Integrating Temporal and Structural Context in Graph Transformers for Relational Deep Learning

Divyansha Lachi*
University of Pennsylvania
Philadelphia, PA, USA
div11@upenn.edu

Mahmoud Mohammadi
SAP
Seattle, WA, USA
mahmoud.mohammadi@sap.com

Joe Meyer
SAP
Palo Alto, CA, USA
joseph.meyer@sap.com

Vinam Arora
University of Pennsylvania
Philadelphia, PA, USA
vinam@upenn.edu

Tom Palczewski
SAP
Palo Alto, CA, USA
tom.palczewski@sap.com

Eva L. Dyer
University of Pennsylvania
Philadelphia, PA, USA
eva.dyer@upenn.edu

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 filtering constraint to exclude future events rather than a modeling signal, and are typically designed for single-task prediction. To address these gaps, we introduce a temporal subgraph sampler that enhances global context by retrieving nodes beyond the immediate neighborhood to capture temporally relevant relationships. In addition, we propose the **Relational Graph Perceiver (RGP)**, a graph transformer architecture for relational deep learning that leverages a cross-attention-based latent bottleneck to efficiently integrate information from both structural and temporal contexts. This latent bottleneck 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. RGP also incorporates a flexible cross-attention decoder that supports joint learning across tasks with disjoint label spaces within a single model. 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 presents 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,

*Work done during internship at SAP.

Lachi et al., Integrating Temporal and Structural Context in Graph Transformers for Relational Deep Learning. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 338, Hybrid Event, December 10–12, 2025.

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]. RelGT [10], a recent method, adapts GTs to handle the structure and temporal homogeneity found in relational databases. However, current relational graph models, whether based on GNNs [11] or GTs [10], primarily focus on spatial structure and often treat time just as a constraint rather than a modeling signal. In particular, 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. To address this limitation, we introduce a temporal subgraph sampler that retrieves temporally relevant nodes, allowing the model to reason about nonlocal events that are structurally distant but contextually similar. To incorporate this rich context, we introduce the **Relational Graph Perceiver (RGP)**, a graph transformer architecture that efficiently integrates structural and temporal information through a cross–attention–based latent bottleneck. Finally, RGP also 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**[12], and **SALT**[13]—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 needing to train separate 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 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 present the **Relational Graph Perceiver (RGP)**, a graph transformer architecture that efficiently integrates structural information and the rich temporal context extracted by our sampler. It uses a cross-attention-based latent bottleneck, enabling scalable global reasoning across heterogeneous relational graphs.
- 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 Relational Graph Perceiver (RGP), a general-purpose transformer architecture for learning on heterogeneous temporal relational graphs. As shown in Figure 1, RGP is built around three key components: (i) a temporal subgraph sampler that retrieves temporally relevant nodes beyond the immediate neighborhood of the seed or query node (Section 2.2); (ii) a Perceiver-style encoder that uses cross-attention to compress structural and temporal context into a fixed-size latent representation (Section 2.3); and (iii) a lightweight, flexible multi-task decoder that enables training across multiple classification tasks with diverse label spaces, without requiring task-specific linear layers (Section 2.4).

Before describing these components in detail, we first explain how relational databases are converted into heterogeneous temporal graphs and tokenized for transformer-based modeling (Section 2.1). 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

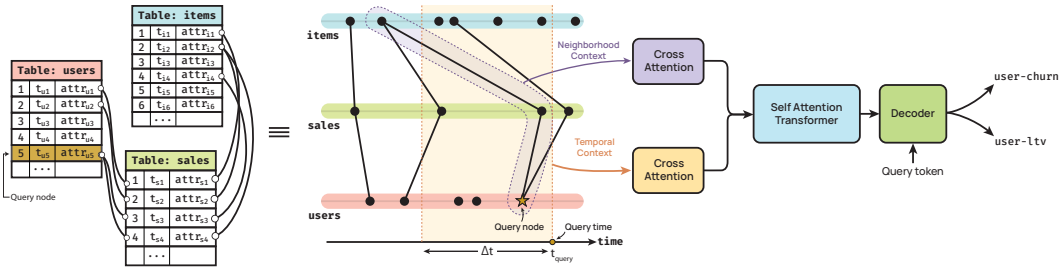

Figure 1: Overview of the RGP architecture. We convert relational databases into heterogeneous temporal graphs, where nodes represent entities (e.g., users, items, or sales) and edges capture interactions between them. Given a seed or query node (e.g., a user), the model applies two parallel cross-attention 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).

