Lab5: FlashAttention

112065528 洪巧雲

Attention

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$$

- QK: 相乘後會是一個分數，再把這個分數去做softmax(Normalization)。
- 將softmax後的結果拿去乘上V後，就會是Queing的結果。(每個row)

Multi-Head Attention

- 將D拆解成很多份，ex: emb_dim=4096，num_heads=32，head_size=128
  - N*128 -> 每個都丟進去做attention，總共做32次。
  - 做完後還需要把它concat起來，這樣就會是原本的size。

實驗目標

這次 lab5 的目地是要比較兩種 Attention 的實現方式 —— Flash2 與 PyTorch，在多種參數組合下的 performance，主要關注以下幾個指標：

- Forward Execution Time
- Backward Execution Time
- FLOPS
- Peak Memory Usage

實驗方法

1. **執行 lab5.py**： 使用 `lab5.py` 執行不同參數下的實驗，測量不同參數組合下的性能。
2. **parameter 設置**： 測試的 parameter 範圍包括：
   - Batch Size：16、32、64
   - Sequence Length：512、1024
   - Number of Attention Heads：16、32
   - Embedding Dimension：1024、2048
   - Implementation：Flash2、PyTorch
   - Causality：True or False
3. **自動化 Script**： 使用 `run_all_benchmarks.sh` 自動遍歷所有參數組合，並將結果保存為 JSON 檔案中。

```
# 定義要測試的參數
 batch_sizes=(16 32 64)
 seq_lens=(512 1024)
 num_heads_list=(16 32)
```

```
 emb_dims=(1024 2048)
 impls=("Pytorch" "Flash2")
 causal_flags=("--causal" "")
```

```
# 創建結果目錄
mkdir -p results

# 遍歷所有參數組合
for batch_size in "${batch_sizes[@]}"; do
  for seq_len in "${seq_lens[@]}"; do
    for num_heads in "${num_heads_list[@]}"; do
      for emb_dim in "${emb_dims[@]}"; do
        for impl in "${impls[@]}"; do
          for causal_flag in "${causal_flags[@]}"; do
            # 確保 emb_dim 能被 num_heads 整除
            if (( emb_dim % num_heads == 0 )); then
              # 設置輸出文件名

output_file="results/result_bs${batch_size}_sl${seq_len}_nh${num_heads
}_ed${emb_dim}_${impl}_causal${causal_flag:+_causal}.json"

              # 運行基準測試
              python lab5.py \
                --batch_size $batch_size \
                --seq_len $seq_len \
                --num_heads $num_heads \
                --emb_dim $emb_dim \
                --impl $impl \
                $causal_flag \
                --repeats 30 \
                --output $output_file
            else
              echo "跳過：emb_dim=$emb_dim 不能被 num_heads=$num_heads
整除。"
            fi
          done
        done
      done
    done
  done
done
```

- 執行這個 `run_all_benchmarks.sh` 前需要先在 Terminal 下: `chmod +x run_all_benchmarks.sh` 。
- 生成的 `.JSON` 檔案如下：

```
{
  "forward": {
```

```
    "time(s)": 0.0011418992032607397,
    "FLOPS(TFLOPs/s)": 30.089992417793418
  },
  "backward": {
    "time(s)": 0.003703039900089304,
    "FLOPS(TFLOPs/s)": 23.196980923140586
  },
  "forward_backward": {
    "time(s)": 0.004844939103350043,
    "FLOPS(TFLOPs/s)": 24.821588408582183
  },
  "peak_memory_usage(MB)": 324.00048828125
}
```


4. **數據處理與可視化**： 最後就是寫一個 python 去讀取 JSON 結果並進行數據可視化的處理，生成視覺化圖表以便分析。這部分我是用:

   ```
   import os
   import json
   import pandas as pd
   import matplotlib.pyplot as plt
   import seaborn as sns
   ```

