Attention Improvements

Problems with full attention

- Attention Matrices can become quite large (N x N) since attention is calculated between all pairs of tokens.
- The Required Memory also increases quadratically
- Not suitable for large input sizes like images, video, audio
- Useless tokens might increase size unnecessarily
- Noise created by useless tokens
- Dot-product of vectors with many dimensions grows large, causing the softmax function to have small gradients

Sparse attention

Idea: Don’t calculate attention scores for every token in relation to every other token -> Only use a subset

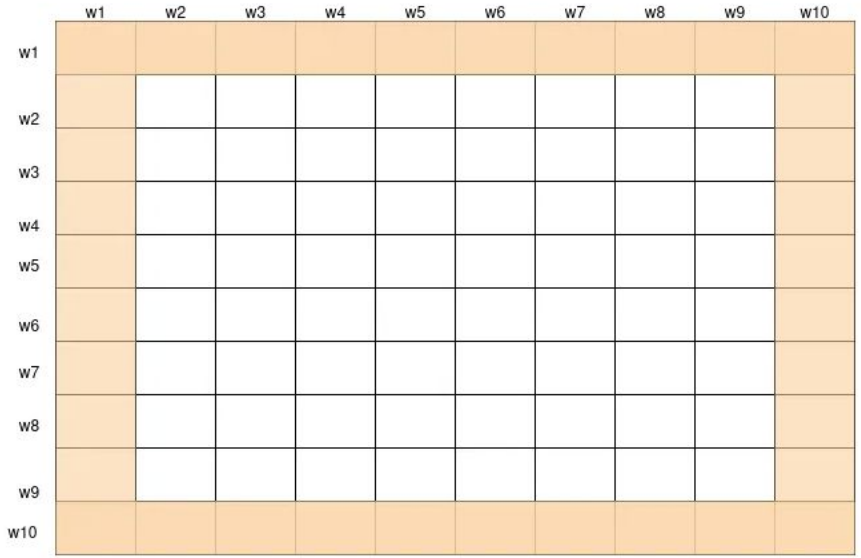
How to choose the subset

1. Global
2. Window
3. Random

Add these all together

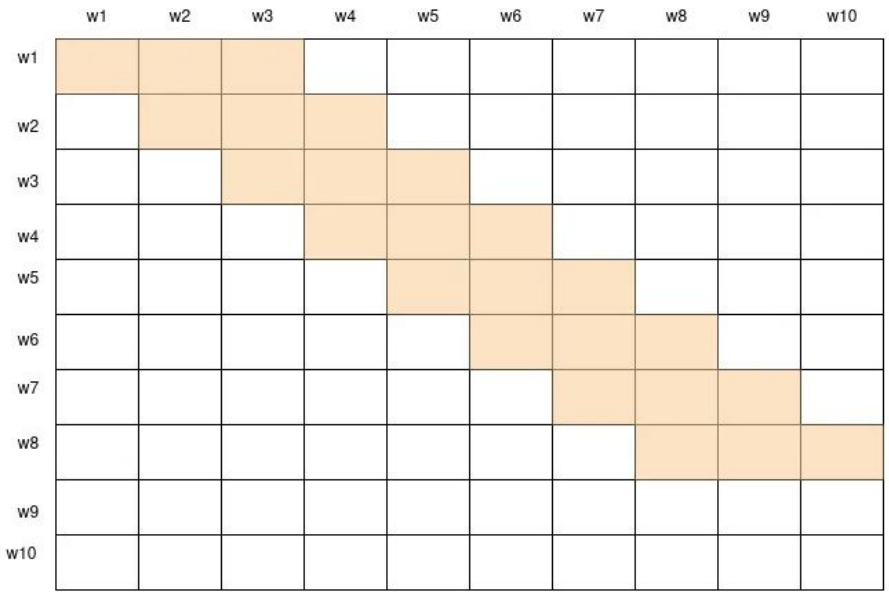
Global tokens

- Typical tokens
  - First
  - EOS
  - Task Specific: Question Token
- Implementation: Set of tokens that are attended by all tokens and attend all tokens
- Task: Serve as a summary of the sequence, simplify long-distance information propagation



