CS542200 Parallel Programming
Lab 6: FlashAttention

Kai-Yuan Jeng 鄭凱元
113062529
kaiyuanjeng@gapp.nthu.edu.tw

1 Experimental Settings

The benchmarks were conducted on a computational environment equipped with a single **NVIDIA Tesla T4** GPU with approximately 15GB of VRAM. The system was configured with NVIDIA Driver version **550.54.15** and **CUDA 12.4**. All tests were executed on this specific hardware configuration to ensure consistency across different parameter settings.

2 Analysis

Execution Time Increases with Sequence Length. Using the Causal setting with a fixed batch size of 16, Figure 1 demonstrates the core advantage of Flash Attention. The Pytorch implementation (red line) exhibits the expected quadratic time complexity $\mathcal{O}(N^2)$, causing execution time to spike dramatically as the sequence length increases (reaching 0.090s at length 2048). In contrast, the Flash1 implementation (blue line) scales much more efficiently, appearing near-linear or sub-quadratic. This confirms that Flash Attention effectively optimizes runtime for long sequences compared to standard attention.

Throughput Gap Increases with Sequence Length. Figure 2 illustrates hardware utilization efficiency. Flash1 (blue) achieves significantly higher throughput, peaking at nearly 9 TFLOPs/s at a sequence length of 2048. The Pytorch implementation (red) saturates much earlier, remaining below 3 TFLOPs/s. This indicates that Flash Attention is far better at keeping the GPU compute units busy, minimizing memory access bottlenecks that typically limit standard attention performance.

Linear Scaling with Batch Size. With Flash1 and the Causal Masking setting, and sequence length fixed at 512, the relationship between batch size and total time is linear (Figure 3). As the batch size increases from 1 to 64, the time increases proportionally (from 0.001s to 0.012s). This suggests that the implementation scales stably with batch size without incurring unexpected overheads or saturation effects within this specific range.

Speedup of Flash1. This bar chart shows the relative speedup of Flash1 over Pytorch (Figure 4). At larger sequence lengths (1024 and 2048), Flash1 is over 3x faster than Pytorch. However, there is a notable anomaly at sequence length 512, where the speedup ratio drops below 1.0 (0.777), indicating that Pytorch was actually faster in that specific configuration. This could be due to overhead dominance in the kernel launch or suboptimal tiling for that specific size before the benefits of Flash Attention kick in at larger scales.

Impact of Causal vs Non-Causal Masking. With the number of heads fixed at 8 and batch size at 16, Figure 5 shows that Causal masking (red line) is faster than Non-Causal masking (blue line). At sequence length 1024, Causal takes 0.009s while Non-Causal takes 0.014s. This performance gain is expected because causal masking effectively requires computing only the lower triangular part of the attention matrix, roughly halving the number of operations compared to the full non-causal computation.

Throughput and Number of Heads. Using Flash1 with Causal Masking (Batch=16, Seq=512), the throughput varies significantly with the number of heads (Figure 6). Performance peaks at 24 heads (5.2 TFLOPs/s) and drops noticeably when increasing to 64 heads (2.8 TFLOPs/s). This indicates a "sweet spot" for parallelism. Having too many heads might lead to smaller, less efficient memory chunks per head, reducing overall hardware utilization.

Linear Relation with Embedding Dimension. With the same settings (Flash1, Causal, Batch=16, Seq=512), the execution time increases linearly as the embedding dimension grows (Figure 7). The trend line from dimension 64 to 2048 is steady and predictable. This confirms that the computational cost scales proportionally with the size of the hidden states (embedding dimension), which is consistent with the theoretical complexity of matrix multiplications involved in attention.

GPU Memory Utilization. With batch size fixed at 16 and using the Causal setting, Figure 8 presents perhaps the most critical insight. The Pytorch implementation (red) shows rapid, quadratic memory growth, consuming over 4.5 GB at sequence length 2048. In contrast, Flash1 (blue) maintains an almost flat, linear memory footprint, staying around 500 MB. This proves Flash Attention's memory efficiency, allowing for training with much longer sequences without encountering Out-Of-Memory (OOM) errors.

