Exploring the benefits of Graph Transformers in Relational Deep Learning

Rafael Alani[1]

Supervisor(s): Kubilay Atasu[1]**, Çağrı Bilgi**[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 22, 2025

Name of the student: Rafael Alani
Final project course: CSE3000 Research Project
Thesis committee: Dr. Kubilay Atasu, Dr. Thomas Höllt, Halil Çağrı Bilgi

Abstract

Heterogeneous datasets hold a large percentage of all digital data that is available. With the rise of the digital medium, they have played a strong part in addressing the need for a structured way of storing data, particularly through the use of relational databases. To better leverage such data, the consensus of researchers has been in favour of using Graph Neural Networks to make predictions and infer possible outcomes. With the rise of the Transformer model and the clear limitations that GNNs inherently have due to their over-smoothing and over-squashing properties, a clear transition occurred into combining and leveraging the properties of both GNNs and Transformer models, Graph Transformers.

While this method has been better researched in the context of homogeneous datasets, much remains unexplored in its heterogeneous counterpart. This paper tackles this by applying Graph Transformers to multiple heterogeneous datasets, examining the differences and advantages between interleaved and cascade architectures of GTs and how the homogeneous positional encodings transfer to the heterogeneous context.

1 Introduction

With the introduction of transformers for graph learning related tasks [1] [2], many different researchers have looked at implementations of machine learning models that combine both Transformers and Graph Neural Network features [3, 4, 5]. This family of models has become its own research topic as improvements in evaluation metrics have been shown consistently with varying degrees of success based on the selected approach of implementation.

The main factor in their popularity has been the apparent limitations that Graph Neural Networks, further referred to in the paper as GNNs, have been unable to circumvent due to their underlying information sharing mechanism between the graph nodes, message passing. A mechanism that does not allow for the creation of deeper GNN models without the drawbacks of over-squashing, early smoothing, and loss of information [6]. Graph transformers, referred to as GTs, use graph neural networks and transformer layers. While the graph layers are responsible for the distribution of information between local nodes, the transformer layers allow the nodes to distribute the information even when the distance, measured in the number of hops between two nodes, is significant. This is due to their information passing mechanism: attention, which works independently of the distance between 2 nodes. The idea of leveraging the benefits of the transformer in the graph learning domain came soon after the initial transformer paper was recognised as a breakthrough [7].

As relational data sets were still mainly converted into large homogeneous tables, followed by applications of Tabular machine learning [8], the question of better alternatives has sparked outstanding research. In parallel with the developments of the GT, researchers have seen improvements by converting heterogeneous datasets into heterogeneous graphs as an intermediary representation. **Relational Deep Learning (RDL)** is a new proposal that re-casts relational data into an exact graph representation, with each entity in the database becoming a node and primary-foreign key links forming edges. Node features are extracted using deep tabular models. This approach allows GNNs to be used as predictive models, fully exploiting the predictive signal encoded in the primary-foreign key links, and moving away from manual feature engineering.

This work aims to bridge the two sides, exploring which features of the Graph Transformer could be used together with heterogeneous graph representation found in RDL to achieve an even better expression of the underlying data. When looking at the benefits and improvements that GT bring compared to their GNN counterparts, a large part of the observed difference in expressivity is attributed to the positional encodings [9] and the architecture, the way the transformer and GNN layers are put together [10]. This paper will tackle both approaches by introducing multiple architectures and positional encodings.

This research will focus on the question,

" What is the most optimal way of combining the benefits of Graph Transformers and Relational Deep Learning for predictive tasks on relational databases? " To do so, research will be conducted on two different architectures that have been prominent since the creation of the GT: the cascade and interleaved architectures. The cascade architecture was chosen for its simple design while still being able to observe improvements over GNNs, while the interleaved architecture was chosen due to its ability of keeping up with more complex GT architectures [10]. To further explore what truly made GT shine on homogeneous datasets, their positional encoding, the question of **" What are the measurable effects of applying positional encodings in the context of Relational Deep Learning? "** will also be answered and complemented by adaptations of algorithms that can be used for the heterogeneous counterparts, recommendations and observations related to the obtained results.

The paper will start with Related Works deemed essential in Section 2. This will be followed by a further dive into explaining field-specific terms in Section 3. The methodology and design choices will be described in Section 4, outlining the contributions of the paper. Section 5 will contain the concrete experimental setup and the obtained results. Further, Section 6 will discuss the observed results, while Section 7 will address the conclusions and possible future works. Last but not least, Section 8 will conclude with all of the required justification as to how the research was done responsibly and the results were derived in a reproducible fashion.

2 Related Research

It has been formally demonstrated that standard GNNs have an expressive power at most equal with the 1-WL (Weisfeiler-Lehman) test by Xu et al. [11]. This has further sparked the development of models that circumvent those limitations such as Yang et al.'s Graphormer [12] or Dwivedi and Bresson [2] GraphTransformer. A wave of general graph transformers followed, with notable mentions such as Dwivedi et al. [13]

that introduced Laplacian Eigenvectors as a positional encoding, Zhang et al. [14] for the use of graph-specific positional features, the use of spectral attention by Kreuzer et al. [15] and many others.

With the development of so many different approaches it's worth mentioning the contribution that Rampávsek et al. [6] has brought to the field through the characterisation and classification of the different possible positional and structural encodings, which were previously poorly structured and their benefits and drawbacks were not put into context through the use of other positional encodings. Rather, they were looked at in a vacuum, mostly against models that did not implement them.

One highly notable contribution to the field was in the theoretical and empirical proof of Wu et al. [17], which has shown that a single attention layer model equivalent can be found for any Graph Transformer that utilises multiple layers of attention. As such, our work was mainly focused on shallower models, which are capable of extracting similar information without the redundancy of numerous layers that have their weights optimised for different intermediary objectives at each layer.

GraphTrans by Wu et al. [18] has been used and compared to in many other studies, due to the simplicity of the cascade architecture. As brought up in Yin and Zhong [10], the vast majority of fusion GT models are limited by the fusion layer themselves, leading to the use of the more simplistic interleaved architecture to be as practical or even more effective.

It is to be noted that this is not the first study that addresses Graph Transformers in the context of heterogeneous datasets; Hu et al. [19] have already created a model that takes in a heterogeneous graph representation and uses a GT model to train on the dataset. The only work that has tried to adapt GTs to heterogeneous graphs [19] did not try to implement positional or structural encodings, already noting performance increases through the use of a *Layered fusion* architecture. There has been no paper that has tried to implement the GT model to the Relational Deep Learning framework proposed by Robinson et al. [20].

3 Background

This section outlines some basic knowledge needed to understand the research blocks that the paper will build upon. In section 3.1, the focus will be on the definition of the different concepts, while section 3.2 will give a more in-depth explanation of the exact concepts that will be used later in the paper. Appendix A.1 contains a visual representation of the Message Passing and Attention mechanisms applied to a Relbench dataset.

3.1 Notations

We define $\mathbf{G} = (\mathcal{V}, \mathcal{E})$ as our heterogeneous graph representation, where $\mathcal{V}$ represents the set of nodes and $\mathcal{E}$ represents the set of edges. The definitions of how the dataset is converted into a heterogeneous graph are outside of the research space and will be taken for granted. The node set $\mathcal{V}$ encompasses n nodes, with the feature matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the dimension of the node features.

Table 1: The basic notation utilised in this study.

