RGP: A Cross-Attention based Graph Transformer for Relational Deep Learning

Proceedings Track Submission

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

In domains such as healthcare, finance, and e-commerce, the temporal dynamics of relational data emerge from complex interactions—such as those between patients and providers or users and products across diverse categories. To be broadly useful, models operating on these data must integrate long-range spatial and temporal dependencies across diverse types of entities, while also supporting multiple predictive tasks. However, existing graph models for relational data primarily focus on spatial structure, treating temporal information merely as a constraint rather than a modeling signal, and are typically designed for single-task prediction.

To address these gaps, we introduce the Relational Graph Perceiver (RGP), a graph transformer architecture for relational deep learning. At its core, RGP employs a Perceiver-style latent bottleneck that integrates signals from different node and edge types into a common latent space, enabling the model to build global context across the entire relational system. It also incorporates a flexible cross-attention decoder that supports joint learning across tasks with disjoint label spaces within a single model. This architecture is complemented by a temporal subgraph sampler, which enhances global context by retrieving nodes beyond the immediate neighborhood. Experiments on RelBench, SALT, and CTU show that RGP delivers state-of-the-art performance, offering a general and scalable solution for relational deep learning with support for diverse predictive tasks.

1 Introduction

Relational data is central to many real-world systems in domains such as healthcare, finance, and e-commerce. These datasets capture interactions between entities such as patients and providers, customers and products, or suppliers and inventory, which unfold over time and span multiple data modalities [1]. The data is typically organized in multi-table relational databases and present a complex modelling challenge which involves both long-range structural dependencies through entity relationships and temporal dynamics through evolving interactions.

Relational Deep Learning (RDL) provides a principled framework for learning from such data by converting relational databases into heterogeneous temporal graphs [2]. In this formulation, nodes correspond to entities (e.g., users, items, visits), and edges represent typed relationships (e.g., purchases, interactions, transactions). While traditional RDL models based on Graph Neural Networks (GNNs) have shown success in capturing local structure via message passing, they suffer from several limitations. In particular, GNNs have limited expressiveness [3–5] and struggle to capture long-range dependencies due to oversquashing [6, 7].

Graph Transformers (GTs) offer a promising alternative by using attention mechanisms for global aggregation, allowing the model to reason across distant parts of the graph [8, 9]. However, most existing GTs are designed for static, homogeneous graphs and are ill-equipped to handle the structural and temporal heterogeneity found in relational databases [10]. Moreover, current relational graph

Submitted to the Fourth Learning on Graphs Conference (LoG 2025, Proceedings Track). Do not distribute.

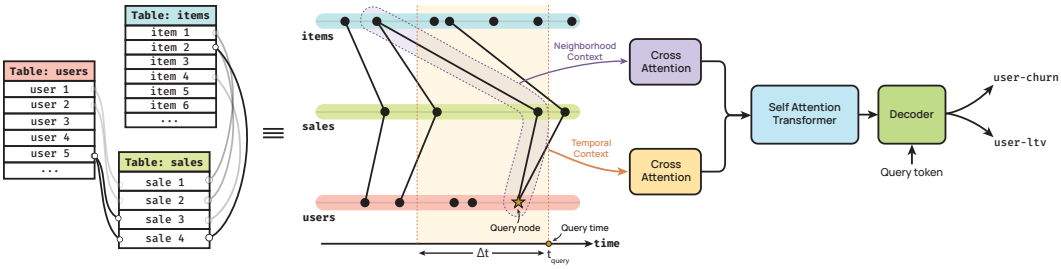

Figure 1: Overview of the RGP architecture. We convert relational databases into heterogeneous temporal graphs, where nodes represent entities (e.g., users, items, or sales) and edges capture interactions between them. Given a seed or query node (e.g., a user), the model applies two parallel cross-attention modules to encode both structural and temporal context into a set of latent tokens. These latents are then processed by a stack of self-attention transformer blocks to enable long-range reasoning. Finally, a lightweight and flexible decoder maps the latent representation to predictions across multiple tasks, such as user churn or lifetime value (LTV).

models, whether GNN [11] or GT [10] based, primarily focus on spatial structure and often treat time as just a constraint rather than as a modeling signal. This limitation is reflected in how context is sampled around a prediction node, where temporal information is typically used to restrict the neighborhood [11] rather than to actively guide the sampling process.

Our Approach. We introduce the **Relational Graph Perceiver (RGP)**, a general-purpose transformer architecture tailored for relational deep learning. RGP extends the Perceiver framework [12] to encode heterogeneous temporal graphs using a fixed-size latent bottleneck, enabling efficient and scalable reasoning across entity types and time. To move beyond local neighborhoods, RGP incorporates a novel *temporal subgraph sampler* that retrieves temporally relevant nodes, allowing the model to reason about nonlocal events that are structurally distant but contextually similar. Finally, RGP supports multi-task learning via a flexible decoder that conditions predictions on task-specific queries and compares them to text-encoded labels using a similarity-based objective.

We evaluate RGP on three diverse benchmarks—**RelBench**[11], **CTU**[13], and **SALT**[14]—spanning binary classification, multi-class classification, and ranking-based tasks. RGP consistently achieves strong performance across all settings, while supporting computationally efficient multi-task learning without the need for training seprate models or linear layers for each task. These results demonstrate the effectiveness of RGP as a scalable, general-purpose architecture for learning from relational data.