Comparison of Forward and Backward Passes. With Flash1 and Causal Masking, across all batch sizes and sequence lengths shown, the Backward pass (red bars) consistently takes significantly longer than the Forward pass (blue bars)—often by a factor of 2x to 3x (Figure 9). This is typical for neural network training, as the backward pass involves calculating gradients for all parameters, which is computationally more expensive than the inference-only forward pass.

Throughput Increases with Head Dimension. With Flash1 and Causal Masking (Batch=16, Seq=512), increasing the head dimension leads to higher throughput (Figure 10). The system achieves 4.2 TFLOPs/s at dimension 64 compared to only 2.4 TFLOPs/s at dimension 16. Larger head dimensions likely allow the GPU's tensor cores to operate more efficiently on larger contiguous blocks of data, improving vectorization and overall speed.

Throughput Heatmap. Using Flash1 and Causal Masking, the heatmap (Figure 11) visualizes the "efficiency zones" of the hardware. The dark red areas (high TFLOPs) are concentrated in the bottom-right, corresponding to large batch sizes and long sequence lengths (e.g., Batch 16, Seq 2048). The lighter yellow areas indicate that for small batches or short sequences, the GPU is underutilized, leading to lower throughput performance.

Efficiency Summary: Flash1 vs Pytorch. With Batch=16, Seq=512, and Causal setting, Figure 12 aggregates the efficiency metrics. Flash1 maintains high TFLOPs (4.0) for both Forward and Backward passes. Pytorch is significantly less efficient, particularly in the Forward pass (1.2 TFLOPs). This demonstrates that Flash Attention is not just faster in terms of raw time, but also much more efficient at utilizing the available compute power of the hardware for both training phases.

