Prallel Programming lab5

學號:112062532 姓名:温佩旻

實驗1: 調整sequence length

參數設定: `batch_size=16、num_heads=32、emb_dim=2048、impl=Pytorch/Flash2、causal=false` ， `sequence length` 分別設定128、256、512、1024。
在上述情況下比較time、peak_memory_usage、FLOPs的變化，並畫成下面三張圖表(圖1、圖2、圖3)。

從圖1可以在sequence length較短時，兩種實作方法的時間差異並不大，Flash隨著sequence length上升，forward time和backward time有非常緩慢的上升，但Pytorch的增長速度明顯快於FlashAttention v2，尤其是在backward time; 在圖2中也可以明顯看到隨著sequence length的增長，Pytorch的peak memory usage上升幅度越來越急遽，而Flash則是緩慢上升; 圖3顯示Pytorch版本的實作在隨著序列長度增加，FLOPs的增加幅度相對緩慢，這是因為FlashAttention 較能夠充分利用 GPU 的計算能力。

上述原因可能是因為Flash核心原則是分塊計算，讓FlashAttention 能夠在相同memory限制下處理更長的sequence，也因為能動態計算及釋放attention matrix，能夠保持peak memory usage的穩定性，因此可以推測FlashAttention在長序列的scenario中表現會相較於在Pytorch中表現得更好。