Notation	Description
$\mathbf{G} = (\mathcal{V}, \mathcal{E})$	A graph $\mathbf{G}$ contains node set $\mathcal{V}$ and edge set $\mathcal{E}$.
$n \in \mathbb{R}$	The number of nodes in the graph.
$\rho_i \in \mathbb{N}$	Function that returns the node type of i.
$\xi_i \in \mathbb{N}$	Function that returns the edge type of i.
$\mathbf{H} \in \mathbb{R}^{n \times d}$	The node features in the graph.
$\tilde{\mathbf{H}} \in \mathbb{R}^{n \times d}$	The updated node representations of the graph.
$\mathbf{A} \in \mathbb{R}^{n \times n}$	The attention matrix learn in the Transformer.
$\mathbf{A}^{\mathbf{G}} \in \mathbb{R}^{n \times n}$	The adjacency matrix in the graph.
$\mathbf{m}_{ij}$	The message from node i to node j.
$\mathbf{e}_{ij}$	The edge feature between node i and node j.
$\mathcal{N}_i$	The neighboring nodes of node i.
ϕ, $\mathbf{W}$, $\mathbf{b}$	The learnable parameters in neural networks.
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	The query, key and value matrices in attention.
$\mathbf{D} \in \mathbb{R}^{n \times 1}$	The degree matrix, indicating the degree of the nodes.
$\|\|$	Concatenate operation.

The dimensions of the node features are independent of ρ_i. The edge set $\mathcal{E}$ is projected through the use of an adjacency matrix $\mathbf{A}^{\mathbf{G}} \in \mathbb{R}^{n \times n}$, where $\mathbf{A}^{\mathbf{G}}_{i,j} = 0$ if there doesn't exist an edge (i, j) in $\mathcal{E}$, and $\mathbf{A}^{\mathbf{G}}_{i,j} = 1$ otherwise.

Given the adjacency matrix $\mathbf{A}^{\mathbf{G}}$ and node embeddings $\mathbf{H}$, a GNN updates node embeddings by spreading information from neighbouring nodes through the use of the ϕ functions. As such, we can define the value of node embedding based on the values of the node embeddings at the previous layer:

$$\mathbf{m}_{ij} = \phi_{msg}(\mathbf{H}_i, \mathbf{H}_j), \tag{1}$$

$$\tilde{\mathbf{H}}_i = \phi_{conv}(\mathbf{H}_i, \{\mathbf{m}_{ij}\}_{j \in \mathcal{N}_i}), \tag{2}$$

$$\tilde{H}_i = \phi_{conv}(H_i, \{m_{ij}\}_{j \in \mathcal{N}_i}), \tag{3}$$

While $\phi_{msg}(\cdot)$ is the function that computes the message value for a respective edge, $\phi_{conv}(\cdot)$ is the selected function that updates the node embeddings based on the messages from this layer.

Transformers are defined through an encoder-decoder architecture, with the core building block being the attention mechanism. [7] The self-attention module operates on a sequence of n tokens, $\mathbf{H} \in \mathbb{R}^{n \times d}$, which can be represented by a function $\phi_\theta(\cdot) : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$. The definition of the function $\phi_\theta(\cdot)$ is shown in Equation 4-6:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \mathbf{K} = \mathbf{H}\mathbf{W}_K, \mathbf{V} = \mathbf{H}\mathbf{W}_V, \tag{4}$$

$$\mathbf{A} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right), \tag{5}$$

$$\tilde{\mathbf{H}} = \mathbf{AV}, \tag{6}$$

The input $\mathbf{H}$ is first transformed into query, key and values, $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d\times d}$, through the use of linear wight matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d}$, respectively. Then, the attention matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is computed by the inner product of the query and key, followed by the normalization of a Softmax$(\cdot)$ function. . The key information that makes attention valuable in capturing long-distance relationships between nodes is $\mathbf{A}_{ij} = 1$ as it indicates the influence of $\mathbf{H}_i$ on $\mathbf{H}_j$ The attention matrix is finally applied to the value matrix to generate new embeddings of the tokens $\tilde{\mathbf{H}} \in \mathbb{R}^{n\times d}$.

After utilising self-attention, Transformers use normalisation layers followed by feed-forward networks (FFN) to prevent exploding gradients and faster convergence. In practice, multi-head attention (MHA) is widely utilised in Transformers to enhance representation learning. In the context of graphs, it has been used by implementations such as GAT [21] while being discredited by others such as Yin and Zhong as something that was just adopted from the natural language processing and computer vision tasks without much thought into it's usefulness in the context of GNNs. Precisely, the $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices obtained from Equation (4) are divided into H independent heads, denoted as $\mathbf{Q}^{(h)}$, $\mathbf{K}^{(h)}$ and $\mathbf{V}^{(h)}$, respectively. The MHA mechanism computes representations as Equation (7):

$$\tilde{\mathbf{H}} = ||_{h=1}^{H} \text{Softmax}\left(\frac{\mathbf{Q}^{(h)}\mathbf{K}^{(h)T}}{\sqrt{d_h}}\right)\mathbf{V}^{(h)}, \tag{7}$$

where $d_h = d/H$ represents the dimension assigned to each head. MHA forms different representations from each head, finalising the process by concatenating the different embeddings from each head.

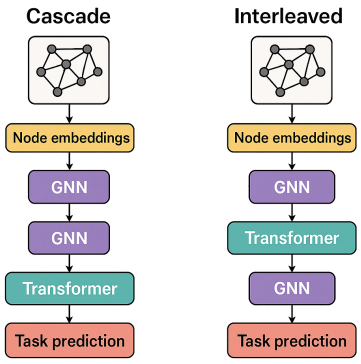


Figure 1: Comparison of layering architectures in cascade versus interleaved models

Graph Transformer Layering Architectures: The integration of MPNN layers with Transformer layers in GTs has been approached in several ways. While a definitive classification is still evolving, a common point of discussion in the literature revolves around how these distinct architectural blocks are combined [6] [10]. Based on current research, prominent layering strategies include:

- **Cascade (Sequential Stacking):** In this approach, a block of MPNN layers is followed by a block of Transformer layers (e.g. GraphTrans [18]).
- **Interleaved:** MPNN layers and Transformer layers are alternated throughout the network's depth (e.g. LGI-GT [10]).
- **Layered Fusion (Parallel):** Within each layer or block, MPNN and Transformer operations are performed in parallel on the same input, and their outputs are fused (e.g. GPS [6]).
- **Late Fusion:** This typically involves separate, deep processing streams for graph-based and sequence-based information, with fusion occurring only towards the end of the model. (e.g. SGFormer [16])

Positional encodings: Through various techniques used to give mathematical values to the structure of a graph, positional encodings try to define the node's role in the structures it is part of. This can be highly beneficial in cases where the MPNN cannot convey the structure well enough. Message passing accounts for the graph structure by using edges as the decider if two nodes $\mathbf{i}$ and $\mathbf{j}$ will exchange information, as seen in Equation 1. No equivalent inherent property guarantees the same for the attention mechanism; as such, Positional Encodings are a way to remind the Transformer layers about the graph structure.

3.2 Detailed knowledge

This section discusses the specifics of the Graph Transformer (GT) architectures relevant to this work while also briefly mentioning other important prerequisites.

Two of the main GNN architectures that are foundational for the Message Passing (MP) layers in many advanced models are the Graph Isomorphism Network (GIN) and Graph Attention Network (GAT). Random Walks and Laplacian Eigenvectors are the chosen Positional Encodings. As their implementation details are not crucial to our contributions, they were described in Appendix A.2.

Cascade models, such as GraphTrans [18], process graph information through several MPNN layers before feeding the resulting node representations to Transformer layers. A significant drawback of this architecture is that it inherits limitations from the initial GNN stack [6]. Problems like over-smoothing or over-squashing in the GNN layers can lead to a loss of discriminative information before the data even reaches the Transformer layers, potentially limiting the global attention mechanism's effectiveness [10].

A Cascade model consists of a block of L GNN structures followed by a block of M Transformer structures:

1.
$$H_{GNN} = (\underbrace{\text{GNN} \circ \cdots \circ \text{GNN}}_{L \text{ times}})(H) \tag{8}$$

2.
$$\tilde{H} = (\underbrace{\text{Transformer} \circ \cdots \circ \text{Transformer}}_{M \text{ times}})(H_{GNN}) \tag{9}$$

where H_{GNN} represents the node embeddings after being processed by all L GNN layers, which then serves as

the input to the subsequent block of M Transformer layers. The base `GNN` function represents Equation 1, 3 and the base `Transformer` function encompasses Equations 4, 5, 6.