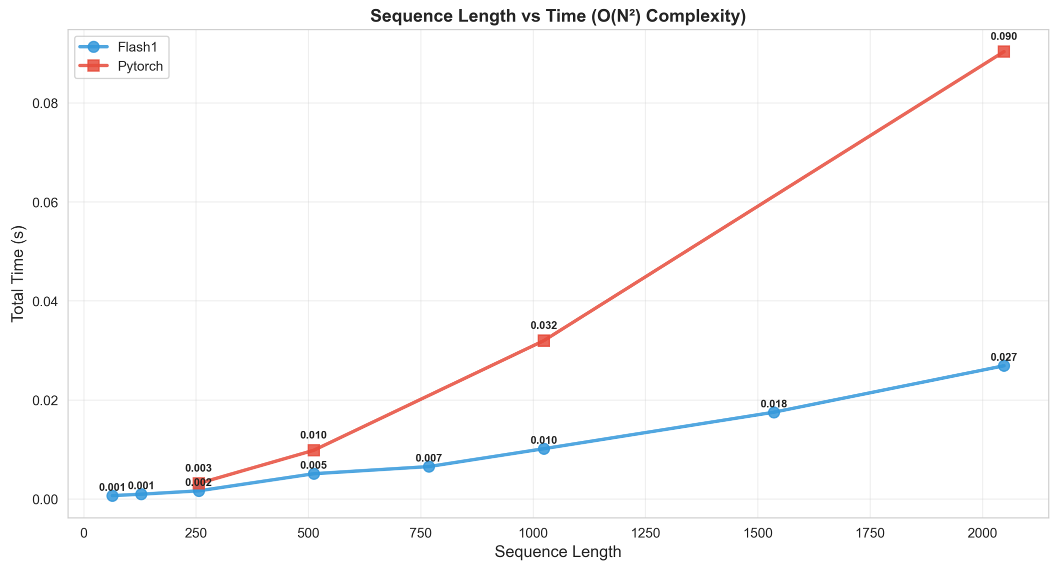

Figure 1: Execution Time Increases with Sequence Length

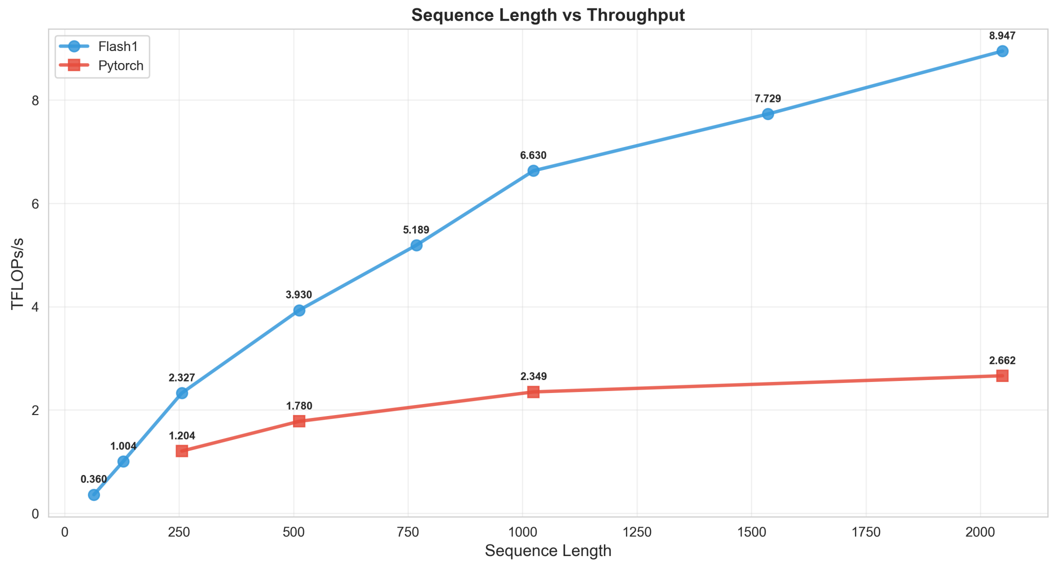

Figure 2: Throughput Gap Increases with Sequence Length

