Relational Deep Learning with Graph Transformers: Exploring Local and Global Message Passing

Ignacio Cuñado Barral[1]
Supervisor(s): Prof. 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: Ignacio Cuñado Barral
Final project course: CSE3000 Research Project
Thesis committee: Prof. Kubilay Atasu, Çağrı Bilgi, Prof. Thomas Höllt

***Abstract*—Graph Transformers have played a key role in the latest graph learning developments. However, their application and performance in Relational Deep Learning (RDL), which has huge potential to remove inefficient data pre-processing pipelines, remain largely unexplored. For this reason, we present adaptations to two well-known Graph Transformer models: a relation-aware local message passing variant (FraudGT) that computes separate attention matrices for each edge and node type; and a simplified global-attention version that ignores heterogeneity (Graphormer). Our analysis demonstrates that local relation-aware attention achieves state-of-the-art results on node classification and regression tasks when evaluated against RelBench tasks, a set of comprehensive RDL benchmarks. We show how local message passing is computationally cheaper, faster, more efficient and more accurate than global attention. Our code is available at https://github.com/ignaciocunado/gt-rdl.**

I. Introduction

Graph Transformers have rapidly become the go-to approach for handling graph reasoning tasks, offering significant improvements over traditional Graph Neural Networks (GNNs) [1] through the use of self-attention. Recent advancements have focused on refining message passing techniques, increasing scalability, efficiency and interpretability [2]. Generally, Graph Transformers can be categorised according to their attention bias. On one hand, local message passing Transformers constrain attention to localised neighbourhoods, while global attention Transformers attend all nodes, sharing information between them. Despite their success, global attention-based Transformers encounter substantial scalability limitations when processing large graphs.

Relational Deep Learning (RDL) uses GNNs to learn directly from relational databases. GNNs have the potential to extract representations that address all database columns, eliminating the need for manual feature engineering, which can introduce human errors, increase costs, and yield suboptimal features [3, 4, 5]. RelBench was introduced to benchmark models in this domain, providing 7 realistic datasets covering node classification, regression, and edge prediction tasks [6]. Recent updates to RelBench have incorporated results from an advanced Relational Graph Neural Network (RelGNN) model [7].

In this work, we investigate how local message-passing compares to global attention mechanisms within Graph Transformers in the context of Relational Deep Learning. Specifically, we implement and evaluate two model variants: FraudGT, which utilises local message passing, and Graphormer, which employs global attention [8, 9]. Our contributions, detailed in section IV, include modifications to the models' attention mechanisms. We thoroughly evaluate their performance on a total of 11 RelBench classification and regression tasks, and conduct an analysis of their runtime and memory usage to measure efficiency, as presented in subsection V-B. Finally, we answer whether global attention mechanisms can simulate or dominate the representational capabilities of purely local message-passing Graph Transformers.

II. Related Work

Since the generalisation of the Transformer to graph learning [10, 11, 12, 13], numerous architectures have integrated self-attention directly into Graph Neural Networks, significantly enhancing their capability to capture complex relationships.

The work on local message passing Transformers has been extensive. Graph Attention Networks introduced Attentional Layers, which perform neighbourhood-constrained self-attention, achieving state-of-the-art results on different graph inductive tasks. Other architectures like LA-MIL and GraphiT further enhanced local attention by integrating novel node and edge encodings for domain-specific tasks like multiple instance learning [14, 15]. Nevertheless, the generalisability and applicability of these methods to Relational Deep Learning is unclear.

Graphormer was one of the first models to introduce global attention, demonstrating the ability of Transformers to learn meaningful representations of graphs [8]. However, the quadratic complexity of Graphormer's layers limits its scalability, restricting its application primarily to graph classification on small homogeneous graphs (±200 nodes and edges). The computational demands of Graphormer mean it often requires days of training on multiple GPUs, which is often infeasible for most applications. Alternative global attention methods, such as the second variant of UGFormer, proposed by Quoc et al., interleave Transformer layers with GNN layers [12]. However, this model was very specialised on text classification tasks. GOAT introduced dimensionality reduction techniques to make global attention more tractable [16]. Despite all the advancements, the scalability of these and other global attention models to larger graphs, particularly for inductive node reasoning tasks, remains underexplored.

Structural encodings have been proved essential in many global attention models [17]. While some approaches propose using pairwise shortest paths, others claim that eigenvectors of the Laplacian Matrix are also a good option [18]. However, the caveat with most of these options is that they do not scale well with graph size, limiting their viability to small graphs. Random Walks-based encodings could solve this problem, as the walk length can be chosen in advance [19]. Even so, applying these methods effectively to large heterogeneous graphs still presents challenges.

Lastly, there is a literature gap in the context of Relational Deep Learning and heterogeneous graphs [3]. Few Transformer and GNN models have been generalised to work with heterogeneous data [20, 21], and most implement only local message passing solutions. In the context of RDL, Chen et al. recently introduced RelGNN, a new architecture based on message passing designed to learn specific patterns in relational graphs, but with no attention mechanism [7]. Architectures like Relphormer solve knowledge graph tasks with a hybrid local-global mechanism [22]. Despite the numerous published surveys that review state-of-the-art models and architectures [23, 24], few works actually compare a purely local message

passing architecture with one that incorporates information from all the nodes in the graph in a standardised set of benchmarks. It is common among these surveys to only explore the theoretical complexity of different types of attention and inductive biases [25], but not to propose or evaluate models which generalise well to different graph topologies.

III. Background

A. Graph Neural Networks and Graph Transformers

Consider an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ denotes the set of edges. The adjacency matrix of G is represented by $A \in \{0,1\}^{N\times N},\ A = A^\top,\ A_{ij} = \mathbb{1}_{\{(i,j)\in\mathcal{E}\}}$. The neighbours of node i are $\mathcal{N}(i) = \{\, j : (i,j) \in \mathcal{E}\}$. Each node $i \in \mathcal{V}$ has an associated d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$, and each edge $(i,j) \in \mathcal{E}$ may also have features denoted by $\mathbf{e}_{ij} \in \mathbb{R}^n$

Graph Neural Networks are an extension of the Artificial Neural Network model, and they aim to learn expressive node (and edge) embeddings. For that reason, they are used for tasks in which the data can be represented as a graph, like molecules or transaction networks. In the traditional GNN framework, the feature representation of each node is updated by iteratively aggregating information from its neighbours until convergence is reached [26]. This is known as message passing, and can be represented as:

$$\begin{aligned} \mathbf{m}_i^{(l)} &= \mathrm{AGG}^{(l)}\big(\{\mathbf{x}_j^{(l-1)} : j \in \mathcal{N}(i)\}\big), \\ \mathbf{x}_i^{(l)} &= \mathrm{UPDATE}^{(l)}\big(\mathbf{x}_i^{(l-1)},\, \mathbf{m}_i^{(l)}\big), \end{aligned} \tag{1}$$

where $\mathbf{m}_i^{(l)}$ and $\mathbf{x}_i^{(l)}$ are the message vector and the feature representation of node i at layer l, respectively. The functions $\mathrm{AGG}^{(l)}$ and $\mathrm{UPDATE}^{(l)}$ aggregate information coming from neighbouring nodes, and update the hidden representation of each node, allowing information to propagate through the graph. Naturally, adding k message-passing layers allows information to reach nodes up to k hops away.

