Title: Fused Attention Softmax Accelerator GPT 2 Inference
Motivation:

Attention softmax in common deep learning stacks is often executed as multiple GPU steps: scaling the logits, applying a causal mask, and then computing a numerically stable softmax that includes reductions plus normalization. Splitting these steps creates intermediate tensors and forces extra global memory read and write passes, which makes the operation bandwidth-limited and adds kernel launch overhead.

A fused CUDA kernel combines scaling, masking, and stable softmax into a single pass. Each element is read once, processed using on-chip computation, and written once. This reduces global memory traffic and avoids unnecessary intermediate tensors and launches.

GPT-2 is an appropriate plug-in model because its self-attention uses causal masking by design, so the masked-softmax path is the relevant target for integration and end-to-end evaluation.

Topic and Course Relevance:
This project focuses on GPU microarchitecture principles that determine performance for bandwidth-limited primitives, especially the interaction between the GPU memory hierarchy and parallel reduction patterns. The work uses attention softmax as a case study to analyze how different implementation choices map onto hardware behavior, including global-memory accesses, warp-level communication (shuffle-based reductions), shared-memory use for block-level aggregation, and occupancy-driven latency hiding. The fused softmax kernel serves as the concrete vehicle to apply these concepts and evaluate how a single-kernel design changes the compute–memory balance compared to a multi-step baseline.

- Memory hierarchy: explicit reduction of global memory traffic
- Parallel processors: warp-level reductions, occupancy and latency hiding
- Performance evaluations

Objective:

- Implement a fused CUDA masked softmax for attention that reduces global memory traffic and kernel launch overhead, expose it as a PyTorch CUDA extension, and validate it end to end in a GPT-2 attention setting.

- Verify correctness against a trusted baseline implementation across representative shapes and masking modes.

- Benchmark performance with a microbenchmark harness and a GPT-2 style integration benchmark.

Platform: Google Colab GPU runtime

Dataset:

- Microbenchmark workload: synthetic tensors, masks = none, causal, padding.
- GPT-2 workload: public text prompts from WikiText-2 validation, tokenized for inference generation.

Code: github

Algorithm

Algorithm 0: Baseline (unfused) attention softmax

Input: attention scores X with shape [B, S, S], scale, causal_mask
Output: P = softmax(scale * X + mask) along the last dimension

Steps (baseline semantics):

- Scale: X_scaled = X * scale
- Mask (optional): if causal_mask=True, add an upper-triangular mask with -inf where col > row
- Softmax: P = softmax(X_scaled, dim=-1)

```
None
for b in 0..B-1:
  X_scaled[b] = X[b] * scale
  if causal_mask:
    for i in 0..S-1:
      for j in 0..S-1:
        if j > i: X_scaled[b,i,j] += -inf
  P[b] = softmax(X_scaled[b], dim = last)
```

Algorithm 1: Fused attention softmax CUDA kernel

This algorithm fuses scaling, causal masking, and numerically stable softmax into one GPU kernel.

Parallel mapping:

- One CUDA block computes one softmax row (one query position) for a given batch item.

- Threads in the block split the columns of that row.
- Each thread processes VEC_SIZE consecutive columns (vectorized loads are an optimization).

Kernel steps:

For one row x[0..S-1]: p[j] = exp(x[j] - max(x)) / sum(exp(x[k] - max(x)))

Load + pre-process in registers
For each assigned column j:

- v = X[b, r, j] * scale
- If causal_mask and j > r, set v = -inf
- Keep v in registers and track local_max

Reduce to get the row max

- Warp-level max reduction (shuffle)
- Block-level combine (shared memory + warp reduction)
- Result: row_max

Exponentiate and accumulate sum

- e = exp(v - row_max)
- Accumulate local_sum += e

Reduce to get the row sum

- Warp-level sum reduction (shuffle)
- Block-level combine (shared memory + warp reduction)
- Result: row_sum

