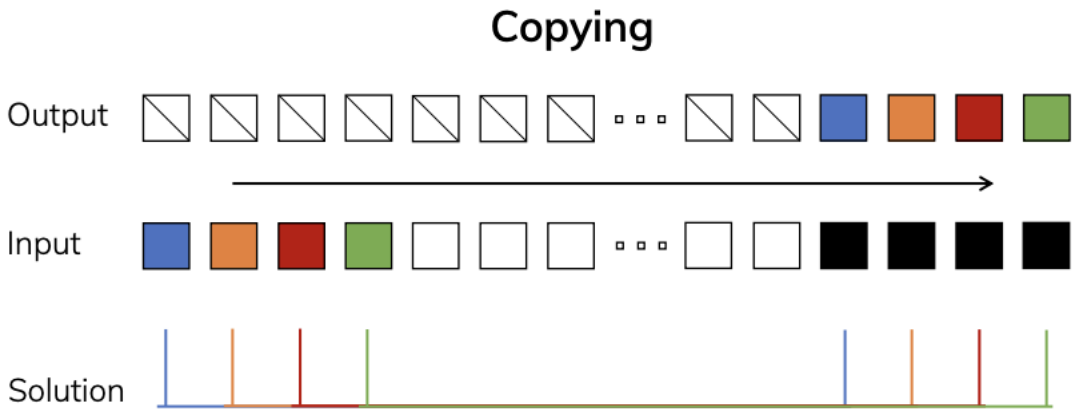
- How to actually use these? Need to define a layer,
  - Stack H of them together (similar to conv layers, multihead attn)
  - Mix with linear layer, place activation function at the end

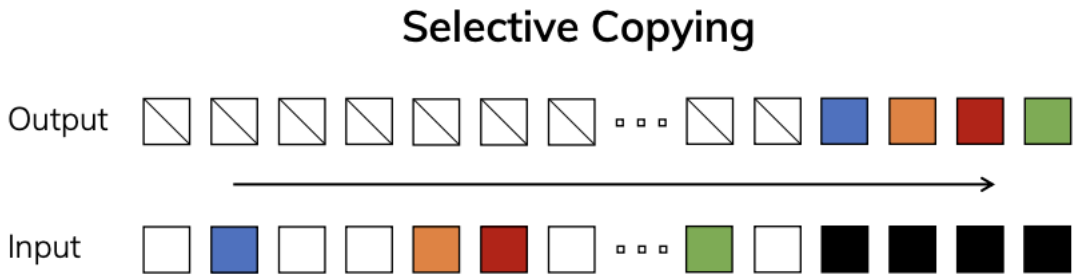
S4 Results: The Good and the Bad

Models like S4 can address **very long sequences**

- "S4 solves the **Path-X task**, an extremely challenging task that involves reasoning about LRDs over sequences of length … 16384. All previous models have failed…"

- But, can struggle with "selective" tasks.





S4 Results: The Good and the Bad

Solution: need some type of context-aware approach

- **Mamba Model**
  - Gu and Dao ‘23, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Thank You!