A multi-objective approach to improving accuracy and efficiency in multitask BERT

Stanford CS224N Default Project

Arpit Singh
Department of Computer Science
Stanford University
arpsingh@stanford.edu

Amitai Porat
Department of Computer Science
Stanford University
aporat@stanford.edu

Lin Ma
Department of Computer Science
Stanford University
linmalin@stanford.edu

Abstract

Our project aims to extend BERT for multitask learning across three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Beyond baseline multitask learning, we further enhance BERT's performance in terms of accuracy and efficiency by focusing on key aspects of model architecture and various training techniques. We explore state-of-the-art methods such as SMART, LoRA, DoRA, and DDP, which have not been extensively evaluated on encoder-based models. Our architecture modifications to BERT significantly improve performance across the three tasks by 18% over the baseline. Our sampled data achieved 88% of the baseline performance with a 96.5% data reduction. Among various resource efficiency techniques, LoRA proved most effective in reducing GPU memory usage and training time, albeit with a trade-off in accuracy as compared to full fine-tuning. While the Distributed Data Parallel (DDP) technique significantly reduces training time, it can negatively impact accuracy.

1 Key Information to include

TA mentor: Josh Singh; *External collaborators:* N/A; *Sharing project:* No; *Team contributions:* Equal Contribution of all team members. Lin worked on multitask implementation, architecture optimization, N hidden layers and data sampling. Amitai worked on the SMART implementation and incorporation into multitask architecture. Arpit worked on PEFT techniques viz LoRA and DoRA as well as DDP implementations and experiments.

2 Introduction

An exceptional power of a Large Language Models (LLMs) is that they can generalize to perform many different tasks across a wide variety of topics through pre-training over large corpora. Fine-tuning LLMs becomes paramount in training LLMs to perform well on domain specific downstream tasks. Modern fine-tuning techniques present new challenges as they face a trade-off between model accuracy improvements and intensive resource consumption.

In our project we address this trade-off by applying various techniques to improve the accuracy and training performance of a multitasked BERT. Specifically, for resource efficient fine-tuning, we apply two Parameter-Efficient Fine-Tuning (PEFT) techniques: Low-Rank Adaptation (LoRA) Hu et al. (2021) and its extension, Weight Decomposed Low-Rank Adaptation (DoRA) Liu et al. (2024), a novel technique. We observe how these techniques impact training time, GPU utilization, and memory consumption during fine-tuning. We also implement architecture optimization and regularization to improve model accuracy. With our architecture modification, accuracy is significantly improved over the baseline. Using PEFT techniques, we see a reduction in training time and resource consumption, though with some impact on accuracy. Our project provides insights into combined efforts to achieve memory and computational efficiency without sacrificing too much accuracy. Further hyperparameter tuning with PEFT techniques is required to optimize accuracy and efficiency simultaneously across all downstream tasks to bring accuracy as close to the baseline as possible.

3 Related Work

After the introduction of BERT [Devlin et al. (2018)], various improvements[Asa Cooper Stickland (2019)] have been applied to generalise it for multitask performance. There is a noticeable gap in research exhaustively covering all aspects of the model development pipeline. In particular, with the advent of decoder-based models like GPT [Alec Radford (2018)] , the latest state-of-the-art techniques have not been well-tested all together on encoder-based models like BERT.

PEFT methods primarily consist of adapter based methods Houlsby et al. (2019), prompt based methods Lester et al. (2021), and LoRA based methods. In adapter-based methods, additional trainable modules are added, while in prompt-based methods, extra soft tokens are added to the initial input for the purpose of fine-tuning. LoRA based methods "apply low-rank matrices to approximate weight changes during fine-tuning and can merge with pre-trained weights prior to inference." DoRA enhances LoRA's learning capacity and stability without adding inference overhead. By decomposing pre-trained weight matrices into lower ranks and decoupling magnitude and direction updates, DoRA maintains state-of-the-art expressivity . An accuracy gap exists between LoRA based methods and full fine-tuning due to the limited learning capacity of LoRA. DoRA tries to bridge this gap by marrying the fine-tuning resource efficiency of LoRA with the expressivity of full fine-tuning learning. Our work applies LoRA and DoRA to fine-tune a multitask BERT and analyzes the impact on performance and efficiency in terms of GPU memory utilization and training time.

The SMART paper Haoming Jiang and Zhao (2019) introduces a regularization penalty to the loss function of our fine-tuning training process. The regularization loss is representative of how sensitive a model is to perturbations in the input embeddings. SMART injects perturbations into the training embeddings and computes a regularization penalty based on the divergence of the model predictions on the original embeddings versus the perturbed embedding values. As part of the loss algorithm SMART runs gradient ascent with respect to the noise embeddings to maximize the regularization penalty at each batch. We adapt it into our architecture to improve model generalization and robustness.

The LIMA paper[Zhou et al. (2023)] demonstrates that fine-tuning a pre-trained language model with a curated high-quality dataset yields superior results, thereby emphasizing the fact that most of the knowledge in large language models (LLMs) is acquired during pre-training. Instead of using millions of prompts the authors fine-tuned a 65B parameter LLaMa model on only 1,000 carefully curated examples. This approach prioritizes quality over quantity when it comes to data and results in better human-preferred responses compared to models fine-tuned on extensive data. Our work derives motivation from this work to achieve high-quality data sampling and improve fine-tuning accuracy while simultaneously reducing training time.

4 Approach

In this and later sections, we will refer to the three downstream tasks as follows: sentiment analysis (**SST**), paraphrase detection (**PARA**), and semantic textual similarity (**STS**).

4.1 Multi-tasking BERT

As a first step we extend BERT to do multitask learning. We add three task-specific heads and set the training loop sequentially in order of SST, PARA and STS. For SST, sentence embeddings are passed through dropout then a softmax classifier, trained with cross-entropy loss. For PARA, individual sentence embeddings are concatenated, followed by dropout and a softmax classifier with binary cross entropy loss. For STS, individual sentence embeddings are used with cosine similarity as the output layer and MSE loss.

We refer to this design as MultitaskBERT v1.

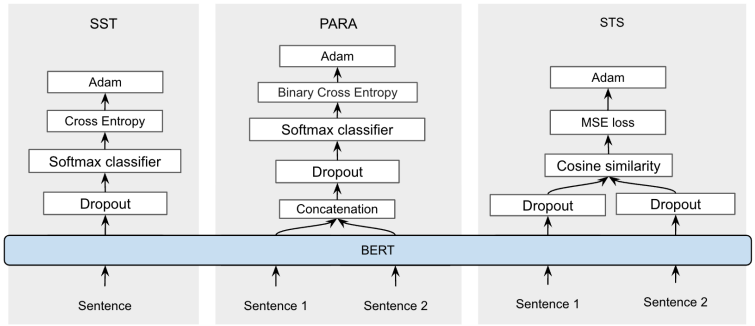


Figure 1: Architecture of our MultitaskBERT v1

4.2 Accuracy Improvements

Architecture optimization: Upon MultitaskBERT v1, we modify the embedding generation strategy for PARA and STS. Each sentence pair is now concatenated with a [SEP] token and then embedded. We add configurable hidden layers to each task head to increase learning capability. Each hidden layer consists of a linear transformation, ReLU activation, and dropout regularization. The configuration provides a convenient way for subsequent N hidden layers experiments.

We refer to this design as MultitaskBERT v2.

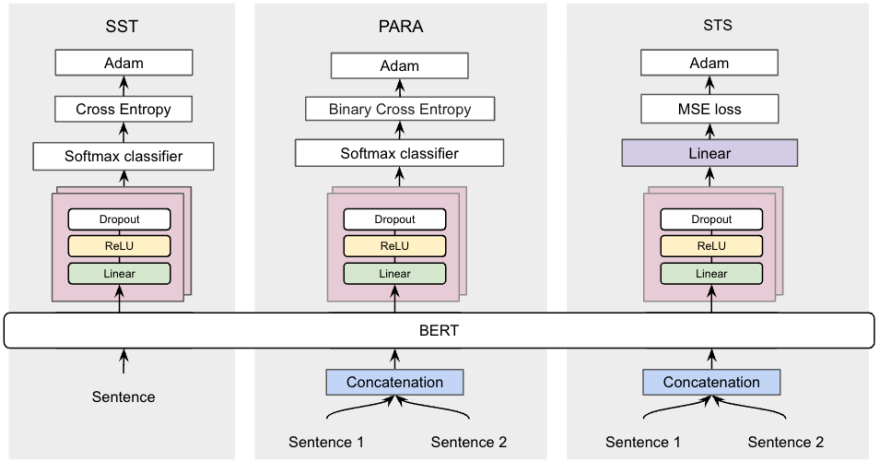


Figure 2: Architecture of our MultitaskBERT v2, with modifications highlighted in color

SMART regularizer: We apply SMART to SST and STS. Both SST and PARA compute embedding similarity for accuracy, while STS computes Pearson correlation [Freedman et al. (2007)]. SMART uses KL divergence to compute loss functions for classification and mean squared error to compute loss for regression tasks. We experiment across SST and STS to assess the efficacy of both loss functions as SST and Para are both classification tasks and STS is a regression task.

N hidden layer: We evaluate whether multi-layer structure performs better for our tasks.

4.3 Efficient Training

Finally, we focus on the following dimensions for enhancing on training efficiency:

Data efficiency via sampling: Curate a sampled dataset to investigate the effectiveness of data sampling and balanced training set in a multitask learning setting, and expediting subsequent experiments.

Memory efficiency via PEFT: Explore LoRA and DoRA techniques on our model for better resource utilization.

Computational efficiency via DDP: Implement Distributed Data Parallel (DDP) Li et al. (2020) to speed up training when multiple GPUs are available. Includes option to use Autocast to take advantage of automatic mixed precision if supporting hardware is present. This addresses the fact that training and experimenting on a single GPU can be time-consuming and expensive.

4.4 Logging and Monitoring

We incorporate PyTorch profiling to capture GPU utilization, memory consumption, and other diagnostic metrics, while utilizing TensorBoard for logging and monitoring of training loss and accuracy.

5 Experiments

5.1 Datasets

We used three datasets in our experiments:

- SST: The Stanford Sentiment Treebank [Socher et al. (2013)] is used and consists of 11,855 single sentences extracted from movie reviews. The dataset is parsed by the Stanford parser8 [Chen and Manning (2014)] and includes a total of 215,154 unique phrases from the resulting parse trees which are annotated by 3 human judges. Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive.

- PARA: The Quora [Zhiguo Wang (2017)] dataset is used and consists of 404,301 question pairs with labels indicating whether text sequences are paraphrases of one another.

- STS: The SemEval STS Benchmark [Cer et al. (2017)] dataset is used and consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning).

5.2 Evaluation method

For SST and PARA, model accuracy is evaluated against labeled dev datasets. For STS the Pearson correlation [Freedman et al. (2007)] is used. The mean of the three accuracy/correlation scores is then used for inter-epoch comparison, and the model with the highest mean score is saved.

For training efficiency experiments, the overall training time, GPU memory consumption and GPU utilization are analyzed and compared between PEFT techniques. For Distributed data parallel(DDP) experiments, GPU meory usage and GPU utilization are measured by capturing the highest utilization across all GPUs, and these metric are used for evaluation.

5.3 Experiment details, results and analysis

Unless specified, the default configurations were used throughout our experiments: batch size 16, learning rate 1e-5, dropout probability 0.3, epochs 10, fine-tune mode "full-model" and 1 Nvidia T4 GPU. The fine-tune mode has two options: 1) "full-model" where all BERT parameters are trainable and 2) "last-linear-layer" where BERT parameters are frozen and only the task-specific layers are trained.

5.3.1 Architecture optimization

Model name	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Training Time	Configuration
MultitaskBERT v1	**0.495**	0.731	0.526	0.663	10.18h	No hidden layers
MultitaskBERT v2	0.491	**0.900**	**0.886**	**0.778**	9.98h	3 hidden layers, 1 for each task

Table 1: Architecture optimization experiment result

The MultitaskBERT v2 surpasses v1 for 17.35% on overall dev accuracy. For SST, during training [Figure 3], the v1 rapidly improves accuracy within the first few epochs. After reaching a plateau around epoch 5, it has a minor dip in accuracy before stabilizing. MultitaskBERT v2 starts with a lower accuracy but steadily improves throughout the training process. Both models reach almost the same accuracy at the end of the training. This finding concludes that the additional layer does not affect fine-tuning for SST.

For PARA, MultitaskBERT v1 accuracy fluctuates throughout all epochs and remains consistently lower than v2. STS presents a similar pattern. Contradicts to the dev accuracy graph, the loss graph of STS shown v1 is better at minimizing the training loss. By looking at the loss graph alone might be misleading when it does not align with accuracy. Lower loss does not always equate to higher accuracy.

This experiment shows that the embedding generation strategy has an enormous impact on model accuracy. MultitaskBERT v2 excels at tasks that benefit from richer contextual information provided by the **sentence pair embeddings**. Its performance on tasks that only utilize **single sentence embeddings** is comparatively less accurate. This indicates a limitation in its ability to capture context from isolated sentences. The [SEP] token used in sentence pair embedding acts as a clear boundary between the two sentences and is an indicator for BERT to process two distinct sequences that need to be processed together. During pre-training BERT is typically trained on tasks that involve multiple segments of text. By using [SEP] token consistently during fine-tuning we ensure that the model is being used in a way that is consistent with its pre-training, which helps improve performance on sentence pair tasks, paraphrase detection, and semantic textual similarity.

The added hidden layer also contributes to the performance boost. To further investigate whether this gain can increase along with more layers, we conduct more experiments in "N hidden layers" [Section 5.3.5].

5.3.2 SMART regularizer

In order to keep experiments focused on the accuracy score impact of SMART, all hyperparameters were held constant except k = {1, 250, 500}, which represents the number of iterations for updating the perturbed embeddings. The injected perturbation size is set to epsilon = 10e-5, sampled from a normal distribution with mean 0, standard deviation 1, and variance 10e-5. The step size was set to 10e-3, as was done in the SMART paper. Like the SMART paper, we use the AdamW Optimizer with learning rate 1e-5. The SMART results were evaluated against the MultitaskBERT v1 architecture [Figure 1] which provided the original baseline model accuracy. Experiments 1-3 focused on fine-tuning with SMART on SST and experiments 4-6 on fine-tuning with SMART on STS. Below test were done on MultitaskBERT v1 which does not have hidden layer for tasks.

The regularization loss is computed using symmetrized KL divergence for SST, and mean squared error for STS, as described in the SMART paper. The loss is defined by the following equation.

$$g_i(\tilde{x_i}, \bar{\Theta}_s)\frac{1}{|B|}\sum_{x\epsilon B} \bigtriangledown_{\tilde{x}} l_s(f(x_i, \bar{\Theta}_s), f(\tilde{x_i}, \bar{\Theta}))$$

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Experiment Setup	Additional Comments
1	0.526	0.409	0.321	0.532	Fine tune on SST. k = 1	Baseline experiment for fine-tuning SST with SMART. Dev Accuracy of .532 outperforms baseline of .523.
2	0.502	0.497	0.297	**.549**	Fine tune on SST. k = 250	SST Accuracy of .502 compared to baseline from MultitaskBERT v1 of .495. Additionally improves dev accuracy from baseline of .523 to .549.
3	0.502	0.497	0.297	0.549	Fine tune on SST. k = 500	Regularization convergence leading to no change on accuracy score from Experiment 2.
4	0.184	0.589	0.471	**0.503**	Fine tune on STS. k = 1	Baseline experiment for fine-tuning STS with SMART. Dev Acc outperforms baseline MultitaskBERT v1 of .496.
5	0.205	0.482	0.421	0.466	Fine tune on STS. k = 250	Under performs compared to Experiment 4 with K=1.
6	0.191	0.526	0.495	0.488	Fine tune on STS. k = 500	Increase number of gradient updates to see if value of regularization penalty has affect on accuracy. Under performs compared to Experiment 4 with K=1.

Table 2: SMART Regularizer experiment results

From Table [2] In Experiments 1-3 we examine whether fine-tuning the baseline multitaskBERT model with SMART regularization on the SST dataset has any impact on accuracy scores. We apply SMART to one fine-tuning task at a time in order to help attribute accuracy score fluctuation to SMART regularization. The distinguishing hyperparameter between experiments k, specifies the number of iterations on the gradient descent loop for the noise embeddings. Fine-tuning SST with SMART produces an improved dev accuracy score at k=1, k=250, and k=500 compared to the baseline. The difference in dev scores between k=1 and k=250, and k=500 can likely be attributed to maximizing the loss penalties through gradient ascent for each batch. The accuracy values converge between k=250 and k=500, indicating that the gradient ascent loop can be terminated earlier with convergence criteria if updates are too small. In future experiments it would be interesting to observe where this convergence occurs. The results do not provide an accuracy score improvement to SST baseline of .495. This might be due to the fact that SMART commits to increasing a model's robustness to perturbations but does not guarantee improvement in accuracy score for a particular task. The overall increase in dev score compared to the baseline indicates that reduced sensitivity may help with generalization to downstream tasks.

In Experiments 4-6 SMART is applied to fine-tuning the STS dataset. The best results to dev accuracy come from k=1. This indicates that the randomly sampled injected noise can be sufficient to calculate regularization penalties. It is possible that maximizing the regularization penalty with k=250 and k=500 causes fine-tuning to a particular task to overfit and thus result in a lower overall dev score across several tasks. In future work it would be interesting to evaluate model accuracy against regularization loss values to understand the impact of tweaking the k iteration loop.

5.3.3 MultitaskBERT v2 with SMART regularizer

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Training Time	Configuration	Model name
-	**0.495**	0.731	0.526	0.663	10.18h		MultitaskBERT v1
-	0.491	0.900	**0.886**	0.778	9.98h		MultitaskBERT v2
1	**0.495**	**0.906**	0.884	**0.781**	11h	SMART on SST	MultitaskBERT v2 with SMART

Table 3: MultitaskBERT v2 with SMART compared to the previous versions

This experiment we fine-tune MultitaskBERT v2 with SMART on SST. Compared to the v1 and v2 models, it produces the highest dev score, resulting in an overall accuracy improvement. While SMART is only applied on SST, the results show that it helps increase the dev accuracy on PARA. It is possible that the dataset elements of PARA are generally

close to the model classification boundary, and thus a regularization penalty created by SMART creates a model that is more robust to perturbations, which would be particularly meaningful in this type of classification task.

5.3.4 Data sampling

We select 10,000 examples (out of 283,011) from PARA train set [Section 5.1] based on their topic category distribution [Appendix A.1], to ensure a representative sample of the original data and a balanced data size across all three tasks. In Experiment 1, PARA is trained on the sample data. SST and STS remain the same. In Experiment 2, we fine-tune on MultitaskBERT v2 which was pre-trained on the full datasets and run 10 epochs per task, with sampled data for PARA, for a total of 30 epochs.

In the table below, "pre-train model" refers to loading an existing model's configurations, whose dev accuracy serves as the initial benchmark for inner-epoch comparisons.

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Experiment Setup
-	0.491	0.900	0.886	0.778	MultitaskBERT v2 [Table 1] as baseline.
1	**0.513**	0.792	0.877	0.748	No pre-train model, fine-tune mode "full-model"
2	0.496	**0.906**	**0.888**	**0.782**	pre-train model "MultitaskBERT v2", fine-tune mode "last linear layer"

Table 4: Data Sampling experiment results

Experiment 1 shows that the model performs better on SST task when data size between tasks is balanced. For PARA, **the sampled data achieved 88% of the baseline performance with a 96.5% data reduction**. Experiment 2 yields our best model, suggests that further fine-tuning a model on task heads only, using the sampled datasets can further improve performance albeit only marginal gains.

5.3.5 N hidden layers

To efficiently evaluate the impact of varying hidden layers, we conduct experiments using the sampled dataset from the previous experiment for PARA. SST and STS remain the same. In the table below, "hl" represents hidden layers, and are allocated evenly among task heads. "lr" represents learning rate and "bs" represents batch size.

Exp.	SST Acc.	PARA Acc.	STS Corr.	Experiment Setup	Additional Comments
1	0.513	0.792	0.877	hl 3, lr 1e-5, bs 16, dropout 0.3, epochs 10.	Baseline experiment.
2	0.495	0.792	0.866	hl 6, lr 1e-5, bs 16, dropout 0.3, epochs 10.	Added additional layer.
3	0.508	0.784	0.876	hl 6, lr 2e-5, bs 32, dropout 0.3, epochs 10.	Increased batch size and learning rate.
4	**0.518**	**0.805**	**0.886**	hl 3, lr 2e-5, bs 32, dropout **0.4**, epochs 20.	Increased batch size, learning rate, dropout probability and epochs.
5	0.509	0.793	0.871	hl 6, lr 2e-5, bs 32, dropout **0.4**, epochs 20.	Increased hidden layer, batch size, learning rate, dropout probability and epochs.

Table 5: N hidden layer experiment results

Experiment 1 we train the model with default configurations [Table 1] as a baseline. Experiment 2 we add one hidden layer to each task and observe a decrease in both SST dev accuracy and STS Pearson correlation. Based on the train loss graph [Figure 4], Experiment 1 consistently shows lower loss values. Both experiments eventually converge to a similar loss at the end of epoch 10. This suggests that Experiment 1 model is generally performing better in terms of learning the underlying patterns in the training data.

Experiment 3 the batch size and learning rate were doubled which results in improved scores for SST and STS compared to Experiment 2, but still a drop compared to Experiment 1. This suggests that increasing the number of hidden layers may necessitate corresponding adjustments to batch size and learning rate to maintain the model performance.

Since convergence remains unclear and dropout's impact unknown, we conducted Experiment 4 and 5 with more epochs [Figure 5]. We increase the dropout probability from 0.3 to 0.4, and run the experiments with batch size 32 and learning rate 2e-5 for 20 epochs. Experiment 4 outperformed all, showing higher and more stable dev scores.

The experiments show us that merely increasing the number of hidden layers does not guarantee improved performance. On the contrary, it can lead to decreased accuracy and increased loss, especially the dataset size might not be sufficient to warrant the increased complexity. For our tasks and data size, a simpler architecture is more effective.

5.3.6 LoRA and DoRA

The first set of experiments are done to setup baseline and evaluate the feasibility of implementing LoRA, DoRA and analyze the impact of varying hyperparameter. Below experiments were conducted on 1 Nvidia T4 GPU on MultitaskBERT v1. We used default hyperparameter setting unless specified.

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Training Time	Experiment Setup	Additional Comments
1	0.429	0.735	0.366	0.510	10h	Learning rate: 4e-5. Batch size: 32.	Experiment to setup baseline
2	0.305	0.693	0.289	0.429	6h 25m	LoRA with MSFT lib [Hu et al. (2022)] (Rank 16). Learning rate: 2e-15. Batch size: 32.	Reduction in accuracy with LoRA
3	0.345	0.677	0.311	0.444	8h 20m	DoRA with Hugging Face PEFT lib [Mangrulkar et al. (2022)]. Learning rate: 4e-15. Batch size: 32.	Overall accuracy improved with higher training time

Table 6: LoRA and DoRA on MultitaskBERT v1 experiment results

For Experiment 1, we observed training time is approximately 10 hours for T4 GPU. Experiment 2 with LoRA achieves shorter training time compared to the DoRA implementation from Experiment 3. Variations in implementations of Huggingface libraries and MSFT library may account for variation in training time. The higher accuracy of DoRA is in line with expectations since it trains a higher number of parameters than LoRA.

Variations in learning rate in Experiment 3 with DoRA compared to previous one is inspired by Smith et al. (2018). In the subsequent experiments, we use the Hugging Face PEFT library due to its diverse range of configurable options.

5.3.7 LoRA and DoRA with Profiling enabled and upgraded hardware

Based on the previous experiment, we conduct the below experiments with profiling enabled such that we can capture hardware performance metrics. All experiments were conducted on MultitaskBERT v2 with 1 A100 GPU.

Experiment 1 is conducted to establish a baseline and represents the accuracy of the full fine-tuning case. The goal of PEFT techniques is to achieve a trade-off by being closer to the baseline accuracy with minimum resource usage.

In the table below "Rank" refers to the rank of the trainable matrix (also called the "attention dimension"). "lora_alpha" represents alpha parameter for LoRA scaling. The "module" represents a "target_module" which is the names of the modules that applied to the adapter. Batch refers to Batch size and LR refers to Learning rate.

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Training Time	Experiment Setup	Additional Comments
1	0.481	0.905	0.880	0.756	2h 3m	Batch 128; LR 4e-5; Epochs 20	Baseline Experiment Peak GPU memory: 4750.8 MB, GPU Utilization 53.91 % .
2	0.473	0.869	0.853	0.731	**1h 53m**	LoRA enabled(Rank 8) lora_alpha 16,module:query,value Batch 128; LR 4e-5; Epochs 20	Num trainable params: 0.2644% Peak GPU memory: **2963 MB**, GPU Utilization 54 %
3	0.477	0.868	0.853	0.733	2h 28m	DoRA enabled(Rank 8), lora_alpha=16,module:query,value Batch 128; LR 4e-5, Epochs 20	Num trainable params: 0.2808% Peak GPU memory: 3503 MB, GPU Utilization 53.47 %

Table 7: Experiment results of LoRA and DoRA with pytorch profiling

We observe that with LoRA the memory used is reduced by almost 37%. It also takes less time to train. The memory used by DoRA increases in comparison to LoRA. The DoRA training time also increases without significant increase in accuracy. GPU utilization in all cases stay same around 54% indicating there is no impact of PEFT on utilization. The lower accuracy on DoRA experiment as compared to full fine tuning might suggest that hyperparameter tuning is required for DoRA to achieve accuracy that are on par with full fine-tuning results of baseline experiment.

From experiments from Table 7 on 1 A100 GPU, we observe that with LoRA/DoRA, number of trainable parameter goes down as reflected in reduction in memory usage. However, DoRA seems to be more resource intensive in terms of training time and memory usage without significant accuracy improvement over LoRA.

5.3.8 LoRA and DoRA with DDP, and Autocast

We implemented Distributed Data Parallel on MultitaskBERT v2 and used all 8 GPUs of single server node on A100 hardware. All experiments used a batch size of 128, a learning rate of 4e-5, 20 epochs, "lora_alpha" of 8, and

"target_modules" as query and value. For LoRA configuration, we set lora_dropout=0.05, init_lora_weights=True, and use_rslora=True for both LoRA and DoRA experiments. GPU utilization and GPU memory represents the maximum utilization and memory respectively among all 8 worker GPUs.

Exp.	SST Acc.	PARA Acc.	STS Corr.	Dev Acc.	Training Time	Experiment Setup	Additional Comments
1	0.425	0.853	0.804	0.694	16m 5s	Baseline experiment	Peak GPU memory: 5545 MB, GPU Utilization 56.83 %
2	0.382	0.816	0.782	0.660	14m 58s	LoRA enabled, Rank 8	Num trainable params: 0.2644% Peak GPU memory: 3397 MB, GPU Utilization 56.32 %
3	0.388	0.826	0.783	**0.665**	18m 0s	DoRA enabled, Rank 8	Num trainable params: 0.2808% Peak GPU memory: 3989 MB, GPU Utilization 59.92 %
4	0.384	0.820	0.771	0.658	**14m 58s**	LoRA enabled, Rank 2	Num trainable params: 0.0662% **Peak GPU memory: 3344 MB**, GPU Utilization 58.2 %
5	0.361	0.802	0.774	0.645	18m 0s	DoRA enabled, Rank 2,	Num trainable params: 0.0828% Peak GPU memory: 3983 MB, **GPU Utilization 61.86 %**
6	0.400	0.828	0.784	**0.670**	18m 26s	Autocast enabled with BF16 DoRA enabled, Rank 8	Num trainable params: 0.2808% Peak GPU memory: 3990 MB, GPU Utilization 60.42 %

Table 8: LoRA and DoRA with DDP, and autocast

The first experiment establishes a baseline for DDP experiments and represents the accuracy of the full fine-tuning case. We vary techniques (LoRA/DoRA) as well as their rank hyperparameter to determine which configuration can get closer to the baseline accuracy with minimum training time and GPU resources.

In above Table 8, we observe that LoRA with lower rank performs better than DoRA with rank 2 while underperforming slightly at rank 8. Training time of DoRA is significantly higher(20 %) than LoRA and consumes more memory as compared to LoRA. With autocast enabled, there is an positive accuracy impact(+0.005) and increase in GPU utilization.

Table 8 also shows that using 8 GPUs with DDP drastically reduces training time due to multi-GPU parallelism. However, accuracy is significantly impacted, indicating a need for further hyperparameter tuning. DoRA (Rank 8) with DDP achieves better accuracy than LoRA, suggesting higher resource usage by DoRA aids in DDP-based training.

Figure 8 compares GPU memory, utilization, and training time, while accuracy is compared in Figure 7 in the Appendix section. Notably, GPU utilization decreases with higher rank, as Rank 8 has lower utilization than Rank 2, due to denser trainable matrices in lower ranks. LoRA consumes lower memory than DoRA and baseline due to less number of trainable parameters. DoRA performs well at higher ranks but consumes more memory and higher training time, as it trains weight parameters in addition to the rank matrix. This undermines the purpose of PEFT if accuracy gains come at the cost of increased training time and memory usage. AMP Autocast acts as a regularizer, providing slight accuracy gains but increasing GPU utilization due to dtype conversion.

Based on above experiments, LoRA with rank 2 seems to be better choice of PEFT technique for Multitask BERT considering resource utilization and accuracy tradeoff.

5.4 Best model

The following table presents the three top-performing models from our training experiments with **the accuracy obtained from the test leaderboard** [1] . The third model is fine-tuned on "MultitaskBERT v2 with SMART" [Table 3], using the same training procedure as "Data Sampling Experiment 2" [Table 4].

Model Name	SST Acc.	PARA Acc.	STS Corr.	Test Acc.
MultitaskBERT v2 with SMART	0.504	0.905	0.880	0.783
Data Sampling Experiment 2	**0.524**	**0.907**	0.880	**0.790**
Combined training techniques	0.511	0.905	**0.883**	0.786

Table 9: Best performed model comparison

6 Conclusion

Our project reveals several important insights into optimizing the performance of a multitask BERT model.

[1] In earlier table 3 and table 4, the overall accuracy were obtained from dev leaderboard.

For our tasks and datasets a single hidden layer is more effective than multiple hidden layers. Paired sentence embeddings are preferred for paraphrase detection and semantic textual similarly tasks. If the data source is limited a good quality of small data can still provide comparable performance.

SMART regularization was shown to enhance dev accuracy and model robustness on the condition that a proper regularization gradient ascent is configured. While SMART did not provide accuracy improvements on fine-tuning for STS correlation compared to the baseline, it was able to increase the overall dev score for this training task. This is a fair result because SMART proposes to deliver on model robustness to perturbations, and not necessarily improved overall accuracy. Experiments show that improved robustness may generalize to downstream tasks outside of a given fine-tuning task. Improved dev scores were consistent in SMART for both SST and STS finetuning. On MultitaskBERT v2, SMART was able to improve the overall dev score to .781, further supporting the idea that it helps the model generalize.

Between PEFT techniques LoRA and DoRA, both successfully reduced the memory usage, however DoRA's increased GPU memory resources did not translate into significant accuracy gains compared to LoRA.

Distributed Data Parallel (DDP) proved to effectively reduce training time through GPU parallelism, and usage of autocast improved accuracy slightly. Both approaches require more hyperparameter tuning to meet baseline accuracy.

In future experiments, a focus on optimizing hyperparameters to balance accuracy and computational efficiency, particularly when using DDP and PEFT techniques. Further investigation into convergence criteria for SMART regularization is needed to determine the optimal termination point for gradient ascent loops, potentially saving computational resources and further optimizing the loss function. Additionally, expanding the analysis to include more datasets and tasks will help validate whether or not our findings generalize. Further more, we would be interested in identifying specific areas for optimization in model architecture and training techniques. Our results underscore the importance of high-quality data sampling, strategic architectural choices, and meticulous tuning to achieve optimal performance in multitask language models.

7 Ethics Statement

One of the goal of our project is to refine and improve multitask finetuning on top of BERT. Ethical challenges arise with the ease of fine-tuning, potentially leading to hastily released models without thorough privacy considerations, especially with multitask fine-tuning, where violations may be harder to detect. Rapid fine-tuning could also worsen biases, reduce model generalizability, and overfit to specific tasks. To address these risks, robust bias detection is crucial, including identifying layers responsible for bias amplification. Transparency in training data and evaluating models on both fine-tuned and non-fine-tuned tasks can help maintain effectiveness across domains.

Sampling data could pose ethical concerns regarding under or misrepresentation, which can lead to biased models that perpetuate or amplify existing societal inequalities. It's crucial to prioritize equitable representation in sampled datasets, ensuring that they are inclusive, diverse, and accurately reflect the real-world complexities they aim to model. Effective sampling strategies also leads to lower training cost which can enable researchers and scientists with limited resources to conduct impactfull research. Training on carefully sampled dataset not only reduce over fitting but also reduce systematic bias which is found to be implicit in large datasets.

Additionally, large-scale model training demands significant GPU memory, contributing to environmental concerns. This was major motivation for us to conduct research in resource utilization. We integrated pytorch profiling specifically to measure resource impact of training and then implemented PEFT and DDP on BERT to efficiently use additional resources when available. Further Mitigation strategies, such as utilizing Fully Sharded Data Parallel (FSDP) technique, could help reduce energy consumption and carbon emissions.

References

Tim Salimans Ilya Sutskever Alec Radford, Karthik Narasimhan. 2018. Improving language understanding by generative pre-training. *https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf*.

Iain Murray Asa Cooper Stickland. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Google Cloud. 2024. General purpose machines - n1 machines. https://cloud.google.com/compute/docs/general-purpose-machines#n1_machines.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

David Freedman, Robert Pisani, and Roger Purves. 2007. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*.

Weizhu Chen Xiaodong Liu Jianfeng Gao Haoming Jiang, Pengcheng He and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimizatio. *arXiv preprint arXiv:1911.03437*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. Pytorch distributed: Experiences on accelerating data parallel training.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2018. Don't decay the learning rate, increase the batch size.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Radu Florian Zhiguo Wang, Wael Hamza. 2017. Bilateral multi-perspective matching for natural language sentences.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment.

A Appendix (optional)

A.1 Data sampling

To sample the data, we used zero-shot classification with the facebook/bart-large-mnli model and provided it with a list of predefined categories. Based on the resulting label distribution, we selected top k from each category, a total of 10,000 records. The distribution by category in our sampled data, listed below:

- Technology: 25.0% (2500 examples)
- Health: 25.0% (2500 examples)
- B2B/SaaS: 15.0% (1500 examples)
- Entertainment: 11.0% (1100 examples)
- Jobs and Career: 8.0% (800 examples)
- Finance: 6.0% (600 examples)
- Education: 5.0% (500 examples)

- Electronics: 2.0% (200 examples)
- Automotive: 2.0% (200 examples)
- CPG (Consumer Packaged Goods): 1.0% (100 examples)

A.2 SMART Implementation References

- https://github.com/namisan/mt-dnn/tree/master/mt_dnn
- https://github.com/archinetai/smart-pytorch/tree/main/smart_pytorch

A.3 PEFT Reference library

- Mangrulkar et al. (2022)
- Hu et al. (2022)

A.4 Figures

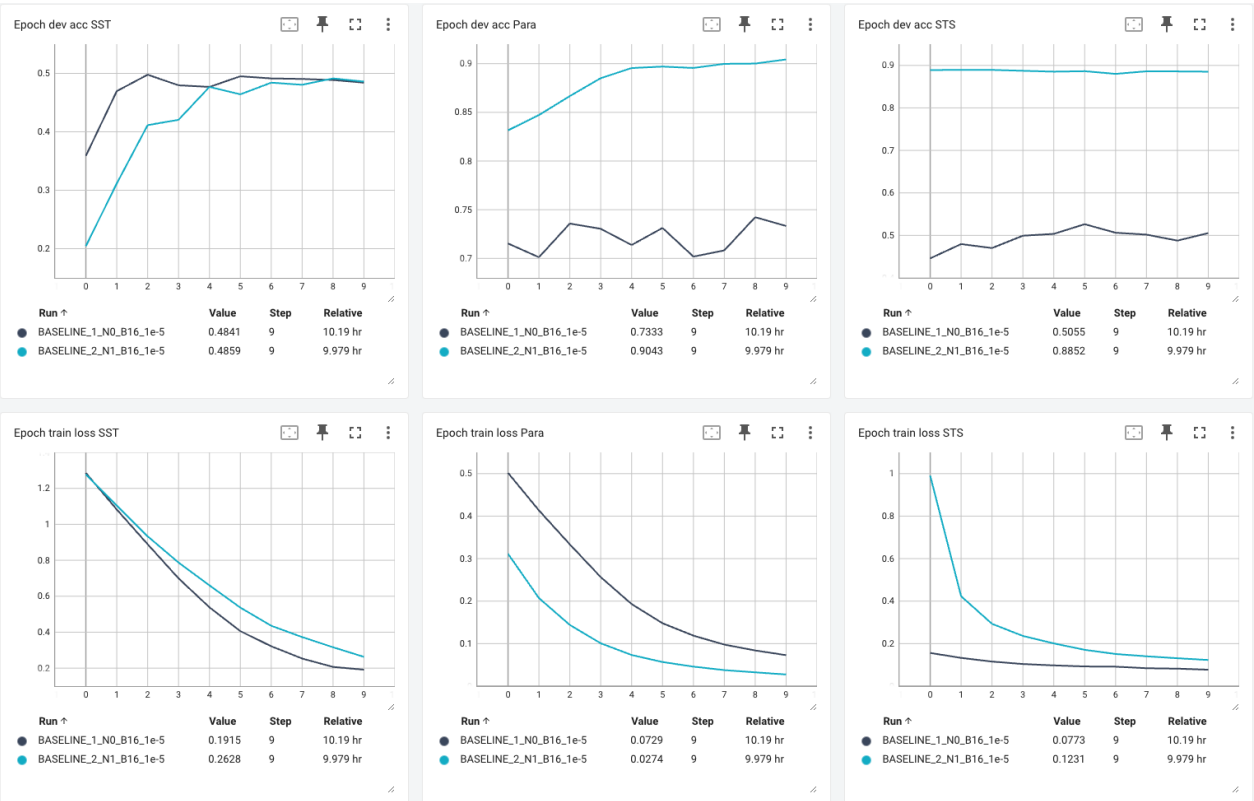


Figure 3: Dev accuracy and train loss per task for experiment 1 and 2 over 10 epochs (1 in black, 2 in blue)

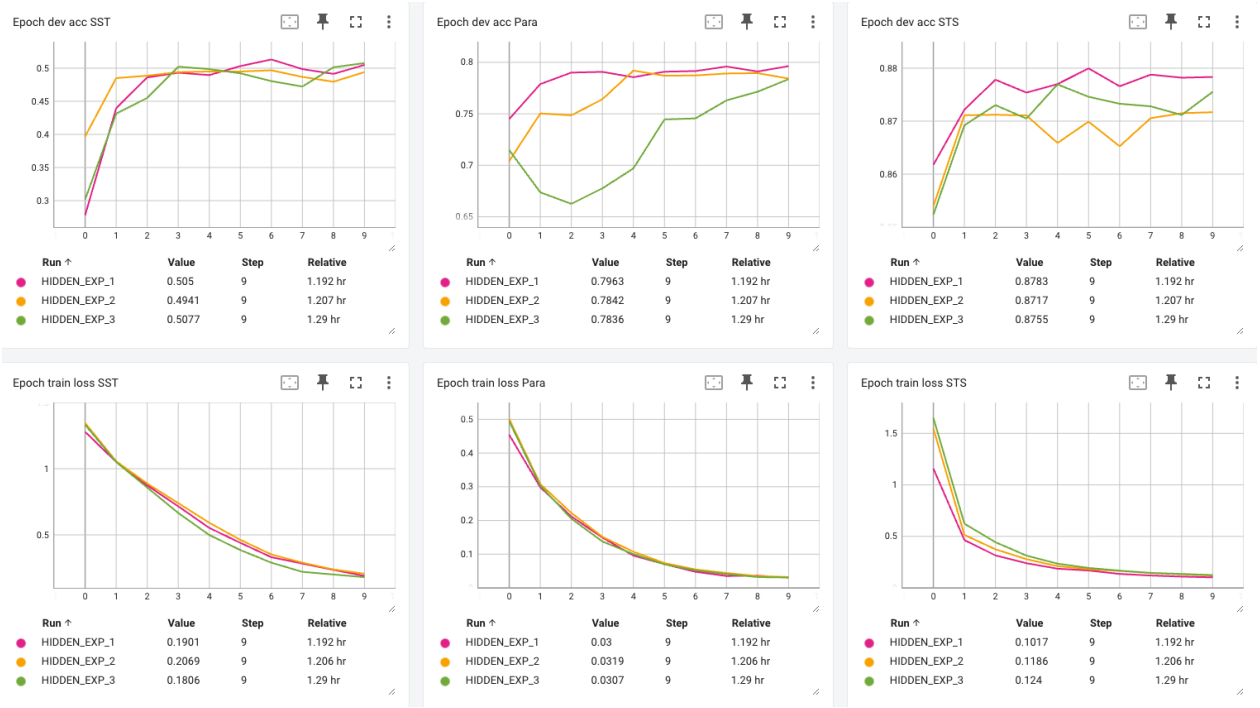


Figure 4: Dev accuracy and train loss per task for N hidden layers experiments 1, 2 and 3

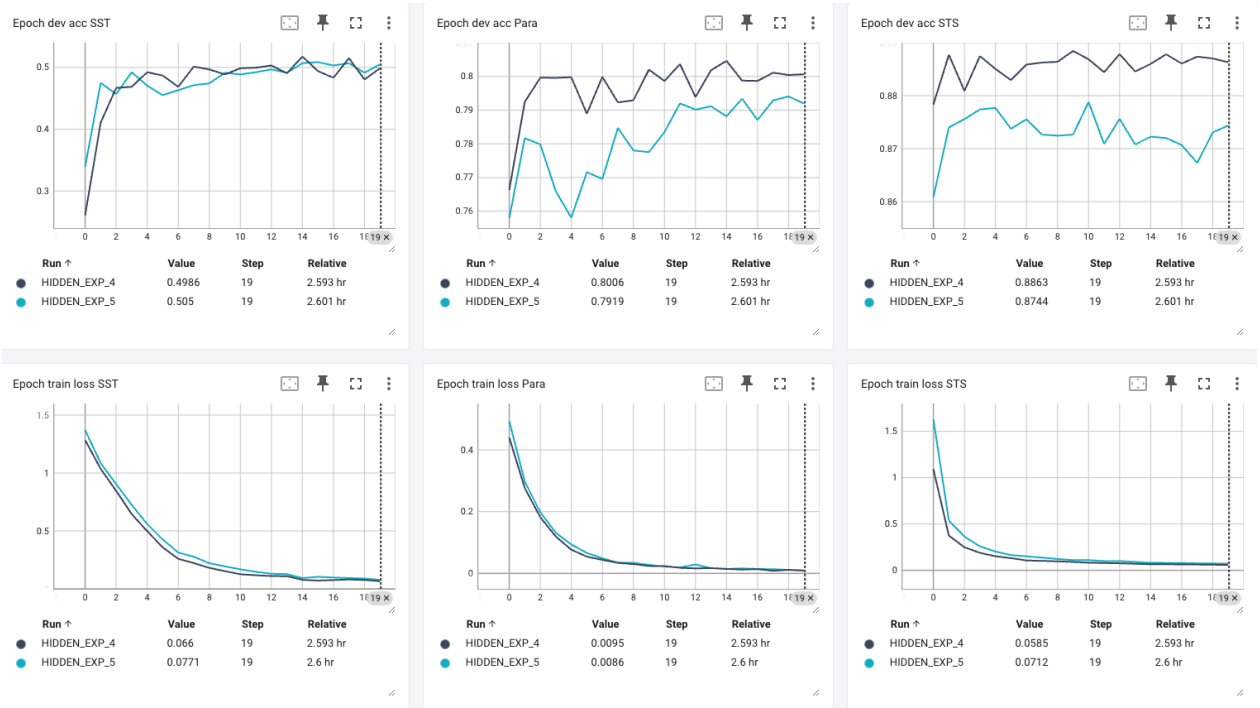
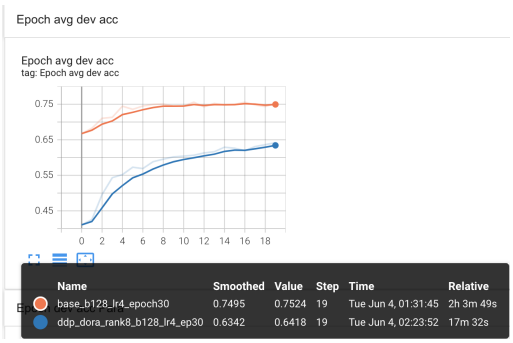


Figure 5: Dev accuracy and train loss per task for N hidden layers experiments 4 and 5

Machine types	vCPUs*	Memory (GB)	Max number of Persistent Disk (PDs)†	Max total PD size (TiB)	Local SSD	Maximum egress bandwidth (Gbps)‡	Tier 1 egress bandwidth (Gbps)
n1-standard-8	8	30	128	257	Yes	16	N/A

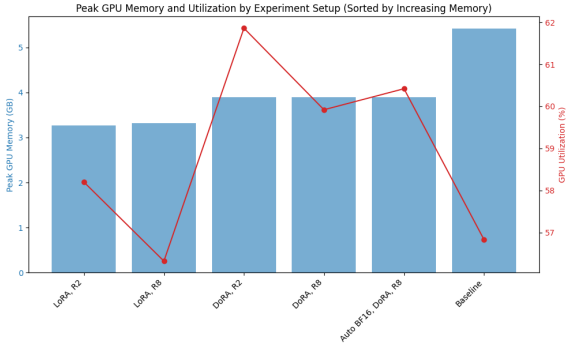
Figure 6: System specification used in training (Cloud (2024)) .



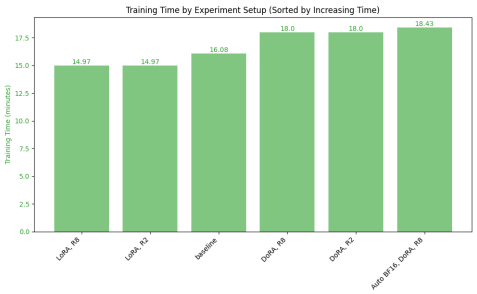
((a)) Baseline accuracy compared with DDP Dora accuracy.

((b)) Dora Dev accuracy compared with Lora rank 8.

Figure 7: Comparison of accuracy metrics with DDP



((a)) Peak GPU memory and GPU utilization



((b)) Training time comparison for each experiment setup

Figure 8: Resource usage in DDP experimental setups(R represents Rank, Auto means AMP Autocast)

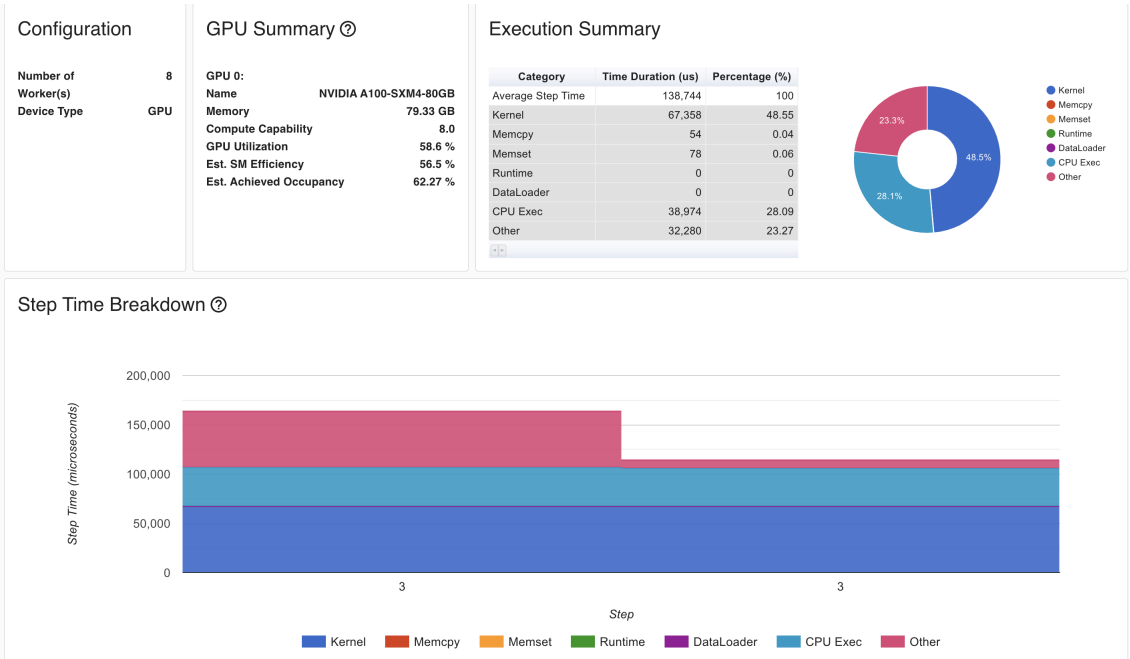


Figure 9: Pytorch profiling of GPU with DDP DoRA training with DDP Dora Rank=8 experiment .

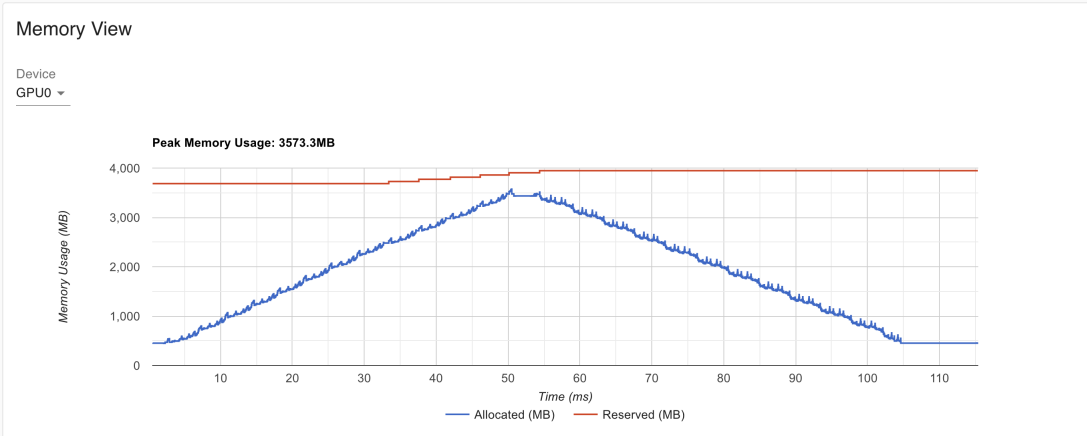


Figure 10: Memory usage with pytorch profiling during DDP Dora(rank=2) experiment .