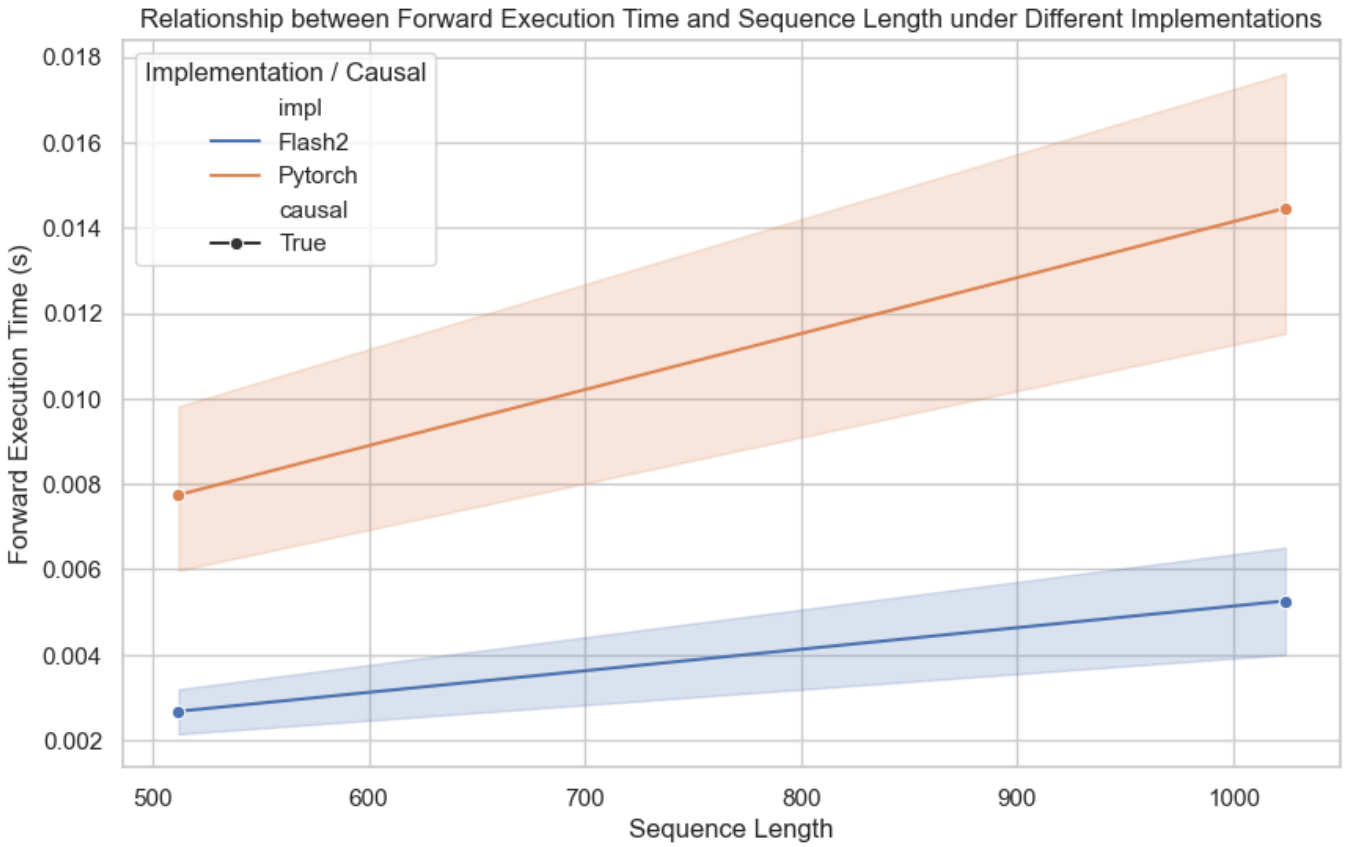
   這幾個 python 套件做的。

lab5.py

- `benchmark_attention` **function**：
  - 這個 function 主要是用來接收 test parameters 如 batch size、Sequence Length、Number of Attention Heads、Embedding Dimension 等等。
  - 並且使用 Flash2 或 PyTorch 的注意力機制實現進行前向與反向計算。
  - 測量以下指標：
    - 前向執行時間
    - 反向執行時間
    - 浮點運算效能（FLOPS）
    - 峰值記憶體使用量（Peak Memory Usage）
  - 最後將測試結果保存為 JSON 檔案。
- `flops` **function**：
  - 計算浮點運算次數（FLOPS），用於評估運算效率。
- `time_fwd_bwd` **function**：
  - 計算前向與反向傳遞的平均時間，返回時間數據供後續分析。