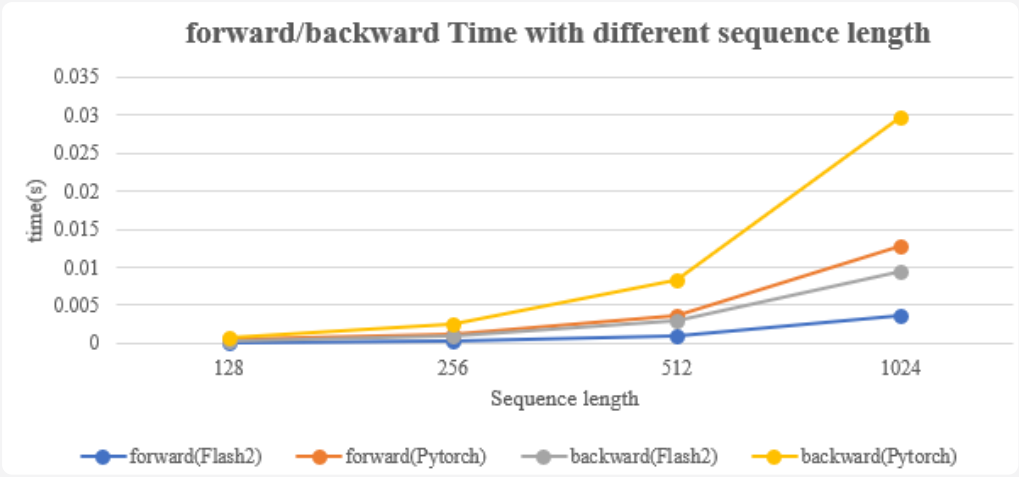

圖1 實驗在不同 sequence length下，Pytorch 和 Flash2 兩種實作方式在**forward/backward time**的變化

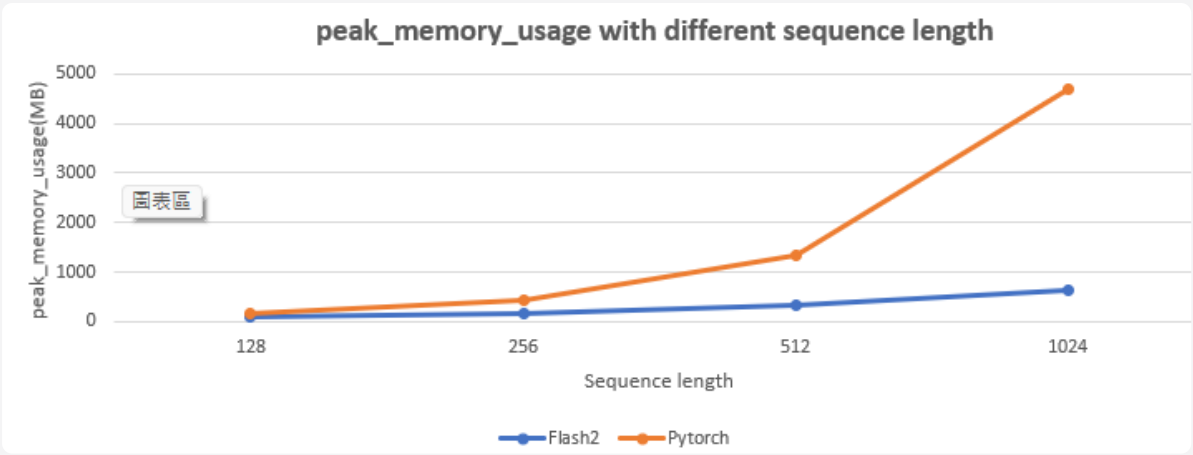

圖2 實驗在不同 sequence length 下，Pytorch 和 Flash2 兩種實作方式在 **peak_memory_usage**的變化

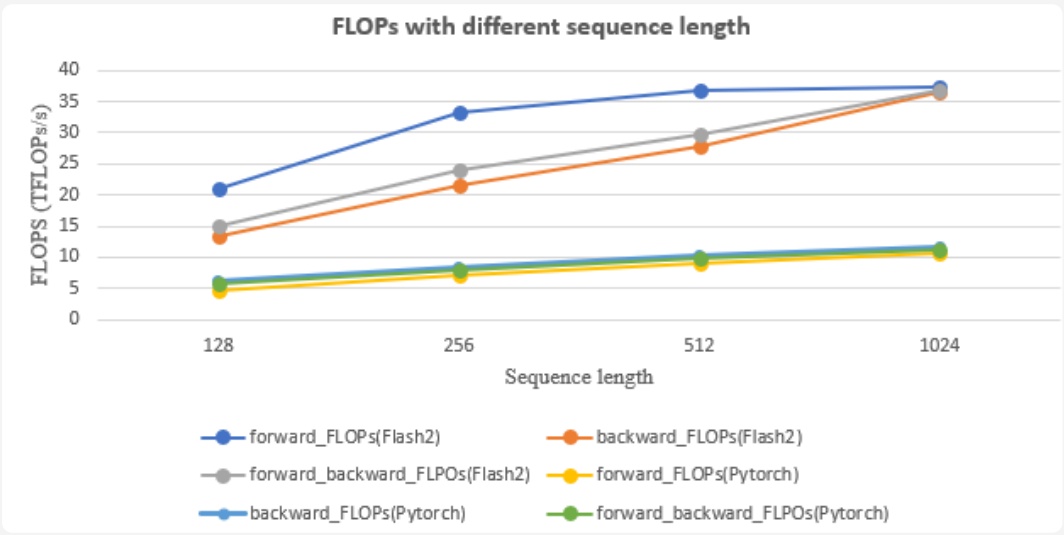

圖3 實驗在不同 sequence length 下，Pytorch 和 Flash2 兩種實作方式在 **FLOPs**的變化

實驗2: set causal和沒有set causal時FLOPs差別

參數設定: `batch_size=16、num_heads=32、emb_dim=2048、impl=Pytorch/Flash2` ，`sequence length=512` ，改變causal。

set causal是讓模型只針對目前和過去的序列進行計算，不會考慮到未來的資訊，可以發現不管是Pytorch還是Flash在沒有設定causal的情況下，FLOPs都會比set causal還要大，這樣的結果是因為在causal的模式下，由於mask的操作減少了計算量。

