Setup:

All of my code for the project is inside of the folder /eecs120/src. Not inside the project120 folder that is a failed implementation.

Here is how I run my code:

```
[tzeeshan@login-i15:~] $srun -p gpu -A eecs120_class_gpu --gres=gpu:V100:1 --time=200:00 --mem=50G --pty /bin/bash -i
srun: job 36538389 queued and waiting for resources
srun: job 36538389 has been allocated resources
[tzeeshan@hpc3-gpu-16-05:~] $module load python/3.10.2
[tzeeshan@hpc3-gpu-16-05:~] $module load cuda/12.2.0
[tzeeshan@hpc3-gpu-16-05:~] $module load gcc/11.2.0
[tzeeshan@hpc3-gpu-16-05:~] $cd eecs120
[tzeeshan@hpc3-gpu-16-05:~/eecs120] $source project120_env/bin/activate
(project120_env) [tzeeshan@hpc3-gpu-16-05:~/eecs120] $cd src
(project120_env) [tzeeshan@hpc3-gpu-16-05:~/eecs120/src] $python run_tests.py
```

Created a file called run_tests.py inside my src to run the following tests for all implementations:

Formatted as: (seq_len, embed_dim)

```
test_cases = [
    (1024, 128),
    (1024, 256),
    (1024, 512),
    (1024, 1024),
    (2048, 256),
    (2048, 512),
    (2048, 1024),
    (2048, 2048),
    (4096, 512),
    (4096, 1024),
    (4096, 2048),
    (4096, 4096)
]
```

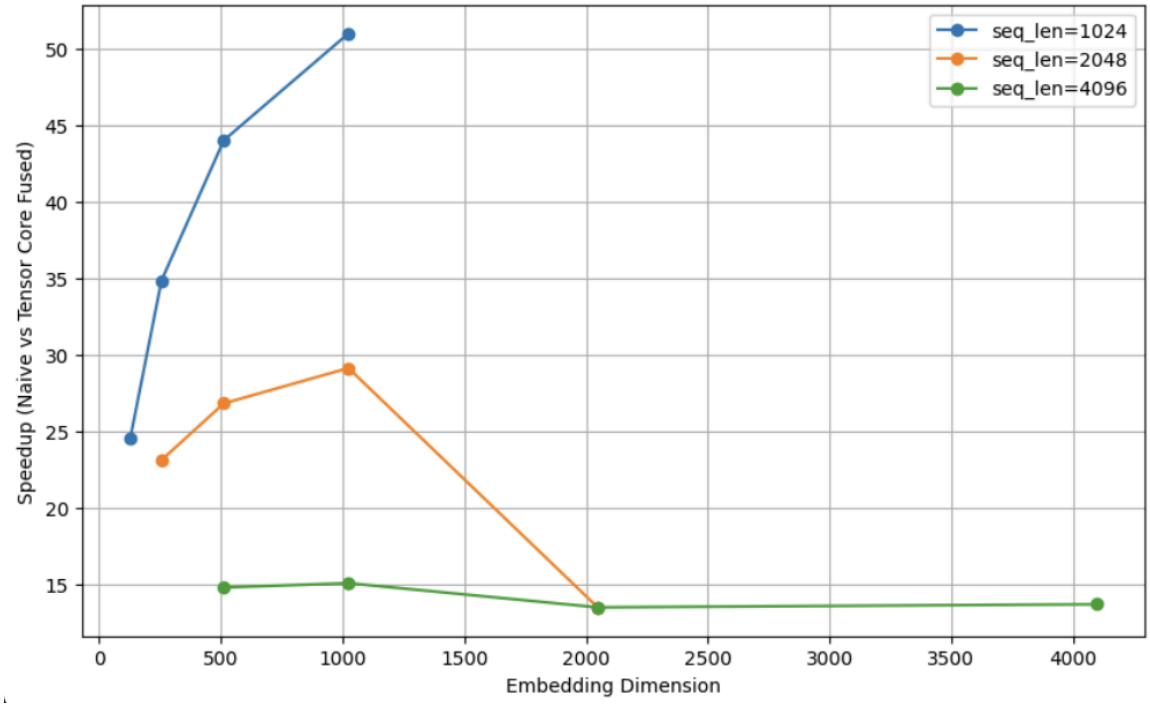
Can also use test_driver.py to run specific dimensions and lengths:

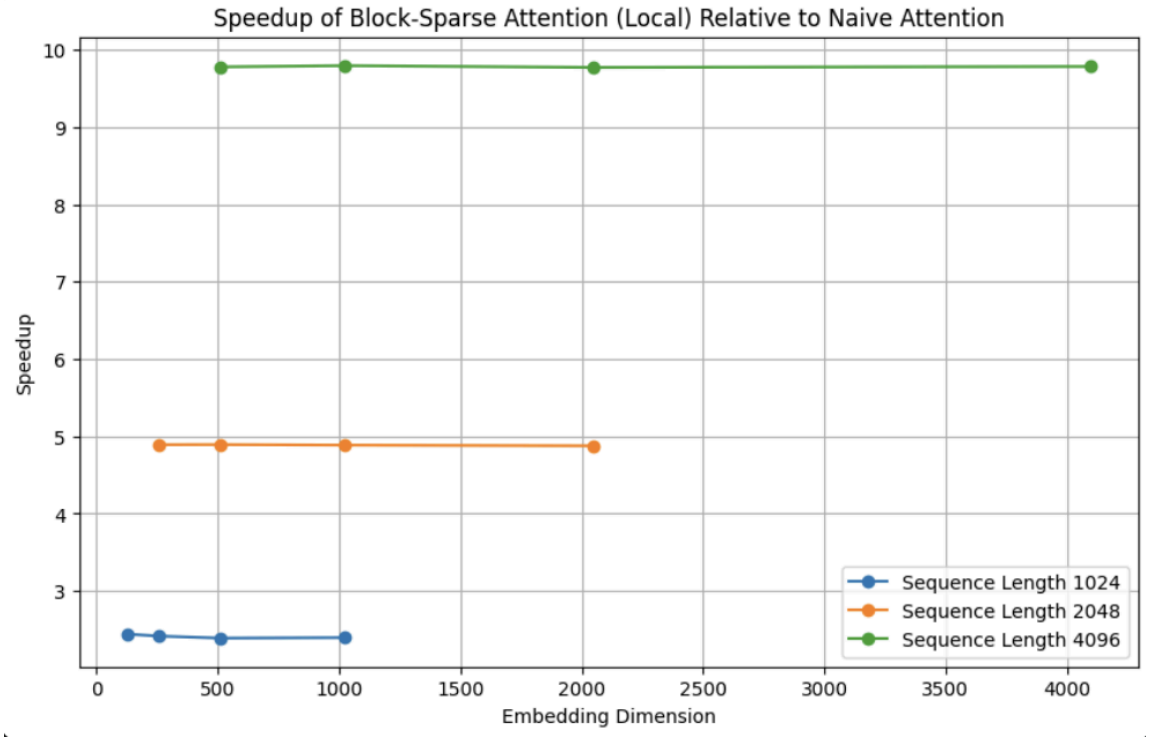
python test_driver.py --embed_dim 128 --seq_len 1024

```
(project120_env) [tzeeshan@hpc3-gpu-16-05:~/eecs120/src] $python run_tests.py
Running test for seq_len=1024, embed_dim=128
Transpose kernel is working correctly

Running with embed_dim=128, seq_len=1024
Vanilla attention time: 0.17950719594955444 ms

===== Testing Naive attention =====
Naive attention time: 56.43729858398437 ms
Relative error: 9.633479294279823e-07
Naive attention implementation is correct!

===== Testing Fused attention =====
Fused attention time: 120.88616943359375 ms
Relative error: 9.731172667670762e-07
Fused attention implementation is correct!

===== Testing Tensor Core fused attention =====
Tensor Core fused attention time: 2.298524856567383 ms
Relative error: 9.633479294279823e-07
Tensor Core fused attention implementation is correct!

===== Testing Block-sparse attention (causal) =====
Block-sparse attention (causal) time: 110.5143798828125 ms
Relative error: 1.5452700853347778
Block-sparse attention (causal) is incorrect
Relative error 1.5452700853347778 is more than 1e-4

===== Testing Block-sparse attention (local) =====
Block-sparse attention (local) time: 23.114239501953126 ms
Relative error: 2.151451587677002
Block-sparse attention (local) is incorrect
Relative error 2.151451587677002 is more than 1e-4

===== Performance Comparison =====
Naive attention: 56.44 ms
Fused attention: 120.89 ms (Speedup vs naive: 0.47x)
TC Fused attention: 2.30 ms (Speedup vs naive: 24.55x, vs fused: 52.59x)
Block-sparse attention (causal): 110.51 ms (Speedup vs naive: 0.51x)
Block-sparse attention (local): 23.11 ms (Speedup vs naive: 2.44x)
Running test for seq_len=1024, embed_dim=256
```

