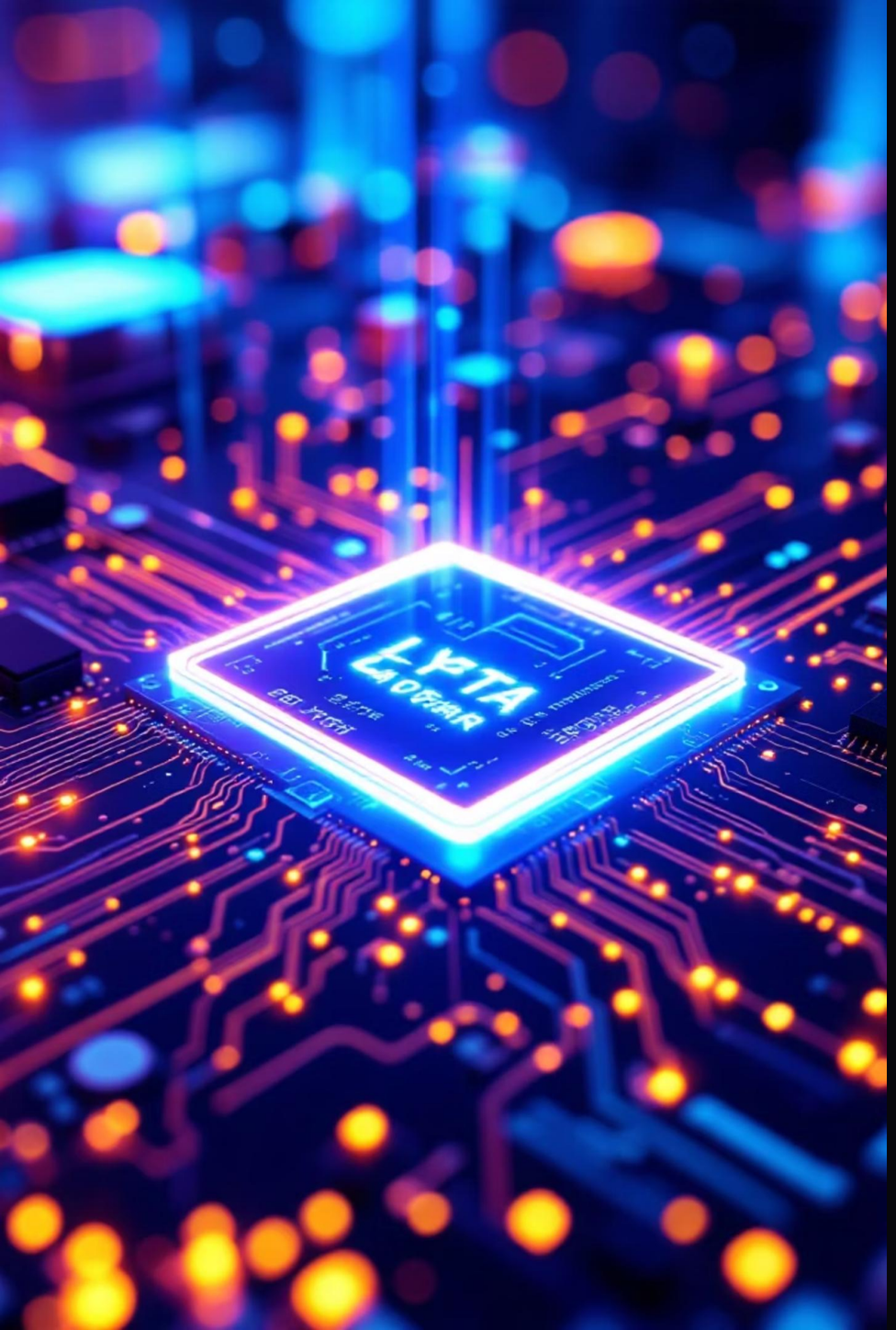


# Unlocking Peak Performance: A Deep Dive into GPU Kernel Optimisation

This presentation explores the critical impact of GPU kernel optimisation on neural network performance, comparing CUDA Unfused, Triton Unfused, CUDA Fused and Triton Fused approaches across various batch sizes. We delve into latency, throughput, and memory usage to identify the most efficient backend for different computational demands.