While traditional GNNs rely strictly on local aggregation, Graph Transformers have emerged as a promising alternative due to their ability to capture long-range dependencies through self attention. Graph Transformers are a generalisation of the original Transformer architecture into Graph Neural Networks, leveraging different attention mechanisms and biases to encode the graph's structure and node positions [11, 23]. In each Transformer layer, multi-head attention is performed followed by a Feed Forward Neural Network (FFN). This can be formally defined as: let $X = [x_1, x_2, \ldots, x_n]^\top \in \mathbb{R}^{n\times d_n}$, be the node feature matrix of $\mathcal{G}$, a graph with n nodes. In each layer l ($l > 0$), the hidden feature matrix is $\mathbf{H}^{(l-1)} \in \mathbb{R}^{n\times d_n}$. Attention linearly projects the input $\mathbf{H}^{(l-1)}$ to query, key, and value spaces with learnable projection matrices $\mathbf{W}$ [10]:

$$\mathbf{Q}^{(l)} = \mathbf{H}^{(l-1)}\,\mathbf{W}_Q^{(l)}, \tag{2}$$

$$\mathbf{K}^{(l)} = \mathbf{H}^{(l-1)}\,\mathbf{W}_K^{(l)}, \tag{3}$$

$$\mathbf{V}^{(l)} = \mathbf{H}^{(l-1)}\,\mathbf{W}_V^{(l)}, \tag{4}$$

where $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)} \in \mathbb{R}^{d_n\times d_h}$. The scaled dot-product is then computed via:

$$\mathrm{Attn}_l(\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)}) = \Big(\mathrm{softmax}\big(\tfrac{\mathbf{Q}^{(l)}(\mathbf{K}^{(l)})^\top}{\sqrt{d_h}}\big)\,\mathbf{V}^{(l)}\Big)\,\mathbf{W}_O^{(l)}, \tag{5}$$

where $\mathbf{W}_O^{(l)} \in \mathbb{R}^{d_n\times d_n}$ acts as an output projection. Dividing by $\sqrt{d_h}$ prevents large dot-products from saturating softmax. Multi-Head Attention (MHA) is then computed for h heads via

$$\mathrm{MHA}\big(\mathbf{H}^{(l-1)}\big) = \big\|_{h=1}^{H} \mathrm{Attn}_l(\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)}), \tag{6}$$

by repeating this process for each head. Note that $\|$ represents the concatenation operator. After MHA has been computed, outputs are forwarded through the FFN with residual connections.

Different attention mechanisms perform either local attention or global attention. This can be seen in Figure 1, depicting the attention computation between two nodes. In this case, **A** (local message passing) shows that only direct neighbours are attended by using techniques like a direct neighbour mask. In contrast, global attention (**B**) attends every node in the graph. A helpful analogy for global attention is to imagine the graph as fully connected, allowing message passing between each pair of nodes [8]. Consequently, global attention is more computationally expensive than local message passing, with a complexity of $\mathcal{O}(n^2)$ for n nodes, since now every vertex has to attend every other node in the graph. However, this approach has the possibility to aggregate information between distant nodes in the graph, potentially improving prediction accuracy.

Since global models tend to disregard the edge structure, they rely on structural and positional encodings to represent the relative position of nodes and the overall structure of the graph in non-Euclidean space, adding to their overall complexity.

Finally, a prediction head is added for downstream tasks like node classification or edge prediction. This head can be, for example, a Multi Layer Perceptron (MLP) or a simpler linear projection layer.

B. Heterogeneous Graphs

Consider a modified version of the graph introduced in subsection III-A. A heterogeneous graph contains different types of nodes and edges. That is, $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R})$, where similarly to before, $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ denotes the set of edges. Each node $v \in \mathcal{V}$ and each edge $(i,j) \in \mathcal{E}$ are now associated with type mapping functions $\tau(v) : \mathcal{V} \to \mathcal{A}$ and $\phi(i,j) : \mathcal{E} \to \mathcal{R}$. In heterogeneous graphs, each node $i \in \mathcal{V}$ of type $\tau(i)$ has an associated $d_{\tau(i)}$-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^{d_{\tau(i)}}$.

C. Relational Deep Learning

Relational Deep Learning represents relational databases as temporal, heterogeneous Relational Entity Graphs (REG). Each row in each relation defines a node, columns define node features, and primary-foreign key links define edges [3].

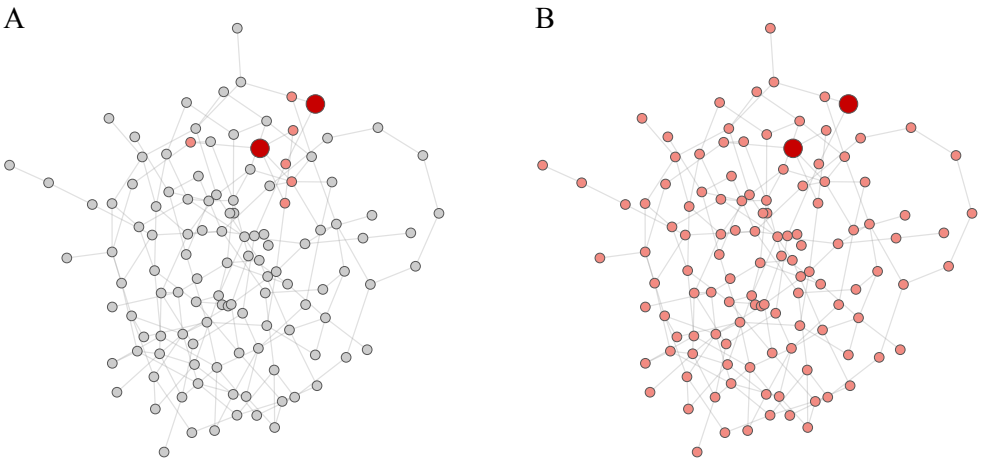

Fig. 1: Comparison of local message-passing and global attention mechanisms in a Graph Neural Network. (a) Local message-passing aggregates information only from nodes in the direct neighbourhood. (b) Global attention enables each node to attend to all other nodes, capturing long-range dependencies.

Since most databases have multiple tables, the graph will have different node and edge types. Moreover, as the number of columns changes per relation, the dimensionality of the feature vectors will also vary.

GNNs can be applied to build end-to-end predictive tasks on the relational entity graph. This practice comes with its own challenges. As mentioned in subsection III-B, heterogeneous data differs from its homogeneous homonymous because there are different types of nodes and edges. If we take the example of the *Rel-F1* dataset (Formula 1 statistics) [27], a node can represent either a driver, race, circuit, constructor, etc. Because of this, different node types will have a different number of features with varying data types.

IV. Contributions

We propose simple adaptations to well-known Graph Transformers that enable performing node classification and regression tasks on heterogeneous data. We start by explaining how we added edge features to REGs. We then introduce how relation-aware attention is calculated on the local MP model, the adaptations for the global attention model, and in subsection V-B we explain how the results support our design choices.

A. Edge Features

When converting a database to a REG, there are no features to add to edges. This is due to the fact that edges represent relationships between tables via foreign-key constraints, which do not contain data per se.

However, the majority of graph attention mechanisms rely on edge features to compute attention weights. For that reason, edge features need to be manually added to the graph.