Experiment Result

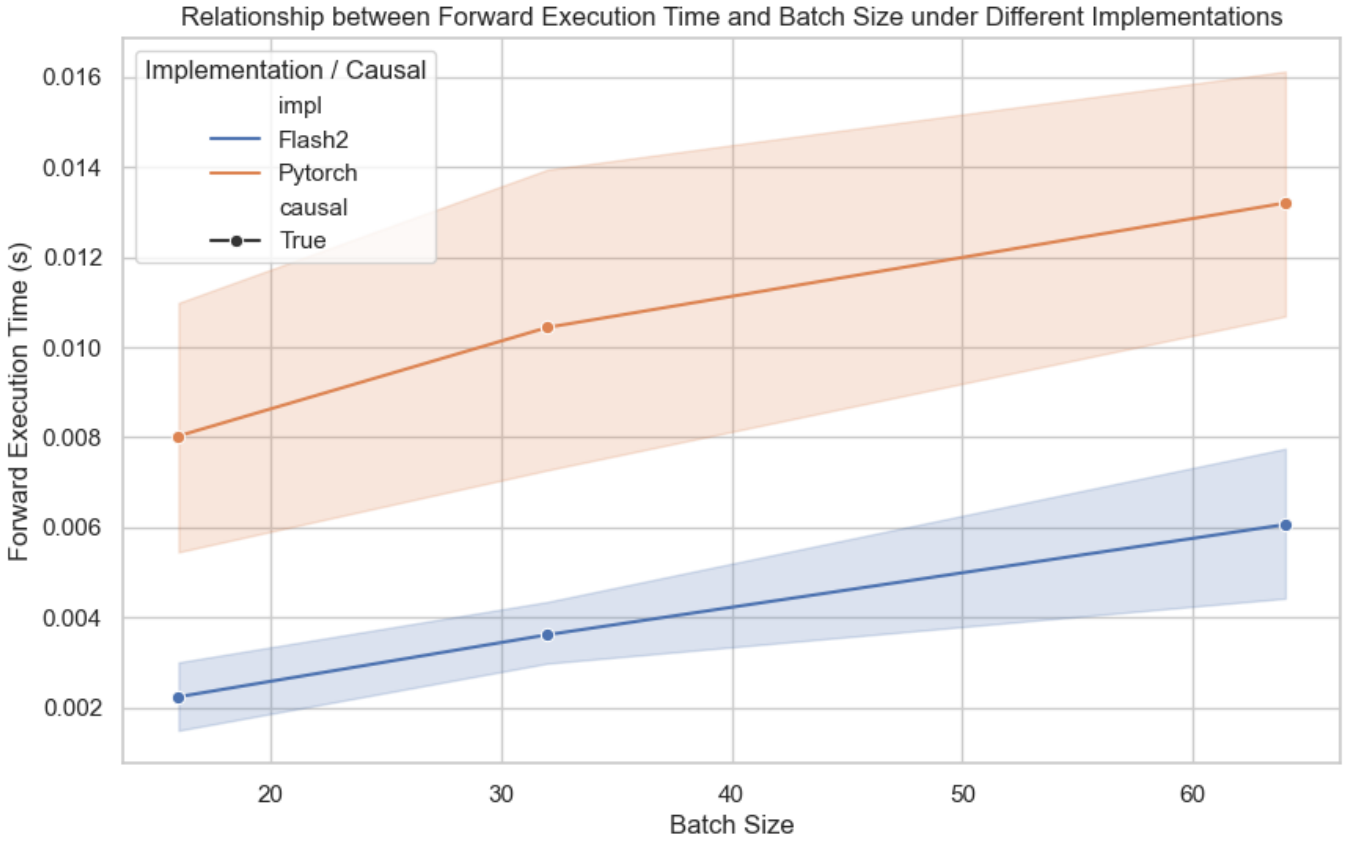
1. 前向執行時間與序列長度的關係



在運行效率方面，Flash2 的執行時間顯著低於 PyTorch。隨著序列長度的增加，Flash2 的執行時間增長速率較為平緩，這表明其在處理長序列時具有更優越的性能。相較之下，PyTorch 的執行時間隨著序列長度的增加呈現出更為明顯的線性增長趨勢，這代表在長序列處理上，Flash2 更加高效。

---

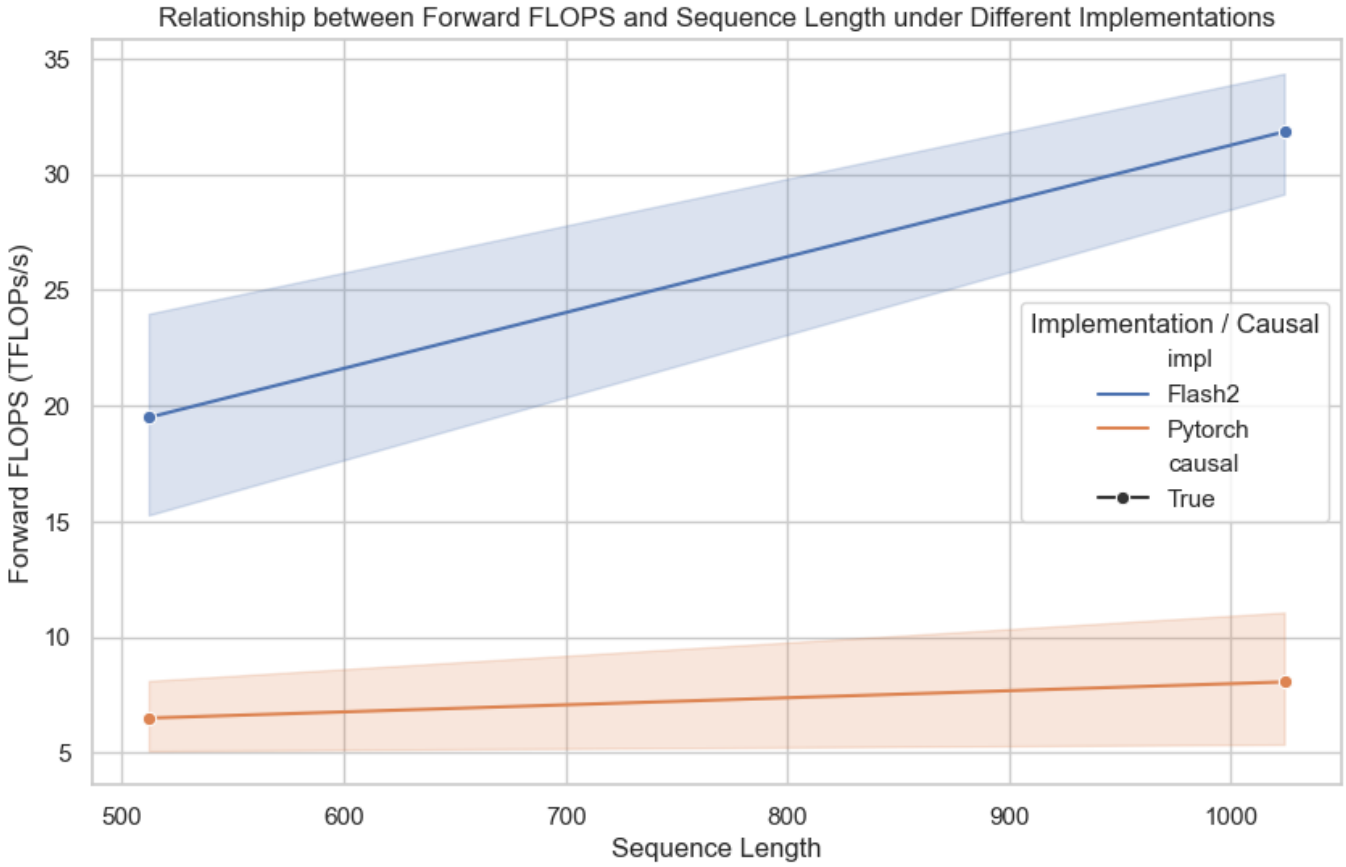
2. 前向執行時間與批量大小的關係



可以從圖表中看到，當批量大小增加時，Flash2 的執行時間增長幅度較小，顯示出其對大批量運算具有良好適應性。相對地，PyTorch 對批量大小的變化更為敏感，執行時間明顯增加。這代表 Flash2 的設計在處理大批量數據時更加高效，能夠有效縮短運算時間。

