Choosing an Efficient Fine-Tuning Strategy for BERT: A Case Study on SST-2 Sentiment Classification

Boyi Zhang, Weifeng Chen, Yunxiang Li
AIAA3102, HKUST(GZ)

***Abstract*—Fine-tuning pre-trained language models is the de-facto approach for downstream NLP tasks. As a small startup serving a movie-review company to mine sentiment from review text, we must respect tight GPU and engineering budgets, making full-parameter fine-tuning of BERT costly. In this project we study parameter-efficient fine-tuning (PEFT) for this setting, using SST-2 as a practical task and bert-base-uncased as the backbone. We build a unified pipeline that supports full fine-tuning, LoRA 16-bit and 8-bit, and 4-bit QLoRA. We compare them in terms of accuracy, training time, and memory usage. Beyond basic benchmarking, we examine learning-rate and LoRA hyperparameter sensitivity, robustness to random seeds and a noise-perturbed "toxic" variant of the dataset, generalization from zero-shot and few-shot to full-data regimes, difficulty-based sampling, and the effect of freezing lower encoder layers. We further study transfer learning by moving SST-2–trained LoRA adapters to MRPC, comparing transferred versus directly trained adapters and varying LoRA target modules, together with an SVD-based analysis of the learned updates. Our experiments show that PEFT methods closely match full fine-tuning on SST-2 while using far fewer trainable parameters, with 16-bit LoRA offering the best speed–memory–performance trade-off in our setup, and transferred adapters provides faster convergence on the target task. We discuss how these findings translate into practical recommendations for resource-constrained classification tasks.**

***Index Terms*—Sentiment analysis, parameter-efficient fine-tuning, LoRA, QLoRA, data-centric learning, transfer learning.**

I. Introduction

Online review platforms increasingly want to understand why users liked or disliked a movie, not just how many stars they assigned. Star ratings might be noisy and inconsistent, while the free-form review text could contain the real sentiment signal. Our startup serves a movie-review company to build an automatic sentiment classifier over review text, so that they can monitor user satisfaction, surface representative examples, and support downstream analytics.

In practice, this startup operates under strict resource constraints: it has access to only a single commodity GPU or shared cloud instances, and it must consider training time limits. Full-parameter fine-tuning of a pre-trained encoder such as BERT is feasible, but it consumes substantial memory and slows iteration. Parameter-efficient fine-tuning (PEFT) methods such as LoRA and QLoRA promise to reduce the number of trainable parameters and the memory footprint by freezing most of the backbone and learning small task-specific adapters instead [1]. The key question is whether these methods can deliver competitive performance on sentiment analysis while respecting the startup's deployment constraints.

We investigate this question on a controlled but realistic problem: fine-tuning bert-base-uncased on the SST-2 binary sentiment classification benchmark. We implement a unified and reproducible training pipeline that supports full fine-tuning, LoRA with 16-bit and 8-bit backbones, and 4-bit QLoRA under the same codebase. The pipeline standardizes data preprocessing, tokenization, optimization settings, logging, and evaluation, and it also supports constructing a noisy "toxic" variant of the dataset through character deletions, emoji insertions, and word shuffling.

Our study is organized around several concrete questions that matter for this startup scenario. How do full fine-tuning, LoRA, and QLoRA compare in efficiency and accuracy on SST-2? How sensitive are they to learning rate and LoRA hyperparameters, and how robust are they to random seeds and simple text corruption? How does performance scale from zero-shot through few-shot to full-data regimes, and what happens when we change the training distribution using label-conditioned, difficulty-based sampling? How much model capacity is actually needed for this task, and what is the effect of freezing lower encoder layers on overfitting and generalization? Finally, if the startup later needs to support new tasks, can SST-2–trained adapters be transferred to another GLUE task such as MRPC, and how does the design of LoRA target modules affect both the transfer accuracy and the structure of the learned updates? Addressing these questions allows us to move from raw benchmark numbers to practical guidance for building sentiment models under real-world constraints.

Our contributions are fourfold. (1) We implement a clean, configurable, and reproducible pipeline for fine-tuning bert-base-uncased on SST-2 with full and parameter-efficient methods, including a Gradio demo and tooling for constructing noisy "toxic" datasets. (2) We provide an extensive empirical comparison of full fine-tuning, LoRA, and QLoRA, analyzing efficiency–performance trade-offs, hyperparameter sensitivity, robustness, data scaling, and capacity effects. (3) We study cross-task transfer by moving SST-2–trained LoRA adapters to MRPC, comparing transferred versus directly trained adapters, and varying LoRA target modules while analyzing their update matrices via SVD-based stable rank. (4) We complement these model-centric results with a data-centric study, and we distill practical guidelines for choosing a fine-tuning strategy in resource-constrained sentiment analysis scenarios, together with limitations and directions for future work.