Window tokens

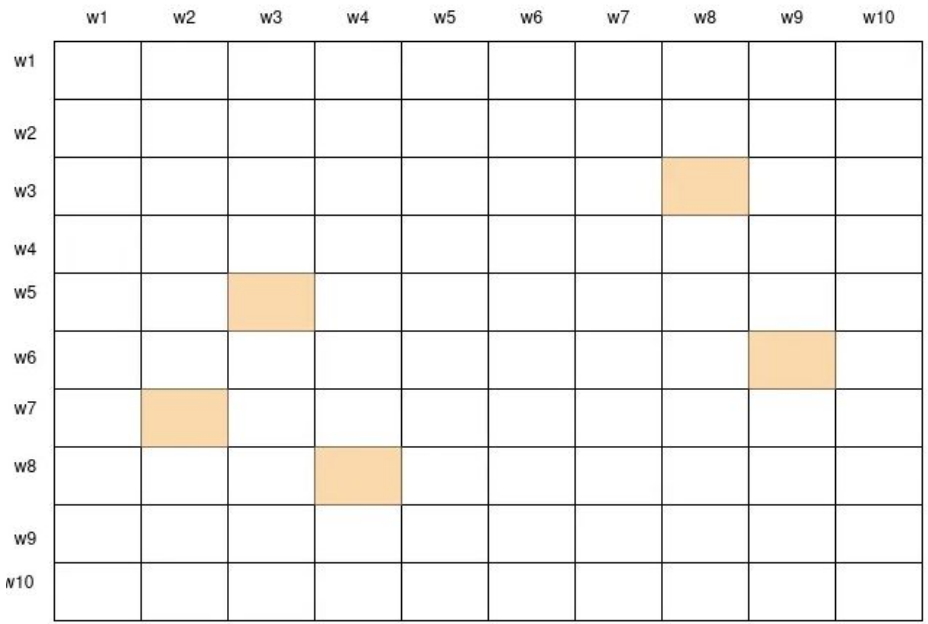
- Tokens: All
- Implementation: Each token attends to a window of neighbouring tokens on both sides
- Task: Captures local context and structure of the sequence