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 (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) and `MultiModalEncoder` is the modality-aware encoder taken from [14]. 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 structural 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.

Note: 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 both structural sampling [11] and temporal context sampling (Section 2.2). When timestamps are missing or partially available, the sampler defaults to standard neighborhood sampling without temporal constraints, allowing the model to operate on both temporal and purely structural relational data. 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 2.3).

2.2 Temporal Sampler

In many real-world relational systems, temporally correlated events often occur across entities that are not directly connected in the underlying graph. For instance, a sudden market shift may influence multiple customer accounts that share no explicit transactional links, or a new product release may simultaneously affect several supplier and retail nodes. Capturing information from such temporally proximate but structurally distant entities can provide valuable context for prediction. However, standard neighborhood sampling methods apply time-restricted sampling around a target node to prevent temporal leakage [10, 11] (Figure 1) by excluding nodes that occur at future timestamp. These methods focus on preserving local graph structure while enforcing temporal constraints, treating time primarily as a boundary condition on graph-based neighborhood sampling.

To complement this, 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 $G = (V, E)$ with node timestamps $T_v : V \to \mathbb{R}$, we define an edge timestamp $T_e(u, v) = f(T_v(u), T_v(v))$, where f can be the mean or maximum of the endpoint timestamps. Using a seed time t_{seed}, we then extract a temporal subgraph by selecting edges within a fixed time window Δt or the k nearest edges in time. For datasets where edge timestamps are already provided, we retain those original values. The full algorithm is provided in Appendix A.1.2.

2.3 Compression via Cross Attention

The temporal sampler introduced in Section 2.2 provides rich context by retrieving nodes that are temporally relevant but structurally distant. To effectively integrate this expanded context without incurring the quadratic cost of full self-attention, we adopt a Perceiver-inspired encoder [15, 16] that compresses both structural and temporal node embeddings into a fixed set of latent representations through cross-attention.

Latent bottleneck. Given the tokenized graph structure described in Section 2.1 and the subgraph sampled around each target entity (using both structural and temporal samplers), we obtain a sequence of input node embeddings $\mathbf{X}_g = [\mathbf{x}_1, \ldots, \mathbf{x}_{N_g}]$. We maintain a shared set of K *learnable* latent tokens $\mathbf{Z}_0 = [\mathbf{z}_{0,1}, \ldots, \mathbf{z}_{0,K}]$, where $K \ll N_g$. These latent tokens attend to all nodes via a cross-attention mechanism:

$$\mathbf{Z}_g^{(1)} = \mathbf{Z}_0 + \text{softmax}\left(\frac{(\mathbf{W}_q\mathbf{Z}_0)(\mathbf{W}_k\mathbf{X}_g)^\top}{\sqrt{d_k}}\right)(\mathbf{W}_v\mathbf{X}_g), \tag{1}$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are learned projection matrices. Through this mechanism, the latent tokens integrate information from the entire graph without requiring dense pairwise interactions among all nodes.

Fusing temporal and structural context. To combine temporal and structural signals, we apply two parallel cross-attention branches—one over temporally sampled nodes and another over structurally sampled nodes. Each branch maintains its own set of learnable latent tokens and uses an independent

cross-attention layer with its own query, key, and value projections. The resulting latent representations from the two branches are summed elementwise to produce a unified latent embedding. This fused latent representation allows the model to integrate information from structurally local and temporally correlated regions of the graph.

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 [17]. Because self-attention is restricted to K tokens, the computational cost scales as:

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

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

2.4 Multi-Task Decoder

After the Perceiver encoder compresses the input subgraphs (structural and temporal neighborhood) 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. For each prediction task, we define a task embedding $\mathbf{q}_{\text{task}} \in \mathbb{R}^d$ that encodes task-specific context (e.g., classification objective or label space). The embedding is obtained by passing the textual task description through a pretrained language model and projecting the resulting representation into the model's latent space. Given this fixed task embedding 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.

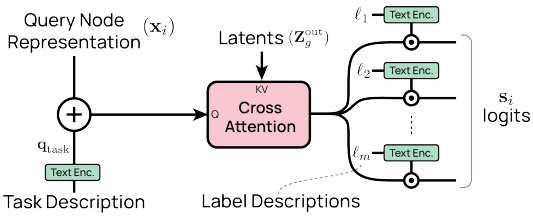

Figure 2: Overview of Flexible multi-task decoder: The decoder receives a query node representation (combined with a task description embedding) and attends to the latents from the perceiver encoder via cross attention.