We propose concatenating source and destination node features into the edge features, so that, if e_{ij} denotes the features of the edge connecting nodes i and j, each with hidden feature vector h:

$$e_{ij} = h_i || h_j \tag{7}$$

B. Local Message Passing

We use an adapted version of FraudGT [9] as our local message passing Transformer. FraudGT's powerful localised attention combined with its edge-based message passing gate offer a strong foundation for our extensions.

In order to build an attention mechanism for heterogeneous data, we propose using relation-aware attention. This mechanism learns projection matrices W_Q, W_K, W_V per node type. In this case, the attention score $\alpha_{ij}^{(h,l)}$ of attention head h in layer l between neighbouring nodes v_i and v_j when type $\tau(i) = \tau(j)$ is:

$$\mathbf{q}_i^{(h,l)} = \left(\mathbf{h}_i^{(l-1)}\right)^T \mathbf{W}_{Q,\tau(i)}^{(h,l)}, \tag{8}$$

$$\mathbf{k}_j^{(h,l)} = \left(\mathbf{h}_j^{(l-1)}\right)^T \mathbf{W}_{K,\tau(j)}^{(h,l)}, \tag{9}$$

$$\hat{\boldsymbol{\alpha}}_{ij}^{(h,l)} = \frac{\mathbf{q}_i^{(h,l)} \odot \mathbf{k}_j^{(h,l)} + \mathbf{b}_{ij}^{(h,l)}}{\sqrt{d_k}},$$

$$\boldsymbol{\alpha}_{ij}^{(h,l)} = \text{softmax}\left(\hat{\boldsymbol{\alpha}}_{ij}^{(h,l)}\right) \tag{10}$$

where $\mathbf{h}_i^{(l-1)}$ denotes the feature of node v_i, and $\mathbf{b}_{ij}^{(h,l)}$ is the edge attention bias defined by:

$$\mathbf{b}_{ij}^{(h,l)} = (\mathbf{e}_{ij}'^{(l)})^T \mathbf{W}_{E,\phi(i,j)}^{(h,l)} \in \mathbb{R}^{d_h} \tag{11}$$

where $\mathbf{W}_{E,\phi(i,j)}^{(h,l)}$ is a learnable matrix per edge-type. We compute the edge-based message passing gate via $\mathbf{G}_{ij}^{(h,l)} = \sigma((\mathbf{e}_{ij}'^{(l)})^T \mathbf{W}_{G,\phi(i,j)}^{(h,l)})$ with the only difference being that we compute it separately for each edge type. $W_G^{(h,l)} \in \mathbb{R}^{d_e \times d_h}$

is another learnable matrix, and σ is the sigmoid function. Finally, MHA is computed via:

$$\boldsymbol{\beta}_{ij}^{(h,l)} = \boldsymbol{\alpha}_{ij}^{(h,l)} (\mathbf{h}_j^{(l-1)})^T \mathbf{W}_{V,\tau(i)}^{(h,l)} \odot \mathbf{G}_{ij}^{(h,l)}$$

$$\mathrm{MHA}(\mathbf{h}_i^{(l-1)}) = \|_h \left(\oplus_{\forall j \in \mathcal{N}(i)} \left(\boldsymbol{\beta}_{ij}^{(h,l)} \right) \right)^T \mathbf{W}_{O_n}^{(l)}, \quad (12)$$

$$\mathrm{MHA}(\mathbf{E}_{ij}^{\prime(l-1)}) = \|_h \left(\hat{\boldsymbol{\alpha}}_{ij}^{(h,l)} \right)^T \mathbf{W}_{O_e}^{(l)}, \quad (13)$$

which applies a direct neighbour mask to only attend those nodes connected by an edge, and where $\mathbf{W}_O$ are learnable matrices. $\oplus$ and $\odot$ denote element-wise addition and the Hadamard product (element-wise multiplication), respectively.

Since REGs are not directed multigraphs, in which there can be multiple edges in both directions between two nodes, ego IDs and port numbering [28, 29] have not been implemented as in the original FraudGT model. In contrast, reverse message passing [30] is already added when constructing the schema graph during the process of converting a database into a REG. These *inverse links* are added to ensure all tables are reachable within the graph [3].

Lastly, the final prediction heads have been devised as a two-layered MLP. In the case of the inductive node regression head, the only change is the addition of L2 regularisation.

C. Global Attention

As discussed earlier, Graphormer's architecture and attention mechanisms need to be adapted to work with heterogeneous graphs [8]. In this case, we decided to adapt our graph structure to work with this model. Adapting heterogeneous graphs into a homogeneous format simplifies the architecture, reducing complexity and computational overhead, enabling efficient utilization of Graphormer's existing mechanisms. This is achieved by treating the graph as homogeneous once initial node embeddings have been built and hidden dimensionality is constant throughout the graph. We build a full-graph hidden node feature matrix $H \in \mathbb{R}^{n \times d_n}$ by concatenating the per-type node features into a single matrix. In this case, n denotes the total number of nodes in the graph (achieved by summing over the node types) and d_h is the dimension of our hidden layers, also known as the number of channels. Then, in the last layer, we can reconstruct the hidden per-type node matrix for downstream tasks.

When it comes to Graphormer's structural encodings, the original model leverages shortest paths to encode the relative position of nodes in a graph and edge features. However, this quickly becomes unfeasible for larger graphs. Therefore, finding scalable spatial encoding methods remains an active research area. Due to computational constraints, we omitted spatial encodings and thus edge features, which might limit the model's capacity to fully capture structural information in larger relational graphs. We believe that investigation into scalable positional encodings for large REGs should be considered for future work and further study.

Centrality encodings have been kept the same in order to add a notion of node importance in the graph. More concretely, Graphormer leverages *degree centrality*, defined as:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)}, \quad (14)$$

which learns embeddings for the in-degree (z^+) and out-degree (z^-) of each node. In this case, we also treat the graph as homogeneous and just sum the number of incoming and outgoing edges without taking their type into account.

Modifying Graphormer's prediction head allows us to use this well-known architecture for node classification and regression rather than graph classification tasks. Therefore, similarly to our local message passing implementation, our inductive heads are two-layered fully connected MLPs, with a final logistic activation function for the classification head.

V. Experiments

In this section we present a thorough set of experiments which evaluate our local and global message-passing implementations. We compare their performance against a set of RelBench benchmarks, and measure their average runtime and memory usage. subsection V-A states the setup used for the experiments, describing the datasets, baselines and training; whilst subsection V-B explores the results.

A. Experimental Setup

Datasets and tasks. RelBench provides a set of 7 realistic databases from publicly available repositories which span different orders of magnitude. Each dataset contains its own machine learning tasks like node classification, regression and edge prediction. Graph sizes vary between 90,000 nodes in the smallest dataset and more than 30,000,000 in the largest one. RelBench provides a comprehensive set of baselines against which models can be tested. Out of the 7 available datasets, 2 of them (*rel-amazon* and *rel-event*) are too heavy to fit in a reasonable amount of memory [31, 32, 33]. Thus, we have chosen not to evaluate our models on these datasets. Due to time and computational constraints, we only evaluated a representative subset of tasks. For node classification, we have randomly chosen *driver-top3* and *driver-dnf* from the F1 dataset, *user-churn* from the HM dataset [34], *study-outcome* from the Trial dataset [35], *user-engagement* from Stack [36] and *user-visits* from Avito [37]. Node regression benchmarks will be run on *driver-position* from F1, *ad-ctr* from Avito, *item-sales* from HM and *site-success* as well as *study-adverse* from Trial. Nevertheless, the proposed architectures and methods are applicable to other heterogeneous graphs.