Our contributions are as follows:

- We present the **Relational Graph Perceiver (RGP)**, the first Perceiver-based graph transformer architecture tailored for heterogeneous temporal graphs. RGP enables efficient global reasoning across relational data.
- We introduce a novel **temporal subgraph sampler** that selects nodes based on contextual timestamp proximity, allowing the model to incorporate nonlocal temporal context beyond structural neighborhoods.
- We develop a **flexible multi-task decoder** that enables joint training across diverse tasks with disjoint label spaces. Our decoder uses task-conditioned queries and similarity-based supervision over text-encoded labels, eliminating the need for task-specific output heads.

2 Method

We now describe the architecture of the Relational Graph Perceiver (RGP), a general-purpose transformer-based model for learning on heterogeneous temporal relational graphs. As shown in Figure 1, RGP is built around three key components: (i) a Perceiver-style encoder that uses cross-attention to compress arbitrarily-sized relational graphs into a fixed-length latent representation (Section 2.2); (ii) a novel temporal sampler that retrieves temporally relevant nodes beyond the immediate neighborhood of the seed/query node (Section 2.3); and (iii) a lightweight, flexible multi-task head that enables training across multiple classification tasks with diverse label spaces, without the need for task-specific linear layers (Section 2.4).

Before encoding, we first transform relational databases into sequences of entity tokens using a standard heterogeneous tokenization scheme, described in Section 2.1. We then sample task-relevant subgraphs for each target entity and compress them into a latent representation using the Perceiver encoder. The overall pipeline is illustrated in Figure 1, and each component is described in the following sections.

2.1 Tokenizing Heterogeneous Temporal Graphs

To process relational data with transformer-based models, we first convert relational databases into graph-structured inputs (Figure 1), enabling end-to-end learning without the need for manual feature engineering. Following prior work in relational deep learning [10, 11], we represent relational databases as *relational entity graphs* (REGs), modeled as heterogeneous temporal graphs.

Relational Graph: A relational database can be formally described as a tuple $(\mathcal{T}, \mathcal{R})$, where $\mathcal{T} = T_1, \ldots, T_n$ is a collection of entity tables, and $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ is a set of inter-table relationships. Each relation $(T_{\text{fkey}}, T_{\text{pkey}}) \in \mathcal{R}$ denotes a foreign-key reference from one table to the primary key of another. Each table T_i contains a set of entities (rows), where each entity is typically defined by (1) a unique identifier, (2) foreign-key references, (3) entity-specific attributes (e.g., numeric, categorical), and (4) timestamp metadata.

We transform this database into a heterogeneous temporal graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi, \tau)$ where $\mathcal{V}$ is the set of nodes (entities), $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges representing primary-foreign key relationships, $\phi : \mathcal{V} \to \mathcal{T}_V$ maps each node to its source table (entity type), $\psi : \mathcal{E} \to \mathcal{T}_E$ assigns relation types to edges, and $\tau : \mathcal{E} \cup \mathcal{V} \to \mathbb{R}$ associates timestamps with both nodes and edges. This graph representation captures both the schema structure and temporal dynamics of the database.

Token Construction. Each node $v_i \in \mathcal{V}$ is mapped to a token embedding $\mathbf{x}_i \in \mathbb{R}^d$ by applying a multi-modal encoder to its raw attributes, followed by the addition of a positional encoding:

$$\mathbf{x}_i = \texttt{MultiModalEncoder}(\mathbf{u}_i) + \text{PE}(v_i),$$

where $\mathbf{u}_i$ denotes the raw attributes of the node (e.g., tabular, categorical, or multi-modal features), `MultiModalEncoder` is the modality-aware encoder taken from [15]. This encoder applies separate encoders for each modality (e.g., numerical, categorical, or text) and aggregates their outputs into a unified embedding using a ResNet (see Appendix A.1.1 for more details). $\text{PE}(v_i)$ captures structural and temporal context such as node centrality, hop distances, or timestamp embeddings. Each input graph is mapped to a full input sequence formed by these node-level tokens.

Positional Encodings. To represent the position of each node in a heterogeneous and temporal relational graph, we combine multiple structure and time-aware signals into a unified encoding. Specifically, for each node v_i, we compute:

- **Node type embedding** $\mathbf{e}_{\text{type}}(v_i)$: a learned embedding based on the node type $\phi(v_i)$.
- **Centrality embedding** $\mathbf{e}_{\text{cent}}(v_i)$: a linear projection of centrality scores (e.g., degree, PageRank).
- **Hop distance embedding** $\mathbf{e}_{\text{hop}}(v_i)$: a learned embedding of the hop distance from a designated entity node (e.g., the seed node or query node in the task).
- **Relative time encoding** $\mathbf{e}_{\text{time}}(v_i)$: a projection of $\tau(v_i) - \tau_{\text{seed}}$ to capture temporal alignment.

We concatenate these components and project them into the final positional encoding:

$$\text{PE}(v_i) = W_{\text{PE}} \cdot [\mathbf{e}_{\text{type}}(v_i) \| \mathbf{e}_{\text{cent}}(v_i) \| \mathbf{e}_{\text{hop}}(v_i) \| \mathbf{e}_{\text{time}}(v_i)] \,,$$

where $W_{\text{PE}} \in \mathbb{R}^{d' \times d}$ is a learned projection matrix and $\|$ denotes concatenation.

Remarks. This multi-element positional encoding allows the model to incorporate fine-grained structural, temporal, and schema-level context across highly diverse relational graphs. While the individual components of this encoding are adapted from established techniques [9, 10], their integration provides a unified representation that is suitable for heterogeneous graph modeling.

Once tokenized, we construct an input subgraph around each target entity using a combination of structural sampling [11] and temporal context sampling (Section 2.3). The resulting node sequence is then passed to our Perceiver-based encoder, which compresses it into a fixed-size latent representation via cross-attention (Section 2.2).