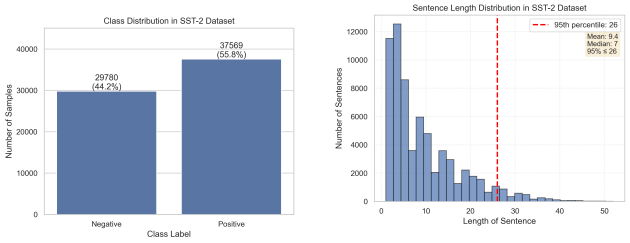

Fig. 1: *2D and 3D t-SNE Projection of Dry Bean Dataset*

II. Methodology and System Design

A. Dataset and Preprocessing Pipeline

We use the GLUE implementation of SST-2 provided by the Hugging Face datasets library [2]. The dataset consists of short movie-review sentences with binary sentiment labels. Figure 1 shows our exploration of the dataset. The label distribution is only mildly imbalanced, and sentence lengths are short (mean $\approx$ 20 tokens), which suggests that the classification task is relatively well-structured and stable. The dataset's cleanliness ultimately explains several findings later, particularly the robustness of models to injected noise and the rapid convergence in few-shot settings.

We follow the standard GLUE setup: the training split (67349 samples) is used for model fitting, the validation split (872 samples) serves as our test set (the original test set has no public labels and is reserved for official leaderboard submissions). All models share the same preprocessing and tokenization pipeline. We load the SST-2 dataset via the datasets library and apply an AutoTokenizer initialized given the backbone model name. Each sentence is tokenized with truncation to a fixed maximum length, and the resulting datasets are converted to the GLUE-style train and validation splits with the labels column. Padding is handled at batch time using a data collator so that full fine-tuning and all PEFT variants see identical tokenized inputs.

To study robustness to input corruption, we construct a noise-perturbed variant of the training data using the transformations implemented in data.py. With a small probability, each original sentence is used to generate additional noisy copies by inserting extra symbols or emojis, introducing character substitutions that mimic typos, removing punctuation and randomly dropping words, and optionally prefixing informal filler phrases, while keeping the original sentiment label. The noisy samples are concatenated with the clean training set, whereas the validation split remain clean.

B. Model Architectures and Fine-Tuning Strategies

Our backbone is bert-base-uncased initialized as a sequence classification model with a two-way linear head on top of the [CLS] representation. We consider three families of fine-tuning strategies that share this architecture.

1) **Full fine-tuning**: In the full fine-tuning baseline, all Transformer and classifier parameters are marked trainable. We use mixed-precision training where appropriate to boost training.
2) **LoRA**: For LoRA experiments, we wrap the backbone with a PEFT adapter using a configurable Lora Config. Low-rank update matrices are inserted into selected linear layers of the Transformer, such as attention projection and feed-forward layers. Only the LoRA parameters and the classification head are updated; the original backbone weights remain frozen [3]. We use two precision settings in practice: a 16-bit variant where the backbone runs in FP16 on Colab T4 GPU, and an 8-bit variant where the frozen backbone weights are quantized to 8 bits while LoRA parameters stay in 16-bit.
3) **QLoRA**: QLoRA combines 4-bit quantization of the frozen backbone with LoRA adapters on top. In our implementation, the backbone weights are loaded in a 4-bit quantized format using the quantization utilities, while LoRA parameters and the classifier head are kept in 16-bit precision. As in the LoRA setting, only the adapters and head are updated during training [4]. This configuration substantially reduces memory footprint at the cost of additional computation overhead from quantization and dequantization.

All methods share the same classifier head architecture and tokenization pipeline so that differences in performance can be attributed to the fine-tuning strategy and precision choices rather than to model structure.

C. Training Setup and Reproducible Codebase

Our experiments are organized around a small, configuration driven codebase. The main entry point is run.ipynb, which loads global settings from config.py, selects a fine-tuning strategy, and then executes helper functions from utils.py. The config.py file centralizes model names, precision modes, learning rates, LoRA ranks, dataset variants, and output paths so that different experiments can be easily reproduced .

The construction of noise-augmented "toxic" training sets is further implemented in data.py. The file util2.py composes time-consuming high-level experiments such as transfer leraning functions and small sweeps over LoRA modules.

Each experiment logs validation accuracy and F1, together with configuration metadata, to CSV files. The notebook plot.ipynb then consumes these logs to generate the figures summarized later in the report.

III. Experiments and Results

A. Efficiency–Performance Trade-offs

1) Experimental setup We compare four fine-tuning strategies on SST-2 using the same bert-base-uncased backbone and identical training hyperparameters. The methods are shown in Table I.

Method	Backbone precision	Quantization level	Trainable parameters
Full fine-tuning	Mixed FP16/FP32	None (full precision)	**(100%)**
LoRA (16-bit)	16-bit (BF16/FP16)	None (full precision)	**(0.8%)**
LoRA (8-bit)	8-bit backbone	8-bit quantization	**(0.8%)**
QLoRA (4-bit)	4-bit backbone	4-bit quantization	**(1.3%)**

TABLE I: *Fine-tuning strategies and backbone precision configurations.*

For each method we measure validation accuracy, wall-clock training time, and peak GPU memory usage on a single T4 GPU. Training time and memory are normalized so that full fine-tuning corresponds to 1.0 on each axis.