Screenshot of my code running

Speedup graphs: speedup = naive_attention_time / implementation_time

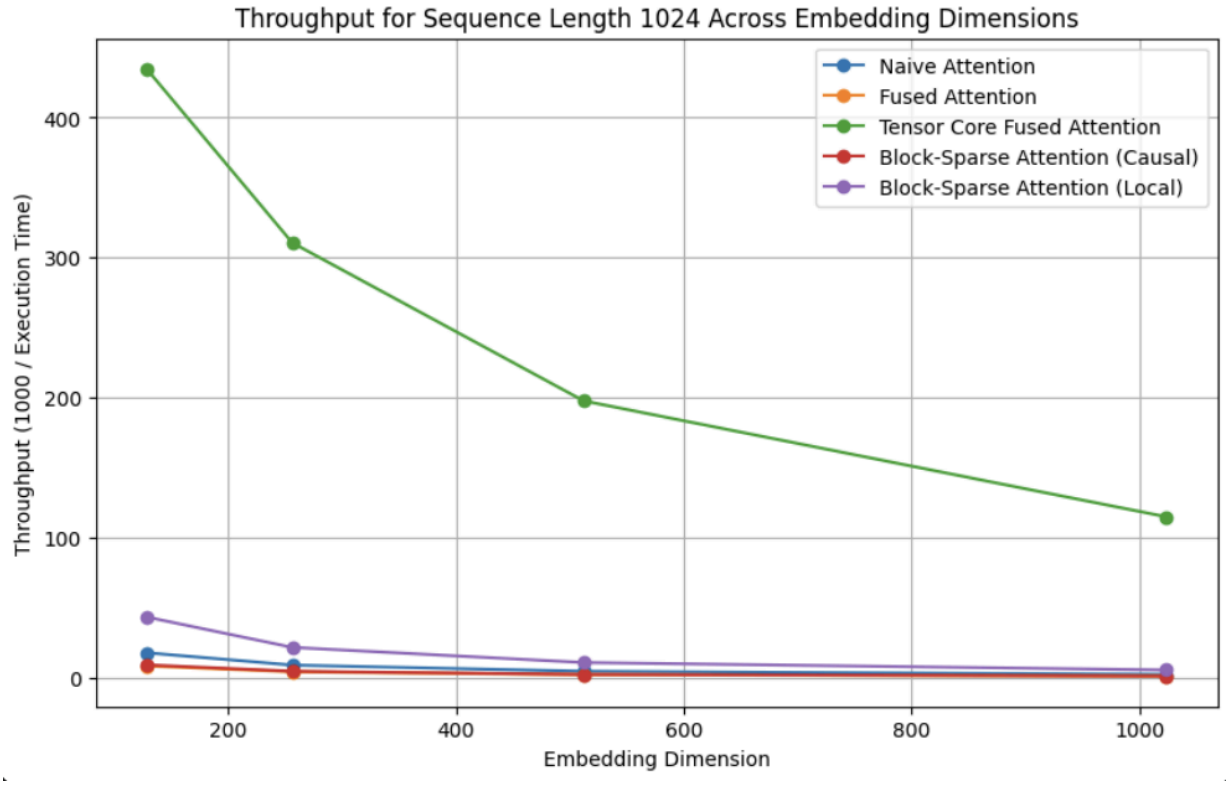


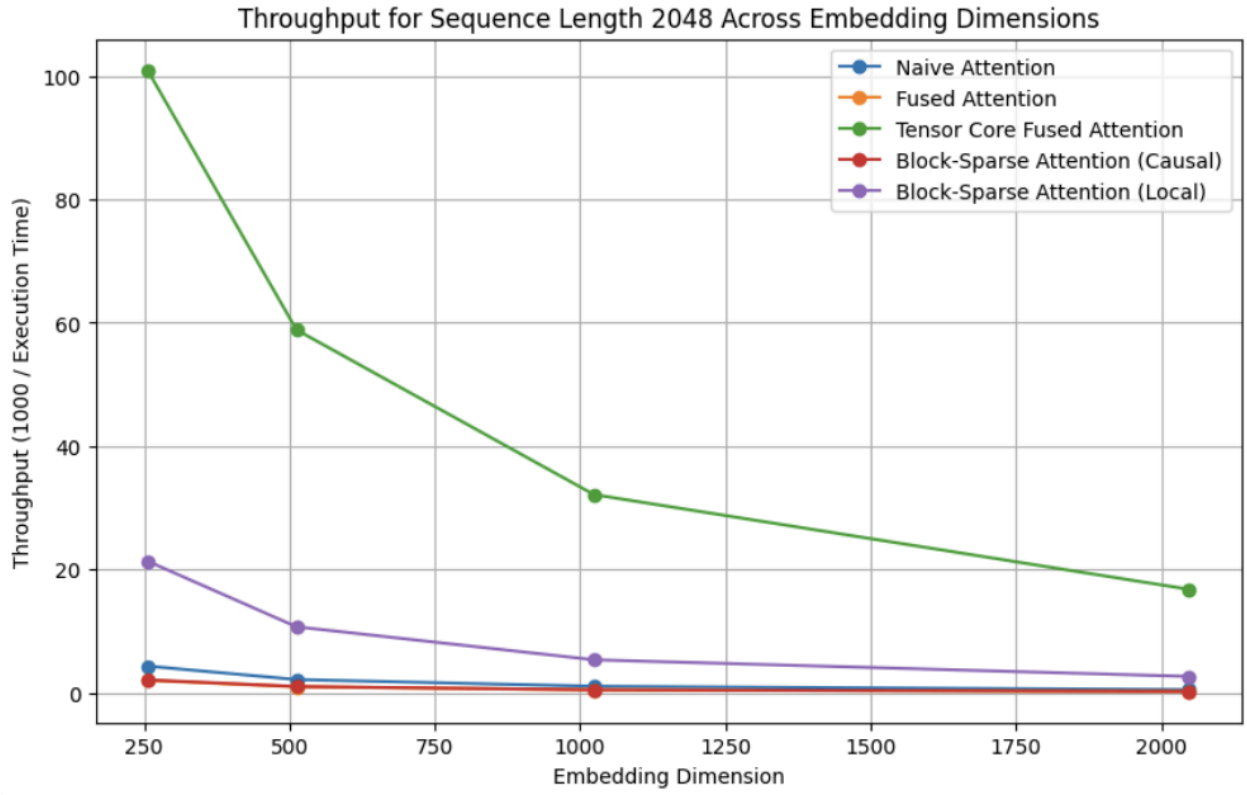


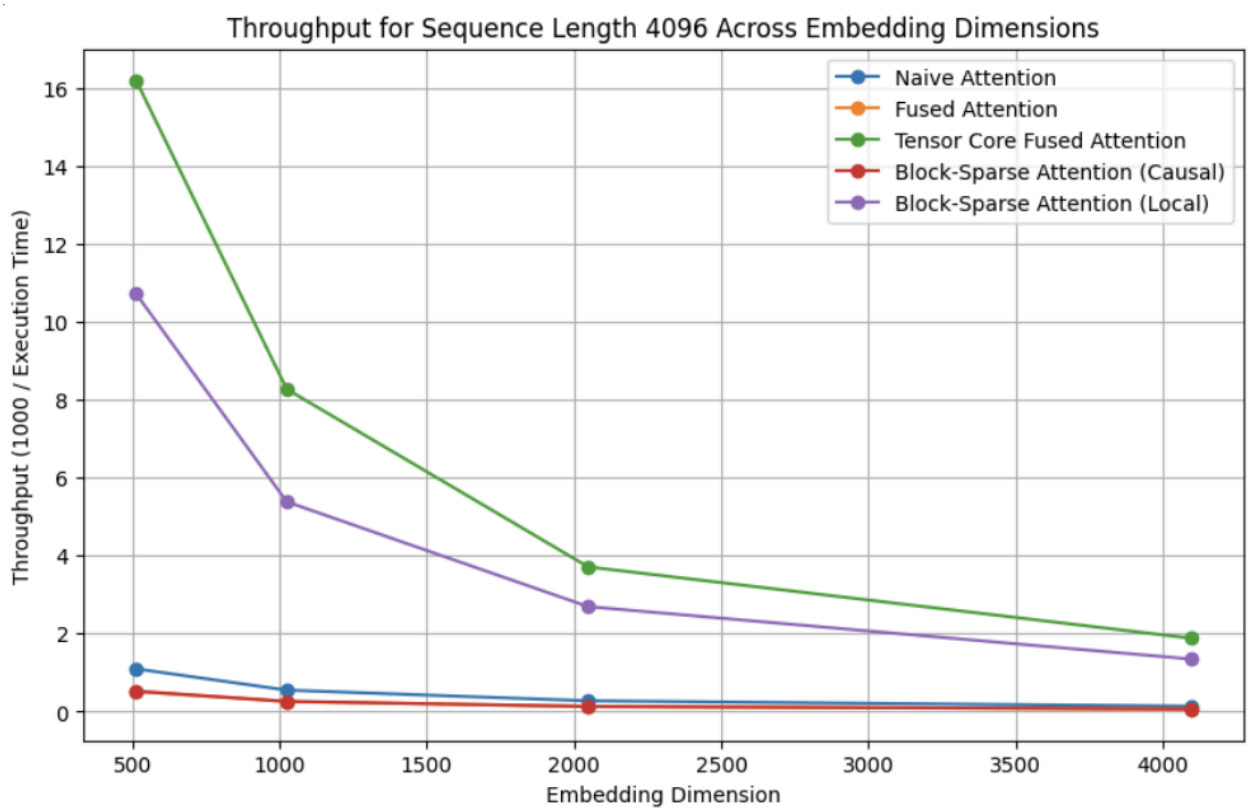
Analysis:

The 2 implementations that had a speedup in time from naive attention were tensor core fused and Block-sparse local. Tensor core saw an increase in speedup from longer dimensions for 1024,2048 seq len but it goes down after dimension exceeds 1048. Block sparse saw the same speedup for each seq len no matter the dimension. The speedup was by far best for 4096 len.

Throughput graphs: throughput = 1000 / execution_time_ms





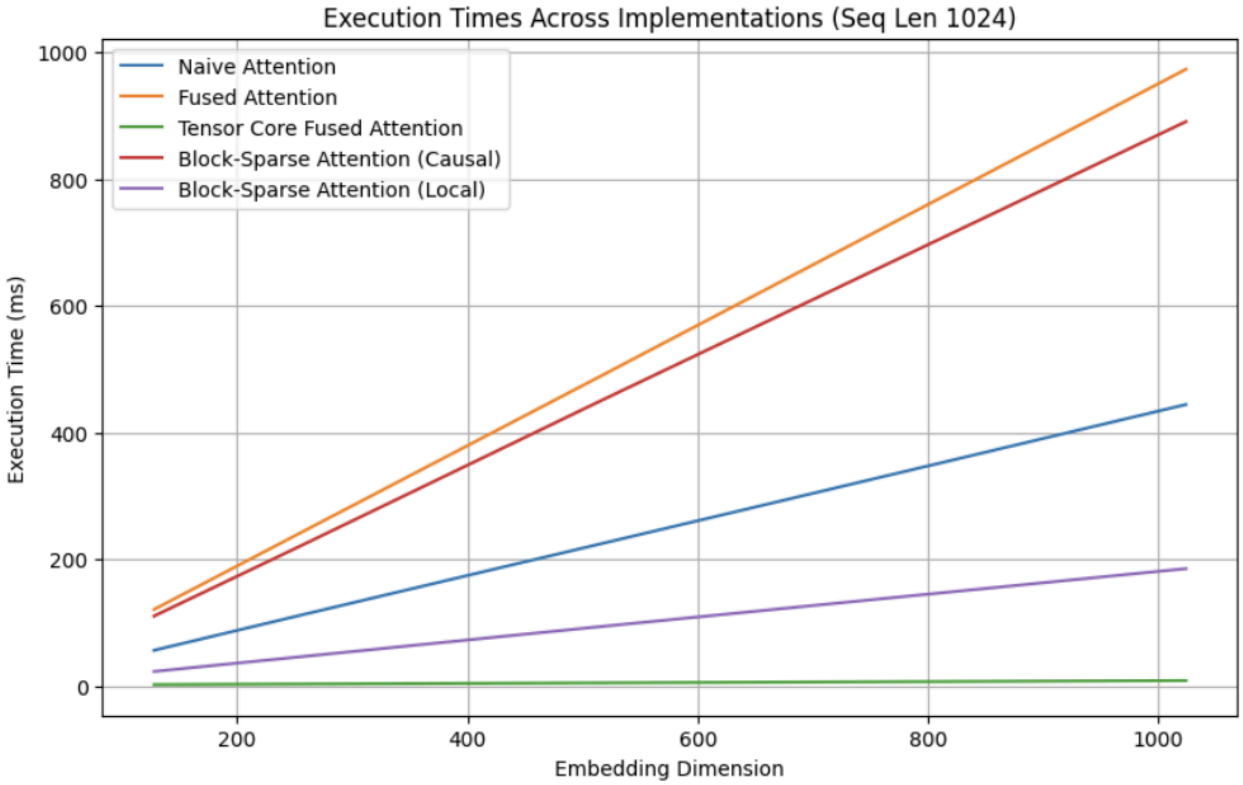


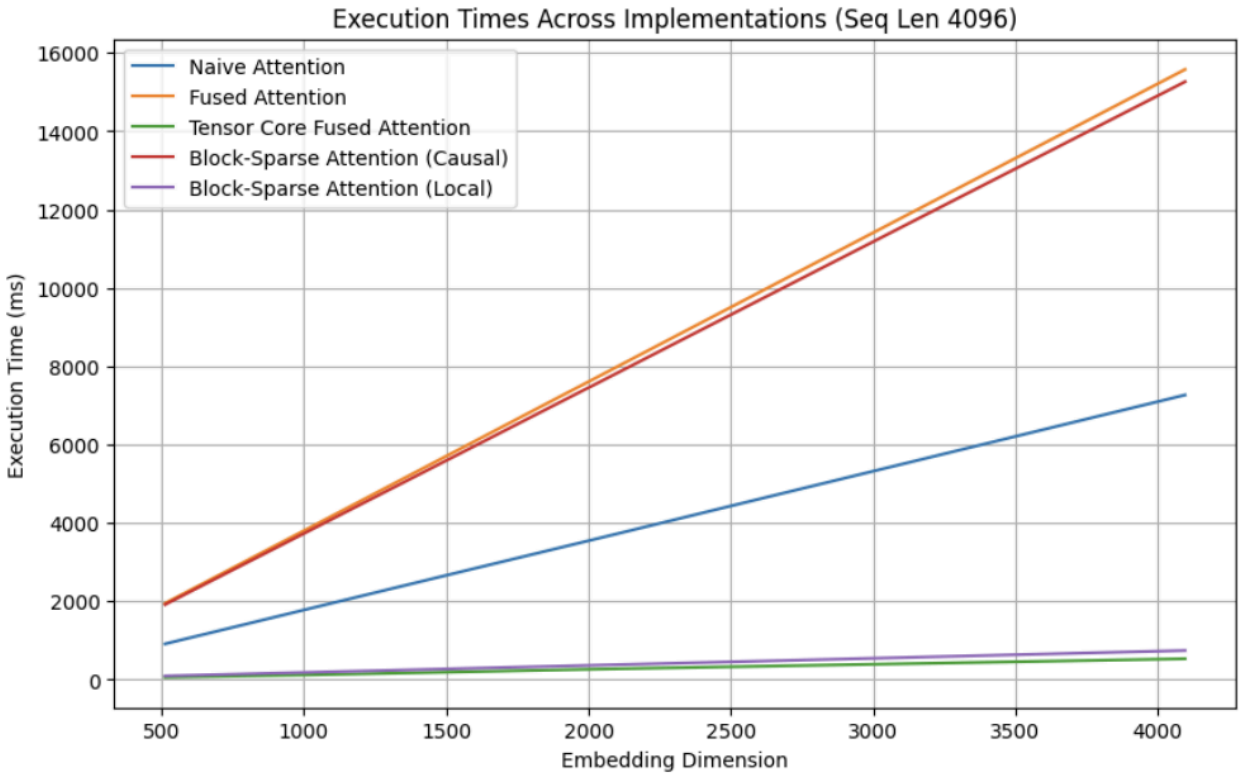
Analysis:
Throughput decreases as embedding dimension increases across all implementations. The throughput for tensor core is by far the fastest for 1024 and 2048 seq len. For 4096 len it is still the fastest but block-spare local is now much closer to it. Block sparse local could likely over take it for longer sequences.