---

3. 前向 FLOPS 與序列長度的關係

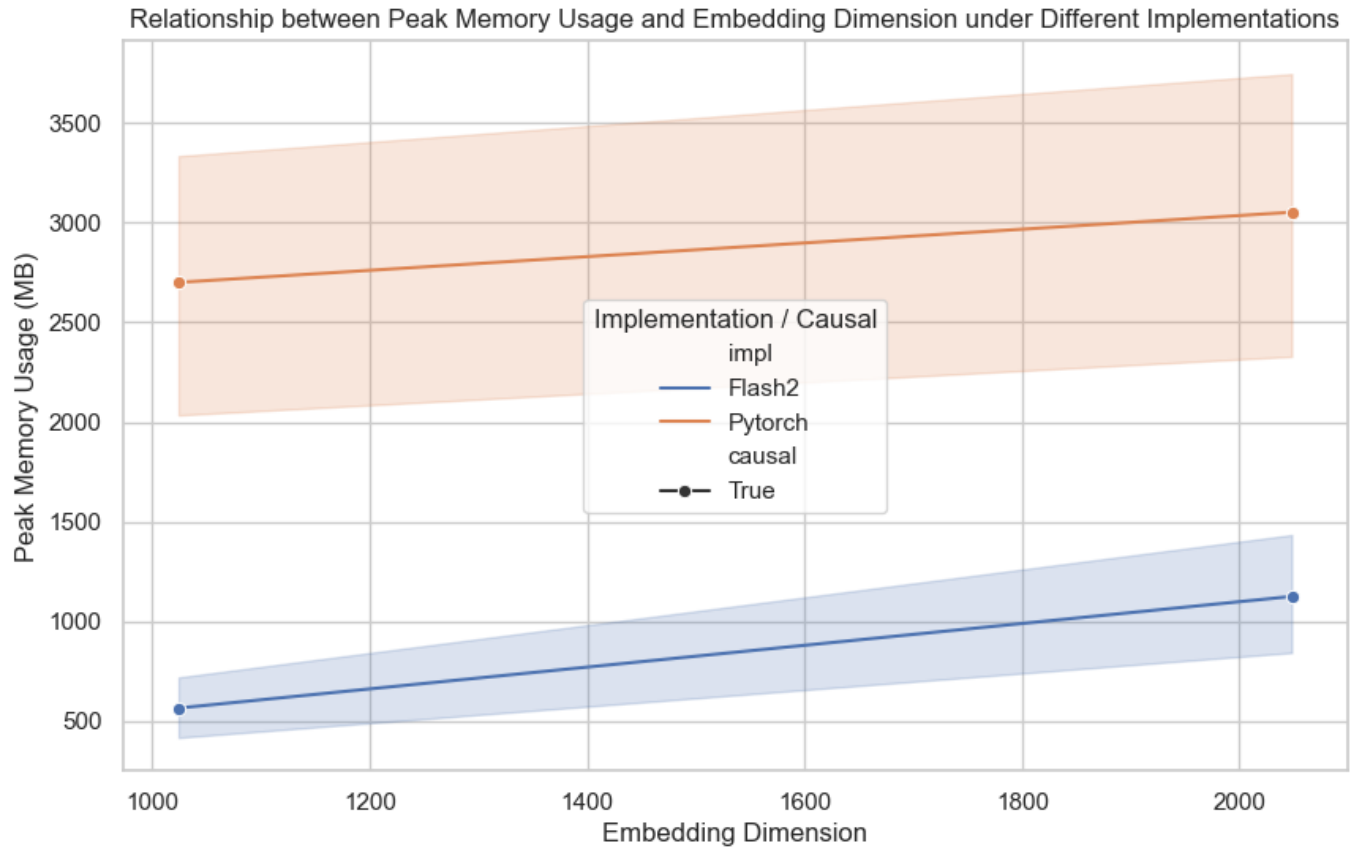


Flash2 通過先進的運算優化技術，使其在處理長序列時展現出更高的效率。這些優化措施包括更高效的算法設計和硬體資源的充分利用，使得 Flash2 能夠在面對長序列時保持低延遲和高吞吐量，顯著提升了整體運算效能。

在浮點運算次數（FLOPS）方面，Flash2 的 FLOPS 隨著序列長度的增加而快速提升，這表明其能夠更有效地利用計算資源。相比之下，PyTorch 的 FLOPS 整體較低，且增長幅度有限，顯示出在高運算需求下，Flash2 能夠實現更高的運算效能，進一步證明其在大規模運算上的優勢。

