Predicting Compiler Speedup via Fine-Tuned CodeRankEmbed + Autoencoder

Javier Sin Pelayo (Author)
Senior Computer Science Student
javier.sinpelayo@uri.edu

Marco Álvarez (Tutor)
Department of Computer Science and Statistics
Associate Professor
malvarez@uri.edu

Christian Esteves (Co-Tutor)
Department of Computer Science and Statistics
Lecturer (temp)
cesteves@uri.edu

Introduction

Modern compilers optimize code by applying a variety of transformation passes, such as loop unrolling, tiling, and distribution, to improve performance. However, these optimizations can lead to different intermediate representations (IR) that may look dissimilar yet be semantically equivalent. In practice, selecting the optimal set of compiler flags or transformations is challenging and time-consuming, often requiring extensive trial-and-error or exhaustive searches over thousands of possible transformations.

Our project aims to automate and accelerate this process by fine-tuning the "nomic-ai/CodeRankEmbed" [1] model (a large language model originally designed for code embedding) to generate semantic embeddings from original C source files extracted from loop groups. It can easily be used by the Hugging Face's PEFT package [2]. In our approach, these embeddings are combined with a vector representing the optimization flags applied (or proposed) during compilation. The concatenated vector is then fed into a multi-layer perceptron (MLP) that predicts the speedup achieved by those transformations. A new addition to take into account for the final report has been the use of an Autoencoder. This addition has provided promising results that will be discussed later.

This report summarizes:

1. **Key challenges & solutions**
    - **Semantic code embeddings**: fine-tune CodeRankEmbed via LoRA to efficiently learn loop semantics.
    - **Imbalanced classes**: adopt a binary classification setup to mitigate the 80/20 slowdown-to-speedup skew.
2. **Modeling approach**
    - A two-way classifier over "slowdown" (≤1×) vs. "speedup" (>1×).

    - Focal loss outperforms other kinds of weighted losses on our data.

3. **Extensive hyperparameter search**
   Learning rate, weight decay, batch size, optimizer (SGD vs. AdamW), scheduler (ReduceLROnPlateau vs default), dropout, and LoRA hyperparameters (rank *r*, α, dropout) were all tuned.

4. **Results**
   On held-out loop groups, our classifier achieves **≈ 88.46 % validation accuracy**, with a **speedup-class F1** around 71.6%.

Methods

In terms of the methods, I'll discuss the dataset details, model's architecture and the required setup for the training.

Dataset Details

- *Nature and Source of the Dataset:*
  Our dataset is built upon a repository of C source files representing loop nests extracted from popular benchmarks (similar to those described in LORE [3] and related work on LoopLearner). We will work with approximately 2,100 unique loops. These loops are transformed using various compiler optimizations, resulting in around 70,000 data points for training.

- *Data Components:*
    - **C Source Files:** Raw source code containing loop nests extracted from real-world applications and benchmarks.
    - **Optimization Flags Vector:** For each loop, a 56-dim multi-hot vector that encodes the set of compiler flags or transformation passes applied.
    - **Labels:** slowdown (≤ 1×) vs. speedup (> 1×).
- *Statistics:*
    - ~2,100 unique loops
    - ~70,000 total samples, 80∶20 slowdown/speedup
    - Stratified split by loop-group (80 / 10 / 10)

Model Architecture

I'll now describe the model's architecture step by step including all the dimensions and layers. It can also be addressed in the form of a flow diagram on *Figure 1*.

1. **CodeRankEmbed + LoRA**
    - SentenceTransformer (**`nomic-ai/CodeRankEmbed`**)
    - LoRA adapters applied to the Transformer's self-attention projections (**`Wqkv`** & **`out_proj`**) with rank r = 4, α = 32, dropout = 0.1.
    - Pool CLS-token → 768-dim

2. **Autoencoder Regularizer**
    - **Encoder**: 768 → 512 → BN → ReLU → 512 → 256
    - **Decoder**: 256 → 512 → BN → ReLU → 512 → 768
    - Adds MSE loss to encourage compact latent code

3. **Flag-Projection**
    - 55 → 32 linear → GELU → Dropout(0.2) → LayerNorm → 32 → 32 linear

4. **Classification Head**
    - Concatenate [256 (code) // 32 (flags)] → 288 dim
    - MLP:
        i. 288 → 144 → BN → GELU → Dropout(0.2)
        ii. 144 → 72 → BN → GELU → Dropout(0.2)
        iii. 72 → 2 → Softmax

Training Setup

- **Loss**: unweighted Focal Loss (γ = 2) on slowdown/speedup.

  Focal Loss is a one-term modification of the usual cross-entropy that *down-weights well-classified/predictable examples and focuses learning on harder, mis-classified ones*. In my two-way slowdown (class 0) vs. speedup (class 1) setup it looks like this:

  $$FL(p_t) = -(1 - pt)^{\gamma} log(p_t)$$

  - 'p_t' is the model's predicted probability for the true class (after softmax)
  - γ ≥ 0 is the focusing parameter (in my case, γ = 2)

  With this loss, the model pays more attention to the minority "speedup" class without having to hand-tune class weights.

- **Optimizer**: AdamW, LR=1e-3, weight_decay=1e-3
- **Scheduler**: ReduceLROnPlateau (factor = 0.5, patience = 3)
- **Batch size**: 128
- **Epochs**: 20

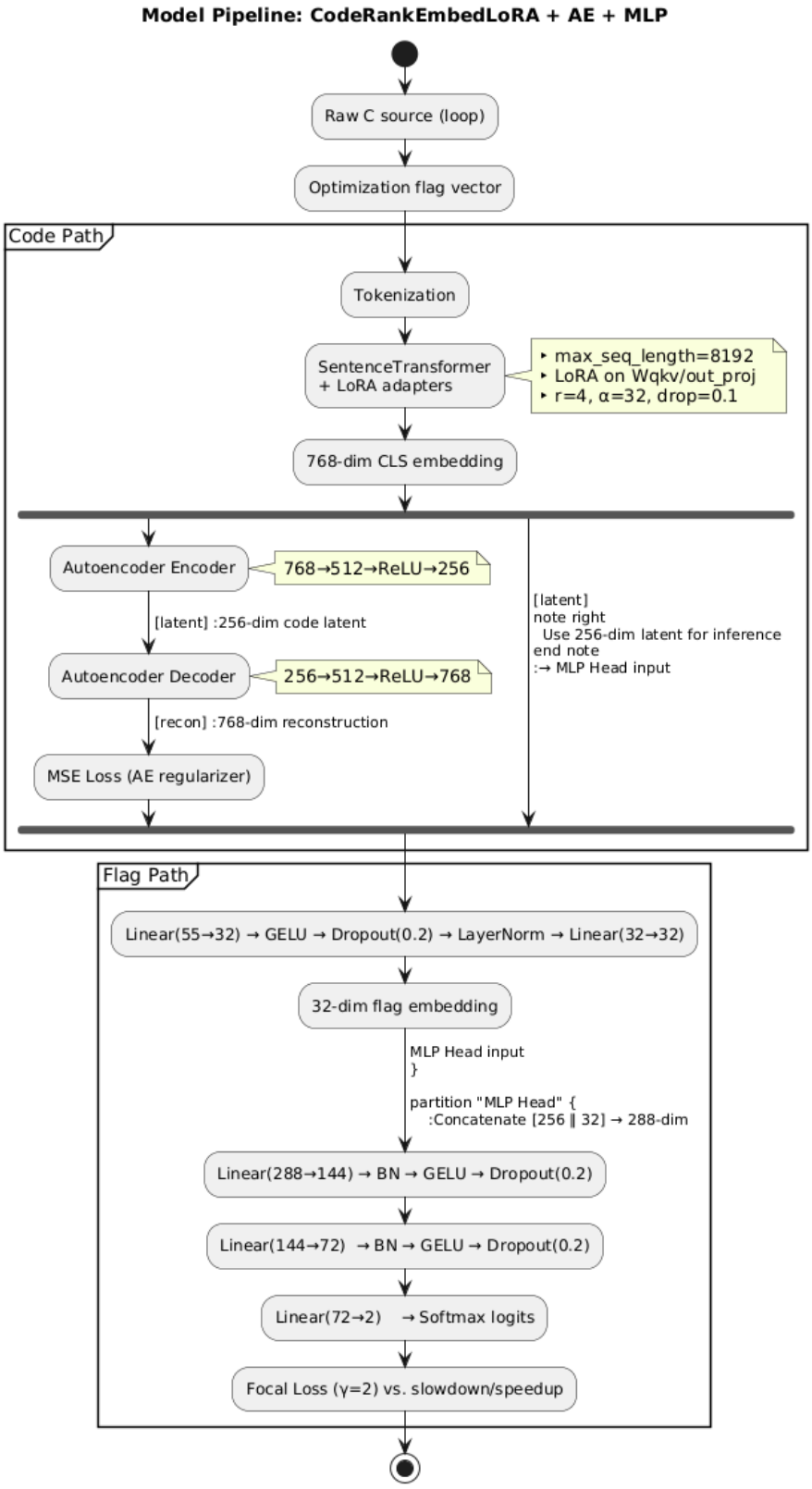

Figure 1: Model's Architecture

Experiments & Analysis

Data Processing

- Filter loop-groups to ≥ 3 transforms → 1,586 groups
- Token count limit ≤ 250
- Class balance: no oversampling (relied on focal loss)

Learning & Hyperparameter Search

- **Search over**
    - LR ∈ {1e-3, 1e-4, 1e-5}
    - Weight decay ∈ {1e-3, 1e-4, 1e-5}
    - Optimizer ∈ {SGD, AdamW}
    - Scheduler ∈ {None, ReduceLROnPlateau}
    - Dropout ∈ {0.1, 0.2, 0.3, 0.4, 0.5}
    - LoRA r ∈ {2, 4, 8}, α ∈ {16, 32, 64}, dropout ∈ {0.1, 0.2}
- **Best config**
    - AdamW, LR=1e-3, wd=1e-3, Dropout=0.2, LoRA(r=4,α=32,drop=0.1)

Results & Discussion

As I had to generate the embeddings by myself adding LoRA to the model, the training process takes over 1h 30min to finish with the available Unity resources thanks to the collaboration with the *University of Rhode Island*. This way, after lots and lots of experiments, I ran the model 5 times and then calculated the average of the runs to see a more representative result. The detailed output obtained can be found in Annex I: Notebook's Output, which is interesting if you want to see the loss evolution throughout the 20 epochs on each of the 5 runs, and the separate results of each run. Here's the summary of the results obtained:

- **5 runs averages (± stddev)**
    - **Val Acc**: 0.8846 ± 0.0444
    - **Speedup** F1: 0.7160 ± 0.0326
        - Precision: 0.7360 ± 0.0287
        - Recall: 0.6960 ± 0.0393
    - Slowdown F1: 0.9260 ± 0.0350
        - Precision: 0.9180 ± 0.0402
        - Recall: 0.9320 ± 0.0325

- **Comparison**
    - vs non-LoRA baseline (*Christian Esteves's approach*) :
        - + 9 pts on SU-Recall
        - + 3 pts on SU-Precision
        - + 7 pts on SU-F1
    - vs original paper (*Learning to Make Compiler Optimizations More Effective* [4]):
        - + 15 pts on SU-Recall
        - + 7 pts on SU-Precision
        - + 11 pts on SU-F1

Conclusion

- **Key takeaways**
    - LoRA-tuned CodeRankEmbed + AE + focal loss works best for 80/10/10 skew.
    - Achieved ~ 88.46% val accuracy, SU-F1 ~ 0.71.
- **Future work**
    - Extend to **multi-class** (more granular speedup buckets, adding the *Neutral* class and maybe differentiate speedups by magnitude).
    - Try **other LLMs** as the backbone such as *Code T5+* or *DeepSeek (Lite)*.

References

Now I'll include the references used for this project.
These include two important papers: "*LORE*" (which talks about the origin of the data), and "*Learning to Make Compiler Optimizations More Effective*" (which talks about the framing and precedent for ML-based prediction of compiler optimizations).

- **[1] nomic-ai. (2023). CodeRankEmbed: A Transformer-Based Code Embedding.** Retrieved May 2, 2025, from https://huggingface.co/nomic-ai/CodeRankEmbed

- **[2] Hugging Face. (2023). PEFT: Parameter-Efficient Fine-Tuning Library.** Retrieved May 2, 2025, from https://github.com/huggingface/peft

- **[3] LORE: A Loop Repository for the Evaluation of Compilers**
  Chen, Z., Gong, Z., Szaday, J. J., Wong, D. C., Padua, D., Nicolau, A., Veidenbaum, A. V., Watkinson, N., Sura, Z., Maleki, S., Torrellas, J., & DeJong, G. (2017). *LORE: A loop repository for the evaluation of compilers*. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)* (pp. 219–228).

- **[4] Learning to Make Compiler Optimizations More Effective**
  Mammadli, R., Selakovic, M., Wolf, F., & Pradel, M. (2021). *Learning to make compiler optimizations more effective*. In *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming* (pp. 9–20).

Acknowledgements

I would like to express my sincere gratitude to *Dr. Marco Álvarez* for inviting me to join this project and for his invaluable mentorship, insightful guidance, and unwavering support throughout my research. I also wish to thank *Christian Esteves* for generously sharing his deep expertise in compiler optimization, for rapidly orienting me to the project's technical context, and for his patient, timely answers to all of my questions. Their combined contributions have been instrumental to the success of this work.

Annex I: Notebook's Output

```
INFO: 2097 loop groups before filtering
INFO: 511 loop groups marked for removal
INFO: 1586 loop groups after filtering, 511 loop groups removed
INFO: dataframe columns: id, mutation_number, benchmark, application, file,
function, line, mutation_encoding, speedup, class_label, mut_fname, ori_fname
Source Loaded Correctly!

===== Run 1/5 =====
INFO: Use pytorch device_name: cuda:0
INFO: Load pretrained SentenceTransformer: nomic-ai/CodeRankEmbed
WARNING: <All keys matched successfully>
INFO: 1 prompts are loaded, with the keys: ['query']

CodeRankEmbedLoRA(
  (code_model): SentenceTransformer(
    (0): PeftModelForFeatureExtraction(
      (base_model): LoraModel(
        (model): Transformer({'max_seq_length': 8192, 'do_lower_case':
False}) with Transformer model: NomicBertModel
      )
    )
    (1): Pooling({'word_embedding_dimension': 768, 'pooling_mode_cls_token':
True, 'pooling_mode_mean_tokens': False, 'pooling_mode_max_tokens': False,
'pooling_mode_mean_sqrt_len_tokens': False,
'pooling_mode_weightedmean_tokens': False, 'pooling_mode_lasttoken': False,
'include_prompt': True})
  )
  (code_ae): Autoencoder(
    (encoder): Sequential(
      (0): Linear(in_features=768, out_features=512, bias=True)
      (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU()
      (3): Linear(in_features=512, out_features=256, bias=True)
    )
    (decoder): Sequential(
      (0): Linear(in_features=256, out_features=512, bias=True)
      (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU()
      (3): Linear(in_features=512, out_features=768, bias=True)
```

```
    )
  )
  (tr_projection): Sequential(
    (0): Linear(in_features=55, out_features=32, bias=True)
    (1): GELU(approximate='none')
    (2): Dropout(p=0.2, inplace=False)
    (3): LayerNorm((32,), eps=1e-05, elementwise_affine=True)
    (4): Linear(in_features=32, out_features=32, bias=True)
  )
  (head): MLPHead(
    (net): Sequential(
      (0): Linear(in_features=288, out_features=144, bias=True)
      (1): BatchNorm1d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): GELU(approximate='none')
      (3): Dropout(p=0.2, inplace=False)
      (4): Linear(in_features=144, out_features=72, bias=True)
      (5): BatchNorm1d(72, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (6): GELU(approximate='none')
      (7): Dropout(p=0.2, inplace=False)
      (8): Linear(in_features=72, out_features=2, bias=True)
    )
  )
)

Epoch  1  Train Loss: 0.123  Val Loss: 0.062
Epoch  2  Train Loss: 0.087  Val Loss: 0.063
Epoch  3  Train Loss: 0.078  Val Loss: 0.058
Epoch  4  Train Loss: 0.073  Val Loss: 0.082
Epoch  5  Train Loss: 0.069  Val Loss: 0.064
Epoch  6  Train Loss: 0.066  Val Loss: 0.055
Epoch  7  Train Loss: 0.064  Val Loss: 0.057
Epoch  8  Train Loss: 0.062  Val Loss: 0.063
Epoch  9  Train Loss: 0.060  Val Loss: 0.053
Epoch 10  Train Loss: 0.059  Val Loss: 0.064
Epoch 11  Train Loss: 0.057  Val Loss: 0.058
Epoch 12  Train Loss: 0.056  Val Loss: 0.056
Epoch 13  Train Loss: 0.055  Val Loss: 0.060
Epoch 14  Train Loss: 0.050  Val Loss: 0.064
Epoch 15  Train Loss: 0.049  Val Loss: 0.060
Epoch 16  Train Loss: 0.048  Val Loss: 0.060
Epoch 17  Train Loss: 0.048  Val Loss: 0.071
```

```
Epoch 18  Train Loss: 0.045  Val Loss: 0.072
Epoch 19  Train Loss: 0.045  Val Loss: 0.065
Epoch 20  Train Loss: 0.044  Val Loss: 0.066
```

Train Accuracy: 0.9385

Train Classification Report:

	precision	recall	f1-score	support
0	0.97	0.95	0.96	47950
1	0.82	0.88	0.85	11570
accuracy			0.94	59520
macro avg	0.89	0.91	0.90	59520
weighted avg	0.94	0.94	0.94	59520

Validation Accuracy: 0.9152

Validation Classification Report:

	precision	recall	f1-score	support
0	0.94	0.95	0.95	4115
1	0.78	0.74	0.76	922
accuracy			0.92	5037
macro avg	0.86	0.85	0.86	5037
weighted avg	0.91	0.92	0.91	5037

Test Accuracy: 0.8217

Test Classification Report:

	precision	recall	f1-score	support
0	0.85	0.90	0.88	2859
1	0.73	0.64	0.68	1213
accuracy			0.82	4072
macro avg	0.79	0.77	0.78	4072
weighted avg	0.82	0.82	0.82	4072

```
===== Run 2/5 =====
Epoch  1  Train Loss: 0.132  Val Loss: 0.061
```

```
Epoch  2  Train Loss: 0.093  Val Loss: 0.055
Epoch  3  Train Loss: 0.085  Val Loss: 0.051
Epoch  4  Train Loss: 0.079  Val Loss: 0.053
Epoch  5  Train Loss: 0.075  Val Loss: 0.053
Epoch  6  Train Loss: 0.072  Val Loss: 0.052
Epoch  7  Train Loss: 0.069  Val Loss: 0.051
Epoch  8  Train Loss: 0.063  Val Loss: 0.052
Epoch  9  Train Loss: 0.061  Val Loss: 0.052
Epoch 10  Train Loss: 0.061  Val Loss: 0.053
Epoch 11  Train Loss: 0.060  Val Loss: 0.053
Epoch 12  Train Loss: 0.057  Val Loss: 0.054
Epoch 13  Train Loss: 0.056  Val Loss: 0.056
Epoch 14  Train Loss: 0.055  Val Loss: 0.056
Epoch 15  Train Loss: 0.055  Val Loss: 0.058
Epoch 16  Train Loss: 0.052  Val Loss: 0.058
Epoch 17  Train Loss: 0.051  Val Loss: 0.060
Epoch 18  Train Loss: 0.051  Val Loss: 0.061
Epoch 19  Train Loss: 0.051  Val Loss: 0.064
Epoch 20  Train Loss: 0.050  Val Loss: 0.062
```

Train Accuracy: 0.9052

Train Classification Report:

	precision	recall	f1-score	support
0	0.95	0.93	0.94	40881
1	0.76	0.83	0.79	11471
accuracy			0.91	52352
macro avg	0.86	0.88	0.87	52352
weighted avg	0.91	0.91	0.91	52352

Validation Accuracy: 0.9170

Validation Classification Report:

	precision	recall	f1-score	support
0	0.95	0.96	0.95	6621
1	0.71	0.67	0.69	1066
accuracy			0.92	7687
macro avg	0.83	0.81	0.82	7687
weighted avg	0.92	0.92	0.92	7687

Test Accuracy: 0.9143

```
Test Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      7485
           1       0.69      0.67      0.68      1183

    accuracy                           0.91      8668
   macro avg       0.82      0.81      0.82      8668
weighted avg       0.91      0.91      0.91      8668
```

===== Run 3/5 =====

```
Epoch  1  Train Loss: 0.126  Val Loss: 0.061
Epoch  2  Train Loss: 0.089  Val Loss: 0.059
Epoch  3  Train Loss: 0.080  Val Loss: 0.059
Epoch  4  Train Loss: 0.075  Val Loss: 0.058
Epoch  5  Train Loss: 0.071  Val Loss: 0.056
Epoch  6  Train Loss: 0.069  Val Loss: 0.057
Epoch  7  Train Loss: 0.065  Val Loss: 0.058
Epoch  8  Train Loss: 0.064  Val Loss: 0.052
Epoch  9  Train Loss: 0.062  Val Loss: 0.056
Epoch 10  Train Loss: 0.061  Val Loss: 0.059
Epoch 11  Train Loss: 0.059  Val Loss: 0.058
Epoch 12  Train Loss: 0.058  Val Loss: 0.060
Epoch 13  Train Loss: 0.053  Val Loss: 0.055
Epoch 14  Train Loss: 0.052  Val Loss: 0.056
Epoch 15  Train Loss: 0.051  Val Loss: 0.056
Epoch 16  Train Loss: 0.051  Val Loss: 0.056
Epoch 17  Train Loss: 0.048  Val Loss: 0.059
Epoch 18  Train Loss: 0.047  Val Loss: 0.060
Epoch 19  Train Loss: 0.047  Val Loss: 0.062
Epoch 20  Train Loss: 0.046  Val Loss: 0.062
```

Train Accuracy: 0.9359

```
Train Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     45109
           1       0.83      0.86      0.84     11467
```

```
    accuracy                           0.94     56576
   macro avg       0.90      0.91      0.90     56576
weighted avg       0.94      0.94      0.94     56576
```

Validation Accuracy: 0.9087

```
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      5179
           1       0.74      0.69      0.72      1034

    accuracy                           0.91      6213
   macro avg       0.84      0.82      0.83      6213
weighted avg       0.91      0.91      0.91      6213
```

Test Accuracy: 0.7487

```
Test Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.78      0.83      4654
           1       0.43      0.65      0.51      1212

    accuracy                           0.75      5866
   macro avg       0.66      0.71      0.67      5866
weighted avg       0.80      0.75      0.77      5866
```

===== **Run 4/5** =====

```
Epoch  1  Train Loss: 0.126  Val Loss: 0.074
Epoch  2  Train Loss: 0.091  Val Loss: 0.087
Epoch  3  Train Loss: 0.081  Val Loss: 0.088
Epoch  4  Train Loss: 0.075  Val Loss: 0.094
Epoch  5  Train Loss: 0.071  Val Loss: 0.110
Epoch  6  Train Loss: 0.063  Val Loss: 0.111
Epoch  7  Train Loss: 0.061  Val Loss: 0.123
Epoch  8  Train Loss: 0.060  Val Loss: 0.104
Epoch  9  Train Loss: 0.059  Val Loss: 0.100
Epoch 10  Train Loss: 0.055  Val Loss: 0.105
Epoch 11  Train Loss: 0.054  Val Loss: 0.116
Epoch 12  Train Loss: 0.053  Val Loss: 0.129
Epoch 13  Train Loss: 0.053  Val Loss: 0.118
Epoch 14  Train Loss: 0.050  Val Loss: 0.126
```

```
Epoch 15  Train Loss: 0.049  Val Loss: 0.125
Epoch 16  Train Loss: 0.049  Val Loss: 0.117
Epoch 17  Train Loss: 0.048  Val Loss: 0.127
Epoch 18  Train Loss: 0.047  Val Loss: 0.126
Epoch 19  Train Loss: 0.047  Val Loss: 0.122
Epoch 20  Train Loss: 0.047  Val Loss: 0.119
```

Train Accuracy: 0.8773

Train Classification Report:

	precision	recall	f1-score	support
0	0.90	0.95	0.93	43468
1	0.73	0.58	0.65	10548
accuracy			0.88	54016
macro avg	0.82	0.77	0.79	54016
weighted avg	0.87	0.88	0.87	54016

Validation Accuracy: 0.8829

Validation Classification Report:

	precision	recall	f1-score	support
0	0.92	0.93	0.92	5054
1	0.75	0.74	0.74	1530
accuracy			0.88	6584
macro avg	0.84	0.83	0.83	6584
weighted avg	0.88	0.88	0.88	6584

Test Accuracy: 0.8501

Test Classification Report:

	precision	recall	f1-score	support
0	0.89	0.93	0.91	6442
1	0.66	0.53	0.59	1636
accuracy			0.85	8078
macro avg	0.77	0.73	0.75	8078
weighted avg	0.84	0.85	0.84	8078

```
===== Run 5/5 =====
Epoch  1  Train Loss: 0.129  Val Loss: 0.117
Epoch  2  Train Loss: 0.090  Val Loss: 0.127
Epoch  3  Train Loss: 0.081  Val Loss: 0.126
Epoch  4  Train Loss: 0.075  Val Loss: 0.119
Epoch  5  Train Loss: 0.071  Val Loss: 0.141
Epoch  6  Train Loss: 0.063  Val Loss: 0.131
Epoch  7  Train Loss: 0.061  Val Loss: 0.140
Epoch  8  Train Loss: 0.061  Val Loss: 0.154
Epoch  9  Train Loss: 0.059  Val Loss: 0.149
Epoch 10  Train Loss: 0.055  Val Loss: 0.169
Epoch 11  Train Loss: 0.055  Val Loss: 0.148
Epoch 12  Train Loss: 0.053  Val Loss: 0.161
Epoch 13  Train Loss: 0.053  Val Loss: 0.180
Epoch 14  Train Loss: 0.051  Val Loss: 0.185
Epoch 15  Train Loss: 0.051  Val Loss: 0.182
Epoch 16  Train Loss: 0.050  Val Loss: 0.191
Epoch 17  Train Loss: 0.049  Val Loss: 0.190
Epoch 18  Train Loss: 0.048  Val Loss: 0.195
Epoch 19  Train Loss: 0.048  Val Loss: 0.199
Epoch 20  Train Loss: 0.047  Val Loss: 0.198

Train Accuracy: 0.8826

Train Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93     43677
           1       0.67      0.78      0.72     10467

    accuracy                           0.88     54144
   macro avg       0.81      0.85      0.82     54144
weighted avg       0.89      0.88      0.89     54144

Validation Accuracy: 0.7993

Validation Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.86      3206
           1       0.70      0.64      0.67      1463

    accuracy                           0.80      4669
```

macro avg	0.77	0.76	0.76	4669
weighted avg	0.80	0.80	0.80	4669

Test Accuracy: 0.8644

Test Classification Report:

	precision	recall	f1-score	support
0	0.93	0.90	0.92	7998
1	0.61	0.71	0.66	1774
accuracy			0.86	9772
macro avg	0.77	0.81	0.79	9772
weighted avg	0.87	0.86	0.87	9772

===== Average over 5 runs =====
Avg Train Accuracy: 0.9079 ± 0.0257
Avg Train SU Precision: 0.7620 ± 0.0591
Avg Train SU Recall: 0.7860 ± 0.1084
Avg Train SU F1-score: 0.7700 ± 0.0756

Avg Train SD Precision: 0.9460 ± 0.0242
Avg Train SD Recall: 0.9400 ± 0.0179
Avg Train SD F1-score: 0.9440 ± 0.0136

Avg Val Accuracy: 0.8846 ± 0.0444
Avg Val SU Precision: 0.7360 ± 0.0287
Avg Val SU Recall: 0.6960 ± 0.0393
Avg Val SU F1-score: 0.7160 ± 0.0326

Avg Val SD Precision: 0.9180 ± 0.0402
Avg Val SD Recall: 0.9320 ± 0.0325
Avg Val SD F1-score: 0.9260 ± 0.0350

Avg Test Accuracy: 0.8398 ± 0.0546
Avg Test SU Precision: 0.6240 ± 0.1046
Avg Test SU Recall: 0.6400 ± 0.0600
Avg Test SU F1-score: 0.6240 ± 0.0659

Avg Test SD Precision: 0.9020 ± 0.0349
Avg Test SD Recall: 0.8920 ± 0.0591
Avg Test SD F1-score: 0.8980 ± 0.0407

Christian’s NON-LoRA approach results
===== Average over 5 runs =====
Avg Train Accuracy: 0.9035 ± 0.0189
Avg Train SU Precision: 0.7406 ± 0.0419
Avg Train SU Recall: 0.6777 ± 0.1128
Avg Train SU F1-score: 0.7050 ± 0.0787

Avg Train SD Precision: 0.9344 ± 0.0194
Avg Train SD Recall: 0.9503 ± 0.0092
Avg Train SD F1-score: 0.9422 ± 0.0108

Avg Val Accuracy: 0.8857 ± 0.0250
Avg Val SU Precision: 0.7028 ± 0.0460
Avg Val SU Recall: 0.6028 ± 0.0788
Avg Val SU F1-score: 0.6460 ± 0.0526

Avg Val SD Precision: 0.9164 ± 0.0240
Avg Val SD Recall: 0.9438 ± 0.0236
Avg Val SD F1-score: 0.9298 ± 0.0211

Avg Test Accuracy: 0.8489 ± 0.0627
Avg Test SU Precision: 0.6187 ± 0.1431
Avg Test SU Recall: 0.5759 ± 0.0680
Avg Test SU F1-score: 0.5853 ± 0.0925

Avg Test SD Precision: 0.9052 ± 0.0375
Avg Test SD Recall: 0.9097 ± 0.0671
Avg Test SD F1-score: 0.9060 ± 0.0431