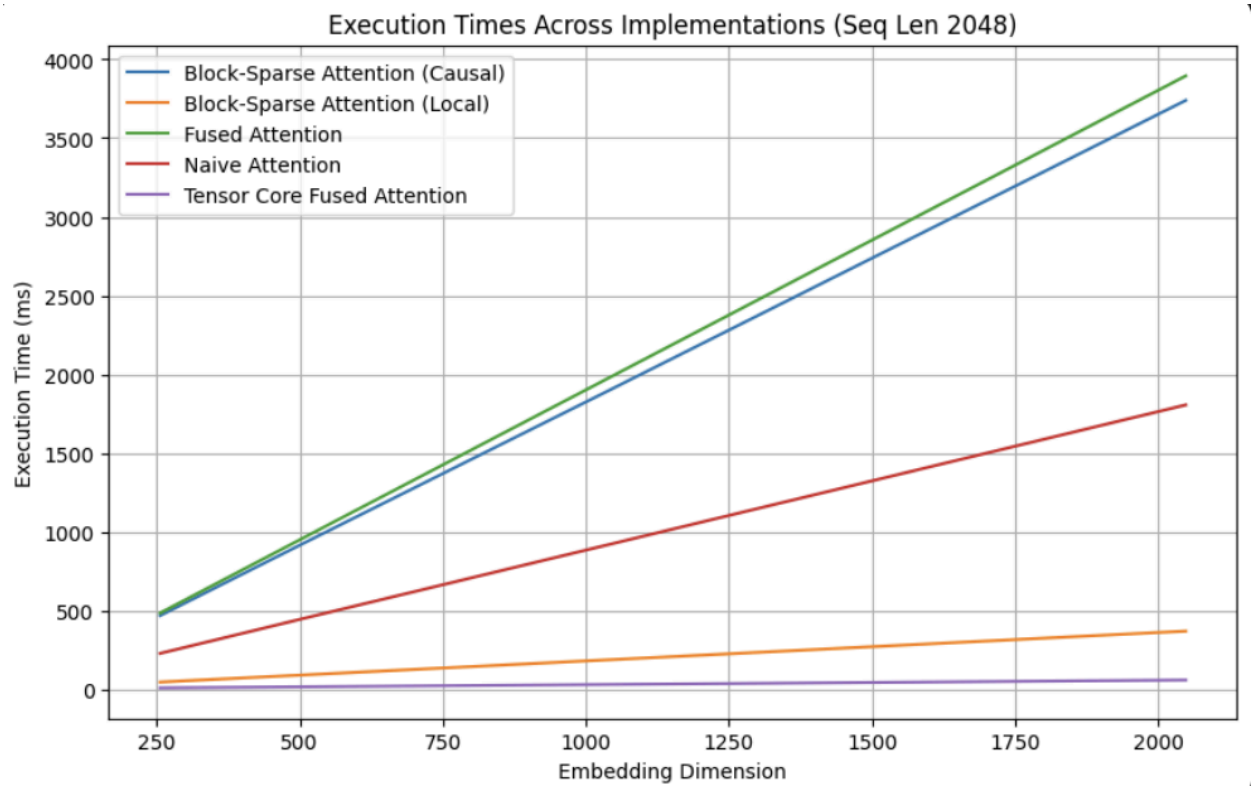
Memory Bandwidth Utilization analysis:
Tensor core fused achieves the best performance due to its efficient use of shared memory and reduced global memory traffic. Block-sparse local also benefits from lower memory bandwidth demands by skipping masked blocks. However, the dense implementations like naive and fused are more memory-bound, since they require access to large attention matrices in global memory. The high execution times for these methods suggest bad memory bandwidth utilization compared to the sparse and tensor core.

Kernel Occupancy analysis:
Higher occupancy can improve performance by hiding memory latency, tensor core fused achieves excellent performance even with moderate occupancy due to its reliance on instruction-level parallelism and tensor core efficiency. Sparse attention methods can get higher occupancy by reducing register pressure, but their performance is constrained by lower arithmetic ability. Dense methods like naive attention suffer from bottlenecks in register usage and global memory access patterns, limiting their occupancy and throughput.

Latency graphs:







Analysis:
Tensor core fused was the fastest implementation. Block-sparse local wasn't too far behind it but saw more increase as the embedding dimension increased. However Block-sparse local saw little increase in time from the larger seq lens. With every larger seq len it got a bit closer to tensor corse fused. If the seq len kept increasing it is likely it would end up being faster at some point. Naive attention is right in the middle for its speed, then fused and Block-sparse causal are both quite slow. I tried to implement masking and shared memory for the fused kernel however I got repeated memory errors and gave up trying to get it to work.

Sparsity-performance tradeoffs:
All of the first 3 implementations, naive, fused, and tensor get very low relative errors all falling far below 1e − 3. On the other hand the sparse implementations all have errors over one. The casual implementation which is slower has errors ranging from 1 to 2. The faster local implementation has even larger errors ranging from 2 to 5. Although the local implementation has a pretty good amount of speedup especially for longer sequence lengths, its relative error value makes it much less valuable than it could be. Using the tensor core fused would likely be much more optimal in terms of accuracy and speed.