---

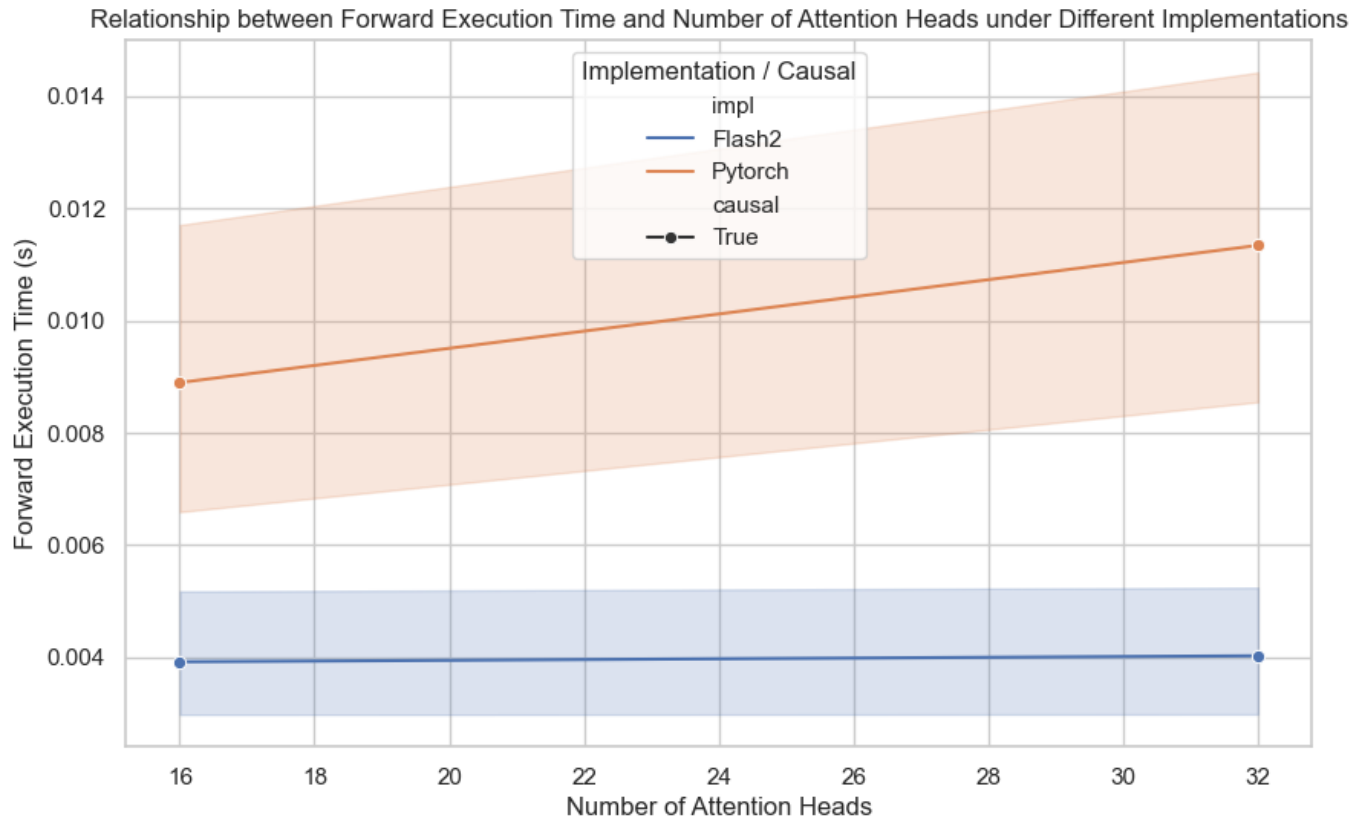
4. 峰值內存使用量與嵌入維度的關係



Flash2 在記憶體使用量上顯著低於 PyTorch，這意味著在相同的運算任務下，Flash2 能夠更節省記憶體資源。隨著嵌入維度的增加，兩者的記憶體使用量均呈線性增長，但 Flash2 的記憶體管理更加高效，使其在處理更大嵌入維度時依然保持優越的性能表現。