Splits. Train, test and validation data splits are automatically computed using a temporal split methodology. Every dataset contains two critical temporal demarcation points: a validation timestamp t_{val} and a test timestamp t_{test}, which are shared across all predictive tasks within a given dataset.

Baselines. For each task, the original RelBench paper published benchmark results that compares its RDL implementation against LightGBM [3, 38, 39]. We evaluate our models

Dataset	Task	RDL	Local Message Passing	Relative Gain	Global Attention	Relative Gain
F1	*driver-top3*	75.54±0.63	**82.99±0.87**	9.86%	76.67±2.80	1.50%
	driver-dnf	72.62±0.27	**73.81±0.70**	1.64%	69.72±2.53	-3.99%
HM	*user-churn*	69.88±0.21	**70.30±0.30**	0.60%	67.49±0.033	-3.42%
Trial	*study-outcome*	68.60±1.01	**69.28±0.32**	0.99%	67.42±0.585	-1.72%
Stack	*user-engagement*	**90.59±0.09**	**90.52±0.10**	-0.08%	87.59±1.90	-3.31%
Avito	*user-visits*	**66.20±0.10**	64.92±0.20	-1.93%	63.72±0.44	-3.75%

(a) Node classification results

Dataset	Task	RDL	Local Message Passing	Relative Gain	Global Attention	Relative Gain
F1	*driver-position*	4.022±0.119	**3.925±0.062**	2.41%	4.025±0.085	-0.07%
HM	*item-sales*	0.056±0.000	**0.052±0.003**	7.14%	0.076±0.000	-35.89%
Trial	*site-success*	0.400±0.020	**0.376±0.024**	6.00%	0.431±0.008	-7.75%
	study-adverse	44.473±0.209	**43.439±0.771**	2.33%	46.596±0.549	-4.77%
Avito	*ad-ctr*	0.041±0.001	**0.037±0.001**	9.76%	0.039±0.001	4.88%

(b) Node regression results

TABLE I: Node classification and regression results for different RelBench tasks. Each data point represents the mean ± 1 standard deviation over 5 runs. Best results are in bold. (a) shows our classification results using the ROC-AUC metric (higher is better). (b) is our node regression results with the MAE metric (lower is better).

against these baselines and assess the differences for statistical significance.

Training. A Bayesian hyperparameter search [40] with 100 trials was run for each model. We then trained each model with 1 or 2 Transformer layers, between 5 and 15 epochs to avoid overfitting. All models were trained with the AdamW optimiser [41]. Generally, global attention uses a lower batch size than local message passing. This is due to the fact that global attention's space complexity is quadratic, so there is a limit on the number of subgraphs we can sample for our batches as all of them need to fit in memory at once. For each node, we sample between 25 and 150 neighbours in two steps. A full overview of the hyperparameters used during training and testing can be seen under section A.

System. All models were trained and tested on a system with a single Nvidia A40 GPU and 64GB of RAM memory and developed with PyTorch Geometric, a framework built on top of PyTorch [42, 43].

Memory and runtime tests. For these tests, we compare the training times and GPU memory usage for our local and global attention implementations. We train both models with the same parameters for 10 epochs on the same machine, and measure the average runtime and memory use over 5 runs on the same 11 tasks. More concretely, we use 1 Transformer layer with 32 hidden dimensions, a batch size of 32 and sample 100 neighbours per node in two steps.

B. Results

We divide our results into classification and regression tasks. For each task, we present our best results averaged over 5 runs for both the local and global attention implementations, as well as the Relational Deep Learning results from the RelBench paper (labelled as RDL). Lastly, to ensure reproducibility and fairness, each run was performed with a different random seed (numbered 1 to 5). Please note that the reported results correspond to the best-performing configuration for each model.

Classification. Table Ia presents our results of predicting a binary label for a given entity at a given seed time. We use the area under the Receiver Operating Characteristic curve (ROC-AUC) metric [44], where a higher score is better than a lower score.

Our local message-passing Transformer consistently outperforms the baseline RDL implementation on most evaluated tasks. On tasks like *user-engagement*, both models show the same performance within statistical significance. Our FraudGT implementation underperforms RDL in the *user-visits* task, with multiple experiments showing that learning plateaus at around 64% AUC. This could likely be due to high class imbalance (90.5% of positives). On the other hand, the global attention implementation is able to reproduce RDL's results in some tasks. However, in general, it conveys signs of unstable learning, as shown by the high standard deviations. Additionally, global attention is more sensitive to random weight initialisation and demonstrates inferior data-driven learning compared to our local message-passing model.

Regression. Table Ib showcases our results of predicting full numerical labels for nodes of a graph at a given seed time. We use the mean absolute error (MAE) metric, where a lower score is better.

Our local message passing Transformer is generally able to outperform the Relational Deep Learning implementation of Robinson *et al.* in regression tasks. It is also clear that global attention performs better in node regression than node classification tasks, as experiments have generally lower standard deviations, suggesting a more stable training. In addition, global attention is able to reproduce the baseline scores in two tasks, indicating room for improvement in the rest.

Trainable Parameters. During the runtime and memory tests, the parameter count per model was recorded in Table II (note that we only record parameters per-datasets, as tasks from the same datasets are trained with the same data).

Dataset	Local Message Passing	Global Attention
F1	968,293	517,698
HM	313,689	2,217,810
Trial	1,640,369	13,831,394
Stack	884,705	4,773,362
Avito	705,347	76,348,674

TABLE II: Number of trainable parameters for the local message passing and global attention models across five datasets: F1, HM, Trial, Stack and Avito.

FraudGT shows a different parameter count on different datasets because attention is computed per node and edge type. If K denotes the number of node types in a graph (or tables in a database), our model will have to learn more attention matrices as K increases (Equation 8), the same applies for edges. Despite this, the number of parameters for Graphormer tends to be larger than FraudGT due to the increased layer complexity and computationally expensive encoding schemes. A more detailed insight into the number of parameters can be found under section B.

Runtime. Training 10 *driver-top3* epochs on Graphormer is, on average, over 2 times slower than on FraudGT as seen on Figure 2. This is even more notable on *ad-ctr*, being over 18 times slower. Results show how global attention is indeed much more complex and expensive, which makes Graphormer slower on training and inference.

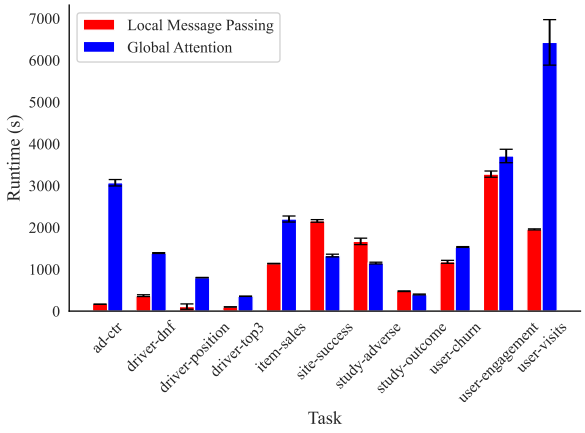

Fig. 2: Mean training time for 10 epochs of our local message passing model (FraudGT) and global attention model (Graphormer) across all evaluated tasks; error bars represent one standard deviation over five independent runs.

Memory usage. As with runtime, Figure 3 shows how Graphormer has a much higher average memory usage than FraudGT often even filling up the memory on our 48GB GPU. Graphormer's memory footprint shows a much higher standard deviation, likely due to a combination between increased complexity and variation in subgraphs due to random batch sampling. Looking at both runtime and memory, we can see that the highest memory spikes are related to the slowest Graphormer trainings.

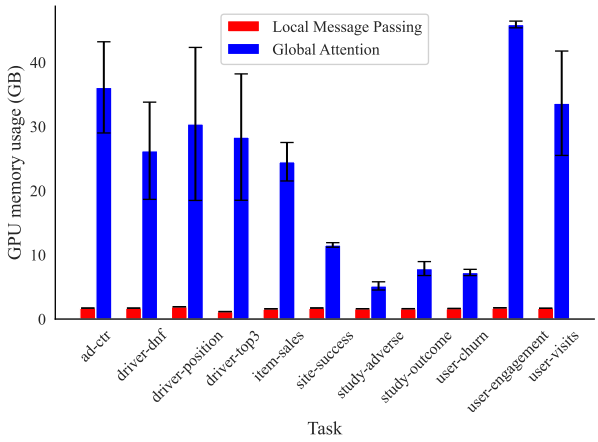

Fig. 3: Mean GPU memory allocated during a 10-epoch training run of our local message passing model (FraudGT) and global attention (Graphormer) across all evaluated tasks; error bars denote ±1 standard deviation over five independent runs.

VI. Discussion

Our results highlight that local message passing models like FraudGT can outperform GNN-based Relational Deep Learning implementations. Their relatively low complexity and memory usage, as well as fast training times make it a good option for a variety of scenarios. The use of purely local self-attention allows FraudGT to efficiently focus on the important nodes, since it is not uncommon that relevant nodes are close together.

In contrast, our findings have shown that naïve implementations of global attention Graph Transformers like Graphormer cannot simulate the expressive power of purely local message passing models. Moreover, even with reduced batch sizes and a limited number of neighbours, training Graphormer demands the maximum memory capacity of even the latest GPUs. This limitation requires significantly smaller batch sizes, and combined with higher per-layer complexity, results in notably longer training and inference times. All of this leads us to conclude that global attention could be better suited for graph classification rather than node classification or regression. This is due to the fact that attending all nodes at the same time may dilute the individual details that allow local message passing to predict unseen node labels with higher accuracy.

Even though our results suggest that global attention may not justify the additional computational resources required over local message-passing methods, there exists the possibility that this could be due to our model choice. Graphormer was originally designed to perform graph classification tasks on small graphs, not individual node inductive tasks on large graphs. In addition, the lack of scalable positional encodings for heterogeneous graphs could also hinder Graphormer's performance on these tasks. For that reason, this and other extensions have been devised as future work.

Lastly, during our analysis, we observed unexpectedly stable baseline performance. Most baselines were easily reproducible with small adaptations basic models like GIN [45], yet difficult to improve upon. To investigate this phenomenon, we conducted a comprehensive exploratory data analysis to inspect label distributions, feature correlations and temporal sampling. We find that temporal splits, inherent to RDL lead to distribution shifts due to natural changes in the data; and that most regression tasks show skewed distributions, making true learning hard. A complete overview of this analysis can be seen under section C.

VII. Conclusions and Future Work

In this work, we have proposed adaptations to popular Graph Transformers in order to use them for end-to-end Relational Deep Learning. We have then compared these implementations against RelBench baselines across a variety of node classification and regression tasks.

We have demonstrated how local message passing has the potential to produce state of the art results on some of these tasks, and how current developments in global attention biases for Graph Transformers are unable to match RDL results. Finally, we show that even if global attention was capable of matching local attention's representational power, choosing it over local message passing would not be a trivial decision due to its increased computational costs.

Further experiments would involve running tests on the remaining tasks and other relational databases to evaluate how local and global attention mechanisms perform on a broader set of tasks. Additionally, the models could be extended to run edge prediction tasks by designing a new link prediction head. This would allow us to determine whether our implementations are as effective at predicting edges as they are in node classification and regression.

Combining local and global attention layers could allow the model to learn the specific node details to perform node inference while keeping a global overview of the graph.

Improving Graphormer. Our global attention model has potential to improve. Looking for ways to make the model less sensitive to random initialisation and investigating towards more stable learning are interesting research directions. This could be achieved, for example, by finding more efficient positional and spatial encodings, or by modifying or even eliminating the existing centrality encodings. These positional biases would likely enhance Graphormer's performance by adding a notion of structure and position to the model, this is needed as nodes are arranged in a non-Euclidean space rather than a sequence. However, this task poses challenges in itself, as graphs tend to be large and heterogeneous, and often the task of computing these encodings is too computationally expensive. Adding a node-type bias should also be considered. This embedding could be computed and introduced in the form of an extra node feature, which would allow the model to keep a notion of heterogeneity. Lastly, training the model with batch accumulation would allow for training the model with a larger batch size, potentially improving results.

VIII. Responsible Research

This project aims to contribute to the ongoing research on Graph Neural Networks, Graph Transformers and end-to-end Relational Deep Learning in a safe, fair and ethical way. The end goal is always to help progress the field of machine learning.

LLM tools (*o4-mini-high* [1]) were used exclusively for the following tasks: summarising and explaining text, generating plotting scripts and aiding in the snowballing process to find related work.

A. Research Ethics

We are committed to the principles of integrity, diversity, and engagement as outlined in the TU Delft's Code of Conduct and the Netherlands Code of Conduct for Research Integrity. Specifically, integrity involves conducting research in a transparent, honest, and responsible manner. To achieve this, we have meticulously documented our methodologies and clearly report all findings, including negative or inconclusive results, to maintain scientific rigour and reliability.

B. Reproducibility

To ensure our experimental setup is reproducible, model hyperparameters have been made public in section A. By running each experiment 5 times with different random seeds (1, 2, 3, 4 and 5), we ensure that our results are not based on lucky initialisations of learnable parameters.

The code used to develop the models is also publicly accessible, and required dependencies can be installed via standard package managers like pip [2]. Lastly, all datasets used come from publicly available repositories licensed for research use.

IX. Acknowledgments

Research reported in this work was partially or completely facilitated by computational resources and support of the Delft AI Cluster (DAIC) [46] at TU Delft (RRID: SCR_025091), but remains the sole responsibility of the authors, not the DAIC team.

References

[1] C. Sanford, B. Fatemi, E. Hall, A. Tsitsulin, M. Kazemi, J. Halcrow, B. Perozzi, and V. Mirrokni, "Understanding transformer reasoning capabilities via graph algorithms," 2024.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[3] M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec, "Relational deep learning: Graph representation learning on relational databases," 2023.

[1] o4-mini-high: https://openai.com/index/introducing-o3-and-o4-mini/

[2] Pip: https://pypi.org/project/pip/

[4] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 979–984, 2016.

[5] J. Adamczyk and F. Malawski, "Comparison of manual and automated feature engineering for daily activity classification in mental disorder diagnosis," *Computing and Informatics*, vol. 40, p. 850–879, Dec. 2021.

[6] J. Robinson, R. Ranjan, W. Hu, K. Huang, J. Han, A. Dobles, M. Fey, J. E. Lenssen, Y. Yuan, Z. Zhang, X. He, and J. Leskovec, "Relbench: A benchmark for deep learning on relational databases," 2024.

[7] T. Chen, C. Kanatsoulis, and J. Leskovec, "Relgnn: Composite message passing for relational deep learning," 2025.

[8] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 28877–28888, Curran Associates, Inc., 2021.

[9] J. Lin, X. Guo, Y. Zhu, S. Mitchell, E. Altman, and J. Shun, "Fraudgt: A simple, effective, and efficient graph transformer for financial fraud detection," in *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, (New York, NY, USA), p. 292–300, Association for Computing Machinery, 2024.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[11] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," 2021.

[12] D. Q. Nguyen, T. D. Nguyen, and D. Phung, "Universal graph transformer self-attention networks," 2022.

[13] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," 2020.

[14] D. Reisenbüchler, S. J. Wagner, M. Boxberg, and T. Peng, "Local attention graph-based transformer for multi-target genetic alteration prediction," 2022.

[15] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," 2021.

[16] K. Kong, J. Chen, J. Kirchenbauer, R. Ni, C. B. Bruss, and T. Goldstein, "GOAT: A global transformer on large-scale graphs," in *Proceedings of the 40th International Conference on Machine Learning* (A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, eds.), vol. 202 of *Proceedings of Machine Learning Research*, pp. 17375–17390, PMLR, 23–29 Jul 2023.

[17] F. Grötschla, J. Xie, and R. Wattenhofer, "Benchmarking positional encodings for gnns and graph transformers," 2024.

[18] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," 2023.

[19] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," 2022.

[20] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, "Simple and efficient heterogeneous graph neural network," 2023.

[21] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," 2020.

[22] Z. Bi, S. Cheng, J. Chen, X. Liang, F. Xiong, and N. Zhang, "Relphormer: Relational graph transformer for knowledge graph representations," *Neurocomputing*, vol. 566, p. 127044, Jan. 2024.

[23] A. Shehzad, F. Xia, S. Abid, C. Peng, S. Yu, D. Zhang, and K. Verspoor, "Graph transformers: A survey," 2024.

[24] C. Yuan, K. Zhao, E. E. Kuruoglu, L. Wang, T. Xu, W. Huang, D. Zhao, H. Cheng, and Y. Rong, "A survey of graph transformers: Architectures, theories and applications," 2025.

[25] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[27] F1DB Contributors, "F1DB." https://github.com/f1db/f1db, 2025. Data from February 2024.

[28] J. You, J. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," 2021.

[29] R. Sato, M. Yamada, and H. Kashima, "Approximation ratios of graph neural networks for combinatorial problems," 2019.

[30] G. Jaume, A. phi Nguyen, M. R. Martínez, J.-P. Thiran, and M. Gabrani, "edgnn: a simple and powerful gnn for directed labeled graphs," 2019.

[31] J. Ni, "Amazon review data." https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/, 2018.

[32] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), (Hong Kong, China), pp. 188–197, Association for Computational Linguistics, Nov. 2019.

[33] A. Carroll, Avalon, B. Hamner, events4you, glhf, and G. Melton, "Event recommendation engine challenge." https://kaggle.com/competitions/event-recommendation-engine-challenge, 2013. Kaggle.

[34] C. G. Ling, ElizabethHMGroup, FridaRim, inversion, J. Ferrando, Maggie, neuraloverflow, and xlsrln, "H&m personalized fashion recommendations." https://kaggle.com/competitions/h-and-m-personalized-fashion-recommendations, 2022.

Kaggle.
[35] Clinical Trials Transformation Initiative (CTTI), "Aggregate Analysis of ClinicalTrials.gov (AACT) Database." https://aact.ctti-clinicaltrials.org/, 2025.
[36] S. E. Community, "Stack exchange data dump." https://archive.org/details/stackexchange, 2024.
[37] I. Guz, night_bat, and W. Kan, "Avito context ad clicks." https://kaggle.com/competitions/avito-context-ad-clicks, 2015. Kaggle.
[38] W. Hu, Y. Yuan, Z. Zhang, A. Nitta, K. Cao, V. Kocijan, J. Sunil, J. Leskovec, and M. Fey, "Pytorch frame: A modular framework for multi-modal tabular learning," 2024.
[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
[40] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," 2018.
[41] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.
[42] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019.
[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.
[44] J. Hanley, "A method of comparing the areas under receiver operating characteristic curves derived from the same cases," *Radiology*, vol. 148, pp. 839–43, 10 1983.
[45] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2019.
[46] Delft AI Cluster (DAIC), "The delft ai cluster (daic), rrid:scr$_0$25091,$''$ 2024.

APPENDIX A
HYPERPARAMETERS

A. FraudGT

Dataset	Task	Layers	Epochs	Learning Rate	Batch Size	Channels	Neighbours	Dropouts (Local, Global, Attn.)
f1	driver-top3	1	15	0.005	256	32	[128,128]	[0,0,0]
	driver-dnf	1	5	0.001	256	32	[128,128]	[0.1, 0.2, 0.3]
	driver-position	1	10	0.003	1028	32	[150,150]	[0.1, 0.2, 0.1]
hm	user-churn	2	15	0.001	256	32	[128,128]	[0,0,0]
	item-sales	2	10	0.001	256	32	[128,128]	[0,0,0]
trial	study-outcome	2	10	0.001	512	32	[128,128]	[0.1,0.1,0.1]
	site-success	2	4	0.001	256	64	[150,150]	[0.2,0.1,0.1]
	study-adverse	2	5	0.001	256	32	[150,150]	[0,0,0]
stack	user-engagement	1	10	0.001	2048	32	[128,128]	[0.1, 0.1, 0.1]
avito	user-visits	3	10	0.001	1028	64	[128,128]	[0,0,0]
	ad-ctr	2	10	0.001	256	64	[150,150]	[0.3,0.1,0.15]

TABLE III: Task-specific optimal hyperparameter configuration for our local message-passing implementation.

B. Graphormer

Dataset	Task	Layers	Epochs	Learning Rate	Batch Size	Channels	Neighbours	Dropouts (Dropout, Attn., Activation)
f1	driver-top3	1	10	0.001	128	32	[25,25]	[0.1,0.1,0.1]
	driver-dnf	1	5	0.002	32	32	[100,100]	[0.2,0.2,0.2]
	driver-position	1	4	0.001	128	32	[25,25]	[0.15,0.15,0.15]
hm	user-churn	1	15	0.0005	32	32	[100,100]	[0.15,0.15,0.15]
	item-sales	1	5	0.0005	64	32	[50, 50]	[0.2,0.2,0.2]
trial	study-outcome	1	5	0.001	64	64	[50, 50]	[0.2,0.2,0.2]
	site-success	1	10	0.001	32	32	[100,100]	[0.0,0.0,0.0]
	study-adverse	1	10	0.001	32	32	[100, 100]	[0.0,0.0,0.0]
stack	user-engagement	2	10	0.001	64	32	[40,40]	[0.2,0.2,0.2]
avito	user-visits	1	10	0.001	64	64	[50, 50]	[0.1,0.1,0.1]
	ad-ctr	1	5	0.001	64	32	[50, 50]	[0.2,0.2,0.2]

TABLE IV: Task-specific optimal hyperparameter configuration for our global attention implementation.

APPENDIX B
TRAINABLE PARAMETERS

This section introduces some insights into how the number of parameters grows with the number of dataset tables and total columns.

Figure 4 shows how the number of parameters grow with the total number of tables in the database. Relation-aware attention, used by the local MP model computes attention per node type, so it is normal that its total number of parameters has a strong correlation with the number of tables. In this case, a Pearson Correlation Coefficient of 0.98 was found. This is not the case, however, with Global Attention, where there seems to exist no correlation between the number of tables and parameter count. As graphs are more densely connected, the size of Graphormer's centrality encodings increases exponentially, making the total number of parameters explode.

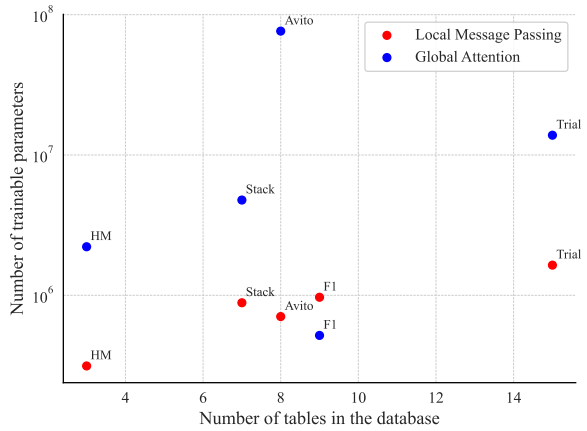

Fig. 4: Scatter plot showing how the number of trainable parameters grows with the number of tables in the training database for the Local Message Passing (red) and Global Attention (blue) models. The y-axis is shown on a log scale to accommodate the wide dynamic range of parameter counts.

This pattern would hold if we were to examine how model size varies with the total number of columns (i.e., the sum of columns across all tables). A similar trend would appear for FraudGT, but of course correlation does not imply causation: sorting the databases by total number of columns or by total number of tables produces the same ordering. Moreover, when taking into account that hidden dimensionality was kept constant during the experiments, it is easy to notice this relationship would not make sense.

APPENDIX C
EXPLORATORY DATA ANALYSIS

This section presents a comprehensive overview of the data-analysis workflow applied to each task and dataset used. Our goal is to describe the challenges that make learning from this data particularly difficult by highlighting the key insights we found.

For each task, we plot the class balance if its a binary classification task. In the case of regression, we plot the distribution of labels to gather further insights. Due to the relational nature of the data, investigating the correlation between features is not always possible due to the presence of text columns.

A. F1

driver-top3 show a similar class distribution between the negative and positive classes, allowing the models to learn meaningful representations and therefore achieving a considerable gain with respect to the baselines.

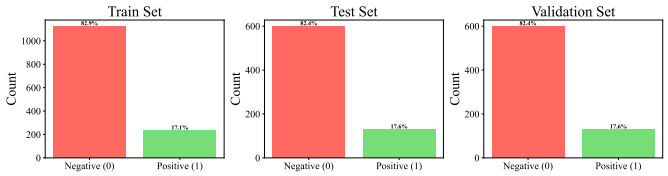

Fig. 5: driver-top3 class distribution

This is not so much the case for **driver-dnf**. A significant distribution shift can be seen in Figure 6 between train, test and validation sets.

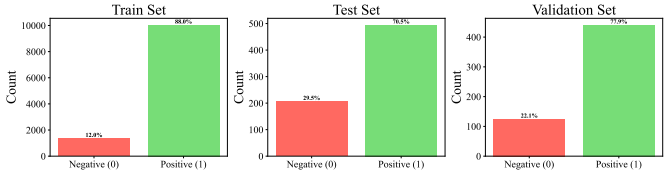

Fig. 6: driver-dnf class distribution

Finally, **driver-position** is where the biggest distribution shift can be seen (Figure 7). Training data from this task goes from 19-06-1950 all the way to 03-09-2004. During this period (specially at the beginning), there were races with up to than 39 drivers on the grid. Training and validation data, however, are much more recent, having a maximum of 24 drivers.

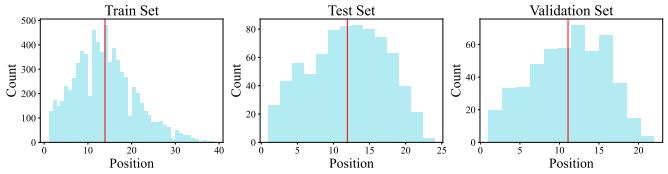

Fig. 7: driver-position label distribution

Simple calculations tell us that over 9.4% of all drivers of the training set finished after 24th place or worse. Models will have to learn that only past races had such a large number of drivers. Examining the distribution of predicted test labels of our best performing model against the ground truth (Figure 8) and comparing it agains train labels distribution reveals that our model just learned to predict the two most common positions from the training set (8 and 15).

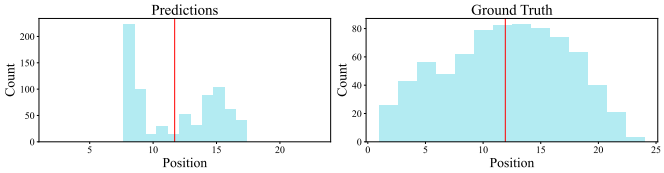

Fig. 8: driver-position predicted test labels distribution against ground truth

This essentially leaves us us with two options: (1) our models and Relational Deep Learning architectures tested so far are not powerful enough, (2) the features in the data are not rich enough to predict average driver finishing positions accurately.

From this information, we can derive that temporal data splits often lead to non-identically distributed subsets due to potential changes in the data distribution over time. This phenomenon is more noticeable as the timespan increases.

B. HM

The label distribution of **item-sales** is extremely skewed as showed by the violin plot in Figure 9.

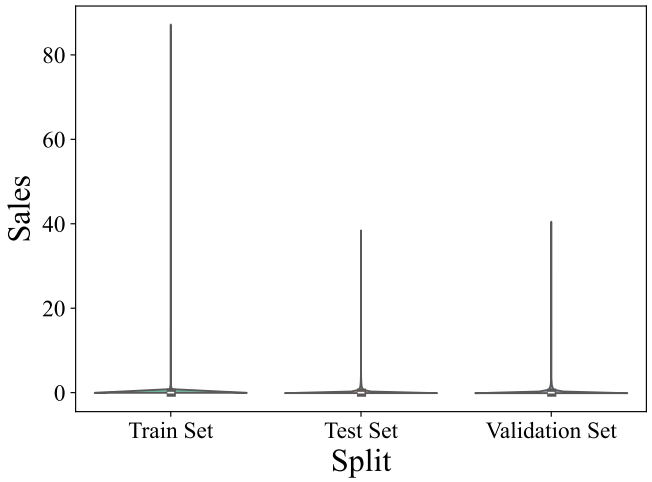

Fig. 9: item-sales violin plot displaying label distribution

The training data from this task consists of a whole year's worth of sales data for different articles, whilst the test and validation splits consist of predicting the total sales for an article for a week in September. This dataset is particularly hard, as coincidentally, the test week had record-low sales out of all the weeks in the train, test and validation sets (last bar in Figure 10). Examples like this demonstrate the importance of having rich features and powerful models.

C. Trial

site-success shows a very similar distribution of train, test and validation labels as shown in Figure 11. This leads to a decent improvement with the local attention model from the baseline.

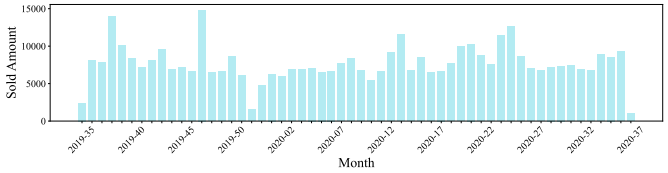

Fig. 10: Weekly sales from the *transactions* table in the HM dataset

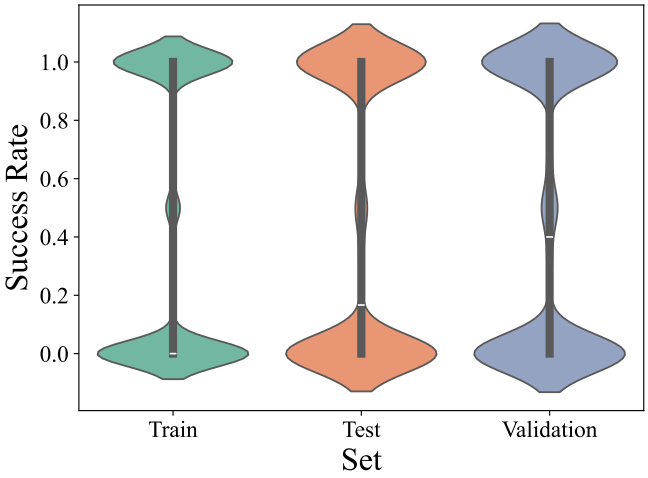

Fig. 11: site-success violin plot displaying label distribution

Nevertheless, on a task where labels range from 0 to 1, and where most are clustered in the extremes, a MAE of 0.400 (baseline) or 0.376 (ours) is not an indicator of good performance. This again arises the question of whether the data is not good enough or the models not powerful enough.

In contrast, **study-adverse** is a collection of rare events, with most training labels clustered around 0 (75th train quantile is only 15 events and the train maximum 28085). This distribution shows again extreme skewness.

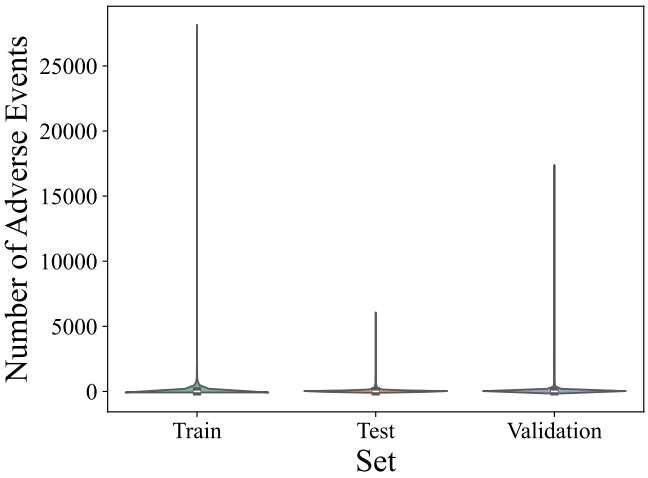

Fig. 12: study-adverse violin plot displaying label distribution

Once again, the distribution shift between train and test data is due to the temporal cut-off. Whilst training samples are approximately evenly distributed between 2001 and 2019, test samples are from 2021. A decrease in the number of rare events could be due to, for example, advances in general medicine which are out of scope for the input features of the model.

D. Stack

user-engagement's class imbalance is probably the most pronounced out of all the tasks. Moreover the large number of important text features in this database means that the results are also dependent on the embedding model used.

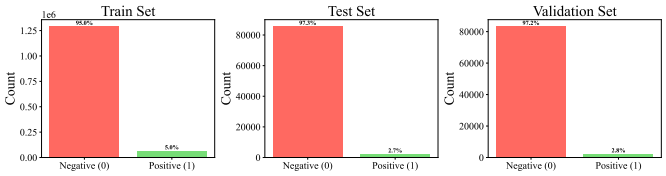

Fig. 13: user-engagement class distribution

Switching from GloVe to BERT would reshape the feature space: BERT's context-aware vectors would allow the model capture more details in the dataset's text features. However, if the downstream Transformer is shallow, an embedding that is too sophisticated may add unwanted cost and latency without contributing to significant accuracy gains.

E. Avito

Lastly, we found **user-visits** to be the hardest baseline to reproduce. Despite trying multiple combinations of hyperparameters, we could not reproduce the results achieved with the RDL implementation. This task also has a considerable imbalance between the positive and negative classes as shown in Figure 15.

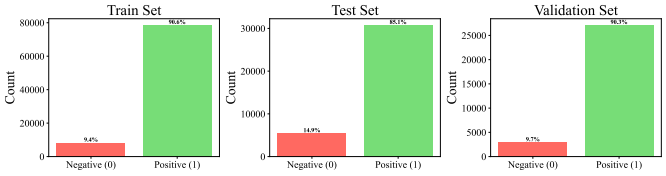

Fig. 14: user-visits class distribution

The dataset itself is characterized by an almost vanishing linear signal (nearly all features correlate with click-through at $|\rho| \lesssim 0.03$), coupled with strong redundancy among hierarchical and ID fields (e.g. SearchLocationID $\Longleftrightarrow$ AdLocationID at $\rho \approx 0.81$). This redundancy could cause representations to collapse and washing out rare but informative connections. Clicks are rare and redundant, thus it is important to have a powerful model which can learn this.

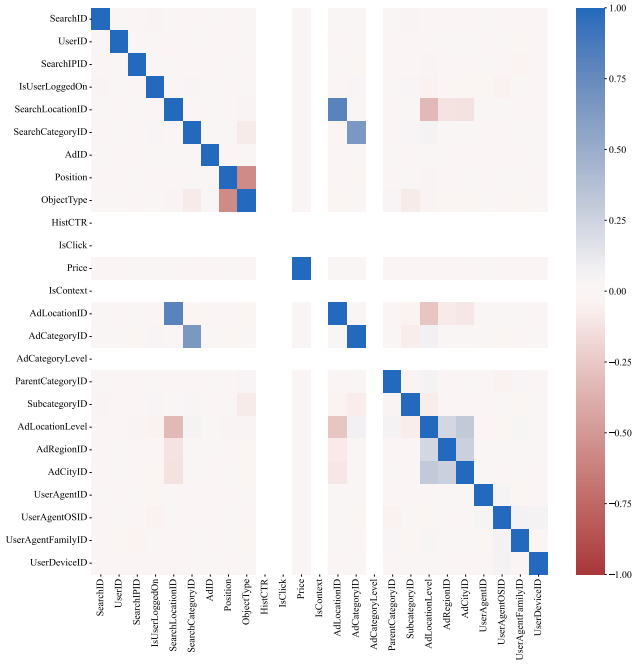

Fig. 15: user-visits numerical labels correlation