All data put in tables:

Implementation	Sequence Length (seq_len)	Embedding Dimension (embed_dim)	Execution Time (ms)	Relative Error
Naive Attention	1024	128	56.44	9.63E-07
Naive Attention	1024	256	112.20	7.72E-07
Naive Attention	1024	512	222.82	1.37E-06
Naive Attention	1024	1024	444.26	1.37E-06
Naive Attention	2048	256	228.81	1.1E-06
Naive Attention	2048	512	455.47	1.54E-06
Naive Attention	2048	1024	905.73	1.59E-06
Naive Attention	2048	2048	1806.39	2E-06
Naive Attention	4096	512	912.46	1.89E-06
Naive Attention	4096	1024	1820.77	1.58E-06
Naive Attention	4096	2048	3630.53	2.72E-06
Naive Attention	4096	4096	7266.81	2.7E-06

Implementation	Sequence Length (seq_len)	Embedding Dimension (embed_dim)	Execution Time (ms)	Relative Error
Fused Attention	1024	128	120.89	9.73E-07
Fused Attention	1024	256	242.67	7.69E-07
Fused Attention	1024	512	485.47	1.39E-06
Fused Attention	1024	1024	973.28	1.39E-06
Fused Attention	2048	256	485.31	1.13E-06
Fused Attention	2048	512	973.10	1.56E-06
Fused Attention	2048	1024	1947.50	1.63E-06
Fused Attention	2048	2048	3895.65	2.01E-06
Fused Attention	4096	512	1948.77	1.91E-06
Fused Attention	4096	1024	3894.31	1.61E-06
Fused Attention	4096	2048	7790.67	2.72E-06
Fused Attention	4096	4096	15579.84	2.7E-06

Implementation	Sequence Length (seq_len)	Embedding Dimension (embed_dim)	Execution Time (ms)	Relative Error
Tensor Core Fused Attention	1024	128	2.30	9.63E-07
Tensor Core Fused Attention	1024	256	3.22	7.72E-07
Tensor Core Fused Attention	1024	512	5.06	1.37E-06
Tensor Core Fused Attention	1024	1024	8.71	1.37E-06
Tensor Core Fused Attention	2048	256	9.91	1.1E-06
Tensor Core Fused Attention	2048	512	16.99	1.54E-06
Tensor Core Fused Attention	2048	1024	31.10	1.59E-06
Tensor Core Fused Attention	2048	2048	59.50	2E-06
Tensor Core Fused Attention	4096	512	61.78	1.89E-06
Tensor Core Fused Attention	4096	1024	120.95	1.58E-06
Tensor Core Fused Attention	4096	2048	269.52	2.72E-06
Tensor Core Fused Attention	4096	4096	531.71	2.7E-06

Implementation	Sequence Length (seq_len)	Embedding Dimension (embed_dim)	Execution Time (ms)	Relative Error
Block-Sparse Attention (Causal)	1024	128	110.51	1.55
Block-Sparse Attention (Causal)	1024	256	221.90	1.47
Block-Sparse Attention (Causal)	1024	512	446.83	1.51
Block-Sparse Attention (Causal)	1024	1024	890.62	1.52
Block-Sparse Attention (Causal)	2048	256	468.81	1.75
Block-Sparse Attention (Causal)	2048	512	937.78	1.67
Block-Sparse Attention (Causal)	2048	1024	1868.83	1.69
Block-Sparse Attention (Causal)	2048	2048	3739.90	1.70
Block-Sparse Attention (Causal)	4096	512	1916.61	1.87
Block-Sparse Attention (Causal)	4096	1024	3820.43	1.93
Block-Sparse Attention (Causal)	4096	2048	7633.66	1.89
Block-Sparse Attention (Causal)	4096	4096	1526.21	1.88

Implementation	Sequence Length (seq_len)	Embedding Dimension (embed_dim)	Execution Time (ms)	Relative Error
Block-Sparse Attention (Local)	1024	128	23.11	2.15
Block-Sparse Attention (Local)	1024	256	46.42	2.09
Block-Sparse Attention (Local)	1024	512	93.20	2.12
Block-Sparse Attention (Local)	1024	1024	185.33	2.13
Block-Sparse Attention (Local)	2048	256	46.77	3.17
Block-Sparse Attention (Local)	2048	512	93.08	3.07
Block-Sparse Attention (Local)	2048	1024	185.35	3.10
Block-Sparse Attention (Local)	2048	2048	370.32	3.10
Block-Sparse Attention (Local)	4096	512	93.32	4.42
Block-Sparse Attention (Local)	4096	1024	185.90	4.50
Block-Sparse Attention (Local)	4096	2048	371.57	4.47
Block-Sparse Attention (Local)	4096	4096	742.81	4.46