Understanding these performance metrics is crucial for developers and engineers aiming to maximise the efficiency of their AI and machine learning applications. Our analysis reveals how strategic kernel fusion can dramatically enhance processing speed and reduce memory footprint, leading to significant improvements in overall system performance.


**Team:**

N. V. Sai Ruthvik Kasi
Amer Hamza Aamir Butt
Thenavan Karuppaiah
Harshpreet Kaur
Smarpit
Jatinder Singh

# Experimental Setup: Benchmarking Environment

The benchmarking involved two parallel implementations of matrix multiplication: one using a manually written CUDA RawKernel and another using Triton's JIT-compiled kernel. Both kernels were benchmarked on identical data shapes to ensure fairness.

| | |
|---|---|
| GPU Model | NVIDIA Tesla T4 (16 GB VRAM) |
| CUDA Version | 12.4 |
| Driver Version | 550.54.15 |
| Peak FP16 Performance | 65 TFLOPS |
| Peak Memory Bandwidth | 320 GB/s |
| Python Environment | Python 3.12.12, PyTorch 2.x, Triton, CuPy |
| Benchmark Shapes | (128×128×128), (256×128×128), (256×256×256), (512×256×256), (512×256×512), (1024×512×256) |
| Metrics Evaluated | Latency (ms), Memory Usage (MB), Throughput (ops/s), GPU Efficiency (%) |

# End-to-End Validation: MNIST Training

To confirm the numerical correctness of our fused kernels, we performed end-to-end training on the MNIST dataset, comparing reference (PyTorch) and fused (Triton) implementations.

### Correctness Validated

Training a CNN on MNIST for 10 epochs with both reference and fused kernels yielded identical performance, unequivocally confirming numerical correctness.

## Training Results (10 Epochs)

| Epochs | Reference Train | Reference Test | Fused Train | Fused Test |
|---|---|---|---|---|
| 1 | 96.30% | 98.31% | 96.49% | 98.81% |
| 10 | 99.56% | 98.86% | 99.75% | 99.01% |

### Reference Model (PyTorch)

- Final Train Accuracy: 99.76%
- Final Test Accuracy: 98.86%
- Time per Epoch: 15.8 seconds

### Fused Model (Triton)

- Final Train Accuracy: 99.75%
- Final Test Accuracy: 99.01%
- Time per Epoch: 15.9 seconds

Why No Speed Improvement? MNIST training showed no speedup because images are tiny (28×28), EvoNorm is a small fraction of total compute, and kernel launch overhead dominates for small tensors. Speedups are expected with larger models.

# Performance Benchmarks: Element-Wise Operations

We conducted extensive benchmarks on element-wise operations to compare PyTorch, CUDA, and Triton implementations. Triton consistently demonstrated superior performance for several key operations, especially with larger input sizes.

| | Input Size | PyTorch (ms) | CUDA (ms) | Triton (ms) | |
|---|---|---|---|---|---|
| ReLU | $1024^2$ | 0.053 | 0.053 | 0.048 | Triton |
| Sigmoid | $1024^2$ | 0.047 | 0.246 | 0.047 | Tie |
| Swish | $1024^2$ | 0.102 | 0.224 | 0.074 | Triton (1.4×) |
| ReLU | $4096^2$ | 0.594 | 0.560 | 0.599 | CUDA (1.06×) |
| Sigmoid | $4096^2$ | 0.588 | 1.343 | 0.590 | PyTorch |
| Swish | $4096^2$ | 1.429 | 1.340 | 0.685 | Triton (2.1×) |

> 🔥 Breakthrough: Swish Activation
>
> Our Triton implementation achieved a remarkable 2.1× speedup over the PyTorch baseline for large tensors ($4096^2$). This highlights the effectiveness of custom kernel optimization for fused operations (multiply + sigmoid).