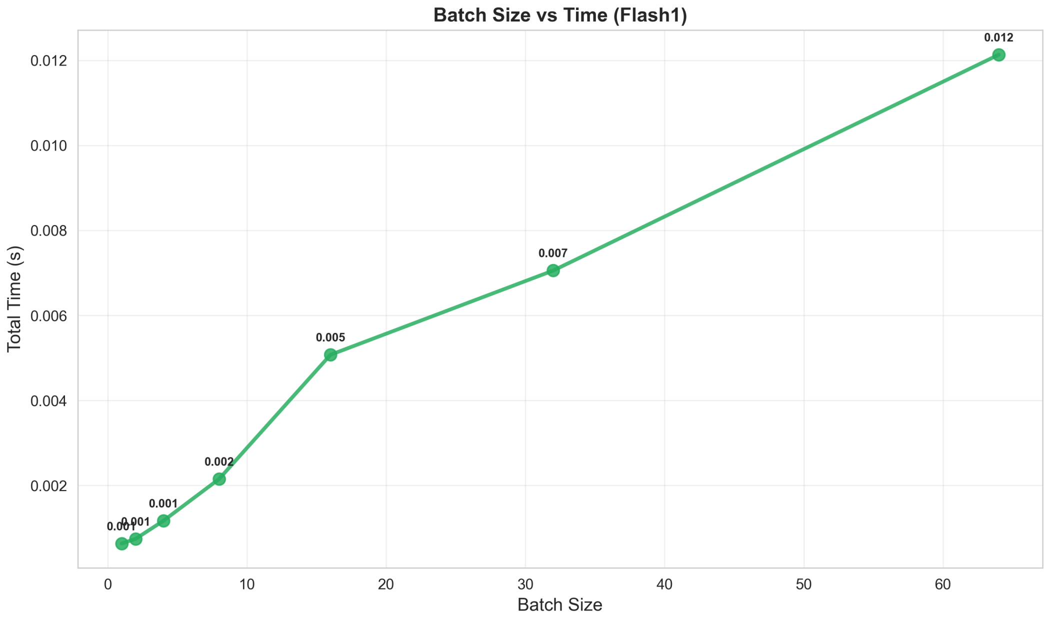

Figure 3: Linear Scaling with Batch Size

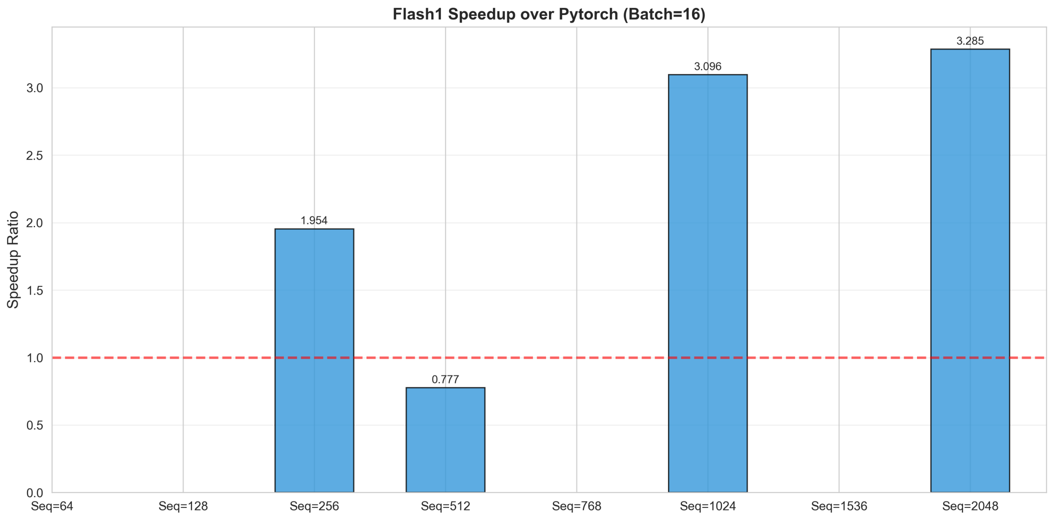

Figure 4: Speedup of Flash1 over Pytorch