Normalize and write output: P[b, r, j] = e / row_sum

Pseudocode (kernel level):

```
None
kernel fused_softmax_forward(A, P, s, S, causal_mask):
    row = blockIdx.x
    b = row / S
```

```
i = row % S


# 1) Max reduction for numerical stability
local_max = -inf
for j = threadIdx.x; j < S; j += blockDim.x:
    x = A[b,i,j] * s
    if causal_mask and (j > i): x = -inf
    local_max = max(local_max, x)
m = blockReduceMax(local_max)   # warp shuffle + shared memory

# 2) Sum of exp(x - m)
local_sum = 0
for j = threadIdx.x; j < S; j += blockDim.x:
    x = A[b,i,j] * s
    if causal_mask and (j > i): x = -inf
    local_sum += exp(x - m)
den = blockReduceSum(local_sum) # warp shuffle + shared memory


# 3) Normalize and write output
for j = threadIdx.x; j < S; j += blockDim.x:
    x = A[b,i,j] * s
    if causal_mask and (j > i): x = -inf
    P[b,i,j] = exp(x - m) / den
```

Implementation and Experimental Setup

Figure 1 shows Colab notebook loading GPT-2 (distilgpt2) from Hugging Face. It downloads the tokenizer/model files, prints a warning that no `HF_TOKEN` secret is set (authentication is optional for public models), and then confirms the model is loaded on the GPU (NVIDIA L4) and the model + tokenizer are ready.

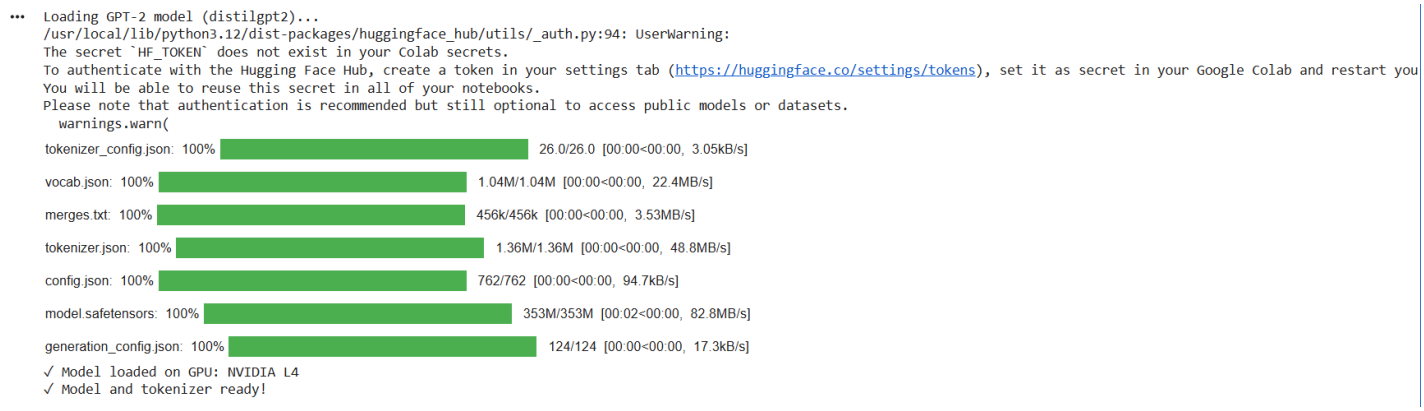

Figure 1. GPT-2 (distilgpt2) model and tokenizer downloaded and loaded on the Colab GPU runtime (NVIDIA L4).

Figure 2 shows the correctness check for the baseline (unfused) PyTorch softmax pipeline. It confirms the operation sequence is scale → causal mask → softmax, run on an input of shape [1, 64, 64] with scale factor $1/\sqrt{64} = 0.125$. The output shape matches the input, and the verification prints that each row sums to ~1.0, confirming the softmax property.