2) Results The comparison is summarized in Figure 2. Full fine-tuning in default FP32 precision reaches an accuracy of 0.9220, with 1454s training time and 6450 MiB peak VRAM (baseline 100% time and 100% memory). LoRA (16-bit) attains similar accuracy, while reducing training time to 25.3% of full FT and memory to 48.6%. LoRA (8-bit) cuts memory more aggressively to 20.3% but training time is similar to full fine-tuning. QLoRA (4-bit) achieves the highest accuracy, with the lowest peak memory (17.4%) and a moderate training time (51.0%).

Because SST-2 is a relatively clean and simple benchmark, all PEFT variants match the accuracy of full fine-tuning. LoRA (16-bit) offers the best accuracy–memory trade-off with roughly half the memory and third percent training time. LoRA (8-bit) and QLoRA (4-bit) minimizes VRAM usage but cost more training time due to Quantization/dequantization overhead. Under the startup-style resource constraints, we therefore adopt LoRA (16-bit) as our default method for later experiments as it remains friendly to both speed and GPU usage

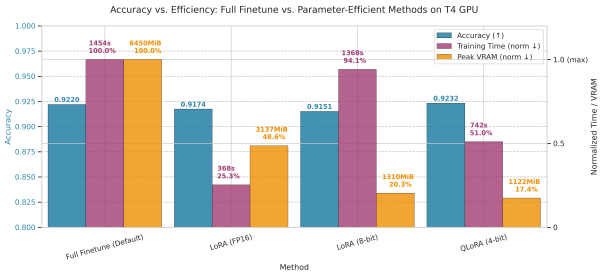

Fig. 2: *Accuracy vs. Efficiency: Full Finetune vs. Parameter-Efficient Methods on T4*

B. Hyperparameters: Learning Rate, LoRA Rank

1) Experimental setup We study hyperparameter sensitivity using the LoRA (16-bit) configuration. For the learning-rate sweep, we vary the LR over a logarithmic grid from 10^{-5} to 10^{-4}. For the rank sweep, we vary the LoRA rank r from small (e.g., 4) to relatively large values (e.g., 64). We also experimented with different choices of LoRA target modules (attention-only vs. FFN-only vs. full), and found that in-domain SST-2 performance is relatively insensitive to this choice; we therefore focus on a reasonable default and study target modules mainly through cross-task transfer in Subection Transfer Learning and Target Modules.

2) Learning rate results Figure 3 shows that very small learning rates lead to clear undertraining: accuracy and F1 start below 0.88 at 10^{-5}. Performance improves steadily as we increase the learning rate, peaking around the 10^{-4} scale, where both metrics exceed 0.91. Beyond this point, further increases bring little benefit, entering a plateau region. We therefore adopt a learning rate at 10^{-4} as the default.

3) LoRA rank results Figure 4 reports the impact of the LoRA rank r. With a very small rank (e.g., r=4), the adapter has limited capacity and validation performance is noticeably lower. Increasing the rank to 16 steadily improves accuracy and F1, with the best scores obtained at r=16. Further increasing the rank to 32 or 64 does not yield additional gains and slightly degrades performance, likely due to mild overfitting on this small dataset while also increasing the number of trainable parameters. These trends suggest that a moderate rank around 16 is a good compromise between expressiveness and efficiency, and we use this setting in our subsequent LoRA experiments.

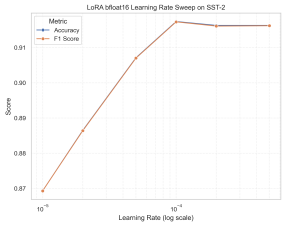

Fig. 3: *LoRA 16-bit Learning Rate Sweep*

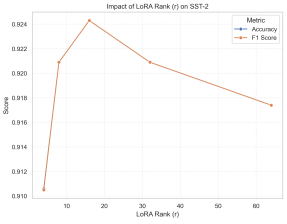

Fig. 4: *LoRA 16-bit rank Sweep*

C. Robustness to Randomness and Input Noise

1) Experimental setup We evaluate robustness using our default LoRA (16-bit) configuration with the best learning rate and rank from the previous section. To study randomness, we repeat training on the full training set with four different seeds and record validation accuracy for each run. For each seed we also compute a 95% bootstrap confidence interval by resampling validation predictions. We use 1,000 bootstrap samples, which is slightly more than the size of the SST-2 validation set to ensure stable interval estimation.

To study robustness to input noise, we compare two training conditions: Clean and Noise-perturbed. We build the robust dataset from data.py, which augments the clean training set with noisy copies generated via emoji insertion, character substitutions, word deletion, and filler phrases. This is applied to 20% of training examples while keeping labels unchanged. In both conditions we evaluate on the same clean validation set and report accuracy and F1.