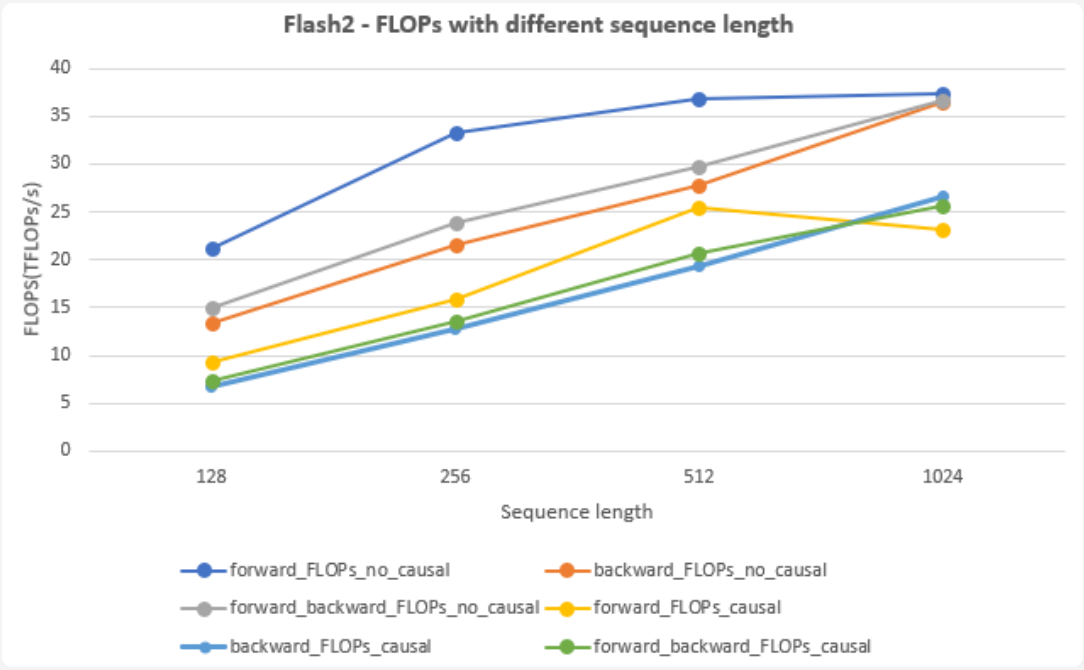

圖4 實驗在不同 sequence length 下，Flash2 有無set causal 在 **FLOPs**的變化

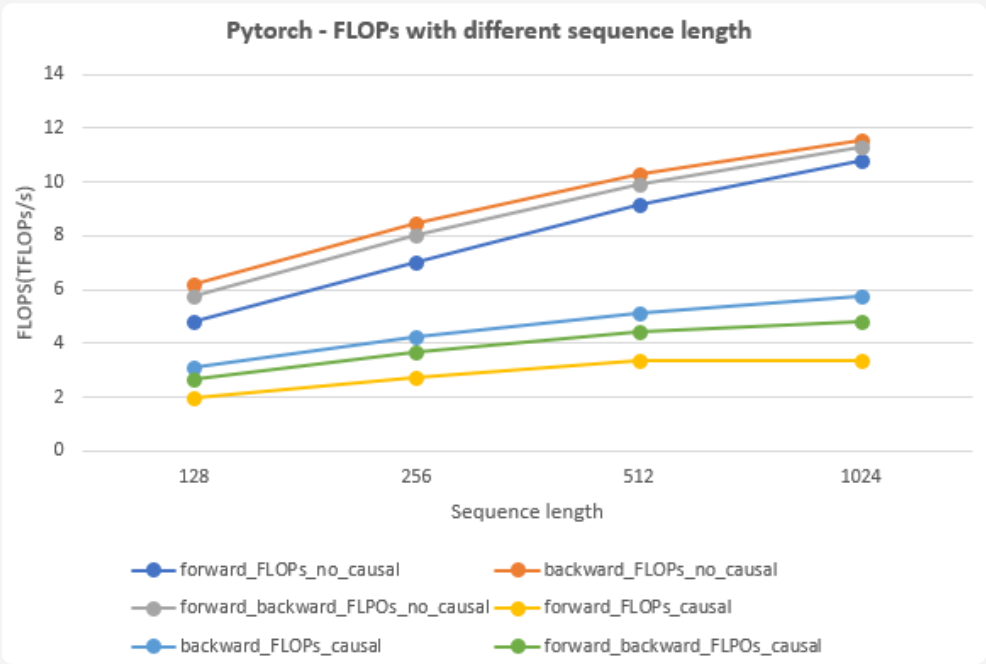

圖5 實驗在不同 sequence length下，Pytorch 有無set causal在**FLOPs**的變化

實驗3: 調整batch size

參數設定: `sequence length=512、num_heads=32、emb_dim=2048、impl=Pytorch/Flash2、causal=true` ， `batch_size` 分別設定8、16、32、64。