window attention score with window size = 3

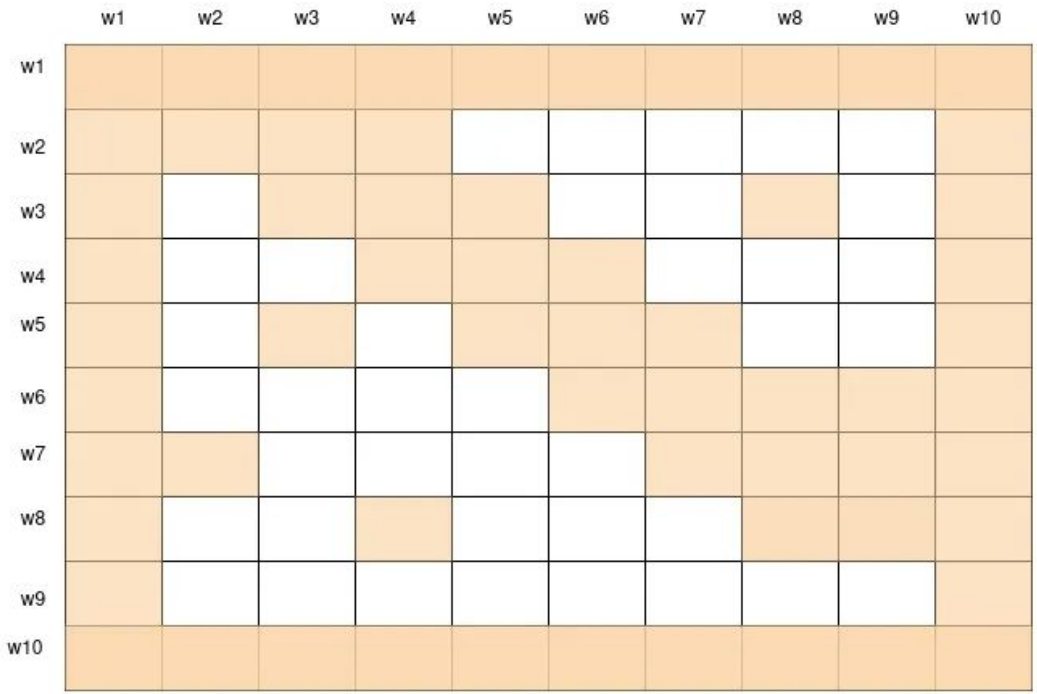
Random tokens

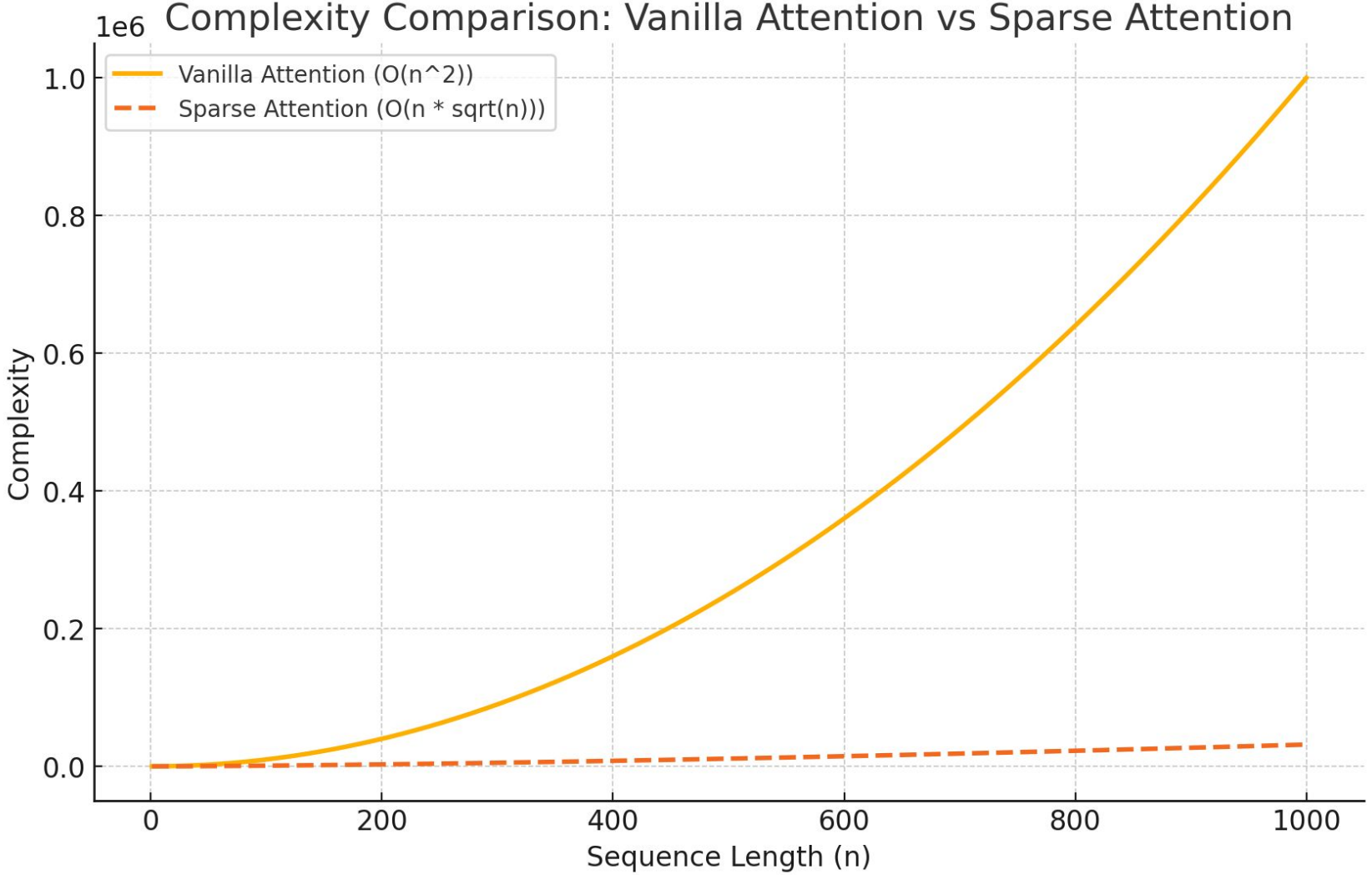
- Tokens: Random
- Implementation: All tokens attend to a set of random tokens
- Task: Introduces some randomness and diversity in the attention pattern, helps connect distant tokens outside of the same window



