Midterm Checkpoint Report
High-Performance Fine-Tuning of BERT for Question Answering

Nikhil Arora (na4063) Devanshi Bhavsar (dnb7638)

1. Project Overview

We study the efficiency–accuracy trade-offs of fine-tuning `bert-base-uncased` for extractive question answering on SQuAD v2 on a single T4 GPU. Experiments are organized into four families:

- **A-series:** Full-parameter fine-tuning with varying attention and batching (baseline, SDPA, `torch.compile`, dynamic padding/bucketing).
- **B-series:** Parameter-efficient fine-tuning (LoRA, BitFit, partial freezing) on top of the optimized A3 baseline.
- **C-series:** Memory-focused techniques (e.g., 8-bit AdamW, gradient checkpointing).
- **D-series:** `torch.compile` backend comparisons on selected A/B configurations.

All experiments build on the Hugging Face `run_qa.py` BERT QA example with the original data pipeline and base hyperparameters. This midterm checkpoint reports A1–A4 and B1 plus initial profiling; the remaining B-, C-, and D-series experiments are planned for the second half of the project.

2. Project Milestones and Status

2.1 Project milestones (from proposal and status)

- M1: Baseline BERT-base SQuAD v2 setup and logging (A1) – **completed**.
- M2: Kernel / compiler optimizations: SDPA, `torch.compile`, dynamic padding/bucketing (A2–A4) – **completed**, A4 bug fix pending.
- M3: PEFT experiments: LoRA, BitFit, partial freezing (B1–B3) – **B1 completed**, B2/B3 pending.
- M4: Memory-focused experiments: 8-bit AdamW and gradient checkpointing (C1–C2) – **pending**.
- M5: Backend comparison for `torch.compile` (D1) – **pending**.
- M6: Final consolidated analysis and write-up – **pending**.

2.2 Milestones completed so far (implemented configurations)

We have implemented and run the following configurations on NYU GPU nodes (T4, FP16):

- **A1 (Baseline):** Full fine-tuning, standard BERT attention, padding to max length, no compiler.
- **A2 (SDPA only):** Replace BERT attention with PyTorch SDPA (`attn_implementation="sdpa"`), full fine-tuning.
- **A3 (SDPA + `torch.compile`):** A2 plus `torch.compile` with `inductor` backend.
- **A4 (Dynamic padding + bucketing):** A3 plus dynamic padding (`pad_to_max_length=False`) and `group_by_length=True`.
- **B1 (LoRA + SDPA + `torch.compile`):** LoRA on `query/key/value` projections using PEFT, on top of SDPA and `torch.compile`. Only LoRA weights are trainable.

Table 1: Core training metrics for A1–A4 (full fine-tuning) and B1 (LoRA).

Config	Trainable Params	Train Runtime	Steps/s	Samples/s	Eval EM	Eval F1
A1: Baseline (std. attn)	109.3M	2h30m	5.49	43.91	73.21	76.62
A2: SDPA only	109.3M	2h37m	5.25	42.00	73.26	76.56
A3: SDPA + `compile`	109.3M	2h25m	5.66	45.28	73.37	76.89
A4: SDPA + `compile` + bucketing	109.3M	1h36m	8.56	68.51	25.82	26.87
A4-nb: SDPA + `compile` + dyn. padding (no bucket)	109.3M	2h08m	6.45	51.59	73.78	77.23
B1: LoRA + SDPA + `compile`	0.44M	1h34m	8.80	70.42	51.12	54.95

2.3 Core training metrics (3 epochs, full train set)

Table 1 summarizes the main metrics from the Hugging Face Trainer logs. All runs use `bert-base-uncased`, batch size 8, SQuAD v2, and 3 training epochs (131,754 training examples).

Key points so far:

- A1–A3 keep accuracy around F1 $\approx$ 76.5–76.9 with modest throughput differences.
- A4 substantially increases throughput (steps/s and samples/s) but collapses accuracy.
- A4-nb restores A1–A3-level accuracy while still improving throughput, indicating the bug is specific to bucketing rather than dynamic padding.
- B1 reduces trainable parameters from $\approx 1.09 \times 10^8$ to 4.42×10^5 (about 0.4% of the full model), with good speed but lower accuracy than A1–A3.

3. Profiling SDPA + `compile` vs. LoRA

To analyze GPU behavior, we ran short profiling jobs on a reduced subset (train: 2,000 examples, 1 epoch; validation: 1,000 examples; FP16, same model and data pipeline). We wrapped the Hugging Face training loop in a PyTorch `profile` context (CPU+CUDA, with shapes) and used the PyTorch Profiler for kernel-level timing.

Table 2: Profiling comparison of A3 (full fine-tuning) and B1 (LoRA) on 2k training and 1k validation examples.