這邊實驗提高batch size，觀察執行forward/backward執行時間的變化(圖6)，可以明顯看到Pytorch的實作中，隨著batch size的提高，時間的上升幅度非常明顯，而Flash基本上變動很小，這是因為FlashAttention有針對memory做優化，可以更好處理batch size上升的問題。

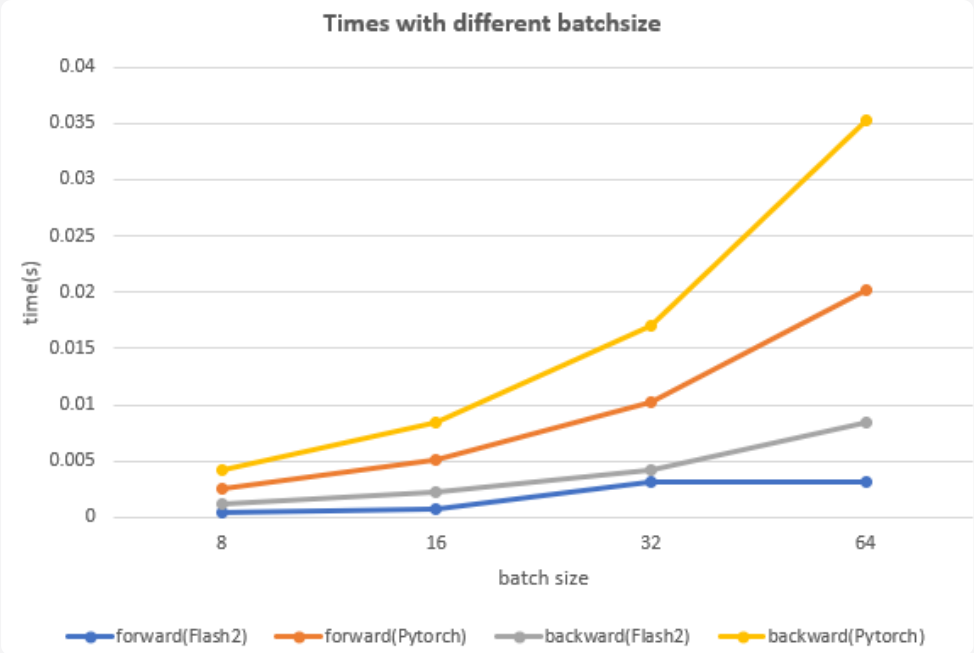

圖6 實驗在不同 batch size下，Pytorch/Flash2 **time**的變化

實驗4: 調整num_heads下

參數設定: `sequence length=512、batch_size=32、emb_dim=2048、impl=Pytorch/Flash2、causal=true` ， `num_heads` 分別設定8、16、32、64。

圖7、圖8、圖9做的實驗是測試不同的num_heads對Pytorch、Flash在time、FLOPs、peak memory usage的差異。可以發現Pytroch隨著head數量增加，執行時間顯著上升，但FLOPs卻明顯下降，代表Pytorch的實作在這邊效率低，而且peak memory usage有明顯上升，但Flash都呈現相較平穩的狀態，代表Pytorch在計算效率低及比較沒有對memory進行有效優化，也間接證明FlashAttention 適合多head的情況。