Interleaved models aim to mitigate these issues by alternating GNN and Transformer layers [10]. This design allows for local information to be refined by GNN and then immediately processed for global context by Transformer layers, or vice versa. This can help maintain richer node representations throughout the network, enabling deeper architectures without the significant information degradation that cascade models might suffer. The interleaved architecture demonstrates robust performance, especially when simpler fusion methods in parallel/layered fusion models might not effectively combine the distinct information captured by MPNNs and Transformers [10].

An interleaved model would consist of $(L + M)/2$ blocks, where a single block would have the following structure, using the definitions above:

1.
$$H_{GNN} = \text{GNN}(H) \tag{10}$$

2.
$$\tilde{H} = \text{Transformer}(H_{GNN}) \tag{11}$$

4 Methodology

This section outlines the architectural design and implementation choices for the Graph Transformer models used in this research. The section begins by detailing the specific implementations of the GNN and Transformer components, explaining how the general mathematical formulations are adapted to operate effectively on heterogeneous graphs. Some general details are presented about the Relbench benchmark, as they have a clear impact on our methodology and how we decided to approach the contributions. We follow by justifying the selection of the cascade and interleaved architectures as the core focus of the study. Finally, we describe the approach taken to integrate and evaluate positional encodings to enhance the structural awareness of the models.

4.1 GNN Layers

The GNN models were chosen based on specific features that their implementations offer. The GIN has been proven to be equally as powerful as 1WL test, while GAT learns to assign different importance weights to different neighbours when aggregating their features, which is even more important in a heterogeneous graph than in a homogeneous one. Some edge types and particular neighbours should have a higher influence on the task at hand. For GAT the addition of edge features has also been explored through the use of the concatenation of the node features of both nodes.

4.2 Transformer Layers

As for the **attention mechanism**, classical multi-head self-attention was chosen as described inside Equation 7. To further accommodate the heterogeneity of our graph, two routes showed potential as we advanced: using a monolithic attention mechanism across all node types or using attention on a node-type basis. Applying a monolithic attention mechanism across all node types might dilute type-specific information or produce less meaningful attention scores if nodes of vastly different characteristics are forced into the same attention space. By limiting attention on a node type-by-type basis:

$$\tilde{\mathbf{H}}_t = ||_{h=1}^{H} \text{Softmax}\left(\frac{\mathbf{Q}_t^{(h)}\mathbf{K}_t^{(h)T}}{\sqrt{d_h}}\right)\mathbf{V}_t^{(h)}, \quad \forall t \in \mathcal{T} \tag{12}$$

We introduce a structured approach that offers semantic coherence and reduces complexity

$$(N_{\text{total}})^2 \text{ to } \sum(N_{\text{type}}^2) \tag{13}$$

, which can be significantly less when the graph is partitioned into reasonable node types like the Relbench datasets.

4.3 Intermediary representation details

RelBench was chosen as the benchmark the study will focus on, a benchmark designed for evaluating machine learning models on relational database tasks, due to its standardised setup and diverse datasets. Relbench also provides us with the heterogeneous graph intermediary representation that we use as a starting point for our application of GTs. The following are the key features that were taken for granted from the Relbench representation to minimise the surface area of the possible factors that influence the findings.

- Each training batch was constructed from different independent connected components sampled from the graph.
- The heterogeneous graph structure was derived from primary and foreign key relationships between different entity types, ensuring no edges connect nodes of the same kind.

$$\forall i, j \in \mathbf{G} \text{ if } \rho(i) = \rho(j) \text{ then } \mathbf{A}_{i,j}^{\mathbf{G}} = 0 \tag{14}$$

- Both the primary-foreign key links and their inverse links were used to create the edges.

$$\forall i, j \in \mathbf{G} \text{ if } \mathbf{A}_{i,j}^{\mathbf{G}} = 1 \text{ then } \mathbf{A}_{j,i}^{\mathbf{G}} = 1 \tag{15}$$

- The temporal restriction based on $VAL_{TIMESTAMP}$ and $TEST_{TIMESTAMP}$ is applied at batch level.
- There are no edge embeddings in the Relbench benchmark.

4.4 Architectural decisions

Based on the findings from the field and past paper research, a ground-up approach will be used in the construction of the models. This work focuses on the **Cascade and Interleaved architectures**. The Cascade model provides a simple implementation, while the Interleaved model offers greater expressive power without sacrificing interpretability [10]. This focused comparison allows us to directly attribute performance changes to specific architectural decisions, grounding our findings. This would guarantee the ability of pinpointing and attributing the observed changes to specific parts of the model, be it hyperparameter changes, implementation additions or simplifications.

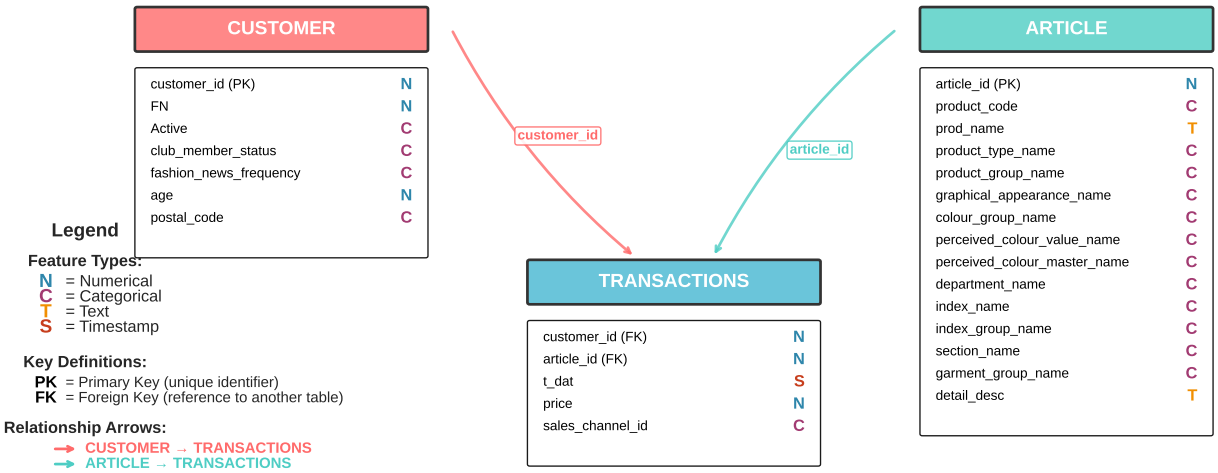


Figure 2: Database schema of the H&M Dataset, highlighting its three main entities and selected features exemplifying the input information for node embeddings.

Our implementation of the **Cascade model**, defined generally in Equations 8 and 9, is specialised to handle heterogeneous graph data. The initial GNN block, corresponding to Equation 8, consists of a shallow stack of message-passing mechanisms. Each GNN layer is followed by a $LayerNorm$ and a $ReLU$ activation function to stabilise training and introduce non-linearity. Thus the definition of the H_{GNN} function becoming:

$$H_{GNN} = (\text{ReLU} \circ \cdots \circ \text{ReLU} \circ \text{LayerNorm} \circ \text{GNN})(H) \quad (16)$$

The most significant adaptation occurs in the Transformer block from Equation 9. Rather than applying a single, monolithic attention mechanism across all nodes, our model instantiates a separate Transformer layer for each node type, as seen in 12 This design choice confines the self-attention computation to only nodes of the same type, effectively creating a structured, type-aware attention mechanism. This approach reduces computational complexity (13) and ensures that attention scores are calculated between semantically coherent entities. The same adaptations are also applied to the interleaved model.

4.5 Positional Encodings

One of the principal ways through which GTs have been able to improve over their Message Passing counterparts.

To further highlight and build upon these models we have created and adapted existing Positional encodings as to observe their influence in the way the models are trained and perform. The two chosen **positional encodings** were Laplacian Eigenvectors and Random Walks. **Laplacian Eigenvectors** were chosen for their global positional awareness, while their **Random Walk** counterpart was chosen due to their local structure context. With both in mind, the performance changes can be attributed more clearly to the specific type of information provided to the model.