Final attention scores

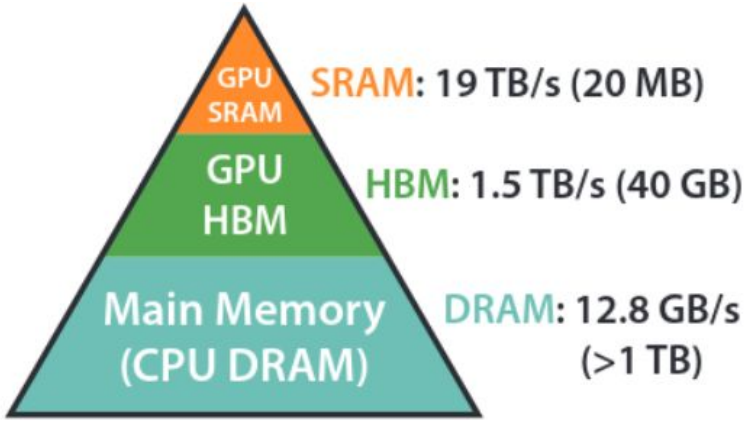
- Overlap all 3 attention scores




Complexity Comparison: Vanilla Attention vs Sparse Attention
1e6
Vanilla Attention (O(n^2))
Sparse Attention (O(n * sqrt(n)))
Complexity
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
Sequence Length (n)

Flash Attention: Problem

- Modern GPUs *Fast computation* vs. *slow memory*
- *Small, fast SRAM* vs. *large, slow HBM* (High Bandwidth Memory)
- **Memory access bottleneck**

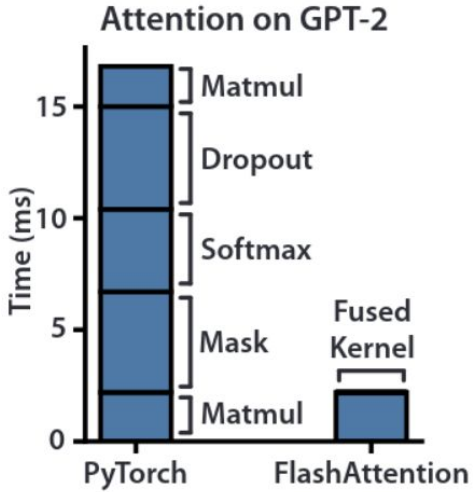


Memory Hierarchy with Bandwidth & Memory Size

Flash Attention: Idea

- Idea: *“Move memory access time to computation time”*
- Compute softmax reduction without loading the whole input
- Avoid storing the large intermediate attention matrix on backward pass

- 7.6x speedup over PyTorch for GPT-2



Flash Attention: Solution

Tiling

- Break down attention matrix into smaller blocks
- Softmax reduction is performed *incrementally* (block by block)

Recomputation

- Softmax normalization factor gets stored on forward pass
- Recompute attention on the backward pass
- Recomputation is faster than load + store intermediate attention matrices on the HBM

Tiling enables **Kernel fusion**

- Combines operations that takes places on the same input data into a *CUDA kernel*
- Leads to a single memory access for N operations

Flash attention: Algorithm (1)

1. Initialisation in the HBM of:
   - *Q, K, V, O*: Query, Key, Value and Output matrices
   - ℓ: vector of normalization factors for the softmax computation
   - *m*: vector of maximum values for the softmax computation
2. Block formation of of **all** initialised matrices and vectors

Flash attention: Algorithm (2)

Forward Pass

- For each *K* and *V* block over j
  - **Load** *K_j* and *V_j* from HBM to SRAM
  - For each *Q* block over i
    - **Load** *Q_i*, *O_i*, *ℓ_i* from HBM to SRAM
    - Calculate attention score for the current block
    - Calculate softmax values (**Tiling**)
    - Update the output matrix
    - **Store** the updated normalisation factors and maximum values *ℓ_i* and *m_i* in HBM

Flash attention: Algorithm (3)