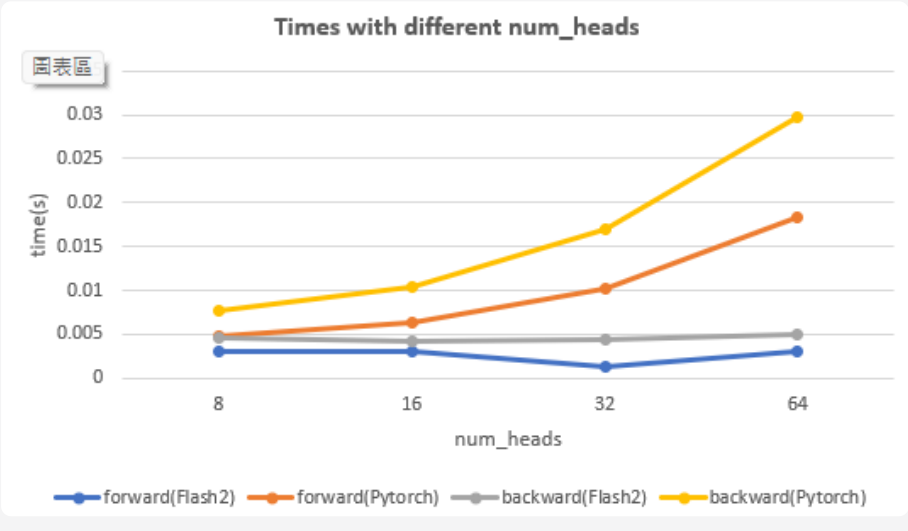

圖7 實驗在不同 num_heads下，Pytorch/Flash2 **time**的變化

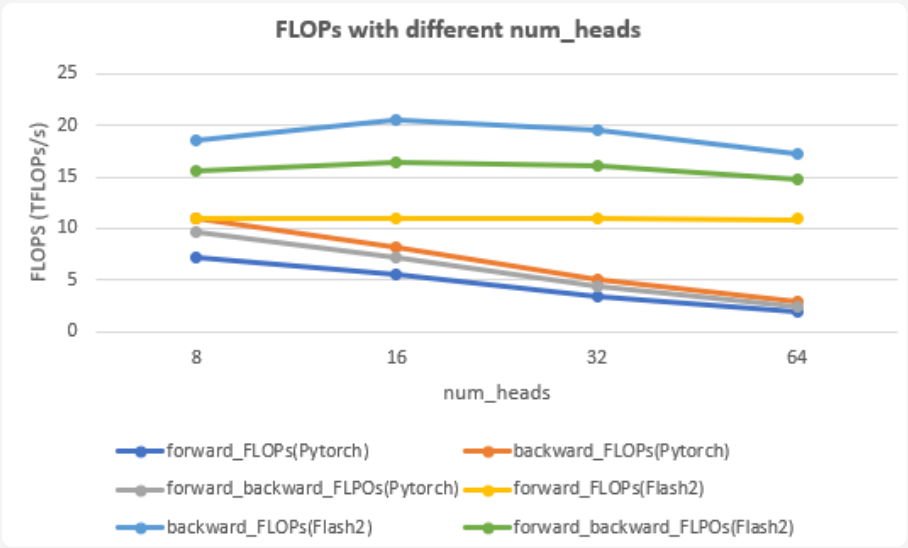

圖8 實驗在不同 num_heads下，Pytorch/Flash2 **FLOPs**的變化

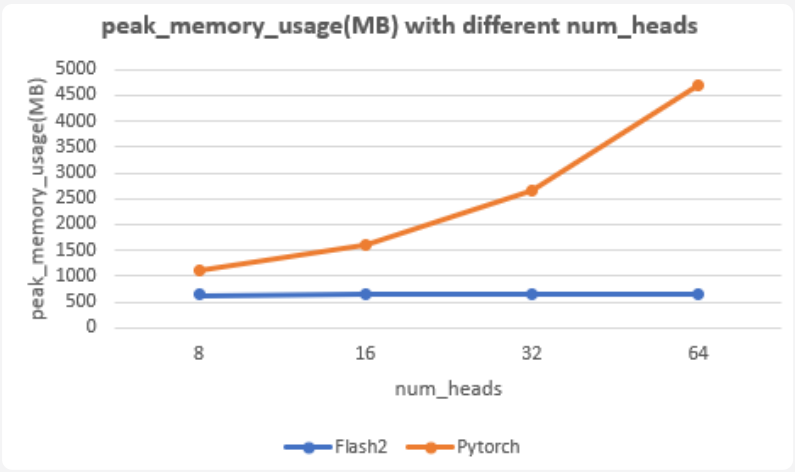

圖9 實驗在不同 num_heads下，Pytorch/Flash2 **peak memory usage**的變化

實驗5: 調整emb_dim

參數設定: `sequence length=512、batch_size=32、num_heads=32、impl=Pytorch/Flash2、causal=true` ， `emb_dim` 分別設定512、1024、2048、4096。