The use of both allows us to isolate the effects of different information types. While random walks' features offer **local structural context**, LapPEs offer their **global counterpart**. Both encodings have been shown to enhance the expressiveness of GNNs beyond the standard 1-WL test capabilities. They are used in tandem to allow us to understand further the importance of the features of the datasets and the differences between their application on heterogeneous and homogeneous graphs. The positional encodings were tested with a sinusoidal and MLP embedding method, while the graph was either enriched by edges between nodes of the same type 2 hops away, or by none. Appendix A.3 explains the whole process.

5 Experimental Setup and Results

This section first describes the experimental setup, such as the datasets and their characteristics, the chosen benchmark, data splits and how to reproduce the experiments in subsection 5.1. Further in Subsection 5.2, we highlight the results obtained by applying the experiments.

5.1 Experimental setup

Relbench was chosen as the benchmark for the study. The main scope of Relbench was to show the ability of RDL to model heterogeneous datasets and obtain SOTA results compared to current Graph Learning methods, and to establish a comprehensive graph-learning benchmark for heterogeneous datasets which previously did not exist. The diverse and vast nature of the datasets present in Relbench makes it an ideal observational medium for research. The chosen datasets for this study are *rel-avito, rel-hm, rel-f1, rel-trial, and rel-stack*, which encompass diverse domains such as e-commerce, sports, medical, and social Q&A platforms. Only the Formula 1

Table 2: Entity classification results (AUROC, higher is better). Relative gains to RDL. Best values are **highlighted**.

Dataset	Task	Split	RDL	Cascade-GIN	Rel. Gain	Interleaved-GIN	Rel. Gain
rel-avito	user-visits	Test	66.20	**69.41**	4.85 %	66.2	−0.12 %
	user-clicks	Test	65.90	**66.47**	0.86 %	65.92	0.03 %
rel-f1	driver-dnf	Test	**72.62**	71.67	−1.31 %	65.16	−10.27 %
	driver-top3	Test	75.54	**78.87**	2.58 %	76.17	−.094 %
rel-hm	user-churn	Test	**69.88**	68.69	−1.7 %	67.92	−2.8 %
rel-stack	user-engagement	Test	90.59	**90.8**	0.23 %	89.63	−1.06 %
	user-badge	Test	**88.86**	88.15	−0.8 %	82.76	−6.82 %

Table 3: Entity regression results (MAE, lower is better). Relative gains to RDL. Best values are **highlighted**.

Dataset	Task	Split	RDL	Cascade-GIN	Rel. Gain	Interleaved-GIN	Rel. Gain
rel-avito	ad-ctr	Test	0.041	**0.038**	6 %	0.039	3.41 %
rel-hm	item-sales	Test	0.056	**0.053**	4.17 %	0.052	6.2 %
rel-f1	driver-position	Test	4.022	**3.955**	1.7 %	4.195	−4.1 %

Table 4: Hyperparameters configured for various Relbench tasks.

Hyperparameter	Task type	
	RDL parameters	Our Parameters
Learning rate	0.005	0.005
Maximum epochs	10	10
Batch size	512	128
Hidden feature size	128	128
Aggregation	summation	summation
Number of layers	2	3
Number of neighbors	128	128
Temporal sampling strategy	uniform	uniform

dataset was used for positional encoding comparisons, acknowledging the scalability limitations of using current positional encoding methods.

These databases vary significantly in scale and structure, with table counts ranging from 3(*rel-hm*) to 15(*rel-trial*) and column counts from 37(*rel-hm*) to 140(*rel-trial*). For instance, the *rel-hm* dataset comprises three interconnected entities: customer, article, and transactions, which are linked through primary-foreign key relationships to capture detailed historical sales data. Each entity within these databases is comprised of various features of different types (e.g., numerical, categorical, timestamp, text) as seen in Figure 2, which contribute to the rich predictive signal.

Data splitting in RELBENCH is handled temporally based on specified validation and test timestamps $VAL_{TIMESTAMP}$ and $TEST_{TIMESTAMP}$. This is to prevent data leakage from future events, ensuring that models are evaluated on information that would be available at the time of prediction. Each of the datasets has different time stamps that can be seen at in the Relbench paper [20].

The training methodology used mirrors that in the Relbench study [20], as it can be seen in Table 4, except for the additional layer, which is used to accommodate for the need of both GNN and Transformer layers in GT. Lowering the batch size was observed to produce similar results with a faster training time. Further in this study, we will refer to the Relbench model as RDL both when addressing the differences in design, but also experimental differences.

It is to be noted that while most of the datasets do have class imbalances, these are cases where this phenomenon is quite extreme, such as the Stack dataset where out of all the data available for the user-engagement classification tasks only about 5% of the training samples (68,020 out of 1,360,850) are positive (user makes a contribution), indicating a significant class imbalance where the majority class (no contribution) is overwhelmingly dominant. Similarly, user-badge also shows a strong imbalance, with around 4.8% positives.

5.2 Results

This section presents the empirical outcomes of our experiments, focusing on the comparative performance of the cascade and interleaved layering techniques, the impact of various positional encodings (PEs), and a brief comparison of GNN architectures within our framework. Performance is evaluated using the Area Under the Receiver Operating Characteristic Curve (AUROC) for classification tasks and the Mean Absolute Error (MAE) for regression tasks.

Architectural Comparison: Cascade vs. Interleaved

The initial architectural comparison, as detailed in Table 2 for classification tasks and Table 3 for regression tasks, reveals distinct trends. For classification tasks the simplistic Cascade-GIN model generally achieves results comparable to or slightly varying from the RDL baseline. For instance, *rel-avito*'s user-visits task shows a 4.85% relative gain in AUROC for Cascade-GIN, while *rel-f1*'s driver-top3 yields a 2.58% gain. Conversely, some tasks, like *rel-hm*'s user-churn, show minor decreases in AUROC, -1.7% for Cascade-GIN.

In contrast, for regression tasks, the cascade model consistently demonstrates notable improvements. Table 3 indicates that the simplistic cascade model achieves a 2-6% reduction in MAE across various regression tasks. For instance, *rel-avito*'s ad-ctr shows a 6.0% MAE reduction for Cascade-GIN, and *rel-hm*'s item-sales shows a 4.17% reduction. Such improvements

can be characterised as statistically significant, supported by the low variance observed between runs.

Another trend that can be observed from the graphs is the interleaved architecture's performance, as seen in both Table 2 and Table 3, generally aligns with or slightly underperforms the cascade architecture.

Impact of Positional Encodings

Our analysis of positional encodings (PEs) involved two types: Laplacian and Random Walks, each tested with the addition of edges connecting nodes of the same type two hops away or none, two adaptation methods (sinusoidal, MLP) were also tested as to see the optimal way to integrate the PE into the current node embeddings, across different exploration lengths (3, 5, 7). The results are presented as follows in Figures 3–4 for the *driver-dnf* task, with the remaining graphs for *driver-top3* and *driver-position* being in Appendix A.4, as they present the same overall trends.

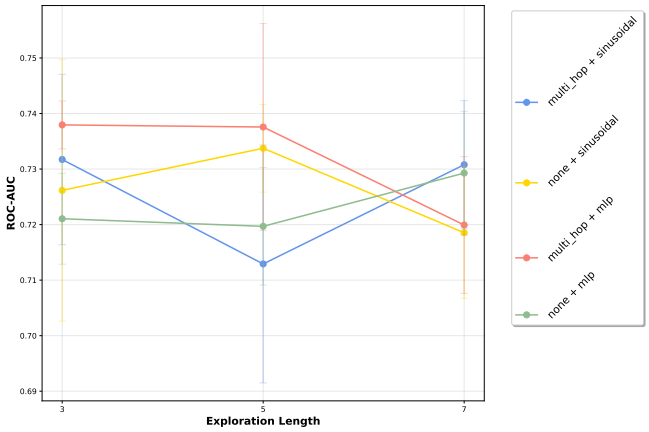


Figure 3: AUROC performance for *driver-dnf* with **Cascade-GIN + Laplacian Eigenvector PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the number of lowest non-zero eigenvalues used for computing the PE.