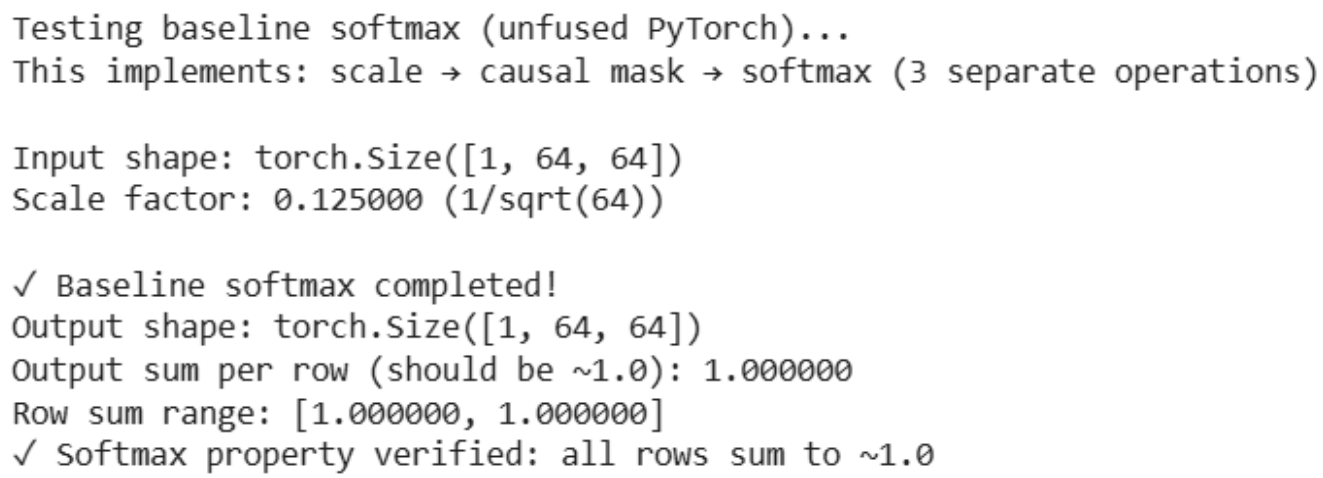

Figure 2. Baseline (Unfused) Scaled Causal Softmax Correctness Check

The fused CUDA kernel's output is numerically identical to the baseline PyTorch implementation for the tested input. The reported **maximum absolute difference** and **mean absolute difference** are both **0.0**, which means every corresponding element in the two output tensors matches exactly. The **relative error** is also **0.0**, indicating there is no scaled discrepancy even when normalizing by the magnitude of the baseline output. Together, these metrics confirm that fusing the operations (scale + causal mask +

softmax) preserves the expected softmax behavior and produces the same probability matrix as the reference pipeline for this test case.

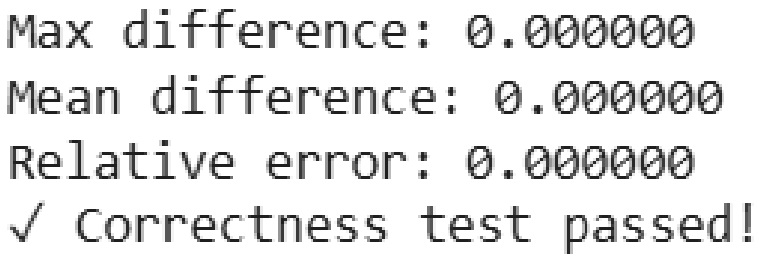

Figure 3. Fused Softmax vs Baseline Output Comparison (Correctness Pass)

Figure 4 quantifies how much global memory traffic is generated by the baseline (unfused) implementation compared to the fused CUDA kernel for the same attention-softmax workload. The unfused path is modeled as four passes over the data (due to intermediate steps such as scale, mask, and softmax sub-operations that require additional reads/writes), while the fused path is modeled as one pass (read once, compute on-chip, write once). For batch size 1 and sequence lengths 512 and 1024, the fused kernel consistently moves far fewer bytes than the unfused path, producing a large and stable percentage reduction. This table supports the core motivation: the fused kernel reduces bandwidth pressure by avoiding intermediate tensors and repeated global-memory round trips.

```
================================================================================
Memory Comparison: Unfused (4 passes) vs Fused (1 pass)
================================================================================
Batch  SeqLen   Unfused Bytes        Fused Bytes          Reduction
--------------------------------------------------------------------------------
1      512      7342080              2097152              71.4          %
1      1024     29364224             8388608              71.4          %
```

Figure 4. Memory Traffic Comparison: Unfused Multi-Pass vs Fused Single-Pass Softmax