圖10、圖11做的實驗是測試不同的emb_dim對Pytorch、Flash在time、memory usage的差異。這邊可以發現Flash在這個實驗中隨著emd_dim的增加，不論是time還是peak memory usage在兩種實作方式來看，上升幅度都差不多，不過還是FlshAttention有比較好的結果。

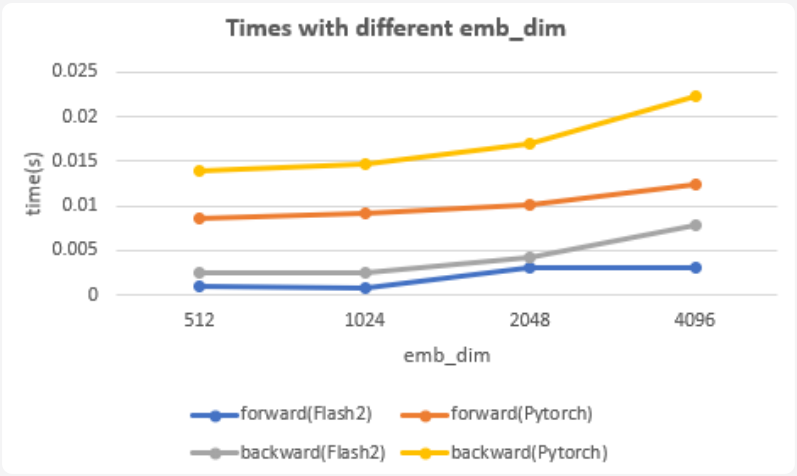

圖10 實驗在不同 emb_dim，Pytorch/Flash2 **time**的變化

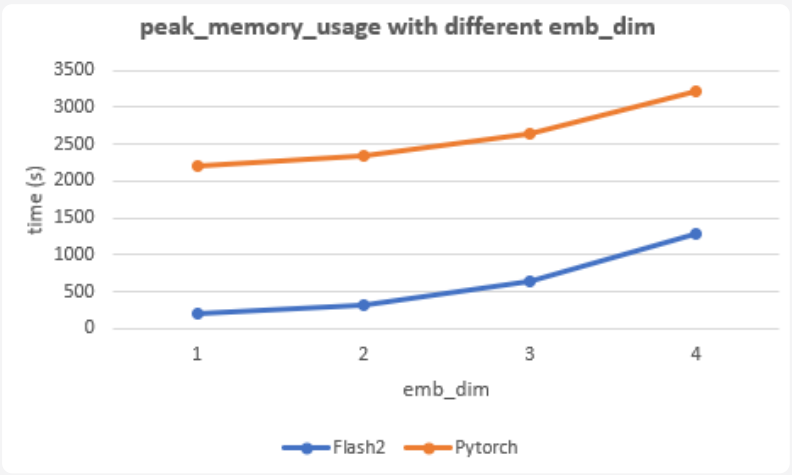

圖11 實驗在不同 emb_dim，Pytorch/Flash2 **peak memory usage**的變化

結論

由上面各種實驗可發現，在長序列和多head的實驗中 FlashAttention表現和Pytorch的相比穩定很多，在batch size的實驗也相對穩定，但在emb_dim的實驗中，似乎就沒有和Pytorch差太多，因此FlashAttention應該會更適合用在長序列和多head場景的應用。