Backward Pass

- **Load** *Q, K, V, O, ℓ, m* and *dO* (output gradient) from HBM
- For each *K* and *V* block over j
  - **Load** *K_j*, *V_j, dK_j, dV_j* from HBM to SRAM
  - For each *Q* block over i
    - **Load** *Q_i*, *O_i*, *ℓ_i* from HBM to SRAM
    - **Recompute** the attention score matrix
    - Calculate the gradients of Q_i, K_j, V_j
    - **Store** these gradients in HBM
  - Return the updated gradients

Flash Attention: Complexity

- IO complexity from $O(N * d + N^2)$ to $O(N^2d^2M^{-1})$ -> up to 9x faster
  - N: length of input sequence
  - d: dimension of the input vectors
  - M: size of SRAM
- Space complexity from $O(N^2)$ to $O(N)$

Flash Attention: Training Speed

- BERT-large: 15% faster than record in MLPerf 1.1
- GPT2: 3x faster than baseline implementations from HuggingFace and MegatronLM
- Long-range area: 2.4x faster than baselines

Flash Attention: Quality

- Scales Transformers to longer sequences => higher quality
- **0.7 improvement** in perplexity on GPT-2
- **6.4 points of lift** from modeling longer sequences on long-document classification
- Longer sequence length (16K)
- Even better: Block-sparse Flash Attention (64K)
- Path-X & Path-256: first result transformer models being able to solve those challenges

Flash Attention: Benchmarks

- Memory footprint scales linearly with seq. length
- 3x faster than standard attention implementation (seq. lengths 128 - 2K) and scales up to 64K
- Faster and more memory efficient than any existing method up to sequence length of 512
- Sequence Length > 1k: approximate attention methods (e.g., Linformer) start to become faster
- Still faster: block-sparse Flash Attention

Flash Attention: Benchmarks

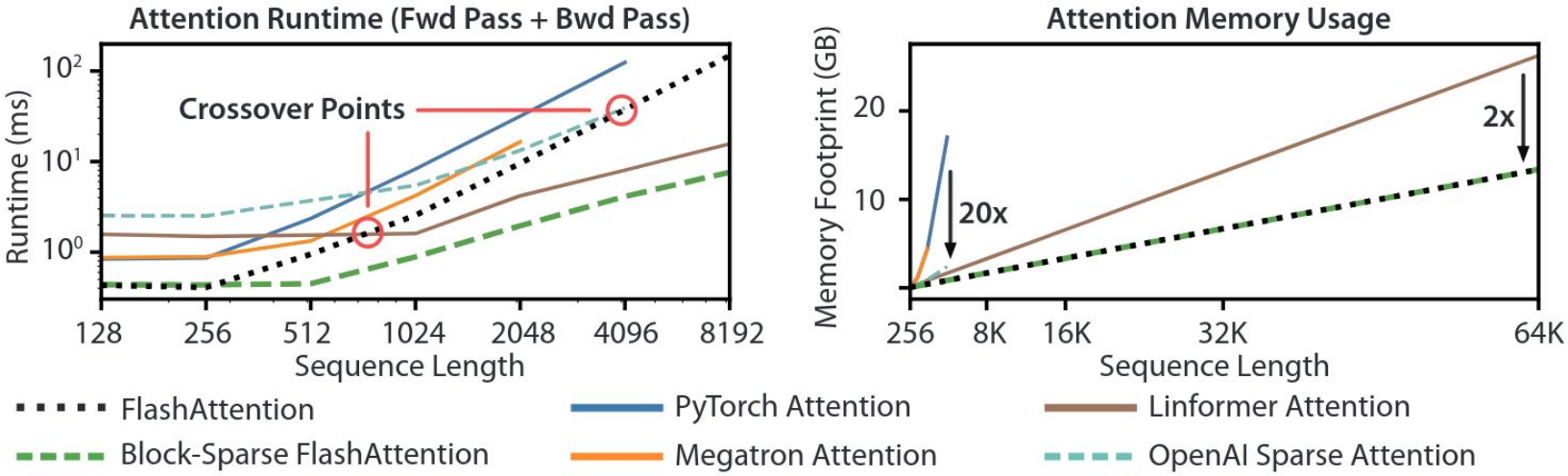


Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

Flash Attention: Limitations

- Required to **write new CUDA kernel** for each new attention implementation
  - Significant engineering effort
- Implementations eventually not transferable across GPU architectures
- Optimal within constants for computing attention on a **single GPU**