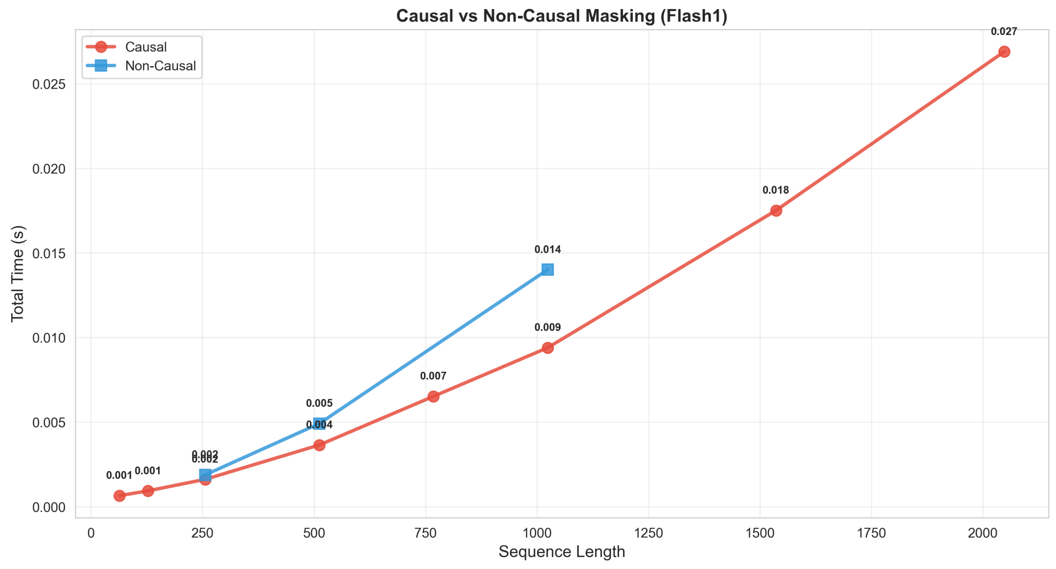

Figure 5: Impact of Causal vs Non-Causal Masking

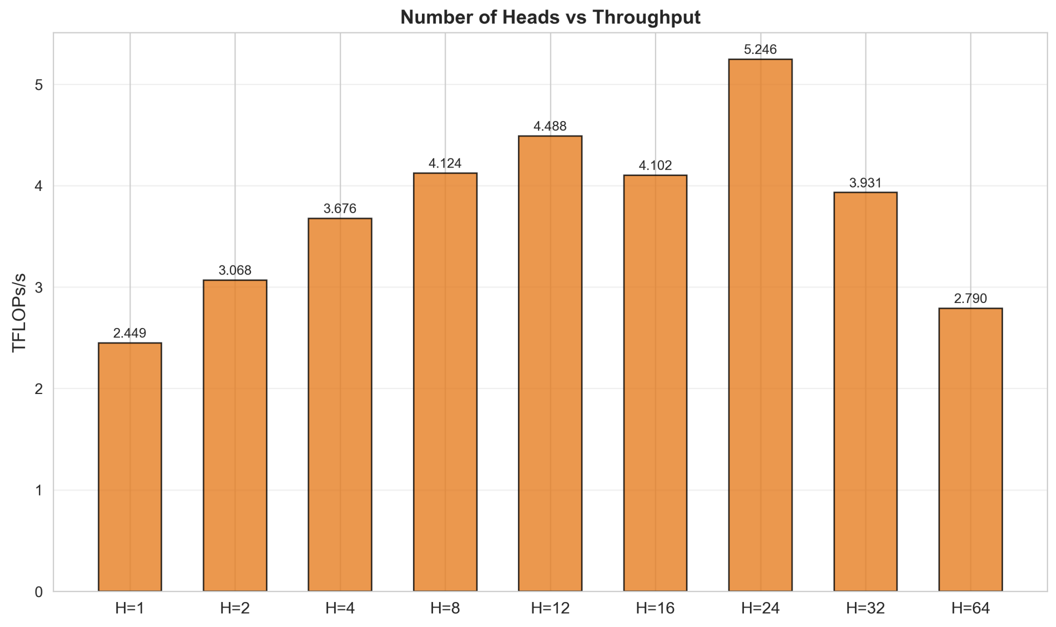

Figure 6: Throughput vs. Number of Heads