---

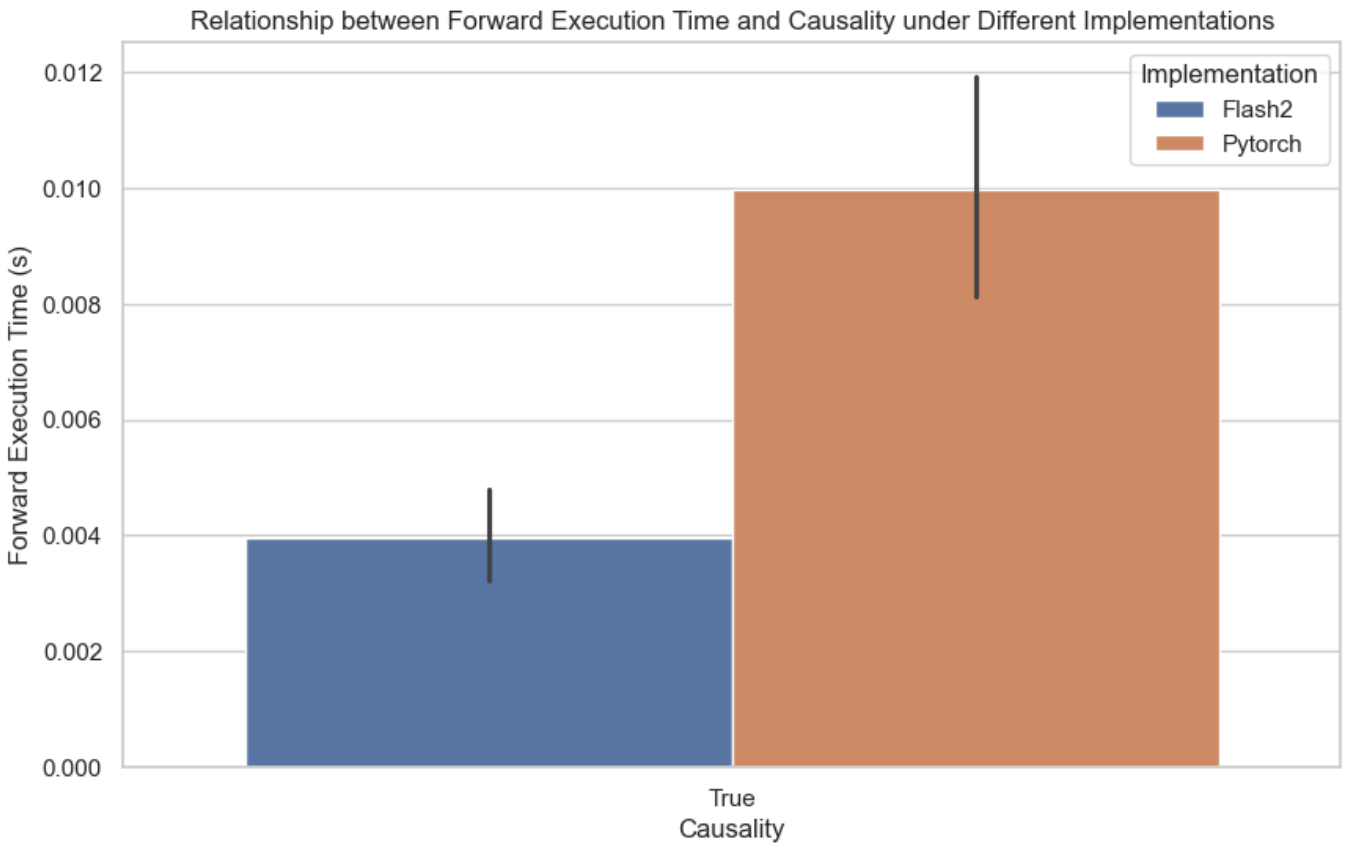
5. 前向執行時間與注意力頭數的關係



當注意力頭數增加時，PyTorch 的執行時間顯著增長，顯示出對多頭注意力機制的計算負荷較大。然而，Flash2 幾乎不受注意力頭數增加的影響，這表明 Flash2 在多頭注意力機制上的優化更加顯著，能夠在保持高效運算的同時，支持更多的注意力頭數，提升模型的表現能力。