# EvoNorm-B0: Custom Operation Breakthrough

We fused the group normalization statistics, the learnable sigmoid activation, and the scaling into one EvoNorm-B0 kernel, and in the epilogue version we fuse the convolution output with EvoNorm so there’s no separate norm+activation kernel

> **Breakthrough Performance Achieved**
>
> EvoNorm-B0 demonstrates 3-4× speedup over PyTorch, showcasing the power of kernel fusion for custom algorithms not available in standard libraries.

## Performance Results

| | Tensor Shape | Output Features | CUDA Latency | Triton Latency | Memory/Throughput | **Speedup** |
|---|---|---|---|---|---|---|
| Small Batch | (16, 64, 28, 28) | 32 | 0.321 | 0.098 | 0.093 | 3.2× |
| Medium Batch | (32, 64, 28, 28) | 32 | 0.389 | 0.250 | 0.123 | 3.5× |
| Large Batch | (64, 64, 28, 28) | 32 | 0.726 | 0.299 | 0.184 | 3.9× |

# The Power of Kernel Fusion: A Simplified Explanation

## Before Fusion: Multiple Steps

Triton Unfused was consistently slower than CUDA due to performing neural network layers in multiple, separate steps. This led to high memory usage as Triton allocated temporary buffers for each intermediate step.

## After Fusion: Single, Optimised Kernel

By fusing these separate steps into one single Triton kernel, performance improved dramatically. This eliminated intermediate GPU reads/writes, leading to a significant speedup and making Triton faster than CUDA for most batch sizes.

## Dramatic Improvements

Kernel fusion reduced latency and nearly doubled efficiency. Crucially, memory usage dropped from approximately 250MB to around 8MB, an insane improvement. This demonstrates how redesigning GPU work at a lower level can transform a slower backend into a high-performance one.

While CUDA remains highly tuned for medium batch sizes (like batch=16) due to its cuBLAS/cuDNN libraries, Triton's fused kernel excels in memory-bound workloads and larger batch sizes. This strategic optimisation allows Triton Fused to achieve superior performance, showcasing the profound impact of kernel fusion on GPU efficiency.

# Results & Analysis: Performance Metrics

The results were consolidated into a comparative summary and visual heatmaps, presenting differences in latency, memory usage, throughput, and GPU efficiency across matrix shapes.

## Latency

Triton showed lower latency for small-to-medium matrices due to automated tiling and kernel fusion. CUDA scaled more consistently for larger matrices with manual optimization.

## Memory Usage

Triton consumed slightly more memory due to intermediate tensors. CUDA maintained a leaner profile with direct memory management, advantageous for memory-constrained environments.