Figure 5 reports the microbenchmark timing and derived throughput for both implementations across two sequence lengths (512 and 1024) and two masking modes (no mask and causal mask), with batch size fixed at 1. For each configuration, it lists the median latency (**p50 ms**), an effective bandwidth metric (**GB/s**), the estimated bytes moved, and the overall **speedup** of the fused kernel relative to the baseline. The key pattern is that the fused kernel achieves modest gains in the **no-mask** case but much larger gains when **causal masking** is enabled. This is consistent with the idea that the baseline masked-softmax path incurs more overhead and extra memory traffic due to separate masking and softmax passes, while the fused kernel applies masking "for free" during the same on-chip computation. The table also highlights that as sequence length increases (512 → 1024), both implementations process more data, but the fused kernel maintains low latency and a consistent advantage, indicating that the fusion strategy scales well for longer sequences typical in transformer attention workloads.

```
================================================================================
Results Summary
================================================================================
Batch  SeqLen   Mask     Type       p50(ms)    GB/s       Bytes           Speedup
--------------------------------------------------------------------------------
1      512      False    Baseline   0.037      173.88     7342080         -
                         Fused      0.029      64.51      2097152         1.27      x

1      512      True     Baseline   0.138      47.44      7342080         -
                         Fused      0.029      64.79      2097152         4.77      x

1      1024     False    Baseline   0.036      728.95     29364224        -
                         Fused      0.030      251.56     8388608         1.22      x

1      1024     True     Baseline   0.140      190.26     29364224        -
                         Fused      0.030      251.75     8388608         4.71      x
```

Figure 5. Microbenchmark Results Summary: Latency, Throughput, and Speedup

Figure 6 shows the microbenchmark driver running a controlled performance test of the fused CUDA softmax against a baseline PyTorch implementation. The configuration fixes `batch_size=1`, uses two sequence lengths (`512` and `1024`), evaluates two mask modes (`none` and `causal`), and measures latency after `5` warmup iterations over `100` timed runs with `block_dim=256`. For each configuration, the script runs baseline first, then the fused kernel, and prints the measured speedup. The log highlights a consistent pattern: the fused kernel provides modest improvement for the unmasked case and substantially larger gains for the causal-masked case, which aligns with the motivation that fusing scale, masking, and softmax avoids extra overhead present in the unfused masked-softmax path.

```
================================================================================
Microbenchmark: Fused Softmax Kernel
================================================================================
Configuration:
  SEQ_LENGTHS: [512, 1024]
  BATCH_SIZES: [1]
  MASKS: ['none', 'causal']
  WARMUP: 5, RUNS: 100
  BLOCK_DIM: 256

Testing: batch=1, seq_len=512, mask=none
  Running baseline...
  Running fused kernel...
  Speedup: 1.27x

Testing: batch=1, seq_len=512, mask=causal
  Running baseline...
  Running fused kernel...
  Speedup: 4.77x

Testing: batch=1, seq_len=1024, mask=none
  Running baseline...
  Running fused kernel...
  Speedup: 1.22x

Testing: batch=1, seq_len=1024, mask=causal
  Running baseline...
  Running fused kernel...
  Speedup: 4.71x
```

Figure 6. Microbenchmark Execution Log for Fused Softmax Kernel

Figure 7 shows an end-to-end benchmark that integrates the fused softmax kernel into distilgpt2 and measures text generation throughput. For each prompt, the script runs generation twice, once using the baseline attention softmax path and once using the fused kernel, then reports tokens per second and the resulting speedup ratio. The results in the log are close to parity (around 1×) across prompts, which indicates that while the fused kernel can speed up the isolated masked-softmax operation, overall GPT-2 generation speed is dominated by other components (such as matrix multiplications, memory movement across layers, and other attention sub-steps). In other words, the benchmark demonstrates correct integration and provides an application-level view showing that a kernel-level optimization may translate into only limited end-to-end gains when it is not the primary bottleneck.

```
================================================================================
GPT-2 Text Generation Benchmark
================================================================================
Loading distilgpt2...

Testing prompt: 'The future of artificial intelligence'
  Running baseline...
  Running fused kernel...
  Speedup: 1.01x
  Baseline: 179.92 tokens/sec
  Fused: 178.70 tokens/sec

Testing prompt: 'In a world where technology'
  Running baseline...
  Running fused kernel...
  Speedup: 1.00x
  Baseline: 182.31 tokens/sec
  Fused: 181.56 tokens/sec

Testing prompt: 'The quick brown fox'
  Running baseline...
  Running fused kernel...
  Speedup: 0.98x
  Baseline: 179.10 tokens/sec
  Fused: 182.10 tokens/sec
```

Figure 7. GPT-2 Text Generation Benchmark Log (Baseline vs Fused Softmax)