2) Results Figure 5 (left) shows accuracy and 95% bootstrap confidence intervals across seeds; Figure 5 (right) summarizes the distribution. All runs fall in a narrow band around 0.91-0.92 accuracy, and the confidence intervals for different seeds almost completely overlap. This indicates that training is stable with respect to initialization and data shuffling, and that seed choice does not materially affect conclusions on SST-2 full train split.

Table II reports the effect of noise augmentation. The degradation under noise-perturbed condition is less than 0.05 in both metrics. This suggests that the model is quite robust to the moderate character-level and token-level corruptions we introduce. For the startup scenario, this means that mild typos, emojis, and informal phrasing in user reviews are unlikely to significantly harm sentiment accuracy.

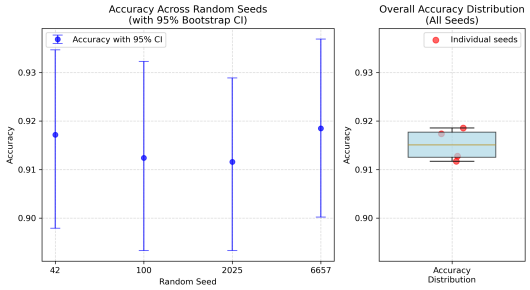

Fig. 5: *Accuracy Robustness to Random Seeds*

Dataset Condition	Accuracy	F1 Score
Clean (No Noise)	0.9186	0.9186
Noise Perturbed	0.9140	0.9139

TABLE II: *Robustness of the Model Under Noise Perturbations*

D. Scaling & Data-Centric Training

1) Generalization scaling We study how performance scales with the number of labeled examples ("shot size") for our default BERT–LoRA (16-bit) model, and compare it to full-parameter fine-tuning of BERT and GPT-2. For each shot size we sample a label-balanced subset of the SST-2 training set, train from scratch on that subset, and evaluate accuracy with bootstrap confidence intervals on the full validation set. Shot sizes range from zero-shot up to the full training set.

The curves in Figure 6 show a typical scaling pattern. Since the pre-trained BERT encoder does not include a sentiment-specific classification head, the zero-shot baseline effectively behaves like random guessing, with accuracy hovering around 50%. All methods start near 0.5 accuracy at 10–50 shots, then improve rapidly as the shot size increases. Around 1000 shots, both BERT–LoRA and the full fine-tuning baselines already approach their final full-data performance, and beyond a few thousand examples the gains are marginal.

The LoRA curve is slightly worse than full fine-tuning in the small-data regime (100-1000 samples), but the gap closes after 1000 examples are available. Furthermore, the LoRA curve exhibits noticeably wider confidence intervals than full fine-tuning in the low-shot regime. Since we freeze the backbone and only train a small set of randomly initialized low-rank adapters, the final solution becomes more sensitive to the particular few-shot subset and random seed. In contrast, full fine-tuning updates all parameters, which tends to average out such randomness and yields more concentrated outcomes. As the shot size increases, the variance of LoRA steadily shrinks and becomes comparable to full fine-tuning, indicating that this instability is mainly a data scarcity effect rather than a fundamental issue with the method. For the startup scenario, this suggests that on an easy benchmark like SST-2, collecting and curating roughly 1000 high-quality labeled reviews may already be sufficient, and blindly scaling annotation volume beyond that brings quickly diminishing returns.

To contextualize the few-shot performance of our BERT-based models, we additionally include GPT-2 (decoder-only) as a full fine-tuning baseline. The motivation is to examine whether architectural differences lead to distinct scaling behaviors as the number of examples increases. Across small shot sizes (10–200 examples), BERT exhibits better sample efficiency than GPT-2. This aligns with the pretraining objectives: masked language modeling provides bidirectional contextual representations that are naturally suited for sentence-level classification. Whereas GPT-2 must adapt its autoregressive representations to a discriminative task, which typically requires more labeled data. However, as the shot size grows beyond 1000 examples, this gap largely disappears. GPT-2 catches up and ultimately reaches higher accuracy, suggesting that with sufficient data, architectural inductive biases become less decisive and both encoder and decoder only models converge toward similar performance levels.

2) Data-centric difficulty-based sampling To go beyond uniform random sampling, we adopt a data-centric approach based on per-example difficulty. We first train a full-data BERT teacher model and compute the loss of each training example, sorting the SST-2 training set from "easiest" (lowest loss) to "hardest" (highest loss), separately within each label. Using this difficulty ordering, we construct several label-balanced 1000-shot subsets and train BERT–LoRA models on each of them.

Naïve strategies that concentrate on a single difficulty region perform poorly. We start from a few-shot model trained on a uniformly sampled 1000-shot subset, we then continue fine-tuning on only the easiest 10% or the hardest 10% of the training set (label-balanced). In both cases, the final accuracy and F1 drop below the baseline obtained without this data-centric continuation step. Intuitively, the easiest examples do not provide enough useful information, while the hardest examples are ambiguous or noisy cases that the teacher model cannot even handle. Randomly sampling 1000 examples from only the middle 80% of the difficulty spectrum also underperforms full-distribution random sampling, indicating that hard truncation distorts the original data distribution in a harmful way.

