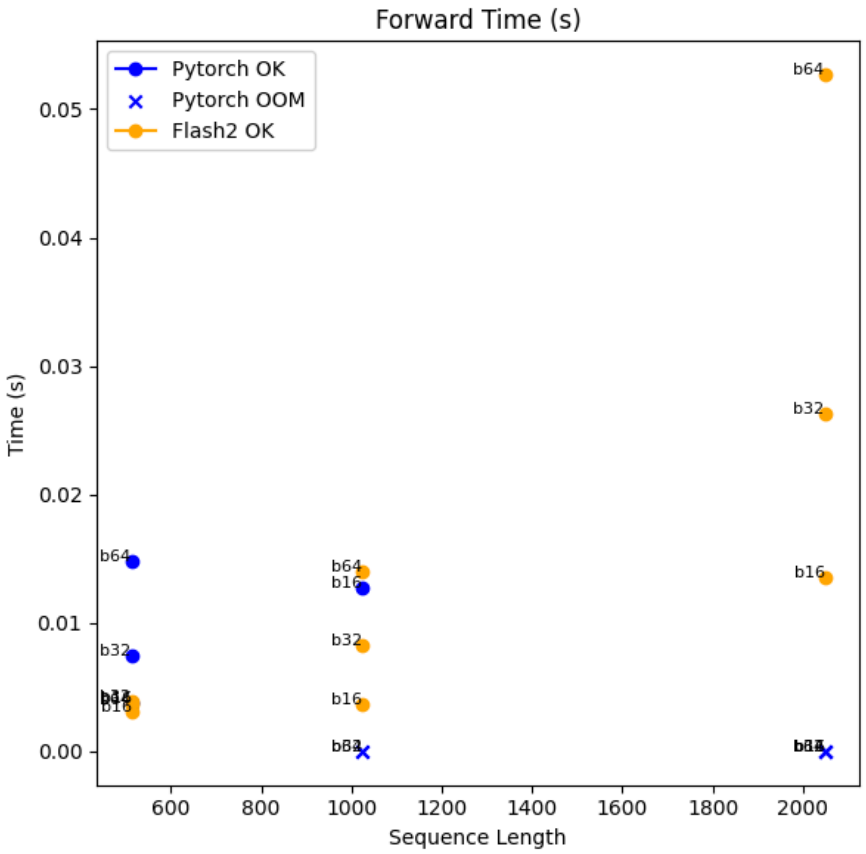
PP-Lab5 Report
Name: 鄔培勇
Student ID: 113062422

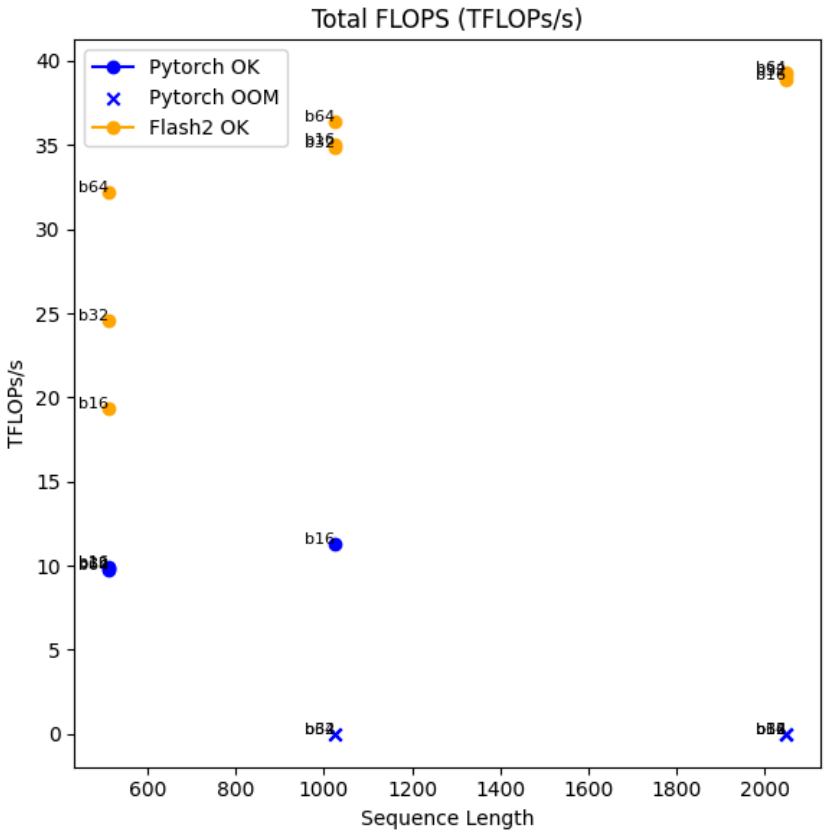
Original Pytorch implementation vs FlashAttention v2 implementation:

