NLP Project Proposal – LoRA Under the Microscope: Drift, Deployment and Sparsity

Ben Cohen – 323855288, benc6116@gmail.com | Gal Avny – 209548815, galavny13@gmail.com

Low-Rank Adaptation (LoRA) is a lightweight alternative to full fine-tuning: instead of adjusting every weight in a large language model, it inserts two small rank-*r* matrices per layer, so only a few million parameters are updated. Researchers report that LoRA usually matches task accuracy while cutting training cost; however, two practical questions are still open. **First, does LoRA truly keep the model's internal representations close to the original, or does it "drift" just as much as full fine-tuning? Second, when several LoRA adapters are kept side-by-side for multitask inference, what is the latency penalty compared with merging them into the base model?** Answers would help both continual-learning research and engineers who must decide between LoRA and heavier approaches.

We will examine these questions on **Llama-2-1.3B**, the 1.3-billion-parameter open-source language model released by Meta that fits inside a student-cluster GPU. On one sentence-pair classification task (GLUE – MRPC) and one question-answering task (SQuAD v2) we will train two versions of the model: (a) **full fine-tuning** and (b) **LoRA with rank 8**. To quantify representational drift, we will take 1000 validation examples, extract hidden states from every transformer layer before and after training, and compare them with centered-kernel alignment and layer-wise cosine similarity. Smaller change suggests less catastrophic forgetting and easier future transfer.

For deployment, we will load the same fine-tuned checkpoints into **vLLM**, a high-throughput inference engine, and measure tokens per second, 95-th-percentile latency and GPU memory at batch sizes 1 – 16. Two modes will be tested: (i) LoRA adapters merged into the base weights and (ii) adapters left separate and selected on-the-fly. This isolates the cost of the popular "just-in-time" LoRA pattern.

Success is defined as follows: LoRA must stay within 3 percentage points of full-tuning accuracy and either show at least 20 % less representational drift or add no more than 30 % inference overhead. If these conditions hold, LoRA can be recommended as both stable and deployable; if not, we will have clear quantitative limits.

Feasibility and Assumptions

Compute: We'll use Google Colab Pro (self-funded), which provides premium GPUs (typically NVIDIA T4 or P100 with 16 GB VRAM) and extended runtimes, sufficient for our needs:

- **Full Fine-Tuning:** Training Llama-2-1.3B requires ~10-12 GB VRAM, comfortably fitting within Colab Pro's 16 GB.
- **LoRA and Inference:** Less intensive tasks (LoRA, vLLM profiling) easily fit as well.
- **Compute Time:** Approximately 80 hours spread across sessions using persistent Colab notebooks.

Data & Models: Public datasets (GLUE-MRPC, SQuAD v2) and Llama-2-1.3B weights downloaded each session.

Software: Open-source libraries (PyTorch 2, PEFT, vLLM, Weights & Biases) easily installable via pip.

Storage: Persistent storage (~100 GB) via Google Drive integrated with Colab Pro. No paid APIs or human evaluators needed.

Note on Feasibility:
The project's viability hinges on using Llama-2-1.3B, whose size ensures manageable VRAM requirements. Colab Pro's standard GPUs sufficiently handle fine-tuning and inference tasks, maintaining original research questions, evaluation metrics (CKA, cosine similarity, vLLM latency), and success criteria.