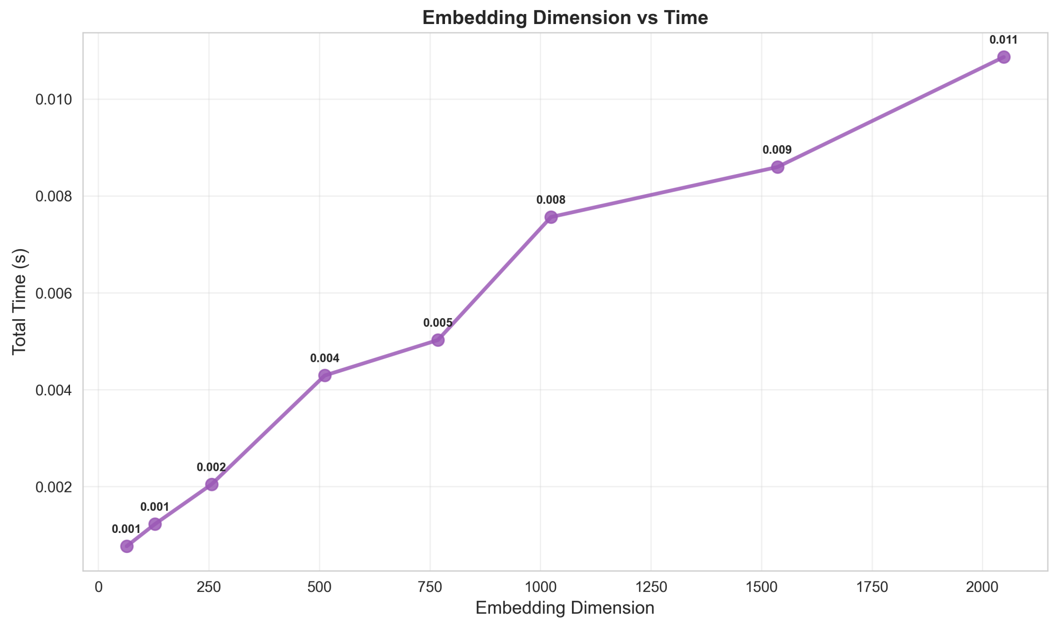

Figure 7: Linear Relation with Embedding Dimension

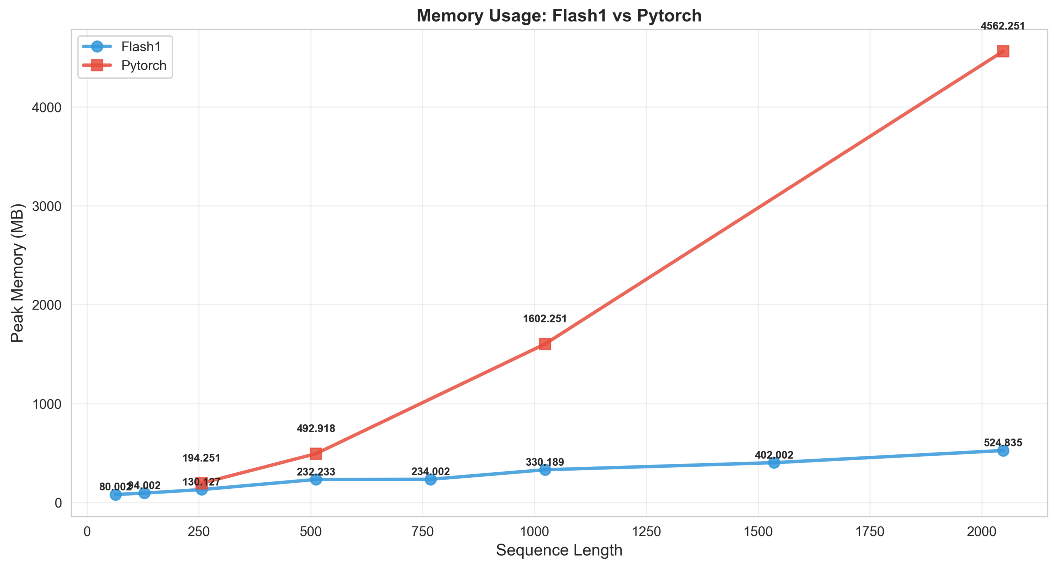

Figure 8: GPU Memory Utilization