For the driver-dnf classification task (Figures 3–4), the inclusion of positional encodings generally shows varying, often marginal, impacts on AUROC. With Laplacian PE 3, multi-hop + sinusoidal performs competitively at walk length 3 while none + mlp is competitive at walk length 7. Multi-hop + mlp also shows a strong initial performance at walk length 3 before decreasing. Similarly, for Random Walk PE (Figure 4), the trends are inconsistent, with none + sinusoidal showing a decline and multi-hop + mlp being relatively stable.

A consistent observation across these PE experiments is the lack of a clear, universal performance gain from the introduced positional encodings. While some configurations show minor improvements or competitive performance at specific walk lengths or with particular PE types, there is no overwhelming evidence that these homogeneous PEs, even with synthetic multi-hop edges significantly and consistently enhance the model's predictive power on these heterogeneous datasets. Furthermore, adding multi-hop edges does not consistently

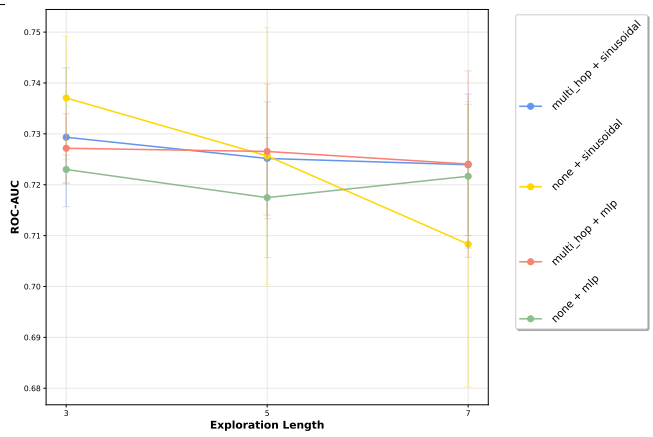


Figure 4: AUROC performance for *driver-dnf* with **Cascade-GIN + Random Walk PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the maximum walk length considered.

improve performance. In some cases, it appears to slightly lower the overall performance of both Laplacian Eigenvector and Random Walk PEs, as suggested by the varying trends in the figures.

GNN Component Comparison

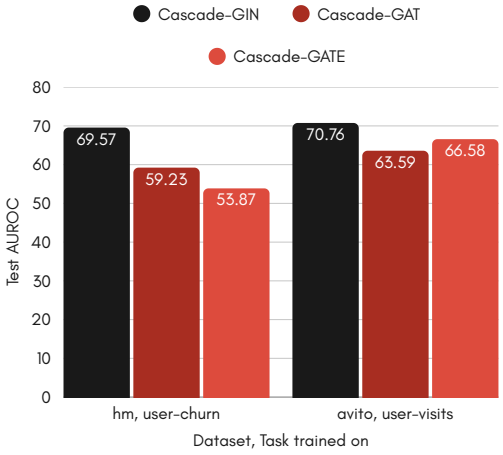


Figure 5: AUROC performance of Cascade models employing GIN, GAT, and GATE GNN modules on classification tasks.

Figures 5 and 6 compare the basic implementation of the Cascade-GIN model presented in 5.2 where we swapped the previously used GNN module of our model for GAT and GATE models. For the classification tasks, Figure 5 indicates a better performance of the Cascade-GIN model. In this context, the GIN model consistently shows the highest AUROC (69.57 for *hm, user-churn* and 70.76 for *avito, user-visits*) compared to GAT and GATE. Similarly, for the regression tasks *hm, item-sales* and *avito, ad-ctr* in Figure 6, the GIN model also exhibits the lowest MAE (0.06100 and 0.03400, respectively). This consistent outperformance of the GIN model strongly suggests

that the GIN-based message passing, as employed within our Cascade architecture, is more effective for these heterogeneous relational datasets compared to GAT and GATE's aggregation mechanisms.

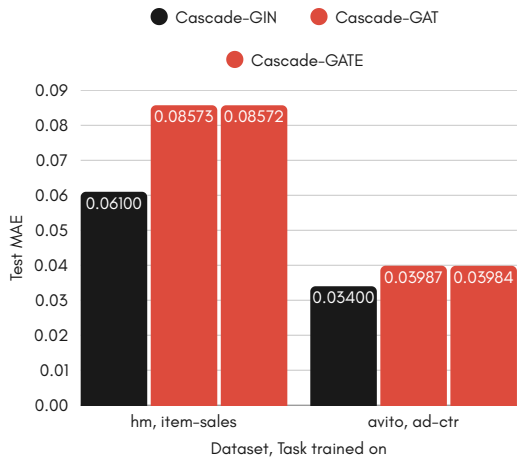


Figure 6: AUROC performance of Cascade models employing GIN, GAT, and GATE GNN modules on regression tasks.

6 Discussion

The findings illuminate several critical aspects regarding the application of GTs to heterogeneous datasets, particularly within the RDL context. Two primary research questions guided the research: the optimal combination of GNN and Transformer benefits and the measurable effects of positional encodings in RDL.

6.1 Architectural Synergy in Relational Deep Learning

The initial comparison between Cascade and Interleaved architectures (Tables 2 and 3) did not reveal a significant performance disparity, both of them showing similar performance in the classification tasks, while the regression tasks showed improvement. This outcome might appear counterintuitive, especially considering literature highlighting cascade models' inherent limitations, such as over-smoothing in deeper GNN stacks. However, the observed similarity is attributable to the shallowness of the models employed in these initial experiments. With a limited number of three layers, the cascade architecture will not suffer significantly from information degradation, allowing its GNN component to effectively capture local structures and its Transformer component to subsequently model global relationships without substantial loss. This would not mean that the Interleaved-GIN model would perform better with more layers; it would only show no degradation in the current results, as observed in [10].

Moreover, the observed superior performance of the GIN-based Cascade model over GAT and GATE, as shown in Figures 5 and 6 suggests that for these heterogeneous relational datasets, the GIN's strong discriminative power, rooted in its ability to distinguish between different graph structures equivalent to the 1-WL test, is highly beneficial.

6.2 Measurable Effects of Positional Encodings in RDL

However, a more critical observation is the general lack of significant performance gain from the implemented PEs, even with the introduction of synthetic two-hop edges. One of the primary reasons for this could again be linked to the node-type-based attention mechanism. The attention mechanism might already effectively learn structural and positional roles within those type-specific subgraphs. In such a scenario, the explicit structural information provided by LE or RW PEs might become redundant or offer marginal new insights, as the Transformer layers are already well-equipped to discern these patterns for the node types they focus on.

Adding such synthetic edges did not have a positive impact, as extracted from Figures 3–4, 9a–9b, 10a–10b. The synthetic edges seem to lower the overall performance of both LE and RW by a small margin. Such a result is not unexpected, as creating new edges that have nothing to do with the graph structure can deter the model from learning based on the actual graph structure.

7 Conclusions and Future Work

This work explored the application of Graph Transformers to Relational Deep Learning, focusing on architectural layering techniques and the influence of positional encodings. The experiments demonstrate that the architectural design influences predictive performance in Relational Deep Learning.

Our findings suggest that for shallower models and Relbench-type heterogeneous datasets, the cascade layering technique, with its GIN-based message passing and node-type-specific attention, provides a simple yet effective solution, particularly for regression tasks. It achieves comparable or superior performance to baselines and other GNN architectures (GAT, GATE) on the evaluated tasks, without the added complexity that might not yield benefits in shallow settings. The simplicity and speed of the cascade model make it a strong candidate for practical applications where efficiency is paramount. The performance of the interleaved model, on the other hand, did not consistently outperform the cascade architecture, indicating that merely alternating GNN and Transformer layers might not be sufficient without more sophisticated mechanisms for global context integration.

When looking at positional encodings, the experiments indicate that homogeneous positional encodings, including Laplacian Eigenvectors and Random Walks, even with the addition of multi-hop edges, do not consistently provide significant measurable performance gains in the heterogeneous graph setting. The impact observed was often marginal or inconsistent across tasks and configurations. Further research would be needed to attribute this to the node type-based attention mechanism or to the chosen PE having no advantage in RDL and in the broader heterogeneous graph context.