In this experiment, I compared the performance of the original PyTorch implementation of attention with the FlashAttention v2 implementation. The goal was to evaluate these implementations in terms of Forward Time, Total FLOPs, and Peak Memory Usage. I experimented with a range of batch sizes (16, 32, 64) and sequence lengths (512, 1024, 2048).

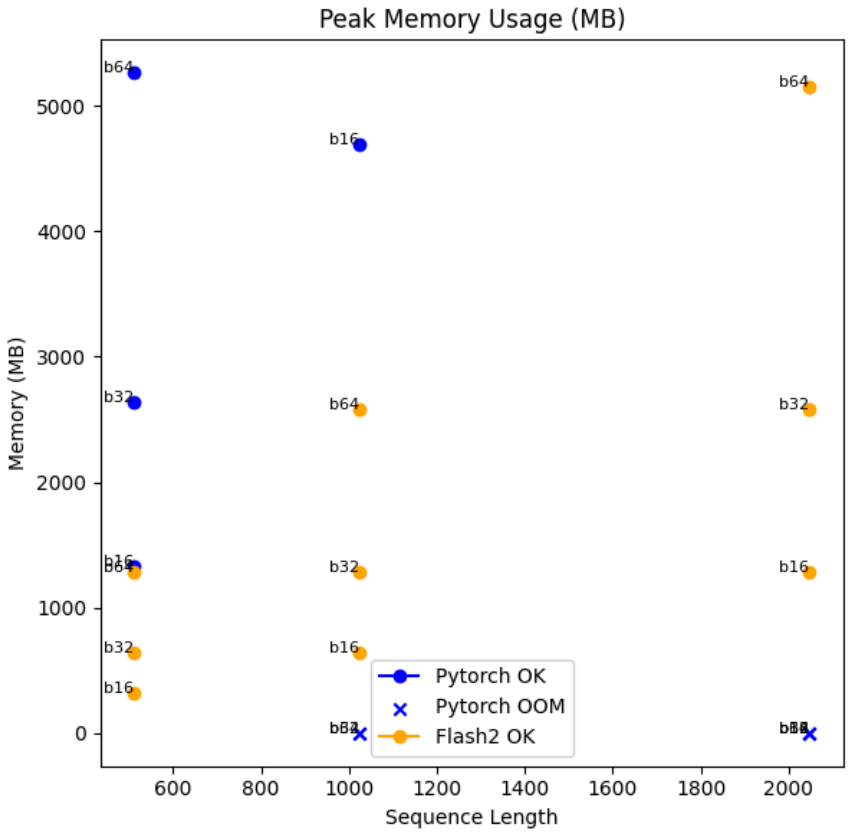


1. FlashAttention v2 Performance:
    - FlashAttention v2 consistently achieves lower forward times compared to PyTorch across all tested configurations.
    - It scales well even at higher sequence lengths and batch sizes, demonstrating robustness and efficiency in handling larger workloads.
2. PyTorch Implementation:
    - While PyTorch successfully handles smaller configurations (e.g., b16 and sequence length ≤ 1024), it struggles with larger setups. For example:
        - At b64 and sequence lengths ≥ 1024, PyTorch encounters OOM errors.

