# How to improve the performance of the fused architecture consisting of a tabular transformer and a graph neural network used for representation learning for multimodal data?

**Dragomir Drashkov**
**Supervisors: Kubilay Atasu, Atahan Akyildiz**
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 23, 2024

Name of the student: Dragomir Drashkov
Final project course: CSE3000 Research Project
Thesis committee: Kubilay Atasu, Atahan Akyildiz, Burcu Ozkan

**Abstract**

The substantial amount of tabular data can be attributed to its storage convenience. There is a high demand for learning useful information from the data. To achieve that, machine learning models, called transformers, have been created. They can find patterns in the data, learn from them, and improve their predictive abilities based on that learning experience. There are also tabular transformers for tabular data. In order to attempt to increase the predictive performance of the transformers, we have combined them with graph neural networks (GNNs), which are again machine learning models, which work on graph data by learning information from the nodes and the edges. A graph representation of the dataset is created and input into the graph neural network. The architecture that fuses these two machine learning models is a more complex machine learning model that combines the transformer and the GNN. The aim is to increase the predictive ability of the model for values from the table or to predict whether an edge in the graph exists, which represents whether a transaction between two users exists. We have built the architecture using certain types of a tabular transformer and a graph neural network, FT-Transformer and GINe respectively, and the next step is to try modifying this architecture by using different models, and different ways of using these layers, for example how many copies we are creating of it. This has the potential to be a versatile model than can be used for different kinds of datasets. We have seen notable improvement in performance when using a different GNN, PNA. The transformer ResNet also shows to be on a similar or slightly better performing level than FT-Transformer when not combined with a GNN. GraphSage in the fused model underperforms significantly due to its weakness to capture simple graph structures.

# 1 Introduction

Tabular data is prominent, since it is a convenient way of storing information [3]. Graph-structured data is widespread as well, naturally occurring in many forms, such as social networks and molecular structures. Even problems that do not inherently involve graphs can often be transformed into graph problems, for instance bank transactions between users, which happens to be what is used in this research. Including this graph representation of the data, called a knowledge graph, has the potential of improving the accuracy of the predictions of the machine learning model.

To train a machine learning model to learn patterns in the data and then to be able to make predictions from unseen data, several methods have been developed - the most notable of which is XGBoost [4], since it is state of the art. Given that, the fact that deep learning models are not as powerful as tree-based models yet for tabular data [3, 19], and that deep learning favors multimodal data, we are trying to outperform it by using tabular transformers combined with graph neural networks.

Transformers [21] have been created in order to avoid using recurrence and convolutions. They rely solely on attention mechanisms, and have proven effective for making predictions on text. Tabular transformers build on transformers by being able to make predictions on tabular data. FT-Transformer [9] is an adaptation of the transformer architecture [21] for tabular data.

Graph neural networks (GNNs) [6, 13, 22] are able to produce informative results on graph data by using message passing between nodes [26]. The GIN (Graph Isomorphism Network) [22] is a neural architecture, a graph neural network, known for its effectiveness,

and GINe [13] is a GNN that builds on GIN. Both the transformer and the GNN can be used for representation learning on their own. In self-supervised learning [15], the model trains itself to learn one part of the input from another part of the input [1]. This is a kind of representation learning. The self-supervised learning objectives are masked cell modelling and link prediction for the transformer and and the GNN respectively. In order to test whether learning from the graph-like properties of the data can enhance the predictive abilities of the machine learning model, we have developed an architecture that combines these two architectures together. What follows is to explain what architecture decisions we have tried out and compare the performance of different architectures - different transformer and GNN models, different number and size of hidden layers. The experiments along with the corresponding results are presented.

# 2 Background

## 2.1 Graph neural networks

The goal of graph neural networks (GNNs) is to learn from the graph structure as much as possible. This happens through a neighborhood aggregation scheme, "where the representation vector of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes" [22]. The aggregation step could be a sum, minimum, maximum, average value, or any other function on the node vectors. While models like GraphSage [11] and GCN (Graph Convolutional Networks) [16] are popular, they have limited ability to capture simple graph structures [22]. This limitation highlights the need for more expressive architectures. Expressive power could be measure by, for example, graph isomorphism recognition ability or subgraph counting and connectivity learning [24]. Isomorphic graphs are those in which every node has the same neighbors in both graphs. The Weisfeiler-Lehman (WL) graph isomorphism test is used to determine if two graphs are isomorphic. The Graph Isomorphism Network (GIN) [22] is proposed as a GNN that matches the power of the WL test and is considered the most expressive GNN architecture.

GINe [13] builds on GIN by pretraining on both local and global neighborhoods to capture a broader range of structures. The graph-level representation is achieved by pooling nodes. The pretrain strategy of GINe on GIN gives state of the art results, while avoiding negative transfer [25], which refers to the undesirable inclusion of negative samples.

Principal Neighborhood Aggregation (PNA) [5] extends the graph neural networks focusing on graph isomorphism tasks and countable feature spaces by allowing the use of continuous features. PNA uses a combination of aggregators, since using only one aggregator cannot capture the distribution of the values of the nodes, and also uses a combination of scalers.

## 2.2 Transformers

Sequence transduction models, such as those used in natural language processing, typically rely on complex recurrent or convolutional networks that include an encoder and a decoder [21]. The most effective models also incorporate an attention mechanism, which helps to focus on relevant parts of the input sequence when generating outputs. However, the Transformer model introduces a novel approach by eliminating the need for recurrence

[1] Neptune.ai - Self-Supervised Learning and Its Applications

or convolutions. Instead, transformers utilize self-attention mechanisms, which assign a relevance score to each word based on its similarity to other words in the sequence. This approach not only improves quality but also reduces training time.

Building on the success of transformers, FT-Transformer (Feature Tokenizer + Transformer) [9] adapts this architecture, performing well on a wide range of tasks. In this model, the feature tokenizer converts input features into embeddings, which the Transformer then processes. This adaptation demonstrates the versatility of the Transformer architecture beyond its original application in language processing. Another innovative approach inspired by transformers is Trompt (tabular prompt) [3]. Trompt leverages the concept of prompt learning, which involves adjusting a large pre-trained model through a set of prompts outside the model without directly modifying the model. In addition to these transformer-based innovations, ResNet [9] is considered a strong baseline, known for its superior performance over traditional Multilayer Perceptrons (MLPs). The implementation details of ResNet are illustrated in equations 1, 2, and 3.

$$ResNet(x) = Prediction\left(ResNetBlock\left(\ldots\left(ResNetBlock\left(Linear(x)\right)\right)\right)\right) \tag{1}$$

$$ResNetBlock(x) = x + Dropout\left(Linear\left(Dropout\left(ReLU\left(Linear\left(BatchNorm(x)\right)\right)\right)\right)\right) \tag{2}$$

$$Prediction(x) = Linear\left(ReLU\left(BatchNorm(x)\right)\right) \tag{3}$$

## 2.3 Software libraries

PyTorch [18] is a deep learning library having many implementations of popular models. PyTorch Frame [14] and PyTorch Geometric [8] build on PyTorch and are used for feature learning on tables and graph respectively. PyTorch Frame contains implementations of popular tabular transformers and PyTorch Geometric contains implementations of popular graph neural networks.

## 2.4 Role of this project within the overall research

Acquiring labeled data for natural language processing and computer vision is challenging and resource-intensive [2]. Consequently, self-supervised learning is increasingly employed in these and other relevant fields [15]. The complete development pipeline for the model in this research involves several stages: designing the model, pretraining it [12], fine-tuning [20], training, and testing its performance. This paper focuses on analyzing architectural decisions, specifically the combinations of transformers and GNNs, and compares their performance outcomes.

## 2.5 Limitations

In related work, transformers have been used together with GNNs, but that model is impossible to be used for tabular data [23]. Despite the existence of numerous transformer models, there is a need to standardize performance measurement, to facilitate accurate and objective comparisons [9]. This could be addressed by using the standard benchmark for evaluating

[2]ENCORD BLOG Self-supervised Learning Explained

models [10]. Moreover, the theoretical properties and limitations of GNNs require better understanding [22]. In order to tackle this, a theoretical framework for analyzing the expressive power of GNNs has been proposed [22]. However, further improvement in analyzing the representational capacity of GNNs' is needed [22]. Additionally, a significant limitation is the necessity to use tables that can be represented with a knowledge graph.

# 3 Contributions

At the start of our project, it was possible to use FT-Transformer for self-supervised learning on the tabular dataset representation and GINe for link prediction on the graph dataset representation. Both models could be used individually or together in an architecture that uses them simultaneously. We added to the transformers the possibility to use ResNet and Trompt, where ResNet can function both independently and in the combined architecture, while Trompt can be used only on its own. For the GNNs, we added PNA and GraphSage, with PNA capable of both standalone and combined use, whereas GraphSage is restricted to the combined architecture.

Our findings revealed that the fused model with FT-Transformer and PNA outperforms others in two out of three metrics: it excelled in accuracy and root mean squared error (RMSE), though PNA alone is superior in mean reciprocal rank (MRR). Additionally, the performance of fused FT-Transformer and PNA model improves with the increase in the number of hidden channels, although at the cost of training speed. Notably, FT-Transformer and ResNet exhibit similar performance individually, but integrating FT-Transformer into the combined architecture significantly outperforms the integration of ResNet.

# 4 Architecture

Our architecture is heavily inspired by DRAGON(Deep Bidirectional Language-Knowledge Graph Pretraining) [23], but unlike DRAGON, it can handle tabular data. In that work, they use masked language modeling and link prediction, combined in a fusing layer. The difference for our architecture is that we use masked cell modeling, where cells in the table are masked and predicted, differing from the text-based input of masked language modeling. Figure 1 illustrates the workflow of our architecture with 3 layers: the first two are fused and in the last one the transformer and GNN are working in parallel, without combining their outputs. The dataset, in tabular format, is preprocessed and then input into the tabular transformer. A graph representation of the data, with users as nodes, is created for the GNN. The initial combination of a transformer and GNN, which is to be improved, is FT-Transformer and GINe. There is a Fused and a Parallel layer. In the parallel layer, the transformer and the GNN are working individually, while in the fused layer, their results are concatenated. Before data is input into the transformer, a transformer encoder, an StypeWiseFeatureEncoder, converts feature values into embeddings, mapping numerical values to a continuous space and categorical values to a discrete space. The Tab Conv in the figure is some functionality that defines how the transformer behaves, and could be, for instance, TransformerEncoderLayer, which applies the Transformer architecture [21] for every feature as described here [9], or TromptConv, which applies the Trompt cell [3]. The GNN Conv could be some convolutional operator or another object that defines how the GNN operates, such as GINEConv, which applies the modified graph isomorphism operator [13], or PNAConv, which applies the Principal Neighbourhood Aggregation graph convolution operator

[5]. For PNA the in-degrees of the nodes need to be calculated before it can be used. In the fusing layer the output from the transformer and the GNN undergo several operations, the crucial one of which is the concatenation of a part of the result of the Tab Conv with the result of the GNN Conv. The fusing layer returns the result of the transformer, the concatenated tensor, and the edge attributes. In the last layer, the output of the GNN goes into a link prediction head, which applies link prediction by performing operations on node and edge features and return positive and negative edge predictions. The result from the transformer goes into the masked cell modeling head - it contains numerical and categorical decoders for the respective type of feature. The results of the link prediction head and the masked cell modeling head are the result of one forward pass of the fused model.

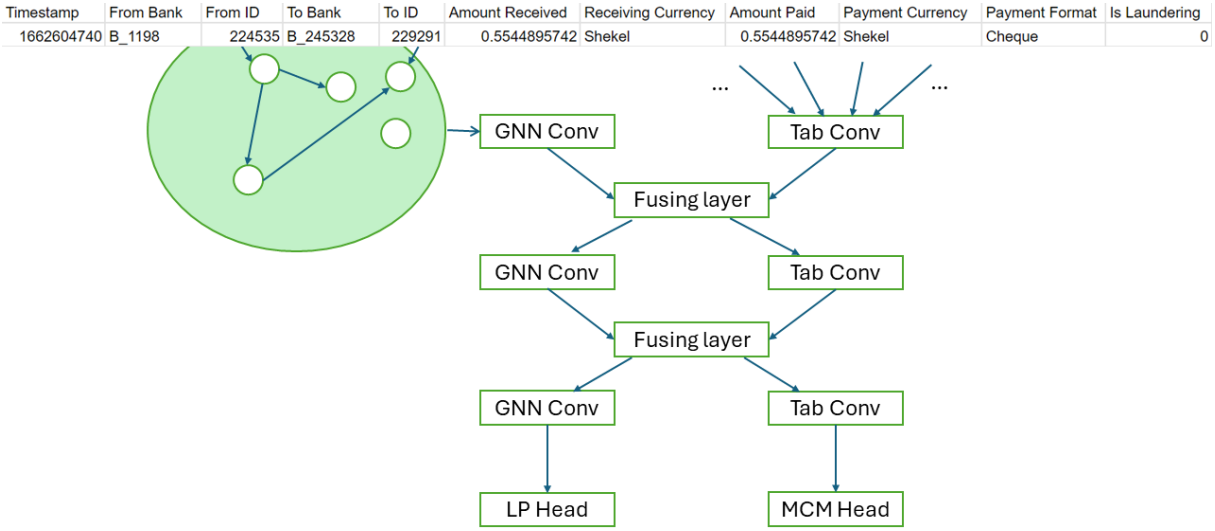


Figure 1: The fused architecture with 3 layers - first 2 are fused, last - not fused, but parallel. GNN Conv - some GNN convolutional operator. Tab Conv - some tabular transformer convolutional operator. LP Head - link prediction head. MCM Head - masked cell modeling head.

# 5 Experiments

In this section, we lay out the conduction of the experiments, including the preparation of the working environment and dataset, choice of the model building blocks, what we do in what order, how we evaluate the performance, and what hardware is used.

## 5.1 Setup

In order to be able to run the experiments, we cloned the project, and then installed the Conda environment, which installs most of the necessary packages, such as PyTorch (for deep learning) and Wandb (for plotting the results and sharing it with with others). Additionally, we pulled as a subproject a modified version of PyTorch Frame that includes masks. It is necessary to use Linux or the Windows Subsystem for Linux, as using other systems may make the installation extremely hard or impossible. Even though the supercomputer runs

on Linux, there are difficulties with installing PyTorch Geometric, so we use CMAKE to build PyG as a standalone C++/CUDA library following this tutorial [3]. Runs on the supercomputer have a significantly higher speed, but very big queue waiting times. Also, on the supercomputer we are able to use the whole 5M dataset, whereas locally, only a fraction of the dataset can be run in a reasonable amount of time.

### 5.2 Dataset

The dataset that is used is the IBM Transactions for Anti Money Laundering (AML) dataset [1]. Before using the data, preprocessing needs to be applied, including standardizing IDs and timestamps, prefixing bank names, dropping rows with missing data, normalizing numerical columns, and creating the graph. Afterwards, the dataset is split into train, test and validation sets. It contains a list of transactions, slightly over 5 million, and what the model is doing is it is predicting the masked cells and predicting whether a transaction between two users exists, where the transaction is represented by an edge in the graph. The table representation is input into the tabular transformer, while the graph representation is input into the GNN.

### 5.3 Architecture decisions

There are numerous combinations that can be tried in order to test their representation learning performance. Possible popular choices from the transformers are: FT-Transformer [9], ResNet [9], Trompt [3] , TabNet [2]. Popular choices among the graph neural networks are GIN [22], GINe [13], PNA [5], GraphSAGE [11], GCN [16]. Among those models, ResNet, Trompt, PNA, and GraphSage were integrated, while FT-Transformer and GINe were the starting point, they were available at the start at the project.

Multi-objective gradient correction (MoCo) [7], which enhances performance by selecting an optimal solution from the Pareto frontier, has also been implemented. This technique optimizes both transformer and GNN weights, leading to a better fused model performance. Additionally, we use the AdamW optimizer [17].

For the Trompt model, multiple runs with different numbers of prompts were conducted, yielding similar results across runs. This aligns with the findings in the Trompt paper [3], which indicate minimal performance differences between models with varying prompt numbers, and identify a single prompt as having the worst performance. Considering this, and the increase in runtime proportional to the number of prompts, eight prompts were found to offer the best trade-off between speed and performance.

### 5.4 Process

Initially, PNA was integrated into the fused model. It proved to be a better choice than GINe. This holds both for link prediction using PNA alone and the fused model, and also for the masked cell modeling combined with link prediction using the fused model (this could be observed in tables 1 and 2). Afterward, Trompt was used in place of FT-Transformer for self supervised learning. Unfortunately, its performance was slightly lower than FT-transformer for lower number of prompts. 128 prompts are promising, but the runtime is so significantly lower, that in the limit of 24 hours imposed by running on the supercomputer, not enough epochs are possible to be run. Afterward, GraphSage was integrated into the

[3] `https://github.com/pyg-team/pyg-lib/blob/master/.github/CONTRIBUTING.md`

fused architecture and ResNet was integrated into the self-supervised model. Around this time, the supercomputer stopped running our jobs. Therefore, we had to resort to alternative methods, such as training with a part of the dataset. That gave bad results, and luckily, the supercomputer got freed up slightly, and we could continue running with the full dataset.

## 5.5 Evaluation

The performance metrics used are hits@k, accuracy, root mean square error (RMSE), mean reciprocal rank (MRR). Hits@k and MRR are used to evaluate the performance of the GNN on the link prediction objective, while RMSE and accuracy are used for measuring the performance of the transformer and the fused model on the masked cell modeling objective. What follows is an explanation of each metric.

### 5.5.1 Hits@k

In link prediction, the goal is to determine whether an edge exists between two nodes in the graph. To achieve that, a neighborhood of nodes are sampled from the whole graph. From this neighborhood the positive edges are samples from the dataset. Negative edges are also created. These are edges that do not exist in the graph. The algorithm explores neighboring nodes by sampling positive edges and negative edges. The goal is that after learning during this process, when an edge is about to be predicted whether it exists or not, the positive should be predicted that they exist and the negative - that they do not exist. When evaluating, the hits@k metric is used. This means what fraction of the positive edges are ranked in the first k positions in the sample under consideration.

### 5.5.2 Mean reciprocal rank (MRR)

Mean reciprocal rank (MRR) (see equation 4) is related to hits@k, in that it averages all the reciprocal ranks that the model has assigned.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{4}$$

### 5.5.3 Accuracy

Accuracy measures the proportion of correctly predicted masked cells within the categorical features.

### 5.5.4 Root mean squared error (RMSE)

RMSE measures the extent to which the model's predictions for the masked cells of the numerical features deviate from the true values.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{5}$$

$n$**:** The number of observations.

$y_i$**:** The actual observed value.

$\hat{y}_i$**:** The predicted value.

### 5.6 Hardware

Empirically we have seen that the runtime grows more than linearly proportional to the size of the dataset. This is due to the linear increase if the model stayed the same, plus the increase in the model size due to the increase in the size of the encoders, plus the increase due to the runtime complexity of the algorithm. Nevertheless, the code is parallelizable, since there are vector and matrix operations, thus its runtime can be reduced by utilizing a GPU. This is possible by using the supercomputer Delft Blue [4]. On the supercomputer we can use 8 CPUs with 16GB of memory allocated per CPU, NVIDIA A100 / V100 Tensor Core GPUs, which allow parallel computations, since PyTorch tensors are used, which are multidimensional vectors or matrices. On the local machine we can use 2.6 GHz 6-core processor, 16GB RAM and NVIDIA Quadro T1000 GPU. This means the local machine is much less powerful, limiting the computational power we can get.

## 6 Results

In this section, we present comparisons of models grouped into different categories. All experiments are done on the full dataset, as the partial dataset does not yield representative results.

### 6.1 Best model

The fused model with FT-Transformer and PNA performs best on two out of three metrics: accuracy and RMSE. For MRR, the best performance is achieved by using PNA alone.

### 6.2 Best transformer and GNN

When FT-Transformer and PNA are used on their own for self-supervised learning and link prediction respectively, they outperform or do not perform worse than their counterpart models. This goes hand in hand with the best performing fused model combines the top transformer (FT-Transformer) and GNN (PNA).

### 6.3 Comparison of transformers

Trompt performs noticeably worse than both FT-Transformer and ResNet, see table 3. The most promising Trompt configuration uses 8 prompts. Increasing the number of layers does not seem to help, whereas increasing the number of prompts does slightly. FT-Transformer and ResNet have similar performance, alternating as the leader in accuracy and RMSE over epochs. However, integrating ResNet into the fused model results in significantly poorer accuracy and RMSE.

[4] `https://doc.dhpc.tudelft.nl/delftblue/`

## 6.4 Comparison of GNNs

PNA is the best-performing GNN, followed by GINe, with GraphSage performing significantly worse, as shown in table 4. This is expected, given the theoretical limitations of GraphSage in capturing simple graph structures.

## 6.5 Comparison of transformers in the fused model

Since RMSE for all runs during all epochs ranges from 0.088 to 0.098, the RMSE difference is significant. Since accuracy ranges from 0.83 to 0.86 for all runs except for the fused with ResNet, then fused with ResNet performs quite bad. See exact values in table 5.

## 6.6 Comparison of GNNs in the fused model

From tables 1 and 2, it could be inferred that the fused architecture performs significantly better when using PNA compared to GINe. This is true both for the link prediction and masked cell modelling self-supervised learning objectives. Since PNA uses multiple aggregators, compared to a single one in GINe, that allows PNA to capture the distribution of the values of the nodes better, leading to the better performance.

## 6.7 Comparison of hidden layer sizes

The fused model with PNA with 256 channels in the hidden layers has a higher MRR than PNA with 128 channels - see table 6. However, PNA with 128 channels outperforms PNA with 256 channels on accuracy and RMSE.

| | **Accuracy ↑** | **Hits@1 ↑** | **Hits@10 ↑** | **Hits@2 ↑** | **Hits@5 ↑** | **MRR ↑** | **RMSE ↓** |
|---|---|---|---|---|---|---|---|
| FT+GINe | 0.80589 | 0.54164 | 0.84305 | 0.69683 | 0.81309 | 0.66502 | **0.04272** |
| FT+PNA | **0.80670** | **0.65163** | **0.88345** | **0.7784** | **0.86145** | **0.74876** | 0.07411 |

Table 1: Fused architecture with masked cell modelling and link prediction results after 3 epochs. Notation: FT = FT-Transformer; MRR = mean reciprocal rank; RMSE = root mean squared error

| | **Hits@1 ↑** | **Hits@10 ↑** | **Hits@2 ↑** | **Hits@5 ↑** | **MRR ↑** |
|---|---|---|---|---|---|
| GINe | 0.65376 | 0.87777 | 0.78442 | 0.86219 | 0.75107 |
| PNA | **0.71286** | **0.89797** | **0.81986** | **0.88254** | **0.79265** |
| FT+GINe | 0.65248 | 0.87399 | 0.78131 | 0.85645 | 0.74901 |
| FT+PNA | 0.71227 | 0.88688 | 0.81499 | 0.87090 | 0.78895 |

Table 2: Link prediction results after 3 epochs. Notation: FT = FT-Transformer; MRR = mean reciprocal rank

| | FT-Transformer | ResNet | Trompt |
|---|---|---|---|
| Accuracy ↑ | **0.8553** | 0.8545 | 0.8497 |
| RMSE ↓ | 0.09194 | **0.09145** | 0.09253 |

Table 3: Comparison of different transformers.

| | GINe | PNA | fused-FT+GraphSage |
|---|---|---|---|
| MRR ↑ | 0.8106 | **0.8434** | 0.4721 |

Table 4: Comparison of different GNNs. The MRR of GraphSage is measured in the fused model, because we did not manage to integrate it in the link prediction algorithm. FT stands for FT-transformer.

| | FT+PNA | ResNet+PNA |
|---|---|---|
| Accuracy ↑ | **0.8544** | 0.7920 |
| RMSE ↓ | **0.0905** | 0.0929 |

Table 5: Comparison of different transformers in the fused model. FT stands for FT-transformer.

| | fused-FT+PNA-128 | fused-FT+PNA-256 |
|---|---|---|
| MRR ↑ | 0.7757 | **0.8056** |
| Accuracy ↑ | **0.8568** | 0.8561 |
| RMSE ↓ | **0.08915** | 0.08941 |

Table 6: Comparison of different hidden layer sizes for PNA. FT stands for FT-transformer.

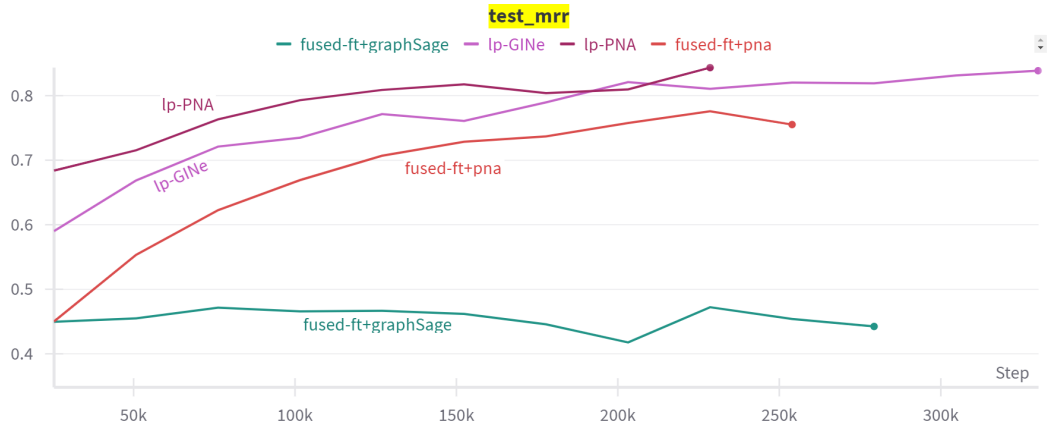


Figure 2: Mean reciprocal rank of the best performing models, and some worse performing for comparison. Names of the models start either with lp, standing for link prediction, and afterward the GNN model is specified, or fused - for the fused model, followed by a specification of the transformer and GNN. "ft" stands for FT-Transformer.

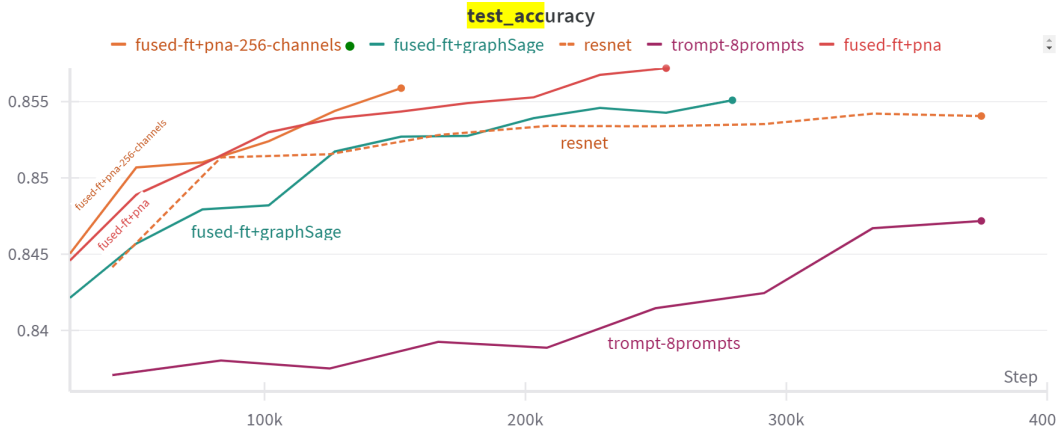


Figure 3: Accuracy of the best performing models, and some worse performing for comparison. Names of the models start either with fused - for the fused model, followed by a specification of the transformer and GNN, or consist of the transformer name, which itself can contain details about its implementation. "ft" stands for FT-Transformer.

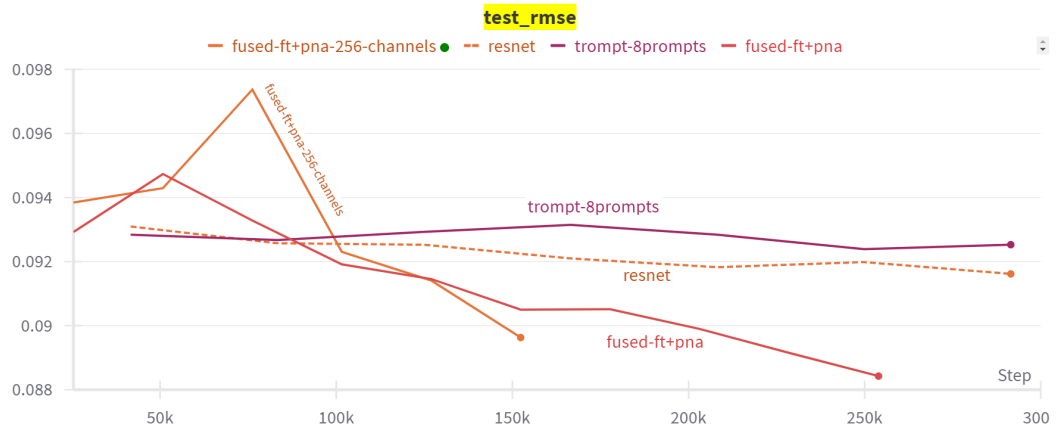


Figure 4: Root mean squared error of the best performing models, and some worse performing for comparison. Names of the models start either with fused - for the fused model, followed by a specification of the transformer and GNN, or consist of the transformer name, which itself can contain details about its implementation. "ft" stands for FT-Transformer.

# 7 Responsible Research

The dataset used in this paper is the IBM Transactions for Anti-Money Laundering (AML) dataset, available on Kaggle. This dataset is popular, as indicated by its statistics on the website, enhancing our confidence in its credibility and suitability for our research. It can also be used by other researchers for further building on this research question. Even more, it is being used for supervised learning using GNNs [6], which is closely related to our work. We have explained the architecture and the workings of the model so that it could be understood and built on by researchers. Moreover, if needed and if it is possible, we can provide the repository allowing for further development using other transformers and GNNs.

Since we are attempting to improve the predictive abilities of a machine learning model on detecting money laundering, this has the potential to create positive impact on society. If researchers use the model for other purposes with a huge societal impact, that would be even more beneficial for people. Since graph structured data is common and naturally occurring, many fields, such as medicine could benefit from this work.

While the power consumption associated with training is not excessive, it is still significantly more than what an average computer uses, therefore we have taken measures to avoid unnecessary computations by ensuring that every experiment is purposeful. Lastly, improving the algorithm's efficiency is necessary to reduce electricity consumption and save users' time. Enhancements in efficiency will also enable the model to run on less powerful machines, increasing its accessibility.

# 8 Future work

To enhance the reliability of the architectural analysis, it is essential to run additional epochs using the the full dataset. The model demonstrates learning capabilities up to at least the 15th epoch. Therefore, extending the training period for all architectures to this duration will ensure that the models are well-trained and approaching their maximum potential. In addition, for statistical significance, it is crucial to run multiple iterations of each experiment and average the results.

Furthermore, Trompt outperforms FT-Transformer in specific scenarios [3], thus the Trompt model requires parameter tuning to identify its optimal performance configuration, because there is variation for different number of prompts and number of layers, or it might be that our dataset is less favorable for Trompt compared to FT-Transformer.

Currently, we have only tested on a transactions dataset. While this dataset type is common, it is important to test on various other types to ensure the architecture's versatility. Ideally, the architecture should perform well across different datasets.

The algorithm's current efficiency prevents timely completion when using larger datasets, restricting us to the HI-Small_Trans.csv dataset, which contains around 5 million rows. Larger datasets have significantly more rows, necessitating substantial algorithm optimizations. This limitation may hinder the model's learning capacity, as larger datasets generally provide more learning opportunities. Optimizations should include eliminating inefficient actions in the code, such as unnecessary data transfers between the CPU and GPU. Other aspects of data handling also require attention [5].

[5] https://towardsdatascience.com/7-tips-for-squeezing-maximum-performance-from-pytorch-ca4a40951259

What is more, integrating XGBoost [4] into the model could be beneficial. XGBoost has proven to outperform deep models. It is worth trying integrating it into the model, since an ensemble of XGBoost with deep models gives superior results among a set of models [19]. XGBoost could either replace components like the transformer or GNN or be used alongside them.

Lastly, to ensure robust performance measurement, a standard benchmark is proposed [10] for example for the evaluation of Trompt [3], one of the tabular transformers which we discussed. Integrating this into our workflow would facilitate objective performance comparisons with other models, allowing researchers and developers to benchmark their models against ours.

# 9 Conclusion

There are machine learning models, called tabular transformers, which can make predictions on an unseen and unknown part of the data, after having learned on other parts of the data. There are also graph neural networks, which can perform link prediction, node classification or any other task they are trained on. Combining tabular transformers with graph neural networks (GNNs) can enhance the predictive abilities on tabular data. The dataset that we use consists of bank transactions, and we are training the model to predict the values in the table, a process called self-supervised learning. We have discovered better-performing models than the starting point, consisting of a tabular transformer and a GNN - the FT-Transformer and GINe fused architecture. Among the GNNs, PNA outperforms GINe, while GraphSage significantly underperforms compared to the other GNNs. While ResNet serves as a viable alternative to FT-Transformer for standalone use, the optimal performance is achieved by the fused model incorporating both FT-Transformer and PNA. This performance is significantly better than the starting point. The are more improvements that can be made, such as running more epochs, running the experiments more times for statistical significance, improving the efficiency of the algorithm, generalizing the model to perform well on more kinds of datasets.

# 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. *Advances in Neural Information Processing Systems*, 36, 2024.

[2] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[3] Kuan-Yu Chen, Ping-Han Chiang, Hsin-Rung Chou, Ting-Wei Chen, and Tien-Hao Chang. Trompt: Towards a better deep neural network for tabular data. *arXiv preprint arXiv:2305.18446*, 2023.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[5] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[6] Béni Egressy, Luc Von Niederhäusern, Jovan Blanuša, Erik Altman, Roger Wattenhofer, and Kubilay Atasu. Provably powerful graph neural networks for directed multigraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11838–11846, 2024.

[7] Heshan Fernando, Han Shen, Miao Liu, Subhajit Chaudhury, Keerthiram Murugesan, and Tianyi Chen. Mitigating gradient bias in multi-objective learning: A provably convergent approach. International Conference on Learning Representations, 2023.

[8] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[9] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[10] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[12] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International conference on machine learning*, pages 2712–2721. PMLR, 2019.

[13] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[14] 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. *arXiv preprint arXiv:2404.00776*, 2024.

[15] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[19] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[20] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[22] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[23] Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D Manning, Percy S Liang, and Jure Leskovec. Deep bidirectional language-knowledge graph pretraining. *Advances in Neural Information Processing Systems*, 35:37309–37323, 2022.

[24] Bingxu Zhang, Changjun Fan, Shixuan Liu, Kuihua Huang, Xiang Zhao, Jincai Huang, and Zhong Liu. The expressive power of graph neural networks: A survey. *arXiv preprint arXiv:2308.08235*, 2023.

[25] Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. A survey on negative transfer. *IEEE/CAA Journal of Automatica Sinica*, 10(2):305–329, 2022.

[26] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

# Appendix A - full sets of runs

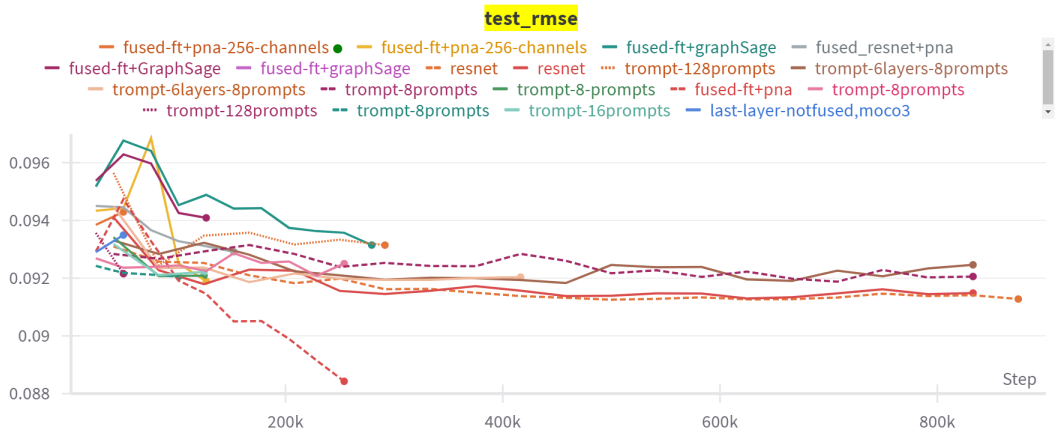


Figure 5: Root mean squared error or models

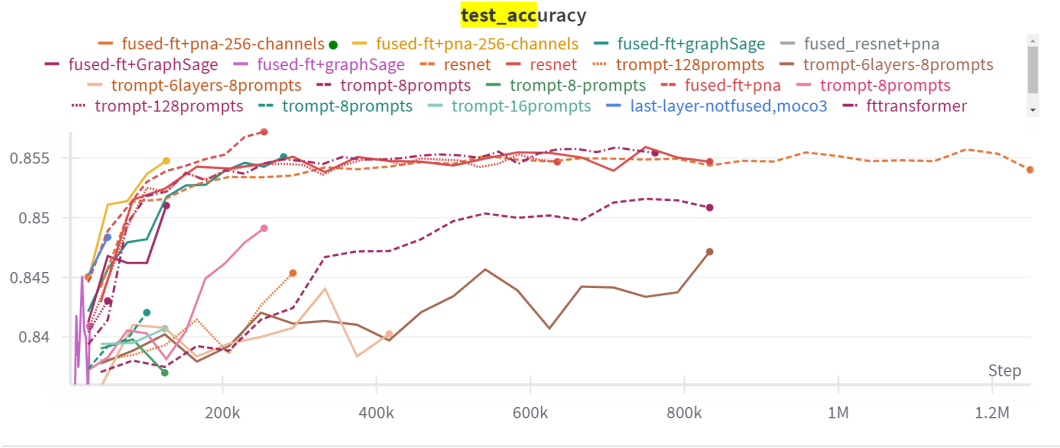


Figure 6: Accuracy of models

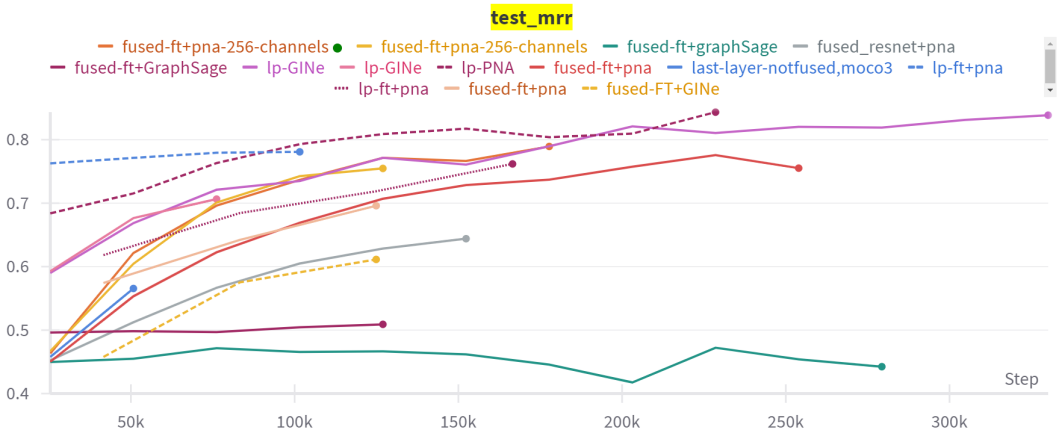


Figure 7: Mean reciprocal rank

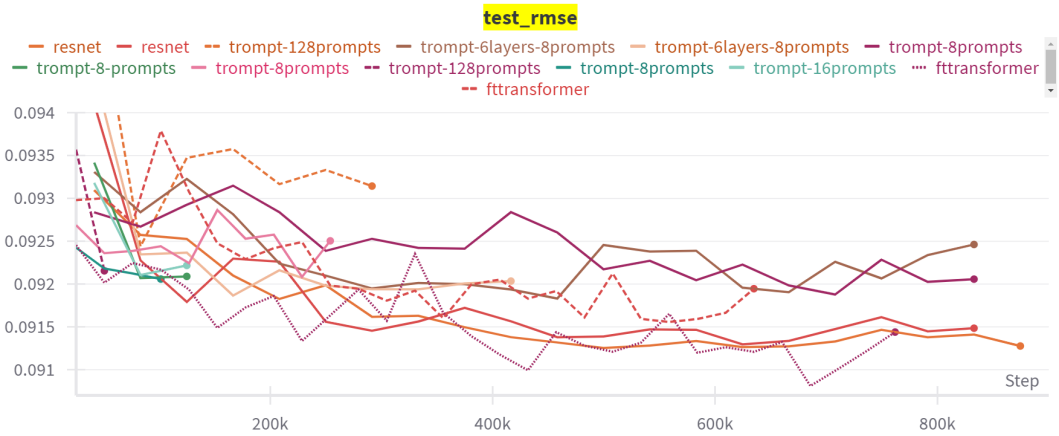


Figure 8: Root mean squared error of transformers

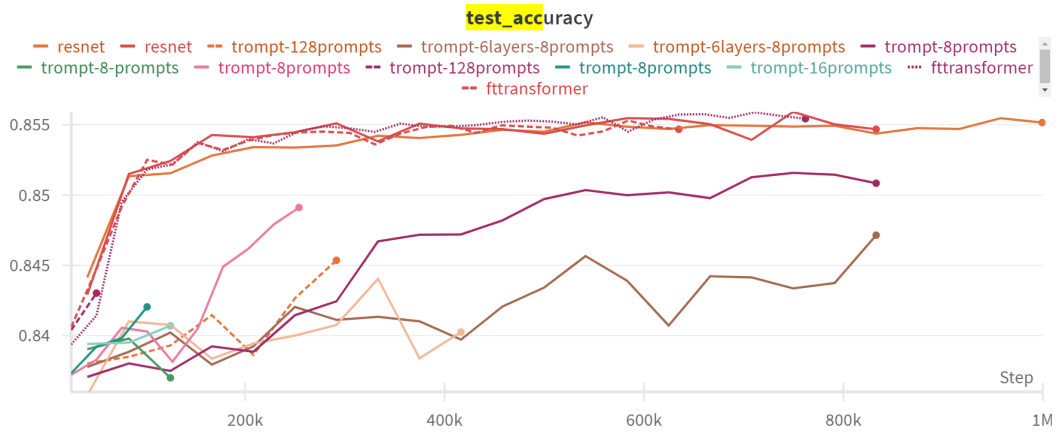


Figure 9: Accuracy of transformers