We then design a mixed-difficulty few-shot strategy that aims to preserve the overall distribution while down-weighting extreme cases: 90% of the subset is sampled from the middle 80% region, and the remaining 10% is split evenly between the easiest and hardest 10%. For this experiment, the BERT–LoRA model is trained from scratch on this mixed subset, rather than being initialized from a previous few-shot model. This mixture retains easy "prototype" examples, a large set of medium-difficulty boundary cases, and a controlled number of hard examples. Table III summarizes our data-centric experiment results. The mixed subset yields better accuracy and F1 than uniform random sampling at the same shot size, showing that carefully balancing difficulty levels can be more effective than both naive truncation and purely random selection. However, more principled difficulty measures could further improve data-centric selection, as loss-based difficulty is itself noisy and checkpoint-dependent.

Overall, on SST-2, a small but well-selected subset can

already match full-data performance, suggesting that data quality and selection matter more than blindly increasing sample size.

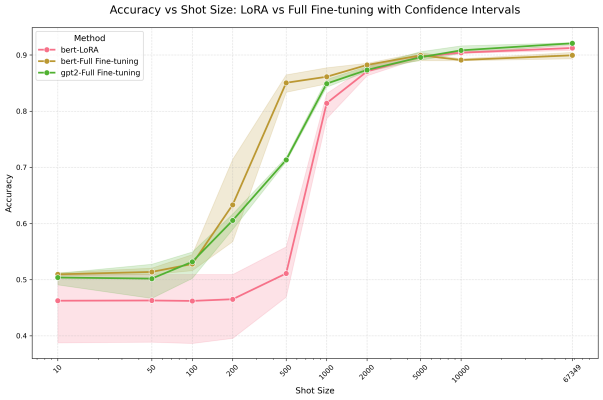

Fig. 6: *Accuracy vs. Shot Size: LoRA vs. Full Finetune with Confidence Intervals*

Setting	Best Accuracy	Best F1 Score
Data-Centric Subset	**0.89335**	**0.89328**
Random Subset	0.86697	0.86695

TABLE III: *Comparison of Data-Centric vs. Random Subset Few-Shot Training*

E. Capacity & Overfitting

1) Experimental setup We explore model capacity and overfitting from two angles. First, we track train vs. validation accuracy of the LoRA 16-bit model over 20 epochs on SST-2. Second, we start from the full fine-tuning baseline and progressively freeze the lowest n encoder layers (keeping the classifier head trainable), measuring both validation accuracy and the fraction of trainable parameters. We conduct the layer-freezing experiment only for full fine-tuning, as in LoRA setup the BERT backbone is already frozen.

2) Overfitting behavior Figure 7 shows that LoRA training accuracy keeps increasing from about 0.89 to above 0.97 as epochs progress, while validation accuracy rises quickly in the first 5–8 epochs and then plateaus around 0.92 with small fluctuations. Additional epochs mainly improve the fit to the training set without yielding consistent gains on validation data, indicating classic overfitting pattern on this small benchmark. In practice, this suggests that a modest number of epochs is sufficient for SST-2, and longer runs mostly waste compute and risk overfitting.

3) Layer freezing and effective capacity Figure 8 summarizes the layer-freezing experiment. With 0 frozen layers, all parameters are trainable and accuracy is around 0.929. Freezing only the bottom 1–4 layers reduces the proportion of trainable parameters from 100% to roughly 55% but keeps validation accuracy above 0.92, with differences well within the noise level of our other experiments. Even when freezing 11 layers which brings the trainable share below 10% , accuracy remains competitive and only drops modestly. This indicates that for SST-2, much of BERT's lower-layer capacity is redundant: a relatively small subset of upper layers plus the classifier head is enough to achieve strong performance, which is encouraging for parameter and compute constrained deployments.

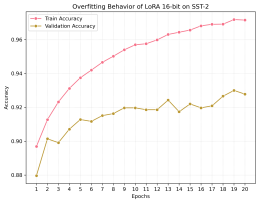

Fig. 7: *Overfitting Behavior of LoRA 16-bit*

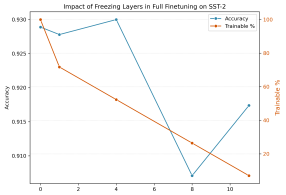

Fig. 8: *Impact of Freezing Layers in Full Finetuning*

F. Transfer Learning

0) Motivation One limitation of our main SST-2 study is that it mostly evaluates in-domain generalization: we fine-tune on SST-2 and then test on the SST-2 validation split. In a real startup setting, however, adapters would ideally be reusable across multiple clients and tasks rather than being tied to a single dataset. This raises two questions: (1) do LoRA adapters trained on SST-2 actually transfer useful representations to other GLUE-style tasks? and (2) how does the design of the adapter affect this transferability? The transfer-learning experiments in this section address these questions by moving SST-2–trained adapters to a different target task and analyzing their behavior through both accuracy and SVD-based rank statistics.

