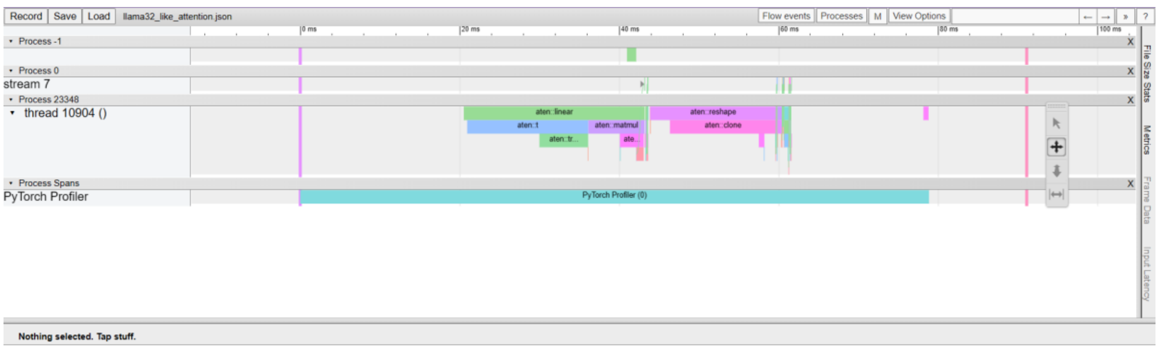
1. Project Title and Team Members
    i. A Fused and IO-Efficient Attention Optimization for Transformer Inference on Llama3
    ii. Team Members
        i. Lyuyongkang Yuan (ly2188@nyu.edu)
        ii. Junwei Yan (jy4831@nyu.edu)
2. Project Milestones (Breakdown of Major Steps)
    i. Understand Llama 3 Attention Implementation
        i. Study Meta's official Llama 3 attention code (ColumnParallelLinear, KV cache design, RoPE)
    ii. Profiling Baseline Attention
        i. Use PyTorch profiler and Nsight to capture kernel timeline for naïve attention
        ii. Identify kernel patterns: Q/K/V GEMM -> QK -> scaling -> mask -> softmax -> P@V -> output GEMM
    iii. Analyze Bottlenecks and Understand FlashAttention
        i. Locate memory bottleneck and excessive global memory traffic from materializing attention scores
        ii. Understand how FlashAttention reduce O(T^2) IO and fuses softmax + matmul
    iv. Implement a Simplified Fused Attention Kernel
        i. Reproduce FlashAttention style tiling and online softmax in CUDA
        ii. Test fused kernel in isolation (not integrated into Llama yet)
    v. Model Level Integration ("Hot swapping" Llama Attention)
        i. Replace Llama-3.2-1B's attention layer with custom fused attention module
        ii. Measure token/s improvement
    vi. Document the Roles of KV Cache and Quantization
        i. Original Llama already support KV cache + mixed precision
        ii. We will explain their latency benefits even if not re implemented from scratch
3. Milestones Completed and Main Results Obtained
    i. Fully analyzed Llama 3 attention logic
        i. We studied Meta's official repo and reproduced the attention flow in a simplified Python/CUDA prototype
        ii. We validated the functional structure:
            Q =WqX, K = WkX, V = WvX -> reshape ->head partition -> repeat_kv -> attention softmax ->projection Wo

ii. Completed baseline profiling
    i. Using PyTorch profiler
        1. Each attention layer launches 15 kernels
        2. Between kernels are micro gaps due to cuLaunchKernel dispath overhead
        3. Attention score matrix (T^2 heads) causes hundreds of MB of global memory reads and writes when seq_len is large
        4. We generated a trace showing:
        5. linear → linear → linear → unsqueeze → expand → transpose→ matmul → div → add → softmax → matmul → transpose → contiguous → linearIdentified bottlenecks
        6. One round attention calculation take ~40ms



4.
    i. Identified bottlenecks
        i. Materializing the full attention matrix causes extreme memory IO (dominant cost at long sequence)
        ii. Launching many small kernels leads to launch overhead accumulation
        iii. HuggingFace uses scaled_dot_product_attention, while Meta's version uses multiple matmuls, both show the same IO issues
        Main insight so far:
        The core bottleneck is IO-bound, not compute bound. Fusing scores + softmax + PV reduces O(T^2) memory movement and eliminates multiple kernel launches.

5. Bottlenecks in Completing Remaining Milestones
    i. Hard to analyze Meta's original attention in isolation

        i. Meta's attention uses: ColumnParallelLinear and RowParallelLinear; Distributed KV cache management; FSDP style tensor sharding assumptions
        ii. Running the module standalone (without full model context) complicates profiling
        iii. We addressed this by: Rebuilding a clean, de-parallelized reference attention module (equivalent to Meta's logic but purely single GPU, using regular Linear ops); Profiling this standalone version to get clean operator-level traces for differential comparison.
    ii. Integrating custom CUDA ops with PyTorch
        i. Setting up C++/CUDA extension + autograd-safe forward path is non trivial
        ii. Ensuring correct tensor shapes, dispatch types, and avoiding unnecessary Python overhead still requires time.
    iii. Online softmax and tiling correctness
        i. FlashAttention involves complex tile wise numerically stable softmax. Maintaining stability in FP16 while matching Llama's behavior is challenging.

6. Contribution of Each Team Member
    i. Lyuyongkang Yuan
        i. Write simplified attention reference module refering to original Llama 3 attention code
        ii. Designed profiler experiments and produced PyTorch traces
        iii. Analyzed baseline bottlenecks and mapped ops to attention phases
    ii. Junwei Yan
        i. Investigated FlashAttention kernel design
        ii. Built profiling environment on local GPUs
        iii. Evaluated memory traffic patterns and documented kernel launch overhead.

7. Planned Work For Final Report
    i. A working fused attention kernel (tile based, online softmax)
    ii. End to end inference bench mark (Pytorch baseline vs fused kernel)
    iii. Analyze of kernel timeline reduction and IO reduction
    iv. Explanation of how KV cache and quantization complement attention fusion