- Even when it completes the forward pass (e.g., b16 and b32), its forward time is higher than that of FlashAttention v2.



1. FlashAttention v2 Implementation:
    - FlashAttention v2 achieves consistently higher TFLOPs/s across all batch sizes (16, 32, 64) and sequence lengths (512, 1024, 2048).
    - It scales effectively even at higher sequence lengths and batch sizes, demonstrating robust performance without memory overhead issues.
    - At b64 and a sequence length of 2048, FlashAttention v2 achieves peak TFLOPs/s (~40), maintaining stability and efficiency.
2. PyTorch Implementation:
    - Even for configurations where it completes the forward pass (e.g., b16, b32), its TFLOPs/s is noticeably lower compared to FlashAttention v2.



1. FlashAttention v2 Implementation:
    - FlashAttention v2 consistently uses less memory compared to PyTorch across all configurations, demonstrating its superior memory management.
2. PyTorch Implementation:
    - PyTorch shows significantly higher memory usage, especially for larger configurations.
    - Even for smaller configurations, such as b16 or b32 at sequence lengths around 1024, PyTorch's peak memory usage is considerably higher than FlashAttention v2.
    - PyTorch's inability to handle larger workloads highlights its limitations in memory optimization.

Conclusion

The comparison between the original PyTorch implementation and FlashAttention v2 highlights significant differences in performance, scalability, and resource efficiency. FlashAttention v2 consistently outperforms PyTorch across all tested configurations in terms of forward time, total FLOPs, and peak memory usage. Its ability to scale efficiently to higher batch sizes and sequence lengths without encountering Out-Of-Memory (OOM) errors underscores its robustness and optimized memory management.

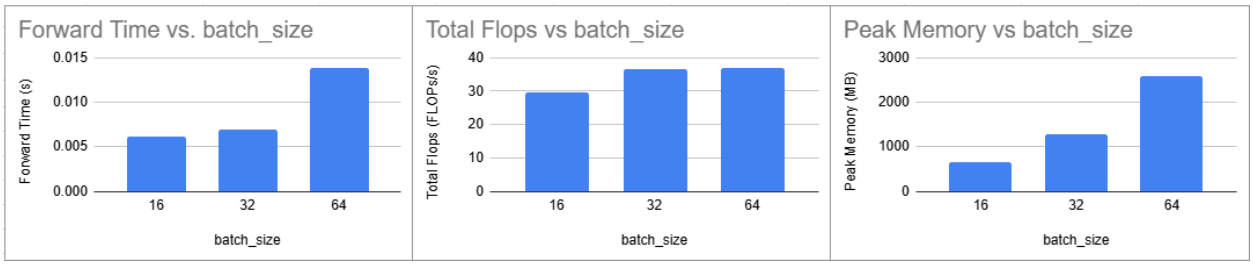
PyTorch, while capable of handling smaller configurations effectively, struggles with larger workloads, encountering OOM errors and exhibiting higher memory usage. Even in cases where it completes the forward pass, its performance in terms of forward time and TFLOPs/s is significantly lower than that of FlashAttention v2.

Varying Parameters Experiment:

In this experiment, I vary the values of some parameters on FlashAttention v2 implementation to gain deeper insights. Unless mentioned, the other parameters are using default values.