---

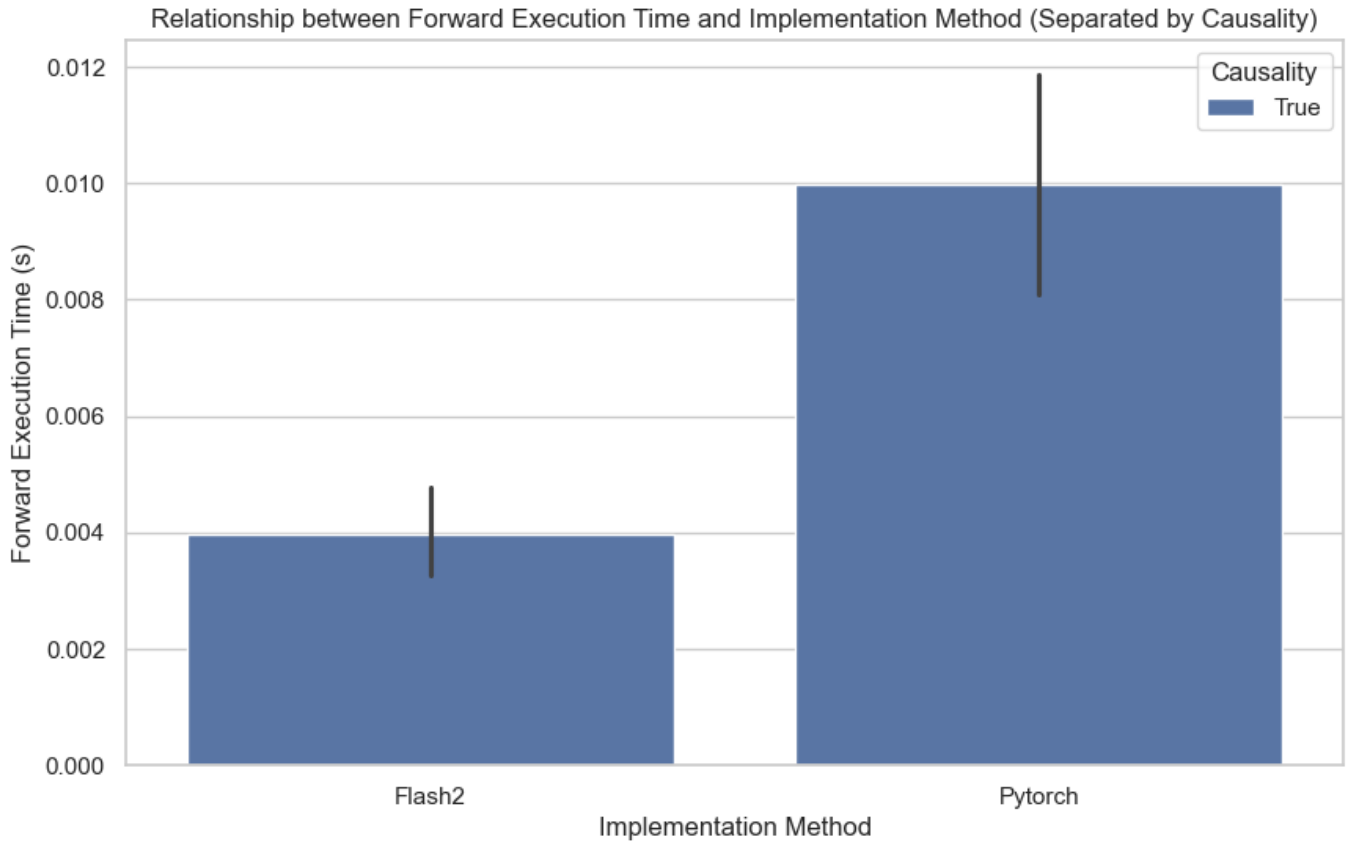
6. 前向執行時間與因果性的關係



在啟用因果性時，PyTorch 的執行時間大幅增加，這對於需要因果注意力的應用來說是一個挑戰。相對地，Flash2 的執行時間僅受到較小的影響，這說明 Flash2 對因果注意力的計算進行了深度優化，使其在需要因果注意力的場景中依然能夠保持高效運算，提升模型的實用性。