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

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 it using a frozen or pretrained text encoder. For example, in a user churn prediction task for the `rel-hm` dataset, a candidate label can be represented as a text string such as "H&M customer will stop making transactions in next week," which is then encoded using a pretrained text encoder to obtain the corresponding label embedding.

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

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:

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

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**

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: RelBench [11], CTU [12], and SALT [13]. 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 [18] with a fixed learning rate of 10^{-3}. Similar to previous work [10], 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 the cross-attention block. All other settings, such as batch size and dropout, 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 publicly available baselines for each setting. Across all benchmarks, we include comparisons to RDL [19], a widely adopted pipeline that combines RelGNN [20] 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. Each dataset has a binary node classification task, where the objective is to predict an entity's future state or behavior based on its multi-relational and temporal context. For instance, in the `user-churn` task for `rel-hm`, the model predicts whether a customer will become inactive (i.e., have no transactions) in the following week. Performance is evaluated using the AUC-ROC metric (refer to Appendix A.3.1 for

Table 2: Results on CTU benchmarks: We report F1 score as the evaluation metric. Best values are in **bold**. Relative gains denote percentage improvement of RGP over DBFormer and LightGBM.

Dataset	Task	LightGBM	XGBoost	TabResNet	Linear	SAGE (RDL)	DBFormer	**RGP (ours)**	Rel. (%) Gain vs. DBF	Rel. (%) Gain vs. LGBM
accidents	temp.	0.170	0.336	0.187	0.583	0.566	0.727	**0.743**	+2.20	+337.06
dallas	temp.	**0.584**	0.512	0.247	0.393	0.424	0.513	0.555	+8.19	-4.96
legalacts	temp.	**0.851**	0.220	0.220	0.721	0.698	0.703	0.736	+4.69	-13.52

results on regression tasks). For RelBench, we report results from the Relational Graph Transformer (RelGT) [21], the current state-of-the-art method on this benchmark. The paper also compared against HGT [22] and a variation of HGT with Laplacian positional encodings [8, 23]. 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. 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 `rel-f1` dataset RGP outperforms 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%**.

In terms of computational cost, RGP scales as $O(KN_gd + LK^2d)$ with $K \ll N_g$, achieving near-linear dependence on the number of input nodes N_g. In contrast, HGT and RelGT require full self-attention with $O(N_g^2 d)$ complexity, quadratic in input size. For comparison, message-passing GNNs operate in $O(M_g d)$ time, where M_g denotes the number of edges. RGP therefore offers a favorable trade-off between efficiency and expressivity, maintaining transformer-level performance while significantly reducing computational overhead.

CTU [12] 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. For example, in the `LegalActs` dataset, the model predicts the `ActKind` (type of court decision) based on case metadata, associated judges, and related legal documents. We compare against baselines reported in the ReDeLEx benchmark [24], which include traditional tabular models (e.g., LightGBM [25], XGBoost [26]), temporal MLPs (e.g., TabResNet [27]), and the GNN-based RDL [11] framework. Following standard practice from ReDeLEx [24], we report F1 scores for all tasks. As shown in Table 2, RGP consistently outperforms the DBFormer baseline across all evaluated CTU datasets. Notably, on the `dallas` dataset, RGP achieves a relative gain of **8.19%**. However, we note that tree-based methods such as LightGBM outperform RDL-based approaches on two out of three CTU tasks. This observation aligns with prior work [24, 28], which shows that boosting-based models tend to perform well on smaller or flatter relational databases with few one-to-many links but many factual (non-key) columns. LightGBM performs strongly on *Dallas* and *LegalActs* because most predictive information in these datasets is contained within a single table composed primarily of categorical and factual metadata. In contrast, *Accidents* links each event to multiple participants with distinct roles and attributes, where outcomes depend on their joint interactions and contextual factors such as location and time. Flattening these relationships into a single table discards this relational structure, limiting the ability of tree-based methods to model inter-entity dependencies. Models like RGP, which explicitly capture cross-entity and temporal relationships, are therefore better suited for such settings.