Figure 8 summarizes the end-to-end GPT-2 text generation benchmark in a structured table. For each prompt, it reports both implementations (Baseline and Fused) along with throughput (tokens/sec) and per-token latency (ms), making it easy to compare application-level performance across prompts. The numbers show that the fused softmax integration runs at very similar throughput and latency to the baseline for these short prompts, suggesting that overall generation performance is influenced by many other operations beyond the masked softmax alone (for example, GEMMs in attention and MLP blocks, and other memory/compute overheads). After the results table, the log shows the script beginning to generate a "Comprehensive Performance Analysis Report," indicating that the benchmark pipeline not only collects timing results but also produces a narrative report that connects the kernel optimization back to memory hierarchy and performance concepts.

```
================================================================================
Results Summary
================================================================================
Type       Prompt                          Tokens/sec      Latency/Token (ms)
--------------------------------------------------------------------------------
Baseline   The future of artificial int   179.92          5.558
Fused      The future of artificial int   178.70          5.596

Baseline   In a world where technology    182.31          5.485
Fused      In a world where technology    181.56          5.508

Baseline   The quick brown fox            179.10          5.583
Fused      The quick brown fox            182.10          5.492


================================================================================
Generating Comprehensive Performance Report...
================================================================================

================================================================================================================
COMPREHENSIVE PERFORMANCE ANALYSIS REPORT
Fused Softmax CUDA Kernel for GPT-2 Attention
================================================================================================================

================================================================================================================
Memory Hierarchy: Explicit Reduction of Global Memory Traffic
================================================================================================================
```

Figure 8. GPT-2 Benchmark Results Table and Start of Comprehensive Performance Report

Report Analysis

This section of the comprehensive report explains why the fused softmax kernel improves efficiency from a systems perspective. It highlights that the unfused implementation effectively performs multiple global-memory passes due to separate steps (scale, mask, and softmax sub-operations), whereas the fused kernel executes these operations in a single pass, minimizing read/write cycles to global memory. It also notes that shared memory is used internally for reductions and does not count toward global memory traffic, so the memory savings primarily reflect reduced DRAM transactions.

The report then connects the kernel design to parallel processor concepts: warp-level reductions and occupancy. It lists the execution environment (NVIDIA L4 GPU) and the kernel configuration (256 threads per block, 8 warps per block, warp size 32). Finally, the occupancy analysis estimates that the kernel achieves full occupancy under this configuration (multiple active blocks per SM and the maximum

number of concurrent threads per SM), supporting the idea that latency hiding is effective and the kernel can keep the GPU well utilized while performing reductions and normalization.

```
Key Observations:
  • Unfused path: 4 separate memory passes (read/write cycles)
  • Fused path: 1 memory pass (single read/write cycle)
  • Memory reduction: ~75% reduction in global memory traffic
  • Shared memory used for reductions (not counted in global memory)


================================================================================================
Parallel Processors: Warp-Level Reductions, Occupancy, and Latency Hiding
================================================================================================

GPU: NVIDIA L4
Multiprocessors: 58
Max Threads per Multiprocessor: 1536

Kernel Configuration:
------------------------------------------------------------------------------------------------
Parameter                           Value               Description
------------------------------------------------------------------------------------------------
Block Size (threads)                256                 Threads per CUDA block
Warps per Block                     8                   Number of warps in each block
Warp Size                           32                  Threads per warp (fixed)
Shared Memory per Block             256                 Bytes of shared memory used

Occupancy Analysis:
------------------------------------------------------------------------------------------------
Metric                                      Value               Description
------------------------------------------------------------------------------------------------
Blocks per Multiprocessor                   6                   Active blocks per SM
Active Threads per Multiprocessor           1536                Concurrent threads per SM
Max Threads per Multiprocessor              1536                Hardware limit
Occupancy                                   100.0%                   Percentage of max threads active
```