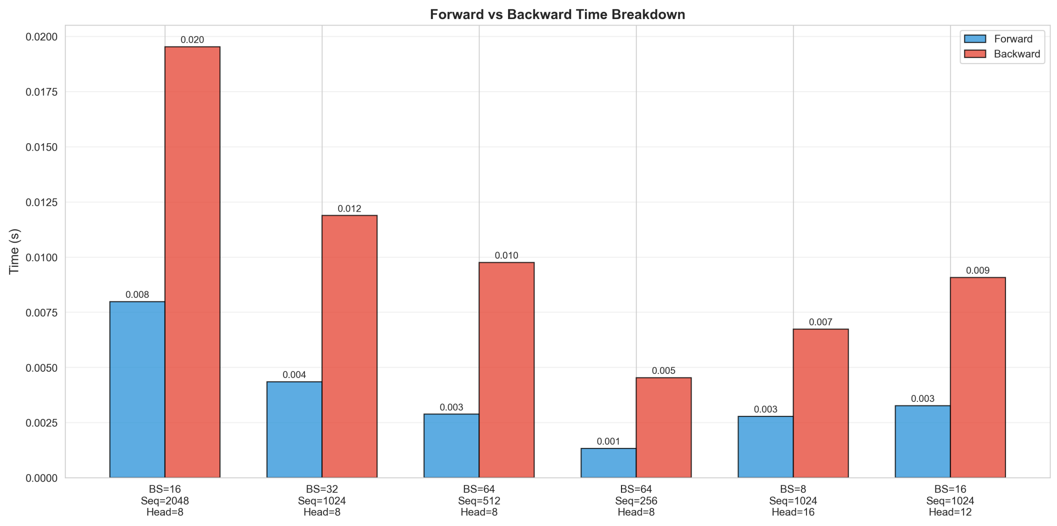

Figure 9: Comparison of Forward and Backward Passes

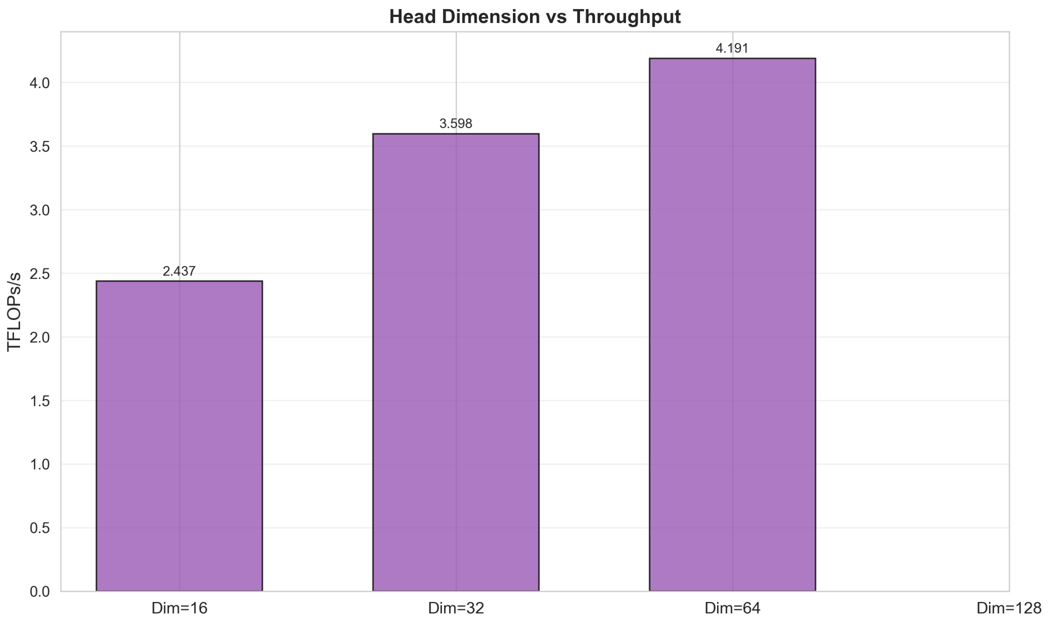

Figure 10: Throughput Increases with Head Dimension