1) Experimental setup To understand how well LoRA adapters transfer across tasks, we run a set of source–target experiments on GLUE. The source task is SST-2 and the target task is the MRPC split from the GLUE benchmark [5], [6]. We start from bert-base-uncased and train LoRA adapters on SST-2 under several target module configurations: attention QKV only (attn-qkv), attention QKV plus attention output (attn-full), FFN only (ffn-only), attention Q or V alone, and a full configuration that covers both attention and FFN layers (full-model). For each configuration we train with multiple seeds and save the resulting LoRA weights.

For transfer, we instantiate a fresh BERT classifier for MRPC, apply the same LoRA configuration, and copy only the LoRA weights from the SST-2 model. We then freeze the backbone and all LoRA parameters, and train only the classification head on MRPC (linear probing) for a few epochs. Validation accuracy on MRPC is averaged over seeds to obtain a transfer score for each target-module setting. As a baseline, we also compare against "LoRA Direct", which initializes LoRA on MRPC from the pre-trained backbone without any SST-2 pre-training and trains on MRPC alone. Finally, we analyze the learned LoRA updates by computing the SVD of each $\Delta W = BA$ matrix and recording its stable rank as a measure of effective dimensionality.

2) Transfer vs. direct LoRA fine-tuning Figure 9 compares LoRA Transfer and LoRA Direct as a function of training epochs on MRPC. In the early epochs, the transferred adapters consistently outperform the direct baseline: after 3–4 epochs, LoRA Transfer reaches around 0.85 accuracy, while

LoRA Direct falls behind. As training continues, the gap narrows and by 8–10 epochs the two methods converge to similar accuracy. This indicates that pre-training LoRA on SST-2 provides a clear sample-efficiency and convergence benefit on the target task, although given enough epochs, direct LoRA fine-tuning can eventually catch up. For a resource constrained setting where fast adaptation matters, source-trained adapters are therefore attractive.

3) Effect of target modules on transfer performance Figure 10 summarizes MRPC linear-probing accuracy for different target-module choices. The full-model configuration achieves the best average accuracy (0.7100), followed by ffn-only (0.7075) and attn-full (0.7026). Configurations that modify only QKV (attn-qkv) or a single projection (attn-q-only, attn-v-only) perform noticeably worse (around 0.68–0.70). Neglecting outliers, this pattern suggests that adapters spanning both attention and feed-forward blocks learn representations that are more reusable across tasks, whereas extremely minimal adapters have weaker transferability.

4) SVD analysis of LoRA updates Figure 11 visualizes the layer-wise stable rank of ΔW for each configuration. Across all settings, the effective rank is highest in the middle Transformer layers and decreases toward the top and bottom layers, suggesting that LoRA mainly introduces high-dimensional updates in the shared semantic layers of BERT. At every layer, attn-full and full-model exhibit consistently larger stable ranks than the other configurations, while attn-qkv and ffn-only are intermediate, and attn-q-only / attn-v-only remain low-rank throughout. Figure 12 summarizes these patterns over all layers: the overall stable-rank distribution is highest for attn-full and full-model. This ordering closely mirrors the transfer accuracies reported earlier, indicating that configurations with higher effective rank in their LoRA updates tend to yield more transferable representations. The SVD analysis therefore provides an interpretable link between the capacity of the adapters (as reflected by ΔW's stable rank) and their ability to support new tasks.

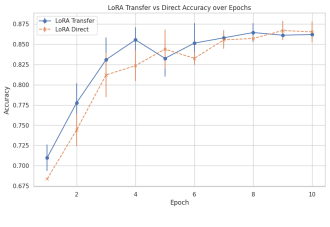

Fig. 9: *LoRA Transfer vs. Direct Accuracy over Epochs*

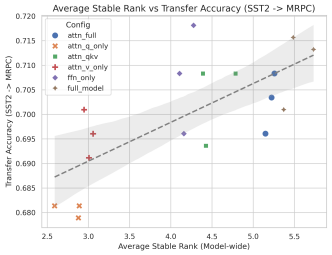

Fig. 10: *LoRA Stable Rank vs. Transfer Accuracy*

IV. Gradio-Based Sentiment Evaluation Interface

To make the models comprehensible and accessible to non-ML stakeholders at the movie-review company, we build a lightweight graphical user interface using Gradio. The interface runs locally and exposes a simple "sentiment evaluation platform" where users can interactively call full fine-tuning,

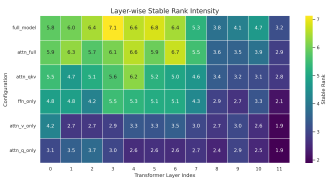

Fig. 11: *Layer-wise stable rank intensity across different adapter configurations*

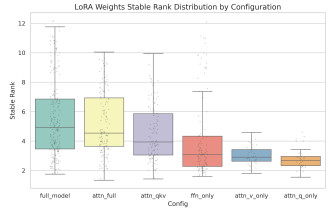

Fig. 12: *Distribution of stable rank values for LoRA weights*

LoRA 16-bit, LoRA 8-bit, QLoRA 4-bit and external API models.

