NNTI Project Report Winter Semester 2025

1 Introduction

We utilized Google Colab for our implementation, leveraging an NVIDIA Tesla T4 GPU with 15.36GB of VRAM, 12GB of system RAM, and CUDA version 12.4.

2 Task 1

This task aims to fine-tune a pretrained chemical language model, MoLFormer, on a regression task to predict molecular lipophilicity. We explored both supervised fine-tuning and an unsupervised fine-tuning stage using Masked Language Modeling (MLM) to improve the model's understanding of SMILES representations. The objective of is to evaluate whether unsupervised fine-tuning using MLM can improve the performance of a pretrained model on a downstream regression task.

2.1 Data Description

The dataset used is the MoleculeNet Lipophilicity dataset, which contains SMILES strings and corresponding lipophilicity values. To ensure robust training and evaluation, the data was split into three sets:

- **Training samples:** 3024
- **Validation samples:** 336
- **Test samples:** 840

These splits were obtained by first partitioning the dataset into 80% for training/validation and 20% for testing, then further dividing the training/validation set into 90% training and 10% validation. Additionally, for tokenization, a maximum sequence length of 128 was chosen. This decision was based on an analysis of 1000 randomly selected SMILES strings, where the longest string had 117 tokens, ensuring that most strings are fully captured without excessive padding.

2.2 Method

Baseline Regression Fine-Tuning. We first add a regression head—a simple linear layer—to the pretrained MoLFormer model. The resulting model is then fine-tuned on the labeled training data using Mean Squared Error (MSE) loss.

Unsupervised Pretraining + Regression Fine-Tuning. In parallel, we perform unsupervised fine-tuning using the Masked Language Modeling (MLM) objective on SMILES strings (using combined training and validation data). After this self-supervised stage, we load the unsupervised fine-tuned base, attach the same regression head, and fine-tune the model on the regression task. This allows us to compare the performance of models with and without the additional unsupervised pretraining.

2.3 Results

We achieved these results by using a batch size of 16 and a learning rate of 1e-7, training the supervised model for 20 epochs and performing 10 epochs of unsupervised fine-tuning.

Table 1 shows that the finetuned regression model achieved a slightly lower Test RMSE (1.1009) compared to the baseline model (1.1352). This improvement indicates that incorporating the unsupervised fine-tuning step helped the model generalize better to unseen data.

Model	Test RMSE
Baseline Regression Model	1.1352
Finetuned Regression Model	1.1009

Table 1: Performance of Baseline and Finetuned Regression Models on the held out test set

In the figures 1 and 2 they represent the training and validation losses curves without and with the unsupervised finetuning respectively.

in 2 after applying the unsupervised fine-tuning using the MLM objective. Here, the loss starts at a

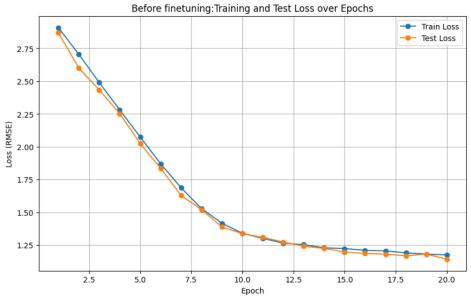

Figure 1: Baseline Training loss over epochs.

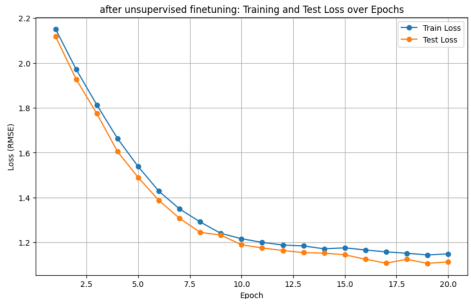

Figure 2: unsupervised Finetuned Training loss over epochs.

lower value and converges to a better final RMSE. This improvement implies that the additional unsupervised stage helped the model learn more robust representations of SMILES strings, enhancing its performance on the regression task.

3 Task 2

This tasks aims to find the most useful points from a secondary training dataset using Influence Function (Koh and Liang, 2017) and then fine tune the model from Task 1 with both the most influential points from the secondary dataset and the original train set described in 2.1.

3.0.1 Finding the Influential Points

The method used for selecting the most influential points is described in section 4.1.1. We used the unsupervised fine-tuned model from Task-1 as the pre-trained model and trained for regression head with both train data from section 2.1 and secondary data. Next we followed the method described in section 4.1.1 for finding the influence score for all the samples in secondary dataset. We use the loss gradient over full test set instead of separately using loss gradients for each test samples as suggested in (Koh and Liang, 2017) due to time limitation.

3.0.2 Results

In the secondary dataset, 161 samples resulted in positive influence score. We fine tuned the model similar to Task 1, using all the positive samples, top 100 positive samples and full dataset. The result is presented in table 2 which shows that using only the most influential points does improve model optimization significantly.

Data Selection	Test RMSE
Train + Secondary	1.0780
Train + Secondary (all positive)	1.1178
Train + Secondary (top 100 positive)	0.9426

Table 2: Performance of Baseline and Finetuned Regression Models on the held out test set

4 Task 3

4.1 Data Selection Strategies

In this section, we explain the data selection strategies used to find useful data points from the external dataset.

4.1.1 Influence Functions

Influence function estimates how much influence a training sample has on the model's predictions. The method descibed in (Koh and Liang, 2017) proposed computing influence without needing to retrain the model. Given a trained model, the effect of upweighting a training sample by a small amount is computed as $I_{\text{up,params}}(z) = -H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$, where $H_{\hat{\theta}}$ is the Hessian of the empirical risk. This allows the estimation of how removing a sample would alter predictions without needing to retrain the model. Since direct Hessian inversion is computationally expensive, the LiSSA approximation (Agarwal et al., 2017) which is based on a recursive formulation of Taylor expansion is used for approximating the inverse Hessian-vector product without directly computing the hessian or the inverse of the hessian.

4.1.2 TS-DShapley

TS-DShapley (Schoch et al., 2023) is a Shapley based data selection method that reduces the computational cost of Shapley based data valuation by introducing two key components. First, it trains a simple model using representations extracted from the target language model to approximate data values which avoids the need to retrain the full model. Second, it employs Monte Carlo sampling

on data subsets to estimate marginal contributions efficiently. The computed values are aggregated to approximate Shapley values for the entire training set. Harmful data points are identified by progressively removing instances with the lowest estimated contributions until optimal set of samples are found which then is used to fine tune the target model.

4.2 Fine-tuning Strategies

In this section, we explore three parameter-efficient fine-tuning techniques that update only a fraction of the parameters during training. The base model used in the following techniques was the unsupervised Fine-tuned model from task 1.

4.2.1 Full Fine-tuning

For baseline comparison, we also conducted experiment using full fine-tuning, where all model parameters were updated. This allowed us to benchmark our fine-tuning approaches versus the performance of full fine-tuning.

4.2.2 BitFit

Bitfit (Ben-Zaken et al., 2022) Updates only the bias parameters of the model and task-specific layers.and maintains most of the pre-trained weights, which reduces the computations overhead.

4.2.3 LoRa

LoRA (Low-Rank Adaptation) (Hu et al., 2021) is a method designed to efficiently fine-tune large language models by freezing the original model weights and introducing small trainable low-rank matrices within the self-attention layers. Instead of updating the full weight matrix W, LoRA decomposes the weight update into two much smaller matrices A and B, where $W' = W + AB$ and the rank r is significantly smaller than the model's dimension d, reducing the number of trainable parameters. This approach minimizes computational cost. By only optimizing the added parameters rather than modifying the entire model, LoRA enables efficient adaptation with lower memory usage and faster training.

4.2.4 IA3

IA3 (Liu et al., 2022) fine-tunes models by freezing all weights and introducing learned scaling vectors that modify intermediate activations. These vectors rescale the keys and values in attention layers and the intermediate activations in position-wise feed-forward networks.

4.3 Hyperparameters and Training

Key Hyperparameters

- **LoRA Rank:** 4
- **LoRA Alpha:** 8
- **LoRA Dropout:** 0.1
- **Learning Rate:** 1×10^{-7}
- **Epochs:** 20
- **Early Stopping:** Patience of 5 epochs with a minimum delta of 0.001

these values of rank and alpha introduce a balance by introducing enough flexibility for adaptation while keeping the number of trainable parameters very low, maintaining efficiency.

Optimizer and Scheduler We used Adam with a learning rate of 1×10^{-7}, combined with a linear learning rate scheduler that decays the learning rate linearly over epochs.

Early Stopping An EarlyStopping mechanism monitored the validation loss. Training would halt if the validation loss did not improve beyond a minimum threshold (*min_delta*) for a set number of epochs (*patience*).

Loss Metric We computed Mean Squared Error (MSE) during training and took its square root to report RMSE as our evaluation metric.

4.3.1 Results

Data Selection Strategy	Full Finetuning	BitFit	LoRA	IA3
No selection	1.0780	2.4351	**1.0972**	1.1514
Influence Method	**0.9426**	2.6366	1.1175	**1.1469**
TS-DShapley	1.2223	**2.3313**	1.1471	1.1494

Table 3: Comparison of Finetuning Strategies Across Data Selection Strategies.

Table 3 presents RMSE values across different fine-tuning and data selection strategies. where LoRA's performance remains very close to full finetuning across all data selection methods (e.g., 1.0972 vs. 1.0780 with no selection), indicating that LoRA retains much of the benefit of updating all parameters while only modifying a fraction of them. BitFit, by contrast, shows a more pronounced performance drop (e.g., 2.4351 vs. 1.0780 without data selection), which is expected since

BitFit trains only the bias parameters and therefore updates far fewer weights than the other approaches. IA3 also achieves results close to LoRA, significantly outperforming BitFit; for instance, under no selection, IA3's performance (1.1514) remains near LoRA's (1.0972) and still close to full finetuning (1.0780). A similar trend emerges with the Influence Method and TS-DShapley, highlighting IA3's ability to stay near LoRA's performance across different data selection strategies while still being more parameter-efficient than full finetuning. Despite the different selection methods, LoRA and IA3 remain consistently near full fine-tuning results, underscoring their robustness, while BitFit exhibits greater variability and lower performance overall.

Fine-tuning Method	**Time (minutes)**
Full Fine-tuning	29m
BitFit	18m
LoRA	18m
IA3	23m

Table 4: Computation Time for TS-DShapley Selected Points Across Different Fine-Tuning Methods.

In Table 4, we see that full fine-tuning is the most time-consuming at 29 minutes, which makes sense given it updates all parameters in the model. Both BitFit and LoRA require just 18 minutes, reflecting their highly parameter-efficient nature (they only update a small fraction of the weights). IA3 takes slightly longer (23 minutes), likely due to its intermediate level of parameter updates compared to BitFit and LoRA, but it still finishes more quickly than full fine-tuning. Overall, this underscores how parameter-efficient methods can significantly reduce training time compared to updating the entire model.

References

Naman Agarwal, Brian Bullins, and Elad Hazan. 2017. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(116):1–40.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language models. *arXiv preprint arXiv:2203.09517*.

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*.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *arXiv preprint arXiv:2205.05638*.

Stephanie Schoch, Ritwick Mishra, and Yangfeng Ji. 2023. Data selection for fine-tuning large language models using transferred shapley values. *arXiv preprint arXiv:2306.10165*.