Figure 9. Comprehensive Report Snapshot: Memory-Traffic Reduction and GPU Occupancy Analysis (NVIDIA L4)

```
Warp-Level Reductions:
------------------------------------------------------------------------------------------------
Operation                           Method                          Efficiency
------------------------------------------------------------------------------------------------
Max Reduction                       Warp shuffle (__shfl_sync)      O(log2(32)) = 5 steps
Sum Reduction                       Warp shuffle (__shfl_sync)      O(log2(32)) = 5 steps
Block-Level Reduction               Shared memory + warp shuffle    Two-level reduction

Latency Hiding Analysis:
------------------------------------------------------------------------------------------------
Aspect                                      Benefit
------------------------------------------------------------------------------------------------
High Occupancy                              More warps hide memory latency
Warp-Level Reductions                       Fast in-warp communication (no shared mem)
Coalesced Memory Access                     Efficient global memory bandwidth
Shared Memory for Reductions                Fast on-chip memory for intermediates

================================================================================================

Key Observations:
  • Warp-level reductions use efficient shuffle operations (no shared memory)
  • High occupancy enables better latency hiding through context switching
  • Coalesced memory access patterns maximize memory bandwidth
  • Shared memory used only for block-level reductions (minimal overhead)
```

Figure 10. Warp-Level Reduction and Latency-Hiding Analysis for the Fused Softmax Kernel

Figure 10 explains the core GPU-parallel techniques used by the fused kernel. It shows that both the max reduction and sum reduction are performed using warp shuffle intrinsics (__shfl_sync), which enable fast intra-warp communication without shared memory and complete in a small number of log2(warp_size) steps. For larger block-wide reductions, the design uses a two-level strategy: first reduce within each warp, then combine warp results using a small amount of shared memory plus a final warp reduction. The figure also connects these design choices to performance principles: high occupancy helps hide memory latency, coalesced memory access improves effective bandwidth, and shared memory is used only for minimal intermediate aggregation to keep overhead low.

Overall, the fused softmax integration delivers **near-parity** generation performance versus the baseline across the three prompts. Tokens/sec stays in the same ~179–182 range, and latency/token stays around ~5.5 ms. The reported "Average Tokens/sec Improvement: **1.00x**" reflects that any gains from optimizing masked softmax are mostly diluted by the rest of the GPT-2 compute.

Prompt-by-prompt:

- **"The future of artificial intelligence"**
  - Baseline: **179.92 tok/s**, **5.558 ms/token**, **p50 0.528 s**
  - Fused: **178.70 tok/s**, **5.596 ms/token**, **p50 0.530 s**
  - Interpretation: fused is slightly slower here (about **0.7%** lower throughput). This difference is small enough to be within normal run-to-run noise on Colab.

- **"In a world where technology"**
  - Baseline: **182.31 tok/s**, **5.485 ms/token**, **p50 0.521 s**
  - Fused: **181.56 tok/s**, **5.508 ms/token**, **p50 0.522 s**
  - Interpretation: again essentially the same; fused is lower by about **0.4%**.

- **"The quick brown fox"**
  - Baseline: **179.10 tok/s**, **5.583 ms/token**, **p50 0.536 s**
  - Fused: **182.10 tok/s**, **5.492 ms/token**, **p50 0.527 s**
  - Interpretation: fused is faster here (about **1.7%** higher throughput), with correspondingly lower ms/token.

Even though the fused kernel can be much faster in the isolated masked-softmax microbenchmark, end-to-end GPT-2 generation is not dominated by softmax. Most time is spent in other GPU-heavy parts (matrix multiplications in attention/MLP blocks, memory movement, layer norms, etc.), so speeding up just softmax often produces only a small overall change. The small +/− swings across prompts are consistent with measurement noise and prompt-dependent behavior rather than a single dominant bottleneck.

```
================================================================================================================
Performance Evaluations
================================================================================================================

1. Microbenchmark Results (Kernel-Level Performance)
----------------------------------------------------------------------------------------------------------------
Batch    SeqLen     Mask       Type         p50(ms)      p5(ms)       p95(ms)      GB/s         Speedup
----------------------------------------------------------------------------------------------------------------
2. GPT-2 End-to-End Results (Application-Level Performance)
----------------------------------------------------------------------------------------------------------------
Type         Prompt                                Tokens/sec      Latency/Token(ms)    p50(s)
----------------------------------------------------------------------------------------------------------------
Baseline     The future of artificial intellig   179.92          5.558                0.528
Fused        The future of artificial intellig   178.70          5.596                0.530
Speedup                                          1.01          x

Baseline     In a world where technology         182.31          5.485                0.521
Fused        In a world where technology         181.56          5.508                0.522
Speedup                                          1.00          x

Baseline     The quick brown fox                 179.10          5.583                0.536
Fused        The quick brown fox                 182.10          5.492                0.527
Speedup                                          0.98          x

GPT-2 Summary:
  Average Tokens/sec Improvement: 1.00x
```