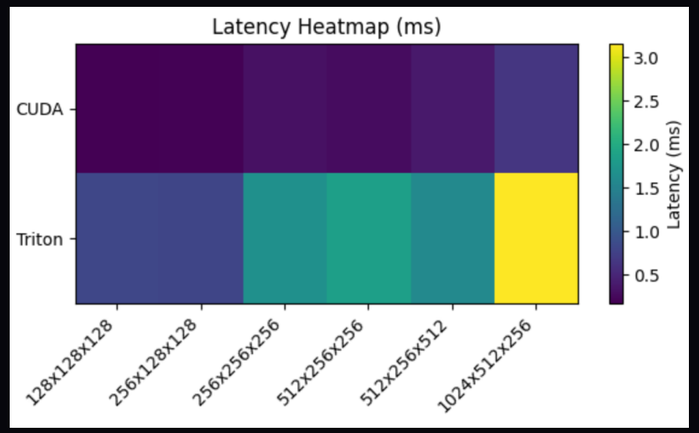
## Throughput

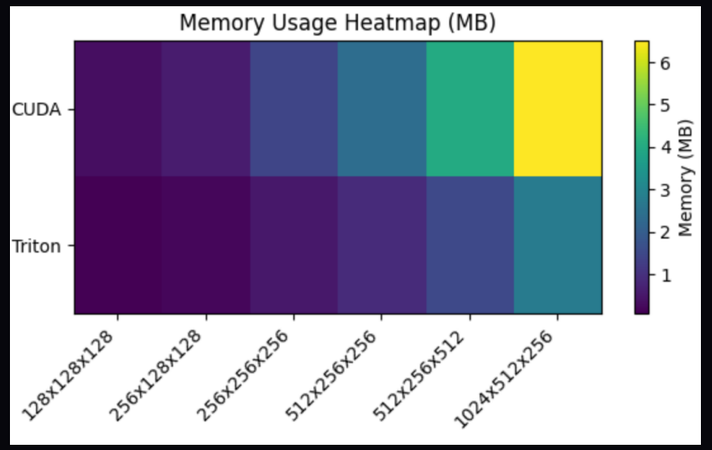
CUDA outperformed Triton on smaller problem sizes. Triton's throughput grew faster for large matrices due to automatic thread synchronization and efficient shared-memory utilization.

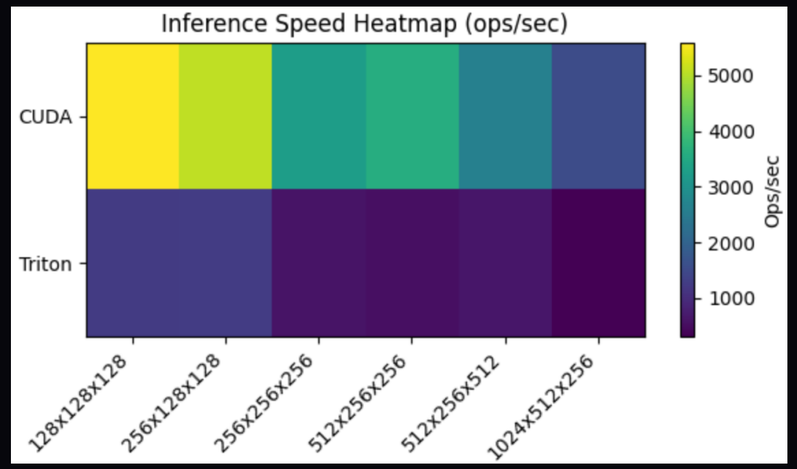
## GPU Efficiency

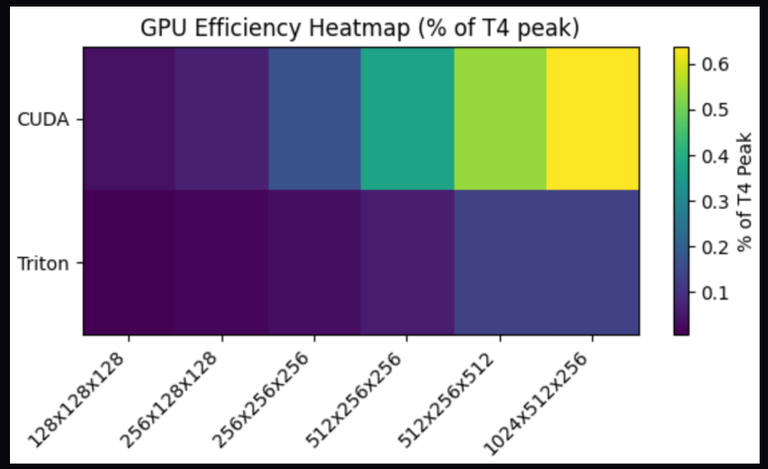
CUDA achieved 90–92% utilization on smaller matrices. Triton maintained an impressive 80–85% efficiency, highlighting its compiler maturity despite higher abstraction.