2.2 Compression via Cross Attention

Graph transformers often suffer from over-globalization when applying full self-attention across all node pairs, leading to performance degradation [16]. In this regime, information from distant and potentially irrelevant nodes is overly mixed into each node's representation. To address this, prior work introduces handcrafted attention masks or distance-based biases that restrict the attention scope and enforce locality [17]. More recently, virtual nodes have emerged as a strong inductive bias for enabling global reasoning in graph models [18, 19], while also reducing computational cost. This is especially important since standard self-attention has a quadratic complexity of $\mathcal{O}(n^2)$ with respect to the number of input nodes, making it infeasible for large graphs.

To address this, we adopt a Perceiver-inspired encoder [20, 21], which avoids imposing fixed attention constraints or relying on heuristic-based virtual nodes. The core idea of the Perceiver architecture is to introduce a fixed-size set of learnable latent tokens—or "virtual nodes"—that serve as an information bottleneck.

Given the tokenized graph structure described in Section 2.1 and the sampled subgraph around each target entity (using both structural and temporal samplers; see Section 2.3), we obtain a sequence of input node embeddings $\mathbf{X}_g = [\mathbf{x}_1, \ldots, \mathbf{x}_{N_g}]$ for each subgraph. These embeddings are then passed to the Perceiver encoder, which compresses them into a fixed-size latent representation via cross-attention.

We maintain a shared sequence of K learned latent tokens $\mathbf{Z}_0 = [\mathbf{z}_{0,1}, \ldots, \mathbf{z}_{0,K}]$, where each $\mathbf{z}_{0,i} \in \mathbb{R}^D$ and K is considerably smaller than the number of input nodes. This reduces computational and memory costs by decoupling the number of attention operations from the input size. The node embeddings of each input subsampled graph sequence are compressed into these latent tokens via a cross-attention operation:

$$\mathbf{Z}_g^{(1)} \leftarrow \text{Cross-Attn}(\mathbf{Q}_g, \mathbf{K}_g, \mathbf{V}_g) = \mathbf{Z}^{(0)} + \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}_g^\top}{\sqrt{d_k}}\right)\mathbf{V}_g, \tag{1}$$

where $\mathbf{Q} = \mathbf{W}_q\mathbf{Z}_0$ are linear projections of the latent tokens, and $\mathbf{K}_g = \mathbf{W}_k\mathbf{X}_g$, $\mathbf{V}_g = \mathbf{W}_v\mathbf{X}_g$ are projections of the input node embeddings. Through this mechanism, the latent tokens integrate information from the entire graph without requiring dense pairwise interactions among all nodes.

We apply this cross-attention operation in two parallel branches: one operating on nodes sampled via structural neighborhood sampling, and the other on nodes retrieved via the temporal context sampler. Each branch uses an independent cross-attention layer with its own key and value projections. The resulting latent representations from the two branches are summed elementwise to produce a unified latent embedding.

Following this compression step, we process the latents with a stack of L self-attention blocks operating purely in the latent space, yielding the final set of compressed latent tokens $\mathbf{Z}_g^{\text{out}}$. We use standard Transformer blocks with pre-layer normalization and feed-forward layers [22].

The total complexity of this encoder is:

$$\mathcal{O}(KN_g + LK^2) \ll \mathcal{O}(N_g^2), K \ll N_g, \tag{2}$$

resulting in substantial savings in both compute and memory while still allowing the model to integrate information from all nodes in the input graph.

2.3 Temporal Sampler

As described in Section 2.2, the Perceiver encoder operates on a sampled input graph centered around a target entity. In this section, we describe in more detail how this input subgraph is constructed.

In this work, we incorporate two complementary sampling strategies for constructing the input sequence from heterogeneous temporal graphs. Standard neighborhood sampling methods apply time-restricted sampling around a target node to prevent temporal leakage [10, 11] (Figure 1). These methods focus on preserving local graph structure while enforcing temporal constraints, treating time primarily as a boundary condition on graph-based neighborhood sampling.

In contrast, we introduce a second sampling mechanism—the *Time-Context Sampler*, which explicitly leverages temporal proximity as a signal, independent of graph connectivity. This sampler selects

edges (and their associated nodes) based on their closeness in time to a reference timestamp, regardless of whether they are direct neighbors of the target node. This enables the model to incorporate temporally co-occurring events that may reflect broader contextual information—such as concurrent user activity, market trends, or environmental conditions—that can be critical for accurate prediction.

Formally, given a graph $G = (V, E)$, node timestamps $T_v : V \to \mathbb{R}$, and a seed time t_{seed}, we assign timestamps to edges based on the node timestamps and select a subgraph using either a fixed time window Δt or the k closest edges in time. The full algorithm is provided in Appendix A.1.2.

The nodes selected by the temporal sampler are then processed by the model in parallel with the structurally sampled neighborhood nodes, as described in the previous section.

2.4 Multi-Task Decoder

After the Perceiver encoder compresses the input subgraph into a fixed-length latent representation, this representation must be mapped to task-specific predictions. To support diverse prediction objectives across multiple tasks, we adopt a flexible multi-task decoder that combines cross-attention with similarity-based label supervision. Instead of maintaining a separate output head for each task, we use a shared decoding mechanism that conditions predictions on both the task description and the target node representation. This design enables efficient parameter sharing across tasks while allowing task-specific behavior to emerge through the attention mechanism and label embeddings. Given a task-specific query embedding $\mathbf{q}_{\text{task}} \in \mathbb{R}^d$ and the tokenized representation of the target node $\mathbf{x}_i \in \mathbb{R}^d$ (from Section 2.1), we compute a task-aware query via element-wise summation:

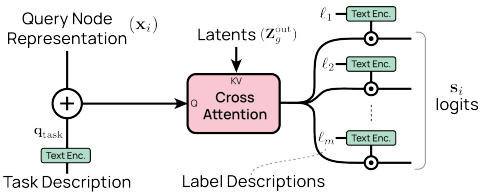

Figure 2: Overview of Flexible multi-task decoder: The decoder receives a query node representation (combined with a task description embedding) and attends to the latents from the perceiver encoder via cross attention.

$$\mathbf{q}_i = \mathbf{x}_i + \mathbf{q}_{\text{task}}.$$

This query is used to attend to the encoder's latent $\mathbf{Z}^{\text{out}} \in \mathbb{R}^{K \times d}$ using a cross-attention mechanism:

$$\mathbf{z}_i = \text{CrossAttn}(\mathbf{q}_i, \mathbf{Z}_g^{\text{out}}),$$

where $\mathbf{z}_i \in \mathbb{R}^d$ is the task-conditioned representation of the node.

To enable generalization across tasks without task-specific output layers, we represent each candidate label as a text string and encode them using a frozen or pretrained text encoder:

$$\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}, \quad \mathbf{E}_{\text{label}} = \text{TextEncoder}(\mathcal{L}) \in \mathbb{R}^{m \times d},$$

where ℓ_j is the j-th label and m is the number of possible labels for the current task.

We compute the logits by taking the dot product between the output node embedding $\mathbf{z}_i$ and each label embedding:

$$\mathbf{s}_i = \mathbf{E}_{\text{label}} \mathbf{z}_i \in \mathbb{R}^m.$$

These logits are passed through a softmax function to compute class probabilities, and the loss is computed using standard objectives such as cross-entropy for classification. This unified decoding framework eliminates the need for task-specific classifiers or training separate models for each task, enabling scalable and efficient multi-task learning.

3 Results

We evaluate the Relational Graph Perceiver (RGP) on a diverse suite of heterogeneous temporal graph datasets spanning multiple domains and benchmarks. Specifically, we consider three sources:

Table 1: Results on Relbench: We report the Area Under the ROC Curve (AUC) as the evaluation metric. Best values are shown in **bold**. Relative gains indicate the percentage improvement of RGP over RelGT.

Dataset	Task	RDL	HGT	HGT+PE	RelGT	**RGP (ours)**	% Rel Gain vs. RelGT
rel-f1	driver-dnf	0.7262	0.7142	0.7109	0.7587	**0.7844**	+3.39
	driver-top3	0.7554	0.6389	0.8340	0.8352	**0.8789**	+5.22
rel-avito	user-clicks	0.6590	0.6584	0.6387	0.6830	**0.6943**	+1.66
	user-visits	0.6620	0.6426	0.6507	**0.6678**	0.6662	-0.24
rel-event	user-repeat	0.7689	0.6717	0.6590	0.7609	**0.7894**	+3.75
	user-ignore	0.8162	0.8348	0.8161	0.8157	**0.8439**	+3.46
rel-trial	study-outcome	0.6860	0.5679	0.5691	0.6861	**0.7027**	+2.42
rel-amazon	user-churn	0.7042	0.6608	0.6589	0.7039	**0.7089**	+0.71
	item-churn	**0.8281**	0.7824	0.7840	0.8255	0.8262	+0.08
rel-stack	user-engagement	0.9021	0.8898	0.8818	**0.9053**	0.9045	-0.09
	user-badge	**0.8966**	0.8652	0.8636	0.8624	0.8868	+2.83
rel-hm	user-churn	0.6988	0.6773	0.6491	0.6927	**0.7025**	+1.41
						Average Gain vs. RelGT (%)	**2.20**

RelBench [11], CTU [23], and SALT [14]. Our experiments focus on the node (or entity) classification task, where the goal is to predict categorical attributes associated with entities in relational graphs. As these benchmarks originate from distinct application areas and have only recently been introduced, they differ significantly in terms of available baselines and evaluation metrics. We provide a detailed discussion of the dataset-specific metrics and baseline comparisons in Section 3.2.

3.1 Experimental Setup

We implement RGP within the RDL pipeline [11] by replacing the original GNN component with our architecture, while preserving the underlying task logic, database loaders, and training infrastructure. RGP is trained using the AdamW optimizer [24] with a fixed learning rate of 10^{-3}. Similar to previous work we only tune a few key architectural hyperparameters: total number of layers in the model $L \in \{2, 4, 6\}$ and the number of latent tokens $n \in \{8, 16, 32\}$ in cross attention block. All other settings such as batch size, dropout, and learning rate remain fixed across datasets. For a complete list of hyperparameters, see Section A.2.1 in the appendix.

3.2 Results on Benchmarks

We report benchmark results for RGP across three representative datasets—RelBench, CTU, and SALT, each reflecting a different application domain and evaluation metric. Due to the diversity of these benchmarks, we group results by benchmark and compare RGP against the publically available baselines for each setting. Across all benchmarks, we include comparisons to RDL [25], a widely adopted pipeline that combines RelGNN [26] with GraphSAGE aggregation, serving as a strong reference point for relational deep learning tasks.

RelBench. [11] is a recently introduced benchmark for relational deep learning that includes seven datasets derived from structured domains such as e-commerce, social networks, and sports. All classification tasks in this benchmark are binary classification problems, with performance evaluated using the Area Under the ROC Curve (AUC-ROC). For RelBench, we report results from the Relational Graph Transformer (RelGT) [27], the current state-of-the-art method on this benchmark. The paper also compared against HGT [28] and a variation of HGT with laplacian positional encodings [8]. As shown in Table 1, RGP achieves state-of-the-art performance in RelBench, outperforming RelGT on 10 out of 12 tasks. Notably, RGP yields an average relative improvement of **2.2%** over RelGT across all tasks, highlighting the benefit of our transformer-based approach. Gains are particularly pronounced on smaller datasets, where self-attention-based models like RelGT are more prone to overfitting. For example, on the driver-top3 task from the F1 dataset, RGP outperforms

Table 3: Results on CTU benchmarks: We report F1 score as the evaluation metric. Best values are in **bold**. Relative gains are percentage improvement over DBFormer.