- Batch Size

batch_size	Forward Time (s)	Total Flops (FLOPs/s)	Peak Memory (MB)
16	0.006150229772	29.77291386	644.0004883
32	0.006949217493	36.62848594	1288.000488
64	0.01381614593	36.76811675	2576.000488



The batch size affects the model's forward time, FLOPs, and peak memory consumption. As the batch size increases:

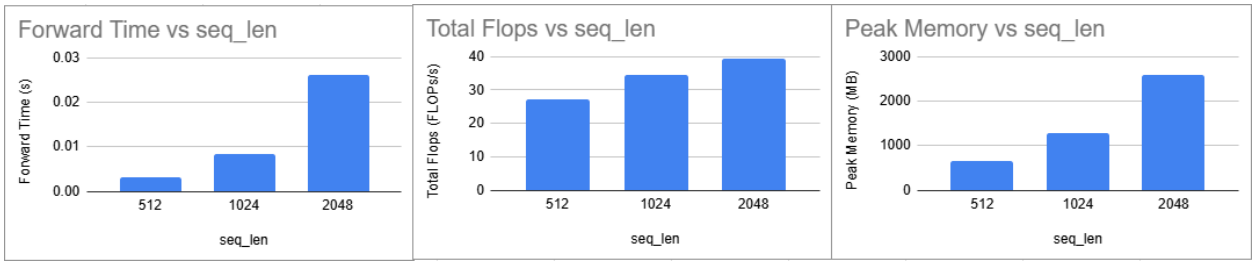
- Forward Time: Increases approximately linearly with batch size, as expected due to more data being processed.
- FLOPs: Shows a minor increase, stabilizing beyond a certain point.
- Peak Memory: Scales proportionally with batch size.

Increasing the batch size improves computational throughput but requires more memory and increases forward time.

- Sequence Length

seq_len	Forward Time (s)	Total Flops (FLOPs/s)	Peak Memory (MB)

512	0.003047129015	27.07925792	644.0004883
1024	0.008463573332	34.64288926	1288.000488
2048	0.02620695792	39.29274602	2576.000488



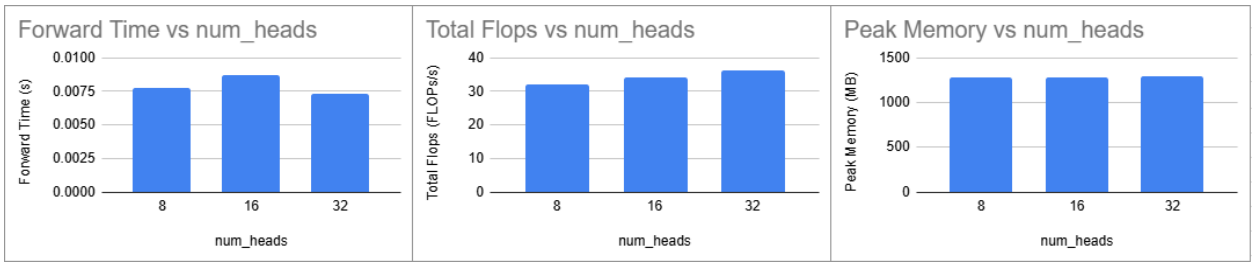
The sequence length has a significant impact on model performance and memory usage:

- Forward Time: Increases non-linearly with sequence length.
- FLOPs: Increases significantly, demonstrating higher computational complexity with longer sequences.
- Peak Memory: Scales linearly with sequence length.

Longer sequences require exponentially more forward time and computational resources, highlighting the need for optimization in handling long sequences.

- Number of Heads

num_heads	Forward Time (s)	Total Flops (FLOPs/s)	Peak Memory (MB)
8	0.007773728793	31.90310299	1282.000488
16	0.008723722522	34.30936054	1284.000488
32	0.007317186023	36.11834562	1288.000488



The number of attention heads directly influences computational efficiency and memory use:

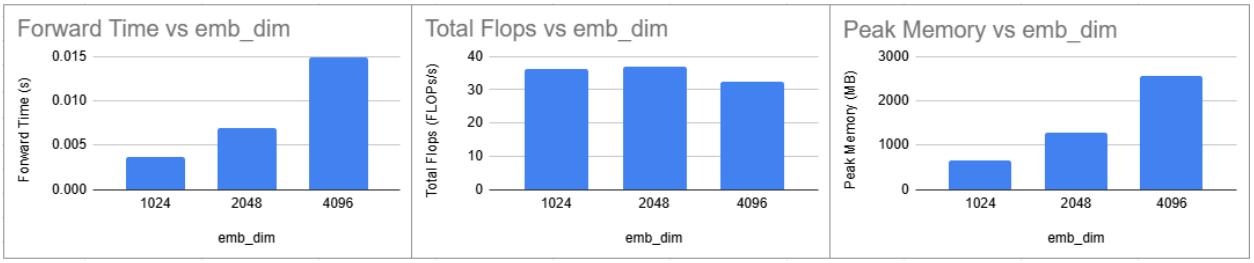
- Forward Time: Varies slightly, with 32 heads having the lowest time.

- FLOPs: Increases with more heads, peaking at 32 heads.
- Peak Memory: Remains stable around 1282–1288 MB across configurations.

Increasing the number of heads improves model efficiency up to a point, but memory usage remains stable, making it a less significant bottleneck.

- Embedding Dimensions (16 heads)

emb_dim	Forward Time (s)	Total Flops (FLOPs/s)	Peak Memory (MB)
1024	0.003675101822	36.09480533	644.0004883
2048	0.006868742655	36.77233137	1284.000488
4096	0.01491965304	32.34884396	2564.000488



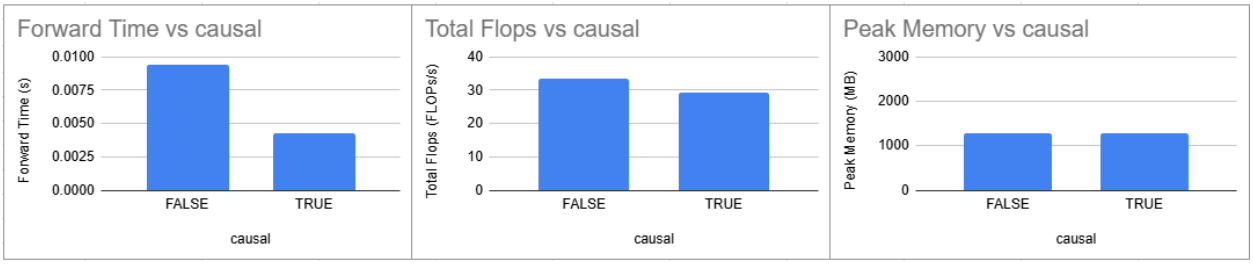
The embedding dimension affects both computational time and memory consumption:

- Forward Time: Increases significantly with embedding dimensions.
- FLOPs: Peaks at 2048 dimensions but decreases slightly at 4096 dimensions.
- Peak Memory: Scales proportionally with embedding dimension.

Higher embedding dimensions increase forward time and memory requirements, showing diminishing returns in FLOPs efficiency at very high dimensions.

- Causal Attention

causal	Forward Time (s)	Total Flops (FLOPs/s)	Peak Memory (MB)
FALSE	0.009378869335	33.51131742	1288.000488
TRUE	0.004313171282	29.28614372	1288.000488



Causal attention significantly affects forward time and computational efficiency:

- Forward Time: Causal attention is faster than the non-causal counterpart.
- FLOPs: Lower for causal attention.
- Peak Memory: Remains consistent at 1288 MB regardless of causal attention.

Causal attention is computationally more efficient and faster, making it a preferred option when applicable.

Conclusion

This experiment reveals that:

- Batch size and sequence length have the largest impact on peak memory, making them critical for memory-constrained environments.
- Increasing the number of heads and embedding dimensions boosts computational complexity, but their returns diminish beyond a certain point.
- Causal attention offers a significant advantage in computational efficiency and forward time without additional memory costs.

Overall, optimizing these parameters should consider the trade-offs between computational throughput, memory usage, and model accuracy.