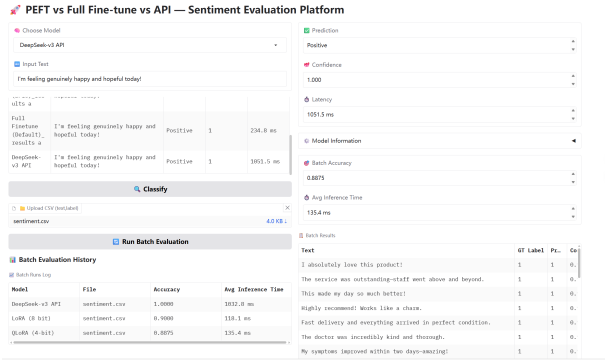

Fig. 13: *Gradio-Based Sentiment Evaluation Interface*

The GUI has three main components as shown in Figure 13. First, a single-example panel allows users to choose a model from a dropdown , type arbitrary review text, and obtain the predicted sentiment, confidence score, and measured latency. Model metadata such as total parameter count is displayed alongside, reinforcing the connection between parameter-efficient methods and deployment cost.

Second, a batch evaluation panel supports CSV uploads for testing. Users can upload a file of labeled reviews, run batch inference, and obtain overall accuracy and average per-example latency for each model. The interface keeps a history of evaluation runs, making it easy to compare between models. For example, the local LoRA 16-bit model achieves similar accuracy to the API while maintaining noticeably lower latency and no per-call cost. This concretely demonstrates one of our key claims: in a startup setting, a small local PEFT model can be both faster and cheaper than depending on a hosted API.

Finally, the Gradio app is implemented in a modular way, mirroring our training pipeline. Adding a new model checkpoint or swapping in a different backbone requires only registering a new entry in the model selector. This makes the interface a practical tool to explore trade-offs—accuracy, latency, and resource usage for product stakeholders who may not be familiar with the underlying training details.

V. Discussion and Limitations

Our experiments were motivated by a startup-style setting: a small team, one or two commodity GPUs, and the need to

ship a reliable sentiment model without excessive engineering overhead. In this setting, the goal is not to chase the last 0.1% on SST-2, but to understand which fine-tuning strategy and which level of capacity and data investment are "good enough" given limited time and hardware.

A. Choosing a fine-tuning strategy under resource constraints

The efficiency–performance results suggest a clear default for this kind of team. Full fine-tuning is dominated by 16-bit LoRA, it slightly improves accuracy, significantly cuts training time to roughly one third and halves peak VRAM. QLoRA achieves the best memory usage but introduces extra implementation complexity and is much slower than 16-bit LoRA on our scale. For medium-size backbones such as bert-base-uncased, a simple LoRA 16-bit configuration therefore strikes the best balance between performance, speed, and engineering cost. QLoRA becomes attractive mainly when the backbone is much larger or the GPU memory budget is extremely tight. Our 8-bit LoRA variant offers little advantage, its memory savings are modest relative to QLoRA, and training time is close to full fine-tuning. This makes it less compelling in this regime.

B. Hyperparameters and effective capacity

The learning-rate and rank sweeps indicate that 16-bit LoRA exhibits stable performance across a range around 10^{-4}. The performance does not collapse when we move slightly up or down, which is important when hyperparameter tuning budget is small. For LoRA rank, very low ranks underfit and very high ranks give diminishing returns or mild overfitting; a moderate rank around 16 offers a good trade-off between expressiveness and parameter count. The capacity experiments further indicate that both full fine-tuning and LoRA can easily overfit SST-2 if trained for too many epochs, while freezing a substantial number of lower layers leaves accuracy almost unchanged. For practitioners, this means that conservative choices—few epochs, partial freezing (for full-finetuning), moderate LoRA rank—are often sufficient, and aggressive capacity increases mostly waste compute.

C. Data quality, scaling, and robustness

From a data-centric perspective, SST-2 behaves like an easy, clean dataset. The scaling curves show that once we have 1000 well-labeled examples, both full fine-tuning and LoRA are already close to their full-data performance; adding tens of thousands more brings only small gains. This reinforces the idea that, for a small company, it can be more cost-effective to filter a relatively small but high-quality labeled set than to pursue a much larger noisy dataset. The robustness experiments support this view: the LoRA model is stable across random seeds, and moderate character-level noise only slightly degrade performance. At the same time, our data-centric experiments warn against naive difficulty-based sampling. Continuation training on only the easiest or only the hardest examples hurts performance, and truncating the distribution to the middle region also underperforms. A simple mixture that preserves the overall difficulty and sample distribution works better than both extremes and random sample from whole dataset. This suggests that lightweight data selection can help, but only if it respects the underlying data distribution rather than aggressively filtering it.

D. Transferability and adapter design

Looking beyond SST-2, we asked how well LoRA adapters trained on one task can help on another. Transferring adapters from SST-2 to MRPC accelerates early training on the target task: with only a few epochs, transferred LoRA outperforms LoRA trained from scratch, although the gap narrows as we train longer. For a small team that needs to support multiple related text-classification tasks, this kind of adapter reuse can reduce time-to-first-usable-model. However, transfer performance is sensitive to where adapters are inserted. Configurations that cover both attention and feed-forward layers have higher stable rank in their update matrices and achieve better cross-task accuracy, whereas extremely narrow adapters (Q-only or V-only) are low-rank and transfer poorly. This indicates that aggressively shrinking adapters to save a few more parameters greatly undermine performance.

E. Limitations and threats to validity

Our conclusions are conditioned on several limitations. First, most experiments use a single small English classification benchmark (SST-2) and one additional GLUE task for transfer. The dataset is short, clean, and balanced; results may differ on longer documents, highly imbalanced labels, multilingual data, or domains with stronger distribution shift (e.g., social media or domain-specific reviews). Second, we focus on bert-base-uncased; larger encoder or decoder-style models may exhibit different efficiency and overfitting behavior, and QLoRA's benefits become more noticeable there. Third, our robustness tests use character-level noise and a simple loss-based difficulty score from a single teacher checkpoint; real-world noise (sarcasm, domain drift, adversarial inputs) is more complex, and different difficulty measures could change the data-centric findings. Finally, the SVD and stable-rank analysis provides suggestive correlations between effective rank and transferability, but it is a heuristic diagnostic rather than a formal causal explanation.

Despite these limiations, the overall picture is consistent: for a small team deploying sentiment models, simple LoRA 16-bit setup combined with modest hyperparameter tuning, early stopping, and basic data selection offers a strong practical default. QLoRA and more elaborate adapter designs are useful tools when scaling to larger backbones or when cross-task reuse is critical, but most of the accuracy, robustness, and efficiency gains can already be realized with relatively simple choices that respect both model capacity and data quality.

VI. Conclusion

In this project We started with a practical question: under tight GPU and engineering budgets, how should a small team

fine-tune Transformer models for sentiment analysis? On SST-2 with bert-base-uncased, our results show that a simple 16-bit LoRA configuration is a strong default: it slightly outperforms full fine-tuning while using about half the memory and one third of the training time. QLoRA delivers the best VRAM savings but adds complexity and is mainly justified when the backbone is much larger or memory is the primary bottleneck.

Our scaling, robustness, and data-centric experiments suggest that effort is better spent on data quality and light selection than on sheer volume or heavy model tuning. With roughly 1000 clean, representative examples and moderate hyperparameters (LoRA rank=16, learning rate=10^{-4}, few epochs), performance is already close to the full-data optimum, and simple distribution-aware sampling works better than aggressively filtering for only easy or hard cases. Furthermore, transfer experiments indicate that LoRA adapters learned on SST-2 are reusable: they speed up learning on MRPC, especially in early epochs, and adapters that cover both attention and feed-forward layers—with higher effective rank—transfer better than very narrow ones.

For similar real-world text-classification problems, a practical recipe is therefore: use 16-bit LoRA on a medium backbone, tune a small set of hyperparameters, filter a modest but clean dataset, and design adapters that span both attention and FFN blocks to enable cross-task reuse.

VII. Individual Contribution

For transparency, we declare that Weifeng Chen and Boyi Zhang contributed equally and jointly carried out the vast majority of the project, over 85% of the total workload.

Name	Roles and Contributions
Weifeng Chen	• Led the project direction and the overall code framework. • Conducted key experiments on PEFT methods, hyperparameter tuning, model capacity and overfitting analysis, and transfer learning. • Contributed to the writing and refinement of the project report. • Recorded the project demo.
Boyi Zhang	• Contributed to the development and maintenance of the code framework. • Conducted experiments on generalization scaling and data-centric analysis. • Performed data analysis and prepared the presentation slides. • Led the writing and refinement of the project report. • Recorded the project demo.
Yunxiang Li	• Designed and implemented the GUI interface. • Reproduced the experiments. • Prepared the README documentation.

TABLE IV: *Team Member Contributions*

References

[1] L. Wang, S. Chen, L. Jiang, S. Pan, R. Cai, S. Yang, and F. Yang, "Parameter-efficient fine-tuning in large language models: a survey of methodologies," *Artificial Intelligence Review*, vol. 58, no. 8, p. 227, 2025.

[2] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[4] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.

[5] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proceedings of the third international workshop on paraphrasing (IWP2005)*, 2005.

[6] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," in *International Conference on Learning Representations*, 2019.