Config	Trainable Params	Train Runtime (s)	Steps/s	Self CUDA (s)
A3 (SDPA + `compile`)	109.3M	83.40	2.998	35.40
B1 (LoRA + SDPA + `compile`)	0.44M	34.17	7.316	21.83

Observations:

- LoRA (B1) reduces trainable parameters by $\sim 250\times$ and runs about $2.4\times$ faster in steps/s than A3 on the same 2k subset.
- Total CUDA time also drops (21.8 s vs. 35.4 s), indicating real GPU savings beyond Python overhead.
- PyTorch Profiler shows that both A3 and B1 are dominated by matrix multiplication (`aten::mm`, fused GEMM kernels) and SDPA/FMHA kernels. LoRA does not change the dominant kernel types, but reduces overall work in the attention projections and feed-forward layers.

4. Insights So Far

Effect of SDPA and `torch.compile` (A1–A3). Switching from standard attention to SDPA (A2) and then enabling `torch.compile` (A3) yields modest but consistent throughput gains without hurting accuracy:

- A3 increases samples/s vs. A1 (45.3 vs. 43.9) and slightly improves F1 (76.89 vs. 76.62).
- Compile-time overhead is amortized over 3 epochs, leading to a small net runtime improvement.

Dynamic padding and bucketing (A4). Dynamic padding and length-based bucketing (A4) boost throughput (samples/s from 45.3 to 68.5) but collapse EM/F1 to ≈ 26, which we treat as a label-misalignment/batching bug (near-zero HasAns F1) rather than a real speed–quality trade-off. A follow-up run with dynamic padding but *no* bucketing (A4-nb) restores A1–A3-level accuracy (F1 $\approx$ 77.2) while still improving throughput, isolating the issue to bucketing.

LoRA with compiler fusion (B1). LoRA (B1) behaves as expected: profiling shows about 2.4× higher steps/s than A3, lower CUDA time, and updates on only $\sim 0.4\%$ of parameters, but with F1 ≈ 55 vs. ≈ 77 for full fine-tuning. We view this gap as a hyperparameter / training-length issue rather than a fundamental limitation, and closing it (via rank, schedule, and/or more epochs) is a key goal for the second half of the project.

5. Bottlenecks and Open Issues

- **Instrumentation limitations:** Nsight/`nsys` are not available on the cluster modules, so we cannot directly measure SM occupancy or detailed kernel counts. We currently rely on the PyTorch Profiler for kernel timing.
- **Memory metrics:** VRAM usage and optimizer-state memory have not yet been systematically captured; this needs additional lightweight logging.
- **LoRA quality gap:** B1 underperforms full fine-tuning in EM/F1; it requires systematic tuning (rank, LR schedule, and potentially more epochs).
- **Dynamic batching correctness:** The accuracy collapse in A4 strongly suggests a correctness issue in the bucketing/dynamic-padding pipeline (e.g., misaligned labels) that must be fixed.

6. Remaining Milestones and Next Steps

- **Complete PEFT variants (M3):** Run BitFit (B2) and partial layer freezing (B3); tune LoRA rank and learning-rate schedule to reduce the F1/EM gap with A3.
- **Memory experiments (M4, C1–C2):** Integrate 8-bit AdamW and gradient checkpointing on the A3 baseline; log peak VRAM and step-time impact to characterize memory–throughput trade-offs.
- **Backend comparison (M5, D1):** Swap `torch.compile` backend (Inductor vs. NVFuser, as available) and compare step time and kernel mix on the same A3/B1-style configurations.
- **Fix A4 correctness:** Debug the dynamic padding/bucketing pipeline, verify label alignment, and re-run A4 to obtain a "correct but fast" dynamic-batching baseline.
- **Instrumentation improvements (M4/M6):** Add lightweight VRAM logging and refine PyTorch Profiler runs so we can approximate the missing Nsight-style metrics and support the final analysis.

7. Work Contributions

Student A (Nikhil Arora : na4063).

- NYU HPC and `run_qa.py` setup: virtualenv, GPU jobs, dataset caching, baseline sanity runs
- A-series baseline configs: implementation and runs for A1 (std. attn), A2 (SDPA), A3 (SDPA + `compile`)
- Extraction and organization of Trainer metrics for A1–A3; initial SDPA/`compile` throughput vs. accuracy analysis

Student B (Devanshi Bhavsar : dnb7638).

- Dynamic batching configs: implementation and runs for A4 (SDPA + `compile` + bucketing) and A4-nb (dynamic padding only)
- LoRA/PEFT pipeline: B1 setup on BERT self-attention, hyperparameters, all B1 training runs (full and profiling)
- Profiling and results presentation: PyTorch Profiler runs (A3, B1), kernel summaries, table design, and comparison of A1–A4/A4-nb vs. B1

Joint.

- Overall A/B/C/D experiment grid, ablation planning, and selection of configs for profiling
- Debugging and sanity checks: A4 bucketing issue, validation of A4-nb, monitoring training/eval anomalies
- Drafting and editing of the midterm report; planning remaining PEFT, memory, and backend experiments