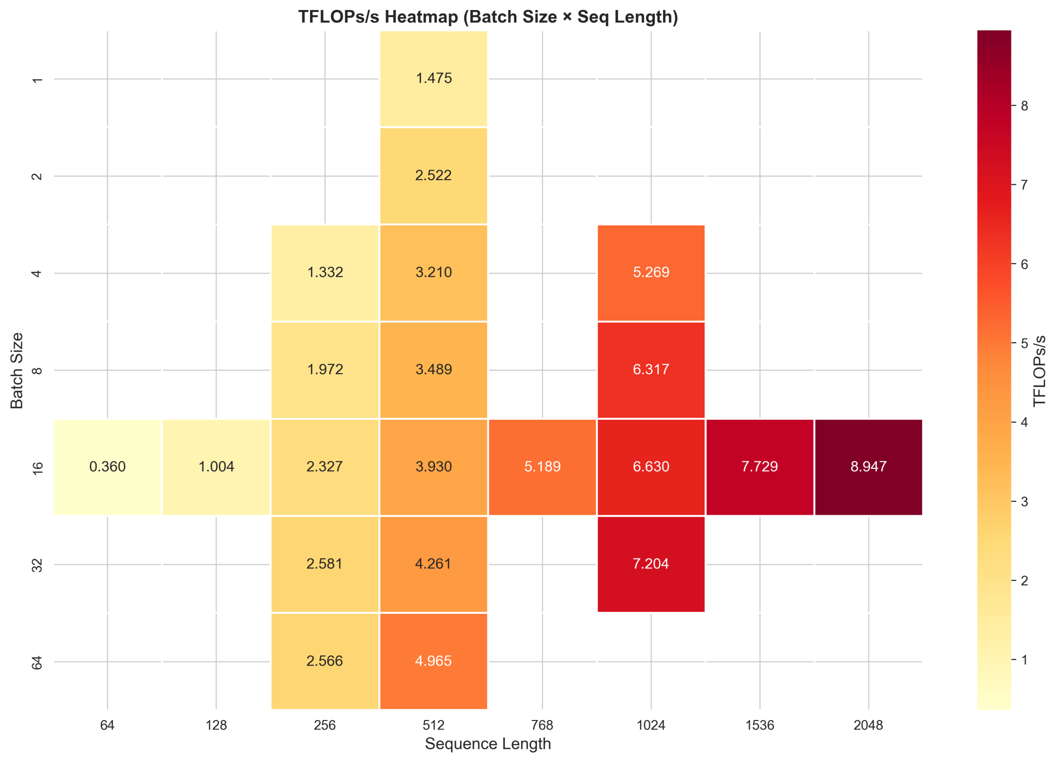

Figure 11: Throughput Heatmap

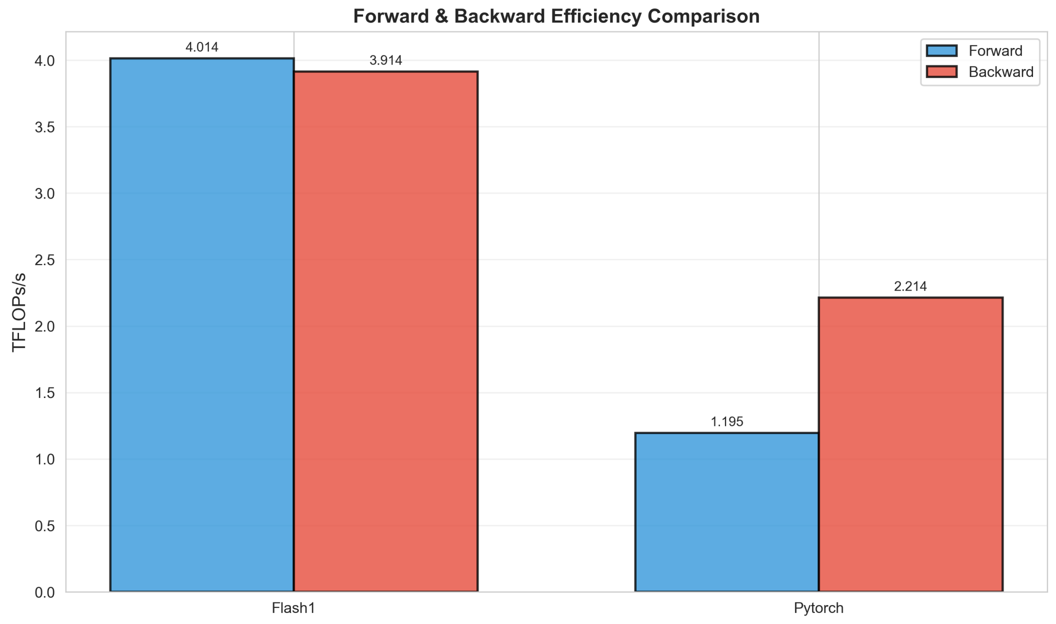

Figure 12: Efficiency Summary: Flash1 vs Pytorch