## 1) Project Title

A Lightweight CUDA Inference Engine for Transformer Models

## 2) Team Members

-Lyuyongkang Yuan ly2188@nyu.edu

-Junwei Yan

## 3) Goal / Objective

Develop a C++/CUDA based inference engine that accelerates the attention mechanism and related computation blocks within Transformer based large language models (e.g., GPT 2, LLaMA).

The project aims to build a drop in replacement for PyTorch attention layers, achieving end to end inference acceleration while maintaining output parity with native implementations.

## 4) Challenges

Fragmented operator pipeline in PyTorch causes excessive GPU memory traffic and kernel launch overhead during inference.

High Python overhead in `generate` or sequential decoding loops limits real time token throughput.

Memory and latency bottlenecks become prominent as sequence lengths grow, especially for autoregressive models.

\*Balancing precision and performance across different GPU architectures (Ampere / Turing / Volta) without sacrificing numerical stability.

## 5) Approach

Our approach combines low level kernel optimization with framework level integration:

1. Baseline: benchmark PyTorch’s native scaled dot product attention under FP32/FP16.

2. Custom CUDA kernels: implement fused attention kernels (QK, scaling, masking, softmax, SV) using warp level primitives and shared memory.

3. FlashAttention inspired online softmax: eliminate intermediate score matrices to reduce memory I/O.

4. Mixed precision and kernel fusion: leverage FP16/TF32 and epilogue fusion for performance while ensuring stable accumulation in FP32.

5. Integration: expose kernels as a PyTorch C++ extension for seamless use in Transformer based models (GPT 2, LLaMA like).

## 6) Implementation Details

Hardware: GPU environments from local systems and NYU HPC clusters

Software: C++ / CUDA, cuBLAS/cublasLt, PyTorch extension (PyBind11), Nsight Compute/Systems for profiling.

Model Scope: tested within a full Transformer inference pipeline — not only operator level microbenchmarks but also token by token decoding.

## 7) Planned Demo

We will demonstrate:

- Inference level acceleration: compare native PyTorch and our engine on total inference latency and token generation throughput.

-Profiling visualization: Nsight timeline and memory bandwidth charts showing kernel reduction and improved utilization.

-Scalability: results across varying sequence lengths and GPU architectures.

## 8) References

1. Dao et al., 2022 — FlashAttention: Fast and Memory‑Efficient Exact Attention with IO‑Awareness.

- Provided IO‑aware tiling and online softmax; our work extends this to a modular C++ engine compatible with Transformer pipelines.

<https://arxiv.org/abs/2205.14135>

2. Dao et al., 2023 — FlashAttention‑2: Faster Attention with Better Parallelism and Work Partitioning.

- Improved intra‑SM scheduling; our engine adopts similar partitioning principles with a focus on PyTorch integration and inference scalability.

<https://arxiv.org/abs/2307.08691>