---

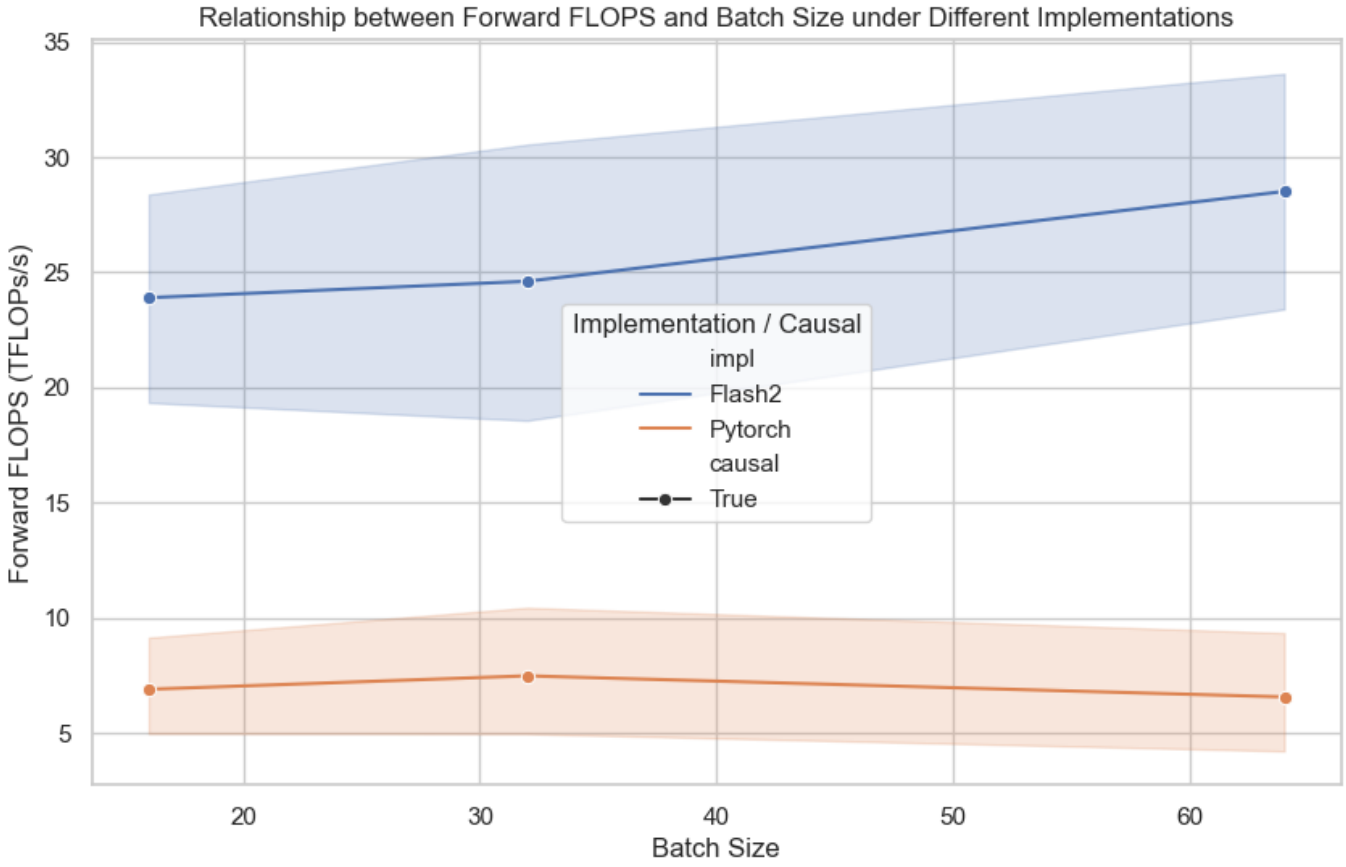
7. 前向執行時間與實現方式的關係

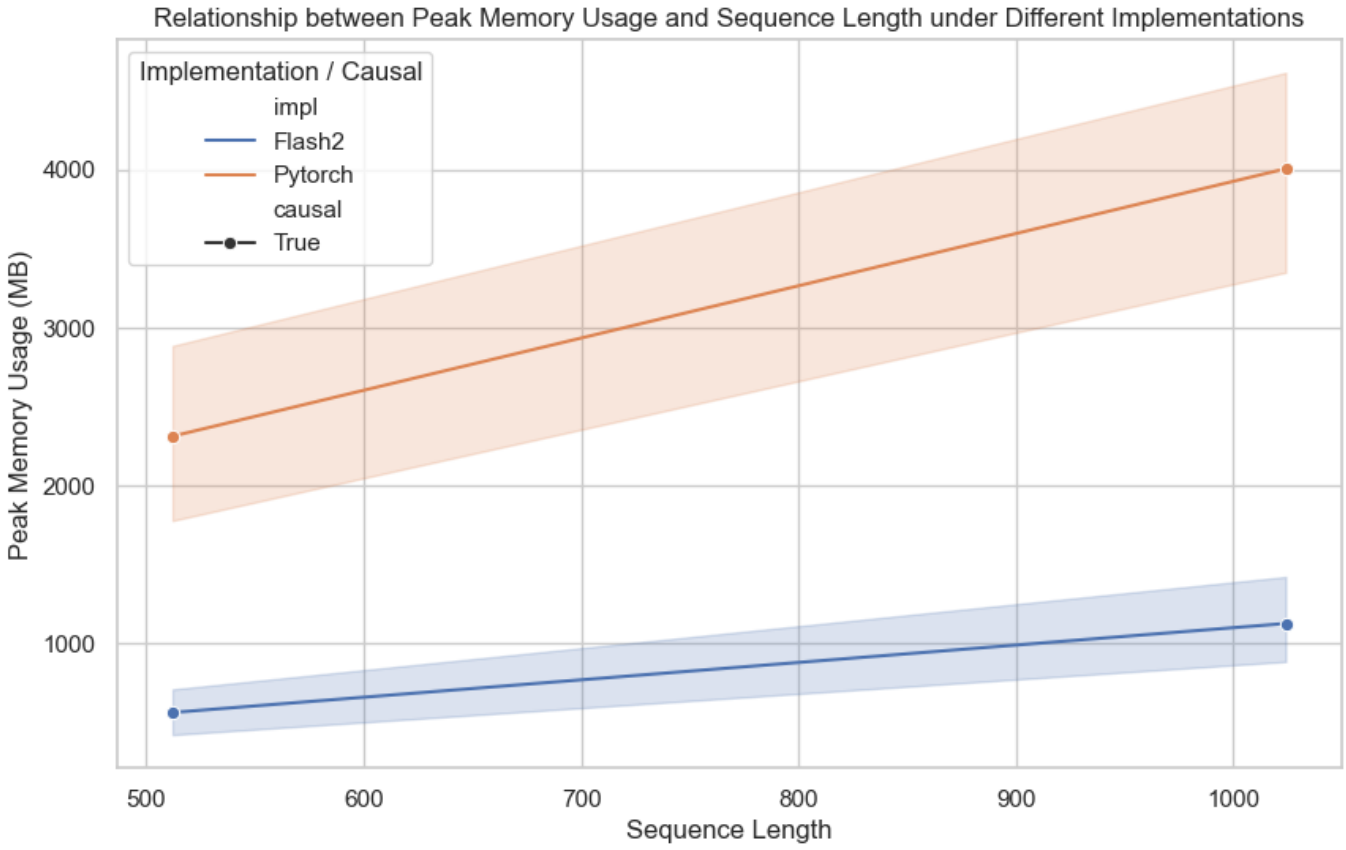


Flash2 的前向執行時間顯著低於 PyTorch，特別是在大規模運算中更為明顯。這主要得益於 Flash2 更加高效的實現方式，包括優化的運算流程和更好的硬體資源利用，從而在大規模數據處理時顯著縮短了前向傳播的時間，提升了整體模型的運行速度。

---

8. Flash2 在批量大小與內存使用上的優勢





隨著批量大小的增長，Flash2 的 FLOPS 顯著高於 PyTorch，這表明 Flash2 能夠更好地利用硬體加速，支持更高的批量大小。此外，Flash2 的內存使用量隨序列長度增加而增長較慢，遠低於 PyTorch 的內存使用量。這種高效的內存管理使 Flash2 更適合處理長序列，提升了在大規模運算中的實用性和效能。

Conclusion

Flash2 在多個關鍵指標上顯著優於 PyTorch，無論是在執行時間、浮點運算次數（FLOPS）還是記憶體使用量方面，尤其在處理大規模參數設置時，Flash2 展現出更高的效率，能夠更快地完成複雜的運算任務。這使得 Flash2 成為需要高性能運算的應用場景中的理想選擇，無論是在研究還是實際應用中，都能提供穩定且卓越的運算表現。此外，Flash2 在資源利用方面表現出色，特別是在內存管理上更加高效，這意味著它能夠處理更大的批量數據和更長的序列，而不會大幅增加系統的負擔。高效的內存利用不僅提升了運算速度，還減少了資源浪費，使得整體系統運行更加流暢。無論是處理大量數據還是運行複雜模型，Flash2 都能夠充分發揮硬體資源的潛力，提供更好的性能表現。更重要的是，Flash2 特別適合那些對計算性能要求極高的應用場景，例如需要處理長序列或多頭注意力機制的任務。在自然語言處理、圖像識別以及其他需要大量計算資源的領域，Flash2 都能夠提供穩定且高效的運算支持。其優化的運算架構和高效的資源利用能力，使得 Flash2 在這些複雜且計算密集的任務中，依然能夠保持卓越的運行效率，滿足現代應用對高性能計算的嚴苛需求。綜合來看，Flash2 的全面性能優勢、高效的資源利用以及適用於高效能計算的廣泛應用場景，使其在現代深度學習和大數據處理領域中，成為不可或缺的強大工具。