Dataset	Task	LightGBM	XGBoost	TabResNet	Linear	SAGE (RDL)	DBFormer	**RGP (ours)**	**Rel.** (%) Gain vs. DBF
accidents	temp.	0.170	0.336	0.187	0.583	0.566	0.727	**0.743**	+2.20
dallas	temp.	**0.584**	0.512	0.247	0.393	0.424	0.513	0.555	+8.19
legalacts	temp.	**0.851**	0.220	0.220	0.721	0.698	0.703	0.736	+4.69

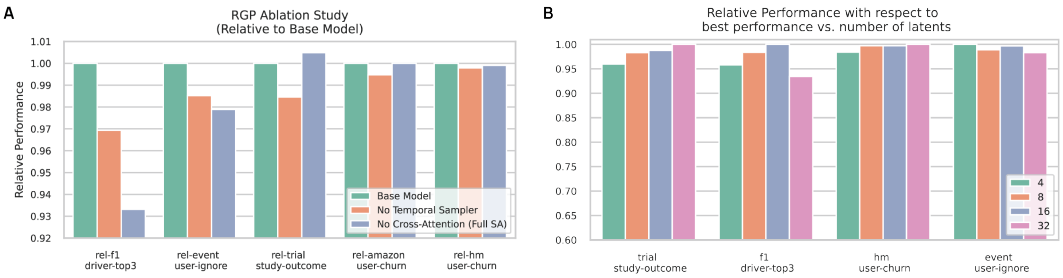

Figure 3: **(A)** Results from ablation study of RGP. We evaluate the impact of removing key components from the full RGP model. Performance is reported relative to the base model. A decrease in performance indicates that the removed component is important to overall model effectiveness. **(B)** Effect of number of latent tokens on model performance across four representative tasks. For each task, we normalize results with respect to the best-performing configuration to compute relative performance.

RelGT by **3.39%**, and on the driver-dnf task, by **5.22%**. Similarly, on the user-repeat task from rel-event, we observe a gain of **3.75%**.

CTU. [23] is a curated repository of heterogeneous graph datasets from domains such as insurance, law and retail. We evaluate on CTU because it includes multi-class classification tasks, enabling us to test RGP beyond binary settings. We compare against baselines reported in the ReDeLEx benchmark [29], which include traditional tabular models (e.g., LightGBM [30], XGBoost [31]), temporal MLPs (e.g., TabResNet [32]), and the GNN-based RDL [11] framework. Following standard practice in from ReDeLEx, we report F1 scores for all tasks. As shown in Table 3, RGP consistently outperforms the DBFormer baseline across all evaluated CTU datasets. Notably, on the *dallas* dataset, RGP achieves a relative gain of **8.19%**.

SALT. [23] is a real-world dataset derived from an Enterprise Resource Planning (ERP) system. It consists of ranking-based entity classification tasks that reflect practical industrial decision-making scenarios. Since these tasks involve ranking objectives, we report Mean Reciprocal Rank (MRR) as the evaluation metric. As graph-based models have not been previously applied to SALT, we compare RGP against standard baselines, including RDL and the Heterogeneous Graph Transformer (HGT) [28]. As shown in Table 2 RGP consistenty outperforms both the baselines.

Summary: Across all three benchmarks, RGP demonstrates robust performance gains over strong baselines in diverse settings—binary classification (RelBench), multi-class classification (CTU), and ranking-based tasks (SALT). These results collectively highlight RGP's versatility and effectiveness as a general-purpose relational graph model.

Table 2: Results on SALT: We report MRR score as the evaluation metric. Best values are in **bold**. Relative gains are percentage improvement over HGT.

Task	RDL	HGT	**RGP (ours)**	% Rel Gain v.s HGT
item-incoterms	0.64	0.75	**0.81**	+8.00
sales-group	0.20	0.31	**0.34**	+9.68
sales-payterms	0.39	**0.60**	0.58	-3.33
sales-shipcond	0.59	0.76	**0.81**	+6.58

3.3 Ablation Studies

To better understand the contributions of key architectural components in RGP, we perform a series of ablation studies. Specifically, we analyze the impact of (1) removing the temporal context sampler

and (2) replacing the Perceiver-style cross-attention bottleneck with full self-attention. For both settings, we report the relative performance compared to the best full model in Figure 3A.

Temporal sampler: The temporal sampler is designed to retrieve nodes based on temporal proximity, complementing the structural neighborhood sampler. As shown in Figure 3A, removing the temporal sampler leads to consistent performance drops, most notably on the *rel-f1* dataset, where we observe a $\sim$3% decline. This dataset contains many cold-start cases, such as newly introduced driver nodes with limited or no structural history. In such cases, structural context alone is insufficient for reliable predictions.

Perceiver encoder: We next evaluate the role of the latent Perceiver encoder by replacing it with a full self-attention (SA) mechanism applied over the input tokens. As shown in Figure 3A, this substitution yields marginal improvements on a few tasks but at the cost of significantly more compute requirements. Additionally, for smaller datasets such as *rel-f1*, full self-attention leads to overfitting and degraded performance, with a $\sim$6.6% drop on the *driver-top3* task. In contrast, the cross-attention bottleneck in RGP offers a more compute-efficient and regularized alternative, preserving competitive performance while maintaining scalability across tasks. On average, cross-attention is 2–6% more compute-efficient in comparison to self attention(see appendix A.2.2 for average run time)

Effect of Latent Token Count: Finally we examine how the number of latent tokens affects performance. We sweep across $n \in \{4, 8, 16, 32\}$ latent tokens on four representative tasks: *study-outcome*, *driver-top3*, *user-churn*, and *user-ignore*. For each task, we normalize results with respect to the best-performing configuration to compute relative performance. As shown in Figure 3B, RGP achieves strong performance even with a relatively small number of latent tokens. However, the optimal number of latents varies across tasks and datasets, and increasing the number does not always lead to better results. This is why we chose to keep it as a tunable hyperparameter. For example, the performance of *driver-top3* peaks at 16 latents and declines at 32, suggesting potential overfitting.

3.4 Multi Task Results

A key motivation behind the flexible decoder introduced in Section 2.4 is to enable scalable multi-task learning across diverse label spaces. Most existing approaches for relational graph learning adopt a task-specific training paradigm, where separate models are trained for each task, even when these tasks share the same underlying graph structure.

In contrast, the decoder in RGP supports task-conditioned decoding by combining cross-attention with a similarity-based objective over text-encoded label embeddings. This unified framework allows a single model to learn multiple tasks simultaneously.

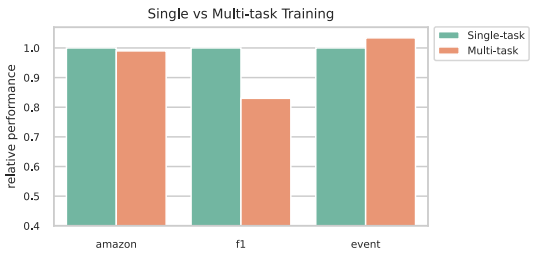

Figure 4: Relative performance of multi-task vs. single-task training across datasets: The Y-axis shows the average multi-task performance normalized with respect to the average single-task performance across all tasks within each dataset

To evaluate the effectiveness of this design, we compare single-task training (one model per task) with multi-task training (a single model trained jointly on all tasks from the same dataset). In our multi-task setting, task supervision is provided via text-based task queries and label embeddings, as described in Section 2.4.

As shown in Figure 4, RGP achieves comparable or improved results under multi-task training, while significantly reducing training cost. For the event dataset, multi-task training even led to slight improvements over single-task models. However, we observed a notable drop in performance on the f1 dataset under the multi-task setting. This can be attributed to the severe imbalance in the number of training samples across tasks: the driver-top3 task has only 1.5k samples, whereas driver-dnf has over 10k. Such imbalance likely causes the shared model to underfit the smaller task, leading to degraded performance on driver-top3.

Overall, these results highlight the generalization ability of our decoder and its parameter-efficient multi-task learning without any retraining or architectural modification, making it a strong candidate for large-scale foundation model training across relational datasets.

4 Related Work

Representing relational datasets as heterogeneous temporal graphs has enabled the use of graph learning methods for tasks like node classification and link prediction. The RelBench benchmark[11] formalizes this paradigm and provides a strong baseline using Heterogeneous GraphSAGE [33] with temporal neighbor sampling, surpassing classical tabular methods like LightGBM [30]. Several architectures have been proposed to better exploit relational structure: RelGNN [26] introduces composite message passing to preserve information across bridge and hub nodes, while ContextGNN [34] combines pair-wise and two-tower encoders for recommendation scenarios.

Graph Transformers (GTs) adapt the self-attention paradigm [22] to graph-structured data, capturing long-range dependencies without iterative neighborhood aggregation [35]. Early GT variants focused on local attention with positional encodings like Laplacian eigenvectors [36], while later designs incorporated global attention mechanisms [9, 17, 37] and scaling strategies such as hierarchical pooling or sparse attention. Heterogeneous GTs such as HGT [28] and Hinormer [38] model multi-type graphs, but face quadratic cost and per-task training inefficiency. Most relevant to our work is the Relational Graph Transformer (RelGT) [10], which introduced a hybrid local/global attention, establishing strong performance on RelBench. Our approach differs by adopting a Perceiver-style latent bottleneck and temporal sampling, enabling greater scalability and multi-task capability.

While GTs highlight the importance of global context, their computational footprint motivates exploring efficiency-focused alternatives. The Perceiver architecture [12] introduces a fixed-size latent array that attends to high-dimensional inputs via cross-attention, followed by latent self-attention, decoupling input size from computational complexity. Perceiver IO [20] extends this framework to handle diverse output domains from a shared latent representation. In graph learning, early Perceiver-inspired adaptations have been proposed for homogeneous graphs or static settings [39], but these typically omit temporal sampling and are not optimized for multi-task inference. Our encoder adapts this latent bottleneck principle specifically to heterogeneous temporal graphs.

Existing RDL methods excel at modeling relational structure but face oversquashing and limited temporal reach. Graph Transformers capture long-range context but remain computationally heavy and often schema-restrictive. Perceiver-based models address scalability but have not been tailored to heterogeneous temporal graphs or multi-task learning in relational settings. This motivates architectures like ours that combine the latent efficiency of Perceivers with relational and temporal inductive biases, while enabling shared encoders across multiple tasks.

5 Conclusion

We introduced RGP, a transformer-based architecture for heterogeneous temporal graphs that integrates a Perceiver-style encoder for global reasoning via a cross-attention bottleneck, a temporal subgraph sampler that incorporates temporally relevant but structurally distant context, and a flexible multi-task decoder that enables prediction across diverse tasks and label spaces within a single model. Across multiple benchmarks, RGP consistently outperforms strong baselines, achieving state-of-the-art results on both binary and multi-class tasks while reducing computational overhead. Our ablation studies further validate the importance of each component, particularly the role of importance of temporally aligned but structurally distant context in improving predictive performance. Beyond accuracy gains, RGP supports flexible multi-task learning without requiring task-specific output heads, positioning it as a strong candidate for large-scale foundation models in relational domains.

References

[1] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 1

[2] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Position: Relational deep learning-graph representation learning on relational databases. In *Forty-first International Conference on Machine Learning*, 2024. 1

[3] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 1

[4] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11313–11320, 2019.

[5] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019. 1

[6] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pages 2528–2547. PMLR, 2023. 1

[7] Shaima Qureshi et al. Limits of depth: Over-smoothing and over-squashing in gnns. *Big Data Mining and Analytics*, 7(1):205–216, 2023. 1

[8] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. 1, 6

[9] Ruiheng Ying, Zhuang Liu, Jiaxuan You, Yingbo Zhou, Chongxuan Li, Meng Jiang, and Jure Leskovec. Do transformers really perform badly for graph representation? In *ICLR 2021*, 2021. 1, 3, 9

[10] Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. *arXiv preprint arXiv:2505.10960*, 2025. 1, 2, 3, 4, 9

[11] Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. Relbench: A benchmark for deep learning on relational databases. In *NeurIPS 2024 Datasets and Benchmarks Track*, 2024. Poster; also available as arXiv:2407.20060. 2, 3, 4, 6, 7, 9, 12

[12] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021. 2, 9

[13] Jan Motl and Oliver Schulte. The ctu prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015. 2

[14] Tassilo Klein, Clemens Biehl, Margarida Costa, Andre Sres, Jonas Kolk, and Johannes Hoffart. Salt: Sales autocompletion linked business tables dataset. *arXiv preprint arXiv:2501.03413*, 2025. 2, 6

[15] Weihua Hu, Yiwen Yuan, Zecheng Zhang, Akihiro Nitta, Kaidi Cao, Vid Kocijan, Jinu Sunil, Jure Leskovec, and Matthias Fey. Pytorch frame: A modular framework for multi-modal tabular learning. *arXiv preprint arXiv:2404.00776*, 2024. 3, 12

[16] Yujie Xing, Xiao Wang, Yibo Li, Hai Huang, and Chuan Shi. Less is more: on the over-globalizing problem in graph transformers. *arXiv preprint arXiv:2405.01102*, 2024. 4

[17] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021. 4, 9

[18] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pages 31613–31632. PMLR, 2023. 4

[19] Chuang Liu, Yibing Zhan, Xueqi Ma, Liang Ding, Dapeng Tao, Jia Wu, and Wenbin Hu. Gapformer: Graph transformer with graph pooling for node classification. In *IJCAI*, pages 2196–2205, 2023. 4

[20] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs. In *ICLR 2022 (Spotlight)*, 2022. Also available as arXiv:2107.14795. 4, 9

[21] Divyansha Lachi, Mehdi Azabou, Vinam Arora, and Eva Dyer. Graphfm: A scalable framework for multi-graph pretraining. *arXiv preprint arXiv:2407.11907*, 2024. 4

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4, 9

[23] Jan Motl and Oliver Schulte. The ctu prague relational learning repository, 2024. URL https://arxiv.org/abs/1511.03086. 6, 7

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[25] Matthias Fey et al. Graph representation learning on relational databases. *arXiv preprint*, 2024. Introduces Relational Deep Learning (RDL) blueprint referenced by RelBench. 6

[26] Chen *et al.*. Relgnn: Composite message passing for relational deep learning. In *ICML 2025 Poster Track*, 2025. 6, 9

[27] Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer, 2025. URL https://arxiv.org/abs/2505.10960. 6

[28] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020. 6, 7, 9

[29] Jakub Peleška and Gustav Šír. Redelex: A framework for relational deep learning exploration. *arXiv preprint arXiv:2506.22199*, 2025. 7

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017. 7, 9

[31] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 7

[32] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34: 18932–18943, 2021. 7

[33] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, pages 1024–1034, 2017. 9

[34] Yiwen Yuan, Zecheng Zhang, Xinwei He, Akihiro Nitta, Weihua Hu, Dong Wang, Manan Shah, Shenyang Huang, Blaž Stojanovič, Alan Krumholz, et al. Contextgnn: Beyond two-tower recommendation systems. *arXiv preprint arXiv:2411.19513*, 2024. 9

[35] Vijay Prakash Dwivedi and Michael M. Bronstein. A generalization of transformer networks to graphs. *AAAI DLG Workshop*, 2021. 9

[36] Vijay Prakash Dwivedi et al. Graph neural networks with learnable structural and positional representations. *ICLR 2022*, 2022. 9

[37] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021. 9

[38] Mao *et al.*. Hinormer: Representation learning on heterogeneous information networks with graph transformer. In *WWW 2023*, 2023. 9

[39] Bin Jiang and Ding Ma. Defining least community as a homogeneous group in complex networks. *Physica A: Statistical Mechanics and its Applications*, 428:154–160, 2015. 9

A Appendix

A.1 Model Details

A.1.1 Multi Model Encoder

In heterogeneous temporal relational graphs derived from relational databases, each node may contain a rich set of multi-modal attributes, such as numerical values, categorical fields, text descriptions, timestamps, and image features. To obtain expressive node-level representations suitable for downstream tasks, we use a modality-aware feature encoder taken from prior work in relational deep learning [11, 15].