SALT [12] 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. For example, in the `sales-shipcond` task, the model predicts the `ShippingCondition` of an order, such as the applicable delivery terms, given information about the customer, product, and sales document context. As graph-based

Table 3: 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

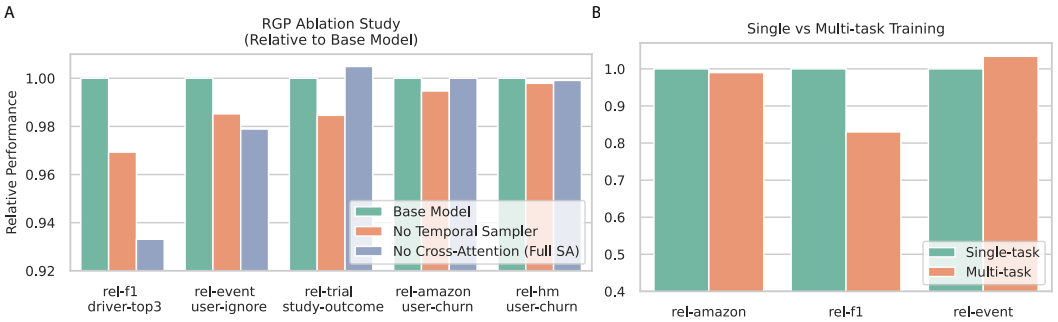

Figure 3: **(A)** Results from ablation study of RGP. We evaluate the impact of removing key components from the full RGP model. Performance is reported relative to the base model. A decrease in performance indicates that the removed component is important to overall model effectiveness. **(B)** Relative 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.

models have not been previously applied to SALT, we compare RGP against standard baselines, including RDL and the Heterogeneous Graph Transformer (HGT) [22]. As shown in Table 3, RGP frequently outperforms both baselines for three out of four tasks. For the `sales-payterms` task, our model is slightly below HGT (0.58 vs. 0.60 MRR) while still outperforming RDL.

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.

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 2–6x more compute-efficient in comparison to self-attention (see Appendix A.2.2 for run times).

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. 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 3B, RGP achieves comparable or improved results under multi-task training. 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 `rel-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 $\sim$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 [29] with temporal neighbor sampling, surpassing classical tabular methods like LightGBM [25]. Several architectures have been proposed to better exploit relational structure: RelGNN [20] introduces composite message passing to preserve information across bridge and hub nodes, while ContextGNN [30] combines pair-wise and two-tower encoders for recommendation scenarios.

Graph Transformers (GTs) adapt the self-attention paradigm [17] to graph-structured data, capturing long-range dependencies without iterative neighborhood aggregation [31]. Early GT variants focused on local attention with positional encodings like Laplacian eigenvectors [32], while later designs incorporated global attention mechanisms [9, 33, 34] and scaling strategies such as hierarchical pooling or sparse attention. Heterogeneous GTs such as HGT [22] and Hinormer [35] 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 [36] 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 [15] 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 [16, 37], 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 a temporal subgraph sampler that expands the relevant input context by retrieving nodes that are temporally proximate yet structurally distant, enabling the incorporation of context that traditional neighborhood sampling fails to capture. Building on this, we proposed the Relational Graph Perceiver (RGP)—a transformer-based architecture that efficiently integrates temporal and structural information via a Perceiver-style cross-attention bottleneck, and supports multi-task prediction across diverse 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 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.

Acknowledgements

Thanks to Shivashriganesh P. Mahato and Keertika Saroj for insightful discussions and feedback on this manuscript. This project was also supported by NSF CAREER Award RI:2146072, NSF award CIF:RI:2212182 as well as generous gifts from the CIFAR Azrieli Global Scholars Program (D.L, V.A and E.L.D).

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. 2

[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. 2

[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. 2

[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. 2

[8] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. 2, 7

[9] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021. 2, 4, 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. 2, 3, 4, 6, 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, 13

[12] Jan Motl and Oliver Schulte. The ctu prague relational learning repository, 2024. URL `https://arxiv.org/abs/1511.03086`. 2, 6, 7

[13] 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

[14] 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, 13

[15] 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

[16] Divyansha Lachi, Mehdi Azabou, Vinam Arora, and Eva Dyer. Graphfm: A scalable framework for multi-graph pretraining. *arXiv preprint arXiv:2407.11907*, 2024. 4, 9

[17] 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. 5, 9

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

[19] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023. 6

[20] Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. Relgnn: Composite message passing for relational deep learning. *arXiv preprint arXiv:2502.06784*, 2025. 6, 9

[21] 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. 7

[22] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020. 7, 8, 9

[23] Vinam Arora, Divyansha Lachi, Shivashriganesh P Mahato, Mehdi Azabou, Zihao Chen, and Eva L Dyer. