Applying Fine-Tuning methods to FTTransformer in Anti Money Laundering applications

Vasco de Graaff[1]

Supervisor(s): Kubilay Atasu[1]**, Atahan Akyıldız**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 26, 2024

Name of the student: Vasco de Graaff
Final project course: CSE3000 Research Project
Thesis committee: Kubilay Atasu, Atahan Akyildiz, Burcu Kulahcioglu Ozkan

Abstract

This research investigates the effectiveness of combining Feature Tokenizer Transformer (FTTransformer)[6] with graph neural networks for anti-money laundering (AML) applications. We explore various fine-tuning techniques, including LoRA[9] and vanilla fine-tuning, on our baseline FTT architecture. Using the IBM AML dataset [1], we compare the performance of different models and fine-tuning approaches. Our results indicate that FTT alone do not outperform GNN's and careful configuration is required when working with datasets of Multi-Modality. This work contributes to the development of more efficient and accurate methods for detecting financial fraud patterns.

1 Introduction

The detection and prevention of money laundering have become increasingly critical in maintaining the integrity of global financial systems. As criminal organizations develop more sophisticated methods to conceal illicit financial activities, the need for advanced analytical tools has never been more pressing. Machine learning, particularly in the domains of Graph Neural Networks (GNNs) and Tabular Transformers, has shown significant promise in addressing these challenges. This research project explores the frontier of anti-money laundering (AML)[1] technology by combining the strengths of GNNs and Feature Tokenizer Transformers (FTT)[6]. We investigate the potential of this fused architecture to enhance the accuracy and efficiency of detecting complex money laundering patterns. Furthermore, we examine the impact of various fine-tuning strategies on model performance, with a particular focus on state-of-the-art techniques such as Low-Rank Adaptation (LoRA)[9].

1.1 Graph Neural Networks (GNNs)

Graph Neural Networks represent a cutting-edge approach to modeling and analyzing structured data. By representing data as nodes and edges within a graph, GNNs can capture intricate relationships and interdependencies that are often crucial in identifying suspicious financial activities. Recent advancements in GNN architectures have incorporated sophisticated techniques such as attention mechanisms and enhanced propagation algorithms. These improvements have led to deeper insights into data connectivity and topology, resulting in enhanced performance across various applications, from social network analysis to bioinformatics. In the context of AML, GNNs offer a powerful tool for understanding the complex web of transactions and relationships that may indicate illicit activities. The work done by the PyTorch Geometric team has significantly advanced the field, leading to several state-of-the-art results in multi-modality tabular learning [10]

1.2 Feature Tokenizer Transformers (FTT)

Feature Tokenizer Transformers represent another significant advancement in machine learning [6], particularly suited to handling large, complex datasets. FTTs combine feature tokenization, which processes input data into a format amenable to machine learning models, with transformer architectures. These transformers are designed to weigh the influence of different parts of data relative to each other, providing a robust framework for learning from diverse data modalities. In the realm of AML, FTTs offer the potential to efficiently process and analyze vast amounts of transactional data, identifying patterns and anomalies that may indicate money laundering activities. Additionally, Tabular Transformers have also shown great performance in Self Supervised applications, simplifying the feature engineering process as previously required when training deep neural networks. [16]

1.3 Fused Architecture

Our research initially explores a novel fused architecture that combines the strengths of both GNNs and FTTs. This approach aims to leverage the relational learning capabilities of GNNs with the powerful feature processing and attention mechanisms of FTTs. By integrating these technologies, we hypothesize that we can create a more robust and effective system for detecting complex money laundering patterns.

The fused architecture was explored to fill in the research gap following the previous results from GNN's in assess AML(Anti Money Laundering) applications from previous work [4]. Figure 1 is the baseline fused architecture given to us by the supervisor.

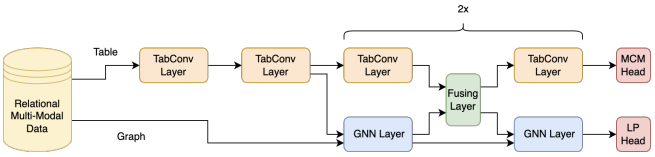

Figure 1: Fused architecture

1.4 Supervised Architecture

In this section we explore the architecture of the Supervised Training model as you can see in Figure 2. This was the model which was given by the supervisor to build on top of.

Encoder

In the decoder, the database entries get transformed. There are three outputs of this layer. The Embedding Encoder which translate the categorical inputs into an embedding, a Linear Encoder which translate numerical features and lastly a timestamp encoder.

Backbone

Next up is the backbone of the architecture, this is where most of the weight update happens and where the learning in the model happens.

Decoder

Lastly, we have the decoder - in this layer the multidimensional output of the backbone gets reduced into the training objective. In the self-supervised head, that is MCM(mask cell modelling), and LP (link prediction). On the other hand, the supervised head, we have a binary classification task, to predict whether or not the input transaction is Money Laundering.

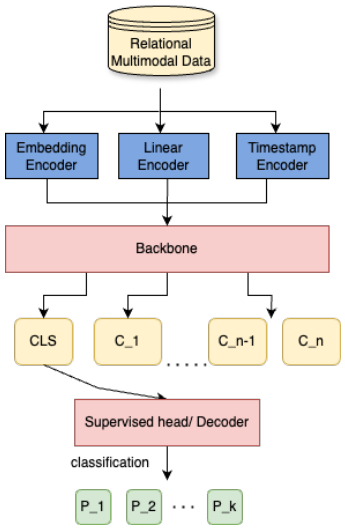

Figure 2: Base Supervised Model

1.5 Research Objective: Fine-tuning

We now look into fine-tuning, the specific technique which has proven to show effective results when trained on Deep Neural Networks [11].

1. What is the effectiveness of supervised training? GNN vs FTT
2. Explore Zero-shot (pre-training) performance
3. Compare performance of pre-training + full fine-tuning vs pre-training + freezing model backbone
4. How does LoRA fine-tuning perform?

2 Background

In this section, we explore previous work that has been done and look into ideas we plan to incorporate into our research. We specifically look into previous papers from the IBM AML paper [1] and the LoRA paper [9].

2.1 IBM anti-money laundering

A significant advancement in the domain of financial fraud detection, particularly in anti-money laundering (AML)[1], where a synthetic financial transaction dataset generator was created. This novel agent-based generator is designed to simulate realistic transactional behaviors in various banking scenarios, incorporating multiple currencies and transaction types to enhance the depth and applicability of training data for machine learning models. Crucially, the research introduces publicly available datasets that serve as a benchmark for comparing the efficacy of different AML detection models, including graph neural networks (GNNs) and gradient-boosted trees (GBT)[7]. By providing a controlled, reproducible environment for model training and evaluation, this work not only enriches the methodological tools available to researchers but also lays the groundwork for more robust and effective AML systems in practical settings, promoting a higher standard of transparency and reprehensibility in financial security research.

2.2 LoRA (Low Rank Adaptation)

Recently, LoRA has been proven to extremely efficient method of fine-tuning LLM (Large Language Models) to new downstream tasks [9]. LoRA works by using the assumption that Deep Neural networks, during fine-tuning, have a low "intrinsic rank" during the weighted updates. This means that the changes of the weights in the original backbone can be represented by a much smaller matrix. This works because of matrix decomposition as you can see below in the formula below.

$$h = W_0x + \Delta Wx = W_0x + BAx$$

The main benefit of LoRA is the increased memory efficiency and the significant reduction in total learnable parameters. Additionally, LoRA adapters can also be hot swapped between each other. This allows you to have multiple LoRA fine-tuned weights that each serve a different purpose and still keep using the same pre-trained model. This has significant downstream benefits when running inference. It allows the models to be swapped easily and quickly without significant memory overhead of loading a new model.

3 Methodology

In this section, we will look into the methodology in which we answer our sub-questions. We look into how different fine-tuning techniques perform when combined with pre-trained models. In this paper [5] the authors explore the benefits of self supervised training and how it has great downstream applications. This is very beneficial when it comes to fine-tuning the model into a new specific objective.

3.1 Vanilla fine-tuning

Vanilla fine-tuning is the standard approach to fine-tuning a pre-trained model, where all the parameters of the model are updated during the fine-tuning process. This technique involves initializing the model with pre-trained weights and then continuing the training process on a specific task and dataset, allowing all the parameters to be updated based on the new data. Importantly, its crucial to benchmark the performance of vanilla fine-tuning versus other more complex techniques.

3.2 Freezing backbone

In this approach, the backbone or the majority of the pre-trained model's layers are frozen, meaning their weights are not updated during fine-tuning. Only a small portion of the model, typically the final few layers, are fine-tuned on the target task and dataset. This technique is often used when the available data for fine-tuning is limited, as it reduces the risk of over fitting and allows the model to leverage the knowledge acquired during pre-training more effectively. This has also been explored in this paper here [15] in the context of object detection. It shows that it improves model performance while maintaining original learning within the backbone while fine-tuning the encoder and decoder.

3.3 Applying PEFT - LoRA

We implement LoRA as a Parameter Efficient Fine-Tuning (PEFT) technique in our baseline architecture. LoRA is integrated by adding low-rank adaptation matrices to the attention layers of our transformer backbone, allowing for efficient fine-tuning with reduced trainable parameters. We use a rank of 4 for LoRA, applied to backbone. This approach was chosen for its memory efficiency, quick adaptation potential, and possible improved generalization. Unlike vanilla fine-tuning, LoRA updates the model with minimal parameter changes, and compared to freezing the backbone, it offers more flexibility in adapting pre-trained knowledge. We expect this will reduce the total training time, thus reducing the training costs of the model.

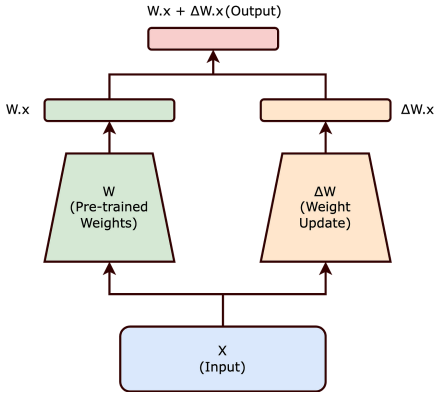

Figure 3: LoRA fine-tuned backbone

4 Experimental Setup

In this section, we look into the experimental setup in order to answer our sub-questions. We first present our evaluation metrics and then look into our fine-tuning objectives. Then we will explore the specific task we are fine-tuning and the evaluation framework for it.

4.1 Setup and Environment

We ran our experiments on the Delft Blue Supercomputer cluster, this consisted of A100 GPU's 80GB which allowed us to schedule tasks via SLURM. However towards the end of the project, queue times got too long and we had to start running experiments locally.A MBP M1 Pro 32GB gave decent computation times when not using the whole dataset. In terms of Software and Libraries, the full list can be found in our conda environment file in the project's repository, the most notable ones were Pytorch, pytorch_frame and wandb. We aim to publish our repository in the near future.

4.2 Dataset

As we have explored in the background, we will be using the IBM AML(anti-money laundering) dataset for these upcoming experiments. The dataset used is the HI_Small_Transaction. This dataset contains approximately 5 million rows, of which around 5000 are positive examples (i.e., transactions identified as money laundering). The dataset is highly imbalanced, with a ratio of 1000:1 for negative to positive examples.

The HI_Small_Transaction dataset includes various features relevant to financial transactions, such as transaction amount, timestamp, sender and receiver IDs, and transaction types. The data is synthesized to mimic real-world financial transactions, providing a controlled environment for testing machine learning models without compromising sensitive information.

4.3 Fine-tuning Task

Fine-tuning can be used to improve the current model further, or adjusting the model to another downstream task. Specifically, we look into using binary classification as the downstream task. This is done by first using the pre-training step, which has a training objective of doing finding the RMSE of the predicted masked columns. Then we fine-tune this model and adjust the task to do binary classification on the fraud dataset. This inherently is quite a difficult task due to the nature of the dataset. Since the IBM AMl dataset was constructed for fraud prediction on GNN's. It makes it quite tough to use the initial FTT code to run binary classification tasks on it.

4.4 Hyper parameter selection

Due to the limited time in this project, we weren't able to do a full hyper parameter search. As a result, we ended up using the baseline hyper parameters provided by our supervisor. In our fine-tune experiments, we keep the number of epochs to 10.

Hyperparameters	**Supervised**	**Self Supervised**
Batch Size	1024	1024
Channels	256	256
Hidden layers	4	4
Epochs	10	10
Learning Rate	1e-3	1e-3
Learnable Parameters	9M	12M

Table 1: Hyper parameter for each model

4.5 Evaluation Metrics

Since binary classification is our main focus, the F1 score is used as the standard metric. To define the components of the F1 score, consider the following formulas:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Accuracy in this case would be a bad metric due to the large class imbalance of the dataset [8]. For example if we predict all rows as 'Not money laundering', we would have an accuracy of over 99%. However, this can be done trivially and doesn't show that the model has learned anything.

In our pre-training tasks, we evaluate our progress using RMSE (root mean squared error) or accuracy. The formula for RMSE is below in equation 4, where n is the number of predictions, y_i is the actual value and $\hat{y}_i$ is the predicted value. Accuracy works here as we are comparing the predicted value with the masked value we applied during our pre-training step.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4}$$

4.6 Experiments

To answer our research question and the sub-questions, we look into exploring these experiments which should give us results into understanding how fine-tuning performs on these tasks.

Experiment 1: Supervised Learning
The first experiment investigates the performance of supervised learning techniques utilizing FTT and GNN's.

Experiment 2: Self Supervised Learning
Next, we assess the effectiveness of pre-training strategies, specifically focusing on the impact of these strategies on a secondary task, which involves predicting the RMSE for the MCM task.

Experiment 3: Fine-tuning
Finally, we explore the results of fine-tuning our models on these specific tasks, examining how freezing the backbone, oversampling the minority class and using PEFT techniques influences the overall model performance.

5 Results

Our experiments yielded results that, while not meeting initial expectations, provided valuable insights into the challenges of applying tabular transformers and fine-tuning techniques to the IBM AML dataset.

5.1 Supervised

The baseline FTTransformer model initially struggled with the highly imbalanced dataset, consistently predicting the majority class. As we can see in the results Table 2, the f1 score of FTT significantly lags behind GNN techniques. The precision of the model also suggest that it is over classifying thus having a precision score of 0.02459. Meaning that 2% of positive predictions are actually Money Laundering Transactions. One interesting result tho is that the recall score of the FTT is actually quite good, meaning that in actual fraud cases, its able to predict it over 90% of the time.

Comparing our results to the original GNN paper on the IBM AML dataset[1], we found that:

- The PNA (Principal Neighbour Aggregation) GNN model significantly outperformed (f1) our FTTransformer approach.
- The GINe (Graph Isomorphism Network extended) model also showed better performance (f1) than our baseline implementation.

5.2 Self Supervised

Our self-supervised pre-training approach showed promise in learning general features from the data, as you can examine in Table 3. The FTT self supervised model achieved an RMSE of 0.09267 and an accuracy of 0.8549 on the mask cell modeling task. While these metrics indicate good performance on the pre-training objective, they did not translate directly to improved performance on the money laundering detection task. Compared with the fused model scoring at 0.0432 for RMSE and 0.8017, we can see that the extra complexity of the extra layers did yield slightly better results.

5.3 Fine-tuning

We applied three fine-tuning methods to our self-supervised model: vanilla fine-tuning, frozen backbone, and LoRA. The results showed minimal differences across methods, with consistently low F1 scores (0.02840 to 0.03844), very low precision (0.0145 to 0.01981), and moderately high recall (0.6493 to 0.6828). These metrics suggest our models are overly sensitive, flagging many transactions as suspicious (high recall) but with poor accuracy (low precision). This pattern indicates struggles with the extreme class imbalance in the dataset and difficulties in effectively transferring pre-trained knowledge to the specific task of money laundering detection. The consistent performance across fine-tuning methods implies that the challenges may be more fundamental, potentially stemming from the mismatch between our model architecture and the graph-optimized nature of the IBM AML dataset. While the high recall is valuable for catching potential fraud, the low precision would lead to numerous false alarms in practice, highlighting the need for more sophisticated approaches to handle class imbalance and improve overall detection accuracy.

5.4 Reflecting on Research Objectives

What is the effectiveness of supervised training? GNN vs FTT
As explored in the Supervised section, the GNN model outperforms FTT model. This is likely due to the missing graph structure in the pre-training process of FTT. This is explored further in the discussion section.

Explore Zero-shot (pre-training) performance
Pre-training/Self Supervised learning performed moderately well at its task objective of MCM, we can see that we are achieving high accuracy and RMSE in Table 3.

Compare performance of pre-training + full fine-tuning vs pre-training + freezing model backbone
Between vanilla fine-tuning and the frozen backbone fine-tuning, there were negligible differences. Both fine-tuning methods struggled outperforming the base supervise model. My hypothesis is that there's not much transfer learning happening and most the learning happens at the decoder layer.

How does LoRA fine-tuning perform?
Similar to the other fine-tuning methods, LoRA did not show a significant improvement in performance metrics. The F1 scores and precision remained low, indicating that the method was not effective in addressing the challenges posed by the

dataset's class imbalance. However, LoRA offered some advantages in terms of training efficiency. The reduction in the number of trainable parameters resulted in slightly shorter training times compared to vanilla fine-tuning. Freezing the backbone did give faster performance since there are even less learnable parameters as a result.

6 Discussion

In this discussion section we explore the challenges in conducting these experiments and look into crucial learning that help us understand the results better. We first explore the issue with the baseline supervise model's loss function, then explore challenges with the dataset, look into why LoRA didn't perform as well as expected and the complications we had with the decoder head of the fused model.

6.1 Loss Function Exploration

Using the base model provided with cross entropy did not perform as expected. The model always predicted the minority class falsely as it predicted everything as being 'Not money Laundering'. After diving deeper into the code, it turns out the cross entropy loss function was not weighting in the weighted distribution. As a result, the update of the loss function was not performing as well as it should. Before this bug was found, I also explored other options such as Focal loss [13], which has shown promising results in the area of object detection models which large class imbalances. However, it wasn't necessary once the fix for the class imbalance was implemented.

6.2 Dataset

One significant limitation of this project was the dataset and the manner in which it was utilized by the base models. The extreme class imbalance of 1000:1 (False to Positive examples) in the training set posed a substantial challenge for making accurate predictions. In traditional finance applications, a more balanced dataset would be preferable to improve model performance.

During the pre-training step, we did not reconstruct the graph, which made it exceedingly difficult to achieve optimal results. The data was synthesized by generating one of the eight types of graphs depicted in Figure 4. As stated in the synthesis of the IBM AML dataset paper [1], this approach hampers the consistent prediction of money laundering patterns because the graph reconstruction process is not adequately addressed.

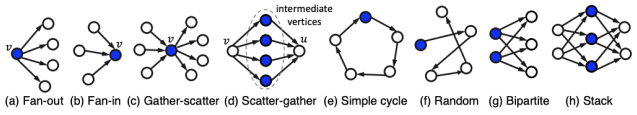

Figure 4: Synthesized Graph from IBM AML [1]

In Figure 5, we can examine 3 rows of the dataset, there is very little information which can be derived from the rows individually. We only know when the transaction happened, which node its from and going to, the amount, currency and

Timestamp	From Bank	From ID	To Bank	To ID	Amount Received	Receiving Currency	Amount Paid	Payment Currency	Payment Format	Is Laundering
1661984400.0(	B_10	0	B_10	0	0.2968	US Dollar	0.2968	US Dollar	Reinvestment	0
1661984400.0(	B_3208	1	B_1	399658	0.0004	US Dollar	0.0004	US Dollar	Cheque	0
1662049020.0(	B_22806	250026	B_6179	250027	0.3505	US Dollar	0.3505	US Dollar	ACH	1

Figure 5: IBM AML processed data

payment format. Since there's also only 1 1 categorical value, payment format, the model is likely to over-fit. As a result,it is extremely hard to classify whether or not a transaction is involved in money laundering without reconstructing the graph and detecting one of the graph structures as in Figure 4.

Upon discussing these results with my supervisor, it was suggested to explore the Amazon Fashion dataset used by other team members. However, the nature of this dataset made it unsuitable for this fine-tuning research. The Amazon Fashion dataset contained a single text column with user reviews and another column for the final rating. The remaining columns were metadata that did not contribute significantly to a robust experimental setup.

Given that the only learnable column was a large text field, a large language model would have been a better fit for classifying the review ratings. This approach, however, overlapped substantially with another teammate's research question. Due to the short timeline, an alternative dataset that better suited the requirements of this research was not identified in time.

6.3 LoRA

In this section we explore LoRA and try to understand the results we got. Looking at the results, we see that LoRA slightly outperforms full fine-tuning. Initially we'd expect that this would make a larger impact, however, after careful examination, we found out that most of the learnable parameters after fine-tuning the base self-supervised model, was in the encoder and decoder. As a result, this didn't make a big impact of the training time as expected. You can see this in Figure 5, there are slightly under 6M learnable parameters in the encoder and decoder, while the self supervised backbone only had 300,000.

```
FTTransformer                                                                          --
├─StypeWiseFeatureEncoder: 1-1                                                         --
│    └─ModuleDict: 2-1                                                                 --
│    │    └─EmbeddingEncoder: 3-1                                                     5,928,832
│    │    └─LinearEncoder: 3-2                                                        256
│    │    └─TimestampEncoder: 3-3                                                     7,296
├─FTTransformerConvs: 1-2                                                              128
│    └─TransformerEncoder: 2-2                                                         --
│    │    └─ModuleList: 3-4                                                           298,752
│    │    └─LayerNorm: 3-5                                                            256
├─SelfSupervisedHead: 1-3                                                              --
│    └─Sequential: 2-3                                                                 --
│    │    └─LayerNorm: 3-6                                                            256
│    │    └─ReLU: 3-7                                                                 --
│    │    └─Linear: 3-8                                                               129
│    └─ModuleList: 2-4                                                                 --
│    │    └─Sequential: 3-9                                                           3,930,886
│    │    └─Sequential: 3-10                                                          2,191
│    │    └─Sequential: 3-11                                                          1,159
│    │    └─Sequential: 3-12                                                          2,191
│    │    └─Sequential: 3-13                                                          2,039,875
==========================================================================================
Total params: 12,212,207
```

Figure 6: Learnable parameters from Self Supervised model

6.4 Decoder Head

Lastly in the discussion, we need to touch on the Decoder head of the Fused Architecture. The training object of the

fused model is to do MCM(Mask Cell Modelling) and LP (Link Prediction). These two objectives are the two outputs of the Self Supervised decoder. As you can imagine, These two training objectives are largely uncorrelated to money laundering patterns. LP at first thought may be related, however, in the way the base architecture implements link prediction, we mask the link and then try to predict whether or not the link exists in the first place. This does not give benefit to the training task of predicting money laundering patterns. As a result, we could not combine the decoder head of the fused architecture with the supervised decoder head to gain valuable results.

7 Conclusions and Future Work

This research paper explores the integration of FTT for anti-money launder(AML) applications using the IBM AML dataset, specifically, looking into various fine-tuning techniques. Our findings highlighted several key insights and challenges in applying these advanced machine learning models to detect money laundering activities. Most importantly, we learned that it is extremely important to first understand the nature of the dataset and understand its characteristics, additionally if it was synthesized, how this is done is also extremely important. This can have downstream effects when training the model if overlooked.

Future work should focus on addressing the limitations of FTT and exploring the necessity of graph reconstruction for training AML applications effectively. Additionally, there has also been a literature survey [2] done to understand better what machine learning methods are used in AML applications. Finally, we suggest potential questions for further research to further push the field forward.

- Investigate whether oversampling the minority class using SMOTE [3] would produce better results with the original dataset.
- Explore the benefits of fine-tuning GNNs specifically for AML applications, building on previous research on Adapter GNNs for fine-tuning [12].
- Examine how other fused architectures with different training objectives perform when fine-tuned, potentially extending the work of our teammates on various FTT and GNN fusion architectures.
- Compare the performance of tree-based learning methods such as XGBoost with transformer-based architectures in AML applications, considering recent findings that suggest simple tree models may still outperform more complex transformer architectures [7].

8 Responsible Research

In developing machine learning techniques for money laundering detection, we must consider the broader implications of our work. This section addresses the environmental impact of our experiments, potential biases in our approach, steps taken to ensure reproducibility, and ethical considerations. We recognize the significant societal impact of such technologies and aim to conduct our research responsibly, with a focus on transparency, fairness, and sustainability.

8.1 Environmental impact

Our research inevitably requires substantial computational resources and associated energy consumption due to the nature of training complex machine learning models. We acknowledge this environmental impact and have taken several steps to mitigate it:

Local development

We initially develop and test our models locally to ensure their functionality before deploying them on the cluster. We do this by running the model on an small version of the dataset (1/100 size). This approach significantly reduces unnecessary energy consumption from running faulty or inefficient code on high-performance computing resources.

Efficient cluster utilization

Efficient cluster utilization: Given the high load and potentially long queue times on the cluster, we optimize our code and experimental design to maximize the efficiency of our cluster usage.

Technological advancements

We recognize that compute efficiency has significantly improved over the past two decades, largely due to Moore's Law. This trend suggests that training equivalent models in the future will likely consume less energy and have a smaller environmental footprint. However, we also acknowledge that model complexity tends to increase over time, potentially offsetting these gains.

8.2 Bias

Addressing bias in AML detection models is crucial to ensure fair and equitable applications of the technology. In [14], data bias led to unfair treatment of specific racial/income groups. Knowing this, we've need to address potential sources of bias, though future work. We need to look into the distribution of the AML dataset and conduct research on the representation of the dataset to real world data. One big issue in open source AML data is that it is very scarce and limited. This is due to the nature of the application, issuing banks and payment service providers are hesitant to open source this data as it may give fraudsters an edge in developing new techniques to commit fraud without getting noticed.

8.3 Reproducibility

Ensuring the reproducibility of our research is essential for scientific integrity and to allow for further advancements in the field. The code for this project is available in our repository which comes with a README file to help you setup and reproduce our results. We aim to publish this repository in the near future. We used an open sourced dataset to conduct the research[1]. The hyper parameters used are also documented in the methodology section. Furthermore, all the logging of the results was done in wandb which can be explored further.

[1] See Appendix A for the dataset link.

Acknowledgments

I would like to thank my two supervisors, Atahan and Professor Atasu. This project would not have been possible without their support and direction. I am also very grateful to my fellow teammates, Ilias, Dragomir, Catalin, and Efe, and what we were able to achieve in such a short amount of time.

Lastly, I would like to express my deepest gratitude to my parents, who have continually supported me throughout my university journey and given me love and advice from the other side of the world in Thailand. They continue to motivate and inspire me to push myself every day in my personal, academic, and professional journey.

Appendix A: Dataset Source

The IBM AML dataset used in this research is available on Kaggle at: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml

References

[1] Erik Altman, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. Realistic synthetic financial transactions for anti-money laundering models, 2024.

[2] Nazanin Bakhshinejad, Reza Soltani, Uyen Nguyen, and Paul Messina. A survey of machine learning based anti-money laundering solutions, 10 2022.

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[4] Béni Egressy, Luc von Niederhäusern, Jovan Blanusa, Erik Altman, Roger Wattenhofer, and Kubilay Atasu. Provably powerful graph neural networks for directed multigraphs, 2024.

[5] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, pages 625–660, 2010.

[6] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting Deep Learning Models for Tabular Data, June 2021.

[7] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022.

[8] Hajo Holzmann and Bernhard Klar. Robust performance metrics for imbalanced classification problems, 2024.

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

[10] Weihua Hu, Yiwen Yuan, Zecheng Zhang, Akihiro Nitta, Kaidi Cao, Vid Kocijan, Jure Leskovec, and Matthias Fey. Pytorch frame: A modular framework for multi-modal tabular learning, 2024.

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

[12] Shengrui Li, Xueting Han, and Jing Bai. Adaptergnn: Parameter-efficient fine-tuning improves generalization in gnns, 2023.

[13] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[14] José Pombal, André F. Cruz, João Bravo, Pedro Saleiro, Mário A. T. Figueiredo, and Pedro Bizarro. Understanding unfairness in fraud detection through model and data bias interactions, 2022.

[15] Cristina Vasconcelos, Vighnesh Birodkar, and Vincent Dumoulin. Proper reuse of image classification features improves object detection, 2022.

[16] Wei-Yao Wang, Wei-Wei Du, Derek Xu, Wei Wang, and Wen-Chih Peng. A survey on self-supervised learning for non-sequential tabular data, 2024.

Model	Dataset	F1 Score ↑	Precision ↑	Recall ↑	Training Time ↓
FTTransformer	IBM 5M	0.02459	0.01246	0.9243	8h 32*
GNN(PNA)	IBM 5M	0.5603	0.5708	0.4854	13h 8m*
GNN(GINe)	IBM 5M	0.2386	0.2382	0.2390	9h 23m*

Table 2: Comparison of Supervised models
*Trained locally on MacBook pro M1

Model	Dataset	RMSE ↓	ACC ↑	Training Time ↓
FTTransformer	IBM 5M	0.09267	0.8549	4h 37m
Fused	IBM 5M	0.0432	0.8017	8h 22m

Table 3: Comparison of Self Supervised models

Finetune method	Model	Dataset	F1 Score ↑	Precision ↑	Recall ↑	Training Time ↓
Vanilla finetuning	FTT	IBM 5M	0.02840	0.0145	0.6611	2h 42m
Frozen backbone	FTT	IBM 5M	0.03844	0.01981	0.6493	2h 30m
PEFT (LoRa)	FTT	IBM 5M	0.03784	0.01946	0.6828	2h 34m

Table 4: Comparison of Fine-tuning methods

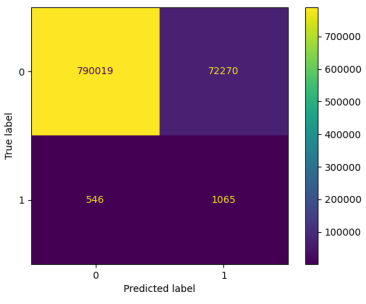

Table 5: Vanilla Fine-tune

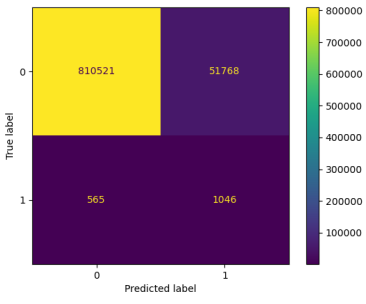

Table 6: Freeze Fine-tune

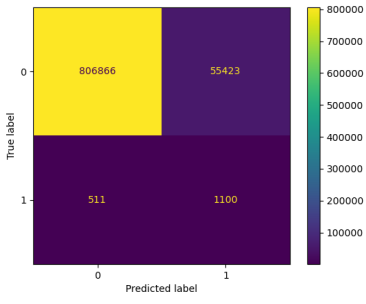

Table 7: LoRA Fine-tune