Project Proposal

High-Performance Machine Learning (HPML)

Project Title

Accelerating BERT Fine-Tuning for Question Answering on SQuAD v2 via CUDA-Level Optimization and Profiling

Team Members (2)

- Nikhil Arora (na4063)
- Devanshi Bhavsar(dnb7638)

Goal / Objective

This project aims to fine-tune **BERT-base** on the **SQuAD v2** dataset efficiently while analyzing and improving GPU-level performance. We combine parameter-efficient fine-tuning (LoRA, BitFit) with CUDA-level optimizations to achieve:

- $\geq$1.7$\times$ faster training throughput
- $\geq$40–50% reduction in GPU memory
- $\leq$1% accuracy drop (F1 / EM)
- Measurable GPU kernel improvements using Nsight and `torch.profiler`

Challenges

- Identifying CUDA bottlenecks in transformer layers.
- Maintaining accuracy with reduced precision and fused kernels.
- Fairly comparing performance across PyTorch compile backends.
- Efficient GPU memory use on T4 (16 GB).

Approach / Techniques

(a) Fine-Tuning Methods

Method	Trainable %	Description
Full Fine-Tuning	100%	Baseline BERT fine-tuning in FP16.
LoRA (r=8)	$\approx$2%	Low-rank adapters in attention layers.
BitFit	$< 0.5\%$	Only bias parameters updated.
Layer Freezing	$\approx$10%	Fine-tunes the last 4 encoder layers only.

(b) CUDA-Level Optimizations

- **Mixed Precision (FP16 / TF32):** Enables Tensor Core acceleration.
- **FlashAttention (SDPA):** I/O-efficient attention reducing quadratic memory.
- `torch.compile` **(Inductor / NVFuser):** Kernel fusion and graph-level optimization.
- **8-bit AdamW:** Optimizer state compression via `bitsandbytes`.
- **Gradient Checkpointing:** Reduces activation memory by recomputation.

(c) System-Level Optimizations

- **Dynamic Sequence Length Optimization:** Dynamic padding + bucketing minimize padding overhead; expected 20–30% compute savings.
- **torch.compile Backend Comparison:** Benchmark `inductor`, `nvfuser`, and `aot_eager` backends for kernel fusion, step time, and launch count.
- **Memory Bandwidth Profiling (Nsight Compute):** Measure achieved throughput (GB/s) and Tensor Core utilization in attention/FFN layers.

Implementation Details

Hardware:	T4 16 GB / A100 40 GB.
Frameworks:	PyTorch 2.x, Hugging Face Transformers + PEFT, `bitsandbytes`.
Dataset:	SQuAD v2 (150k Q&A pairs, including unanswerable).
Training:	Seq_len = 384, batch = 8–16, lr = 2e-5, 3 epochs, FP16.
Metrics:	F1 / EM accuracy, throughput (tokens/s), VRAM, kernel count.
Profiling Tools:	`torch.profiler`, Nsight Systems, Nsight Compute.

Timeline (6 Weeks)

Week	Task / Deliverables
1	Baseline full fine-tuning (FP16): F1/EM metrics, VRAM, profiler trace.
2	LoRA / BitFit fine-tuning: accuracy vs trainable parameters.
3	Add AMP + 8-bit AdamW: measure speed and memory.
4	Integrate FlashAttention + `torch.compile`: backend tests + profiler comparison.
5	Dynamic padding + Nsight memory bandwidth profiling.
6	Combine results, build demo, finalize report.

Division of Work

Member	Focus
A	Model fine-tuning (LoRA, BitFit, layer freezing), integration of CUDA optimizations, and validation of results through profiling.
B	Implementation of CUDA-level optimizations (AMP, FlashAttention, `torch.compile`), fine-tuning experiments, and comparative analysis of accuracy and efficiency.
Both	Dataset preprocessing, performance measurement, visualization, and development of the final Gradio demo and report.

Demo Planned

- **Gradio QA Demo:** User inputs paragraph + question → model highlights answer.
- **Performance Dashboard:** Compare baseline vs optimized (speed, VRAM, accuracy).
- **Profiler Visualization:** Nsight timelines showing kernel fusion and bandwidth gains.

References

Paper	Contribution	Our Extension
Devlin et al. (2019) — BERT	Transformer pre-training for NLP	Baseline QA model.
Dao et al. (2022) — FlashAttention	Memory-efficient attention kernel	Integrate + profile at CUDA level.
Dettmers et al. (2022) — 8-bit Optimizers	Memory-efficient training	Integrate 8-bit AdamW with profiling.
PyTorch 2.x Compiler Docs	`torch.compile` backends	Compare compile backends experimentally.