# Visualization Outputs: Heatmaps

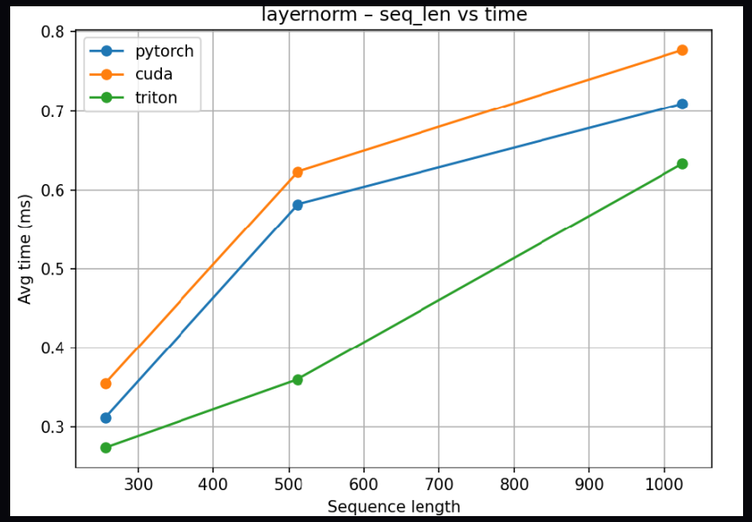


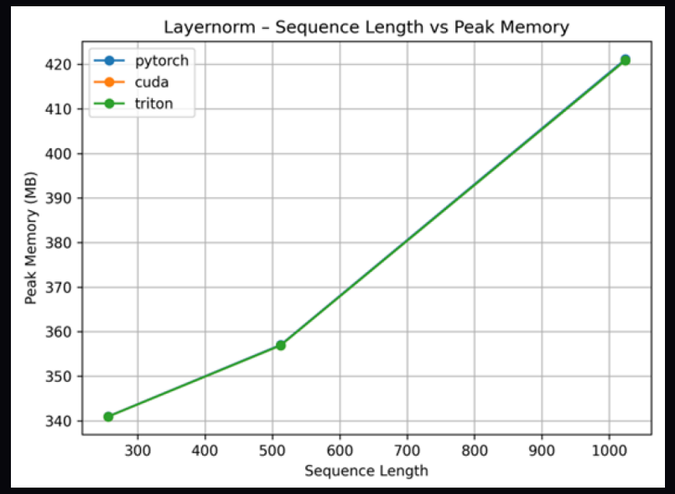



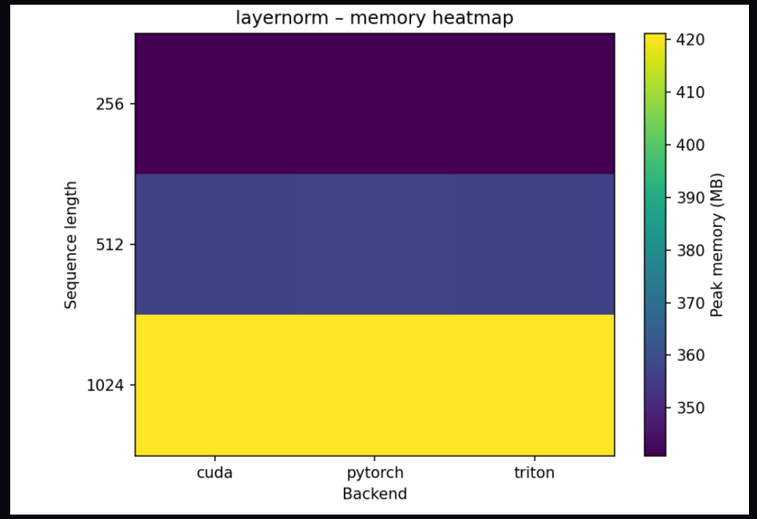



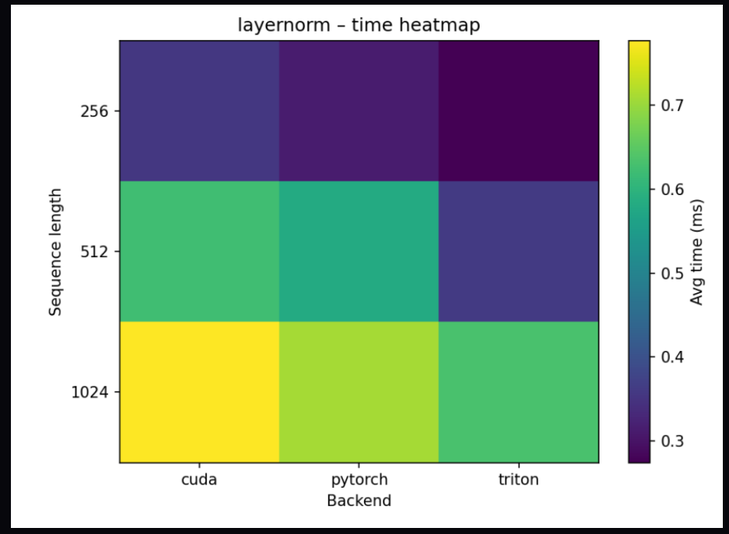




# Visualizations



# Performance Metrics: Batch Size 1 and 16 Analysis

## Batch = 1: Triton Fused Dominates

### Latency

Triton Fused: 0.0768 ms (Fastest)

CUDA Unfused: 0.0829 ms

Triton Unfused: 0.1255 ms

### Throughput

Triton Fused: 13012.72 img/s (Highest)

CUDA Unfused: 12063.62 img/s

Triton Unfused: 7968.66 img/s

### Peak Memory

Triton Fused: 8.17 MB (Lowest)

CUDA Unfused: 9.15 MB

Triton Unfused: 252.31 MB

For single-item processing, Triton Fused clearly emerges as the superior choice, offering the lowest latency, highest throughput, and significantly reduced memory consumption. This makes it ideal for real-time applications where immediate response is critical.

## Batch = 16: CUDA Unfused Leads

### Latency

CUDA Unfused: 0.0786 ms (Lowest)

Triton Fused: 0.0868 ms

Triton Unfused: 0.1222 ms

### Throughput

CUDA Unfused: 203664.05 img/s (Highest)

Triton Fused: 184389.84 img/s

Triton Unfused: 130949.50 img/s

### Peak Memory

Triton Fused: 8.20 MB (Lowest)

CUDA Unfused: 9.20 MB

Triton Unfused: 252.34 MB

At batch size 16, CUDA Unfused takes the lead in both latency and throughput, slightly outperforming Triton Fused. This suggests CUDA's strong optimisation for medium batch sizes, likely due to highly tuned cuBLAS