Moving forward, a pressing need exists for positional encodings specifically designed with heterogeneity and large-scale graphs in mind. Future work should explore effective positional encodings for RDL and heterogeneous graphs that can scale efficiently, and investigate other GNN models known to outperform GIN, such as PNA.

8 Responsible Research

8.1 Ethical implications

This research has considered all ethical standards that might be encountered in such a project. The data used is based on open datasets that were provided to the research and hobbyist community through different methods, collected and compiled by [22], creating a benchmark for heterogeneous graphs. While most of the datasets themselves do not contain any information related to the gender, sex or other features that might be discriminatory, there are others, such as the "Events" dataset, that do. It is to be noted that during the experimental setup, no greater measures were taken to observe direct or indirect biases that might arise from the multitude of features. The biases were not explored further as they were not the main priority of the research.

8.2 Reproducibility and used experimental data

All the runs were seeded with concrete seeds from the pool 42, 123, 456, 789, 1024. All graphs are comprised of five different training runs, all seeds were used once. All packages and libraries have enforced versions to allow for reproducibility. The analysis includes all of the data compiled during the research. Any other results were either faulty due to the current implementation of the code or incomplete training runs. To ease the replication process, the code will be available through an online Git repository: https://github.com/hcagri/relbench-rafael.

8.3 Environmental impact

The inference and training of deep neural networks on large datasets tend to be both computationally and energetically intensive. As the focus has been mainly on shallower models, excessive runs not used in the final analysis were kept to a minimum, with their sole purpose being to explore different hyperparameter values. Thus, such a study's environmental impact was as low as expected.

8.4 Programming

While the code has been checked and debugged using the output of the different tensors and layers, it would be prudent for any further research that uses this study as a basis to familiarise themselves with the code. Most parameters can be passed through the Python activation script, while some will have to be manually edited inside the model or main file. The directories where the datasets are stored will also have to be changed inside the main file. A shell script to run multiple experiments in batches is also provided with examples of how it was used during the research.

Pytorch, together with Pytorch Geometric and Frame, were used due to their tight integration in the development of the Relbench package. Current implementations of RW and LE positional encodings are incompatible with graphs with multiple node or edge types in Pytorch Geometric. As such, the paper has devised its implementations and modifications to adapt, as described in 4.3.

References

[1] Chengxuan Ying et al. *Do Transformers Really Perform Bad for Graph Representation?* arXiv:2106.05234 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2106.05234. URL: http://arxiv.org/abs/2106.05234 (visited on Apr. 26, 2025).

[2] Vijay Prakash Dwivedi and Xavier Bresson. *A Generalization of Transformer Networks to Graphs*. arXiv:2012.09699 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2012.09699. URL: http://arxiv.org/abs/2012.09699 (visited on Apr. 26, 2025).

[3] Junhong Lin et al. "FraudGT: A Simple, Effective, and Efficient Graph Transformer for Financial Fraud Detection". en. In: *Proceedings of the 5th ACM International Conference on AI in Finance*. Brooklyn NY USA: ACM, Nov. 2024, pp. 292–300. ISBN: 979-8-4007-1081-0. DOI: 10.1145/3677052.3698648. URL: https://dl.acm.org/doi/10.1145/3677052.3698648 (visited on Apr. 26, 2025).

[4] Luis Müller et al. *Attending to Graph Transformers*. arXiv:2302.04181 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2302.04181. URL: http://arxiv.org/abs/2302.04181 (visited on Apr. 27, 2025).

[5] Jianan Zhao et al. *Gophormer: Ego-Graph Transformer for Node Classification*. arXiv:2110.13094 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2110.13094. URL: http://arxiv.org/abs/2110.13094 (visited on Apr. 27, 2025).

[6] Ladislav Rampávsek et al. "Recipe for a General, Powerful, Scalable Graph Transformer". en. In: *Advances in Neural Information Processing Systems* 35 (Dec. 2022), pp. 14501–14515. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/5d4834a159f1547b267a05a4e2b7cf5e-Abstract-Conference.html (visited on Apr. 27, 2025).

[7] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. Aug. 2023. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on Apr. 26, 2025).

[8] Yizhou Sun and Jiawei Han. "Mining heterogeneous information networks: a structural analysis approach". In: *SIGKDD Explor. Newsl.* 14.2 (Apr. 2013), pp. 20–28. ISSN: 1931-0145. DOI: 10.1145/2481244.2481248. URL: https://doi.org/10.1145/2481244.2481248 (visited on May 31, 2025).

[9] Ladislav Rampávsek et al. *Recipe for a General, Powerful, Scalable Graph Transformer*. arXiv:2205.12454 [cs]. Jan. 2023. DOI: 10.48550/arXiv.2205.12454. URL: http://arxiv.org/abs/2205.12454 (visited on Apr. 26, 2025).

[10] Shuo Yin and Guoqiang Zhong. "LGI-GT: Graph Transformers with Local and Global Operators Interleaving". en. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. Macau, SAR China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2023, pp. 4504–4512. ISBN: 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/

501. URL: https://www.ijcai.org/proceedings/2023/501 (visited on May 2, 2025).

[11] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* arXiv:1810.00826 [cs]. Feb. 2019. DOI: 10.48550/arXiv.1810.00826. URL: http://arxiv.org/abs/1810.00826 (visited on May 31, 2025).

[12] Junhan Yang et al. "GraphFormers: GNN-nested Transformers for Representation Learning on Textual Graph". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 28798–28810. URL: https://proceedings.neurips.cc/paper/2021/hash/f18a6d1cde4b205199de8729a6637b42-Abstract.html (visited on May 2, 2025).

[13] Vijay Prakash Dwivedi et al. *Benchmarking Graph Neural Networks*. arXiv:2003.00982 [cs]. Dec. 2022. DOI: 10.48550/arXiv.2003.00982. URL: http://arxiv.org/abs/2003.00982 (visited on June 4, 2025).

[14] Jiawei Zhang et al. *Graph-Bert: Only Attention is Needed for Learning Graph Representations*. arXiv:2001.05140 [cs]. Jan. 2020. DOI: 10.48550/arXiv.2001.05140. URL: http://arxiv.org/abs/2001.05140 (visited on June 18, 2025).

[15] Devin Kreuzer et al. "Rethinking Graph Transformers with Spectral Attention". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 21618–21629. URL: https://proceedings.neurips.cc/paper_files/paper/2021/hash/b4fd1d2cb085390fbbadae65e07876a7-Abstract.html (visited on Apr. 27, 2025).

[16] Qitian Wu et al. "SGFormer: Simplifying and Empowering Transformers for Large-Graph Representations". en. In: *Advances in Neural Information Processing Systems* 36 (Dec. 2023), pp. 64753–64773. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/cc57fac10eacadb3b72a907ac48f9a98-Abstract-Conference.html (visited on Apr. 27, 2025).

[17] Qitian Wu et al. *SGFormer: Simplifying and Empowering Transformers for Large-Graph Representations*. arXiv:2306.10759 [cs]. Aug. 2024. DOI: 10.48550/arXiv.2306.10759. URL: http://arxiv.org/abs/2306.10759 (visited on Apr. 26, 2025).

[18] Zhanghao Wu et al. "Representing Long-Range Context for Graph Neural Networks with Global Attention". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 13266–13279. URL: https://proceedings.neurips.cc/paper/2021/hash/6e67691b60ed3e4a55935261314dd534-Abstract.html (visited on Apr. 29, 2025).

[19] Ziniu Hu et al. "Heterogeneous Graph Transformer". In: *Proceedings of The Web Conference 2020*. WWW '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 2704–2710. ISBN: 978-1-4503-7023-3. DOI: 10.1145/3366423.3380027. URL: https://dl.acm.org/doi/10.1145/3366423.3380027 (visited on Apr. 27, 2025).