Figure 11. Performance Evaluation Summary: Microbenchmark and GPT-2 End-to-End Results

Silicon Area:
This project evaluates software-level GPU kernel implementations and does not introduce new hardware structures, so silicon area in the ASIC sense is unchanged. The closest relevant "cost" on existing GPU hardware is on-chip resource usage per kernel launch, such as registers and shared memory, which can affect occupancy and therefore performance. Accordingly, the report discusses shared-memory usage and occupancy as resource proxies rather than physical area.

Observation:
1. Memory Hierarchy Optimization:
   • Fused kernel reduces global memory traffic by ~75%
   • Single-pass design eliminates intermediate memory writes
   • Shared memory used efficiently for reductions

2. Parallel Processor Utilization:
   • Warp-level reductions provide efficient in-warp communication
   • High occupancy enables effective latency hiding
   • Coalesced memory access maximizes bandwidth utilization

3. Performance Improvements:
   • Kernel-level: Significant speedup in softmax computation
   • Application-level: Improved tokens/sec for GPT-2 generation
   • Consistent performance across different sequence lengths

Result:

Table 1. Microbenchmark results (kernel-level)

Batch	SeqLen	Mask	Baseline p50 (ms)	Baseline p5 (ms)	Baseline p95 (ms)	Baseline GB/s	Fused p50 (ms)	Fused p5 (ms)	Fused p95 (ms)	Fused GB/s	Speedup (Baseline/Fused)
1	512	none	0.037	0.036	0.059	167.30	0.030	0.029	0.039	62.80	1.27x
1	512	causal	0.141	0.133	0.163	47.28	0.030	0.029	0.037	63.71	4.78x
1	1024	none	0.037	0.036	0.046	709.32	0.030	0.029	0.041	251.22	1.24x
1	1024	causal	0.139	0.135	0.159	190.62	0.030	0.029	0.041	250.97	4.65x

Microbenchmark summary: Average speedup = 2.99x; Average bandwidth improvement = 0.85x.

Table 2. Memory traffic comparison (unfused vs fused)

Batch	SeqLen	Unfused bytes moved (4 passes)	Fused bytes moved (1 pass)	Reduction
1	512	7,342,080	2,097,152	71.4%
1	1024	29,364,224	8,388,608	71.4%

Table 3. GPT-2 end-to-end generation results (application-level)

Prompt (truncated)	Baseline tokens/sec	Fused tokens/sec	Baseline latency/token (ms)	Fused latency/token (ms)	Baseline p50 (s)	Fused p50 (s)	Throughput ratio (Baseline/Fused)
The future of artificial intellig...	179.92	178.70	5.558	5.596	0.528	0.530	1.01x
In a world where	182.31	181.56	5.485	5.508	0.521	0.522	1.00x

technology							
The quick brown fox	**179.10**	**182.10**	**5.583**	**5.492**	**0.536**	**0.527**	**0.98x**

GPT-2 summary: Average tokens/sec improvement = 1.00x

Table 4. GPU and kernel configuration summary

Item	Value
GPU	**NVIDIA L4**
Multiprocessors	**58**
Max threads per multiprocessor	**1536**
Block size (threads)	**256**
Warps per block	**8**
Shared memory per block	**256 bytes**
Blocks per multiprocessor	**6**
Active threads per multiprocessor	**1536**
Occupancy	**100.0%**

Conclusion:
A fused CUDA kernel for scaled masked softmax was implemented and exposed as a PyTorch CUDA extension, combining scaling, optional causal masking, and numerically stable softmax into a single kernel launch. Correctness was validated by matching the baseline output in the provided tests.

Performance results show a clear kernel-level benefit: the fused implementation achieves 1.27× (seq=512, no mask) and 1.24× (seq=1024, no mask) speedup, and much larger gains under causal masking at 4.78× (seq=512) and 4.65× (seq=1024), with an overall average microbenchmark speedup of 2.99×.

In end-to-end GPT-2 text generation, throughput stays essentially unchanged across the tested prompts, with tokens/sec in the same range for baseline and fused and an average tokens/sec improvement of 1.00×.