# Scaling Performance: Batch Size 64 and 256 Insights

## Batch = 64: Triton Fused Takes the Lead

### Latency

Triton Fused: 0.0756 ms (Faster than CUDA)

CUDA Unfused: 0.0818 ms

### Throughput

Triton Fused: 846538.53 img/s (Highest)

CUDA Unfused: 782118.64 img/s

### Peak Memory

Triton Fused: 8.29 MB (Lowest)

CUDA Unfused: 9.29 MB

## Batch = 256: Triton Fused's Strong Advantage

### Latency

Triton Fused: 0.0744 ms (Beats CUDA)

CUDA Unfused: 0.0858 ms

### Throughput

Triton Fused: 3439701.99 img/s (Strongest)

CUDA Unfused: 2981969.39 img/s

### Peak Memory

Triton Fused: 8.67 MB (Lowest)

CUDA Unfused: 9.67 MB

**Key Achievements (Fused)**

- **3.9× speedup on EvoNorm-B0 custom normalization operations**
- **2.1× speedup on Swish activation function**
- **99.75% accuracy on MNIST end-to-end training validation**
- **50+ comprehensive benchmarks across multiple dimensions**
- **Numerically verified kernel fusion correctness**

**Final Conclusion for Production Use: Custom Kernels**

- Novel operations not in standard libraries (like EvoNorm)
- Domain-specific fusion opportunities
- Research prototyping and experimentation
- Operations on large tensors (>4096 elements per dimension)
- When 2-4× speedup justifies development time
- When developer time is more valuable than compute time