[20] Joshua Robinson et al. *RelBench: A Benchmark for Deep Learning on Relational Databases*. arXiv:2407.20060 [cs]. July 2024. DOI: 10.48550/arXiv.2407.20060. URL: http://arxiv.org/abs/2407.20060 (visited on Apr. 27, 2025).

[21] Petar Velivcković et al. *Graph Attention Networks*. arXiv:1710.10903 [stat]. Feb. 2018. DOI: 10.48550/arXiv.1710.10903. URL: http://arxiv.org/abs/1710.10903 (visited on June 2, 2025).

[22] Matthias Fey et al. "Relational Deep Learning: Graph Representation Learning on Relational Databases". en. In: ().

A Appendix

A.1 Visual representation for Message Passing and Attention

This section is meant to help readers with a more visual representation of the 2 information passing mechanisms found in a GTs, message passing and attention. A customer node is selected from the *H&M dataset* after which all of its transactions and items are received. The attention mechanism operates on the exact node embeddings received from the second GIN layer. The initial node embeddings are simulated while the latter ones actually use *Torch Message Passing and Self-Attention*.

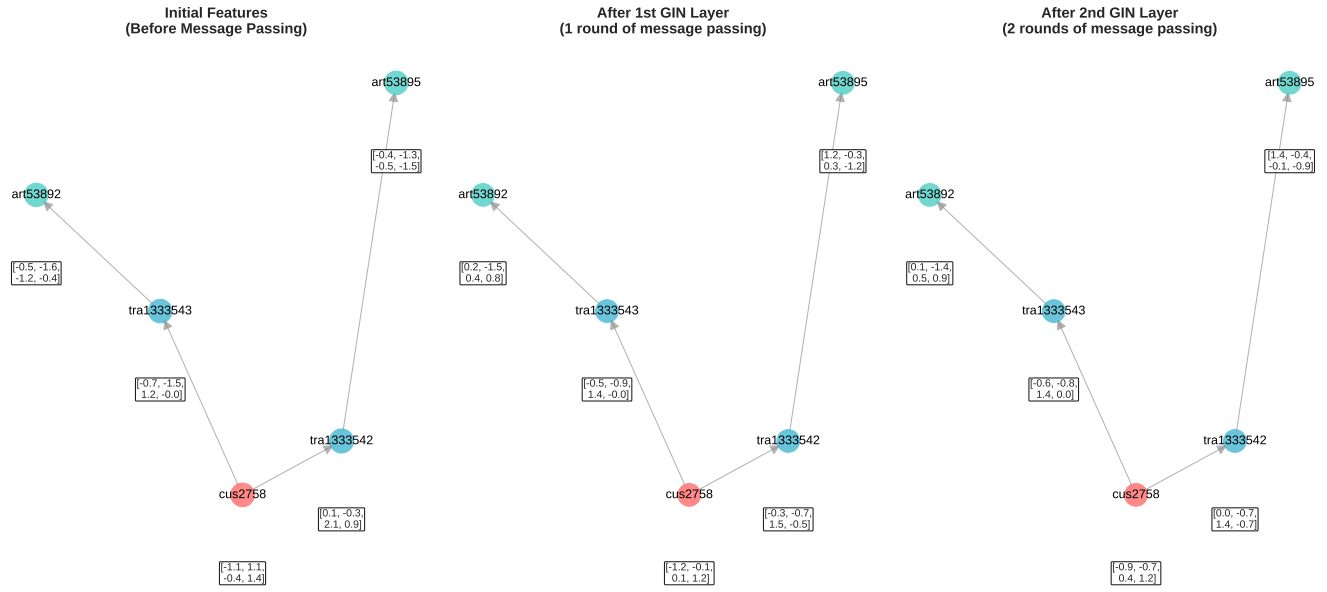


Figure 7: Visual representation of message passing in a Graph Isomorphism Network on actual transaction nodes from the *H&M dataset*. This diagram shows the evolution of node features from their initial state (before message passing) through one and then two rounds of message passing in a GIN layer, illustrating how information is propagated and aggregated across the graph.

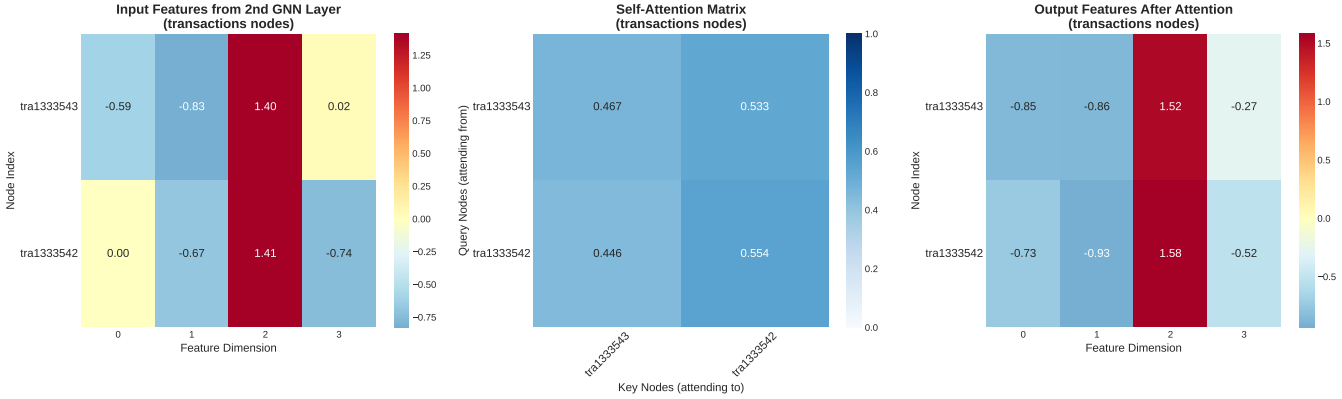


Figure 8: Visual representation of the self-attention mechanism on actual transaction nodes from the *H&M dataset*, illustrating the transformation from input features to output features. The figure shows initial transaction node features from the second GNN layer, the computed self-attention matrix between these nodes, and the resulting updated node features after the attention mechanism has been applied.

A.2 Further detailed knowledge

GIN and GAT

Graph Isomorphism Network (GIN) aims to achieve maximal discriminative power, equivalent to the Weisfeiler-Leman (1-WL) test[11]. It employs a Multi-Layer Perceptron (MLP) to learn the update function. The core idea is that an MLP, under the universal approximation theorem, can learn any function. Combined with a sum aggregator, it can represent injective functions over multisets of neighbour features. This allows GIN to distinguish between graph structures that simpler GNNs cannot. Xu et

al. (2019) noted: "Generally, there may exist many other powerful GNNs. GIN is one such example among many maximally powerful GNNs, while being simple.". The update rule for a node i at layer k in GIN is:

$$\tilde{\mathbf{H}}_i = \text{MLP}\left((1+\epsilon^{(k)})\cdot \mathbf{H}_i + \sum_{u\in\mathcal{N}_i} \mathbf{H}_j\right) \tag{17}$$

where MLP is the MLP for the k-th layer, $\mathbf{H}_i$ is the feature vector of node i from the previous layer, $\mathcal{N}_i$ is the set of neighbors of node i, and ϵ is either a learnable parameter or a fixed scalar. This formulation combines the central node's updated feature with the sum of its neighbours' features before passing it through an MLP.

Graph Attention Network (GAT) [21] utilises masked self-attention mechanisms for local message passing, allowing nodes to assign different importance to different neighbours. The attention coefficients α_{ij} between node i and its neighbour j are computed based on their features, and these coefficients then weight the contributions of neighbours during aggregation. The updated node representation $\tilde{\mathbf{H}}_i$ is then:

$$\tilde{\mathbf{H}}_i = \sigma\left(\sum_{j\in\mathcal{N}_i\cup\{i\}} \alpha_{ij} W \mathbf{H}_j\right) \tag{18}$$

Here, W is a shared linear transformation matrix, $\tilde{\mathbf{a}}$ is a weight vector for the attention mechanism, and σ is a non-linearity [21]. The key difference from the general MP formulation is that $\phi_{msg}(\mathbf{H}_i, \mathbf{H}_j)$ in GAT involves a learnable attention mechanism that computes α_{ij} based on $\mathbf{H}_i$ and $\mathbf{H}_j$, and the message $\mathbf{m}_{ji}$ becomes $\alpha_{ij} W \mathbf{H}_j$. The ϕ_{conv} is then the weighted sum followed by σ.

Laplacian Eigenvectors and Random Walk Positional Encodings

Laplacian Eigenvectors are derived from the spectral decomposition of the graph Laplacian matrix, which is defined as $L = D - \mathbf{A}^{\mathbf{G}}$. laplacian eigenvectors are obtained by solving the eigenvalue problem: $Lu = \lambda u$ More precisely, the eigenvectors corresponding to the k-lowest non-zero eigenvalues are often used for the **i** th node, where **k** is a chosen variable. LapPEs have been shown to be crucial for distinguishing non-isomorphic graphs where standard GNN might fail. They were one of the initial methods used to make transformers graph-aware [6]. By incorporating LapPE we allow the node embeddings to reflect their overall position and role in the graph's structure, giving the node a global positional awareness.

Random Walks Positional Encodings capture the structural role of a node by analysing the probabilities of a random walker returning to that node. They are used to encode the information about the local neighbourhood structure and the node's role within it. The key component is the random walk transition matrix P, which is typically the row-normalised adjacency matrix:

$$P = D^{-1}\mathbf{A}^{\mathbf{G}} \tag{19}$$

The matrix P^k contains the probabilities of transitioning from one node to another in exactly k steps. The RWPE for a node i is often defined using the diagonal elements of the powers of this transition matrix, $(P^k)ii$, which represents the probability of a random walk starting at node i returning to node i after k steps.

For a walk length of L_{max}, the RWPE for node i is given by:

$$PE(i) = [(P)_{ii}, (P^2)_{ii}, ..., (P^{L_{max}})_{ii}] \tag{20}$$

While the positional encoding can be extracted from the sum over rows of diagonal elements of the m-step random walk matrix, giving it a clear purpose within a local cluster, the non-diagonal elements of the matrix can be used to understand its role in the substructure, its position relative to its immediate surroundings.

A.3 Positional adaptations

As no PE have been adapted to work with heterogeneous graphs, the study first aimed to observe how the existing homogeneous implementations interact with such datasets. Due to the aforementioned qualities of our graphs in Equation 14. Our current graph representation G, being passed to such transformation functions, would have no edges to work with as current implementations are looking for edges that connect nodes of the same type. To address this we decide to convert our current graph to a homogeneous version $\mathbf{G}'$ that stores the node type as a feature and transform all nodes and edges to 1 general node and edge type, all other properties remain unchanged. This is only a temporary change that is used in the creation of the Positional Encodings and is reverted back immediately after. As previously mentioned in 4.3, the temporal restrictions are applied separately for each batch; as such, the positional encodings will have to be used at a batch level as well. An implementation that would compute them for the whole graph in advance would suffer from data leakage and would not constitute meaningful results.

Our first positional encoding implementation consists of using G' with the current available homogeneous implementations for both Laplacian and Random Walks. Our model would further use the received embeddings and pass them through as a sinusoidal layer to bring them to the dimensionalities of the node embeddings, from where they will be added to the current node embeddings.

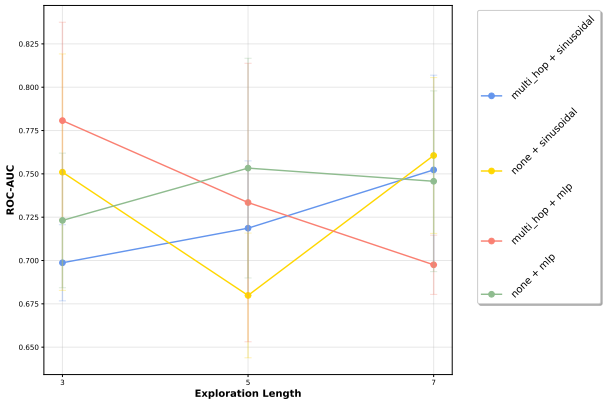
Moving further into our next approach, we still keep G' but we create more complex synthetic edges that can add meaningful information to the current node embeddings. Let $TwoHopNeighbours$ be the function that adds the edges

$$_{ik} = \begin{cases} 1 & \text{if } \left(\exists j \text{ s.t. } (\mathbf{A^G})_{ij} = 1 \wedge (\mathbf{A^G})_{jk} = 1\right) \wedge (\rho(i) = \rho(k)) \\ 0 & \text{otherwise} \end{cases}$$

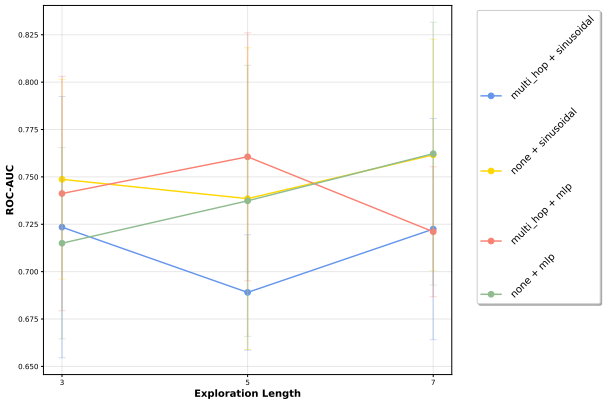
We wish to see if adding such synthetic edges would impact the amount of information the encodings are able to gather.

While using the sinusoidal function to match the embeddings in size we also believed that an approach that uses an MLP to convert the positional encodings to embeddings of the same size as the node embeddings was worth exploring.

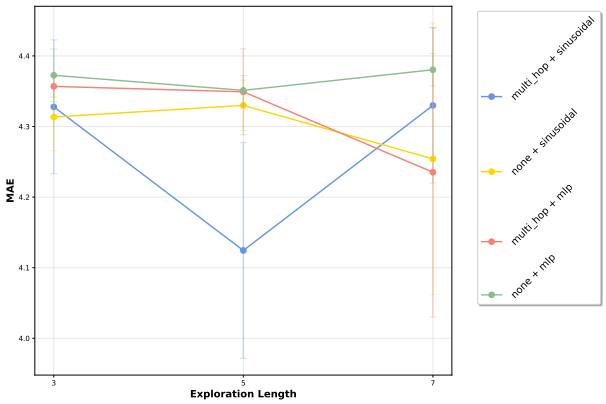
A.4 Positional encodings extras



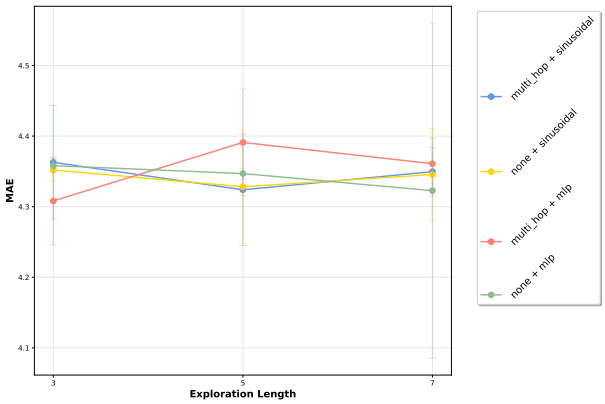
(a) AUROC performance for *driver-top3* with **Cascade-GIN + Laplacian Eigenvector PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the number of lowest non-zero eigenvalues used for computing the PE.



(b) AUROC performance for *driver-top3* with **Cascade-GIN + Random Walk PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the maximum walk length considered.



(a) AUROC performance for *driver-position* with **Cascade-GIN + Laplacian Eigenvector PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the number of lowest non-zero eigenvalues used for computing the PE.



(b) AUROC performance for *driver-position* with **Cascade-GIN + Random Walk PE** across varying exploration lengths. This graph shows the impact of pseudo-edge types (none vs. multi-hop) and embedding adaptation methods (sinusoidal vs. MLP), where "exploration length" refers to the maximum walk length considered.