Given a node $v \in \mathcal{V}$, we denote its raw attributes as $\mathbf{x}_v = \{\mathbf{x}_v^{(m)}\}_{m \in \mathcal{M}_v}$, where $\mathcal{M}_v \subseteq \mathcal{M}$ is the subset of modalities present for node v. Each feature within each modality is independently encoded using a modality-specific function ϕ_m, yielding a set of intermediate embeddings:

$$\mathbf{h}_v^{(m)} = \phi_m\left(\mathbf{x}_v^{(m)}\right), \quad \phi_m : \mathcal{X}^{(m)} \to \mathbb{R}^{d_m}, \tag{3}$$

where d_m is the dimensionality assigned to modality m. Supported modalities include:

- **Numerical features:** encoded via linear layers or small MLPs.
- **Categorical features:** embedded using learned lookup tables.
- **Text features:** embedded via pretrained or fine-tuned language models (e.g., BERT).
- **Timestamps:** embedded as scalar values or periodic functions (e.g., sinusoidal encodings).

The resulting embeddings are concatenated:

$$\mathbf{h}_v^{\text{concat}} = \bigoplus_{m \in \mathcal{M}_v} \mathbf{h}_v^{(m)} \in \mathbb{R}^{\sum_{m \in \mathcal{M}_v} d_m}, \tag{4}$$

and passed through a table-specific (or node-type-specific) projection function—a ResNet in our case—to obtain the final node representation:

$$\mathbf{h}_v^{(0)} = f_{\tau(v)}\left(\mathbf{h}_v^{\text{concat}}\right), \quad f_{\tau(v)} : \mathbb{R}^{\sum d_m} \to \mathbb{R}^d, \tag{5}$$

where $\tau(v)$ denotes the node type (i.e., source table) and d is the unified hidden dimension used across the model.

A.1.2 Time-Context Sampling Algorithm

Edge Timestamp Assignment. Each edge is assigned a timestamp equal to the maximum timestamp of its two endpoint nodes:

$$T_e(u, v) = \max\left(T_v(u), T_v(v)\right).$$

Time-Context-Aware Sampling. Given a reference timestamp t_{seed}, we sample a subgraph by selecting edges based on one of two strategies:

- Time window: include all edges where $|T_e(u, v) - t_{\text{seed}}| \leq \Delta t$.
- Top-k: select the k edges closest in time to t_{seed}.

The resulting subgraph contains all nodes incident to the selected edges.

Algorithm 1 Time-Context-Aware Edge Sampling

Require: Graph $G = (V, E)$, timestamp function $T : V \to \mathbb{R}$, reference time t_{seed}, sampling rule: time window Δt or edge count k
Ensure: Subgraph $G_{\text{sub}} = (V_{\text{sub}}, E_{\text{sub}})$
Initialize $E_{\text{scored}} \leftarrow \emptyset$
for each edge $(u, v) \in E$ **do**
  $t_e \leftarrow \max(T(u), T(v))$
  score $\leftarrow |t_e - t_{\text{seed}}|$
  Add $(u, v, t_e, \text{score})$ to E_{scored}
end for
if time window Δt is specified **then**
  $E_{\text{selected}} \leftarrow \{(u, v) \mid (u, v, t_e, s) \in E_{\text{scored}},\ s \leq \Delta t\}$
else if edge count k is specified **then**
  Sort E_{scored} in ascending order by score
  $E_{\text{selected}} \leftarrow$ first k edges from sorted list
else
  $E_{\text{selected}} \leftarrow E$
end if
$V_{\text{sub}} \leftarrow$ nodes appearing in E_{selected}
return $G_{\text{sub}} = (V_{\text{sub}}, E_{\text{selected}})$

A.2 Experiment details

A.2.1 Hyperparameter

We use a controlled hyperparameter setup to ensure consistent evaluation across diverse datasets without exhaustive tuning. Following prior work, we fix most settings and tune only two architectural hyperparameters: the number of Perceiver layers $L \in \{2, 4, 6\}$ and the number of latent tokens $n \in \{8, 16, 32\}$ in the cross-attention bottleneck. All other hyperparameters are kept fixed across datasets to reflect realistic training scenarios where compute budgets limit per-task tuning.

Table 4 summarizes the fixed hyperparameter settings used in our experiments. All models are trained using the AdamW optimizer with a learning rate of 10^{-3} and a weight decay of 10^{-5}. We use a cosine learning rate scheduler with 10 warmup steps, and all experiments are run for 200 epochs.

Table 4: Summary of hyperparameter settings used in RGP experiments.

Component	Value / Setting
Optimizer and Training Setup	
Optimizer	AdamW
Learning rate	10^{-3}
Weight decay	10^{-5}
Scheduler	Cosine with 10 warmup steps
Epochs	200
Batch size	512
Model Architecture	
Latent token count n	{8, 16, 32} (tuned)
Transformer layers L	{2, 4, 6} (tuned)
Dropout	0.2
hidden dim	128
Temporal Sampler	
Edges per type	10
Temporal decay	0.1

A.2.2 Runtime Comparison: Cross-Attention vs Self-Attention

To quantify the efficiency benefits of using a cross-attention (CA) bottleneck over full self-attention (SA), we measure the total training time across three representative tasks. As shown in Table 5, the Perceiver-based encoder with CA consistently achieves lower runtime compared to its SA counterpart. While SA sometimes offers marginal accuracy improvements on larger datasets, the increase in computational cost is substantial.

Table 5: Average training time for Cross-Attention (CA) vs. Self-Attention (SA) models. Experiments were run on a single B200 GPU.

Dataset	**Cross-Attention (CA)**	**Self-Attention (SA)**
rel-f1	2.7 min	7.5 min
rel-amazon	3.5 hrs	18.5 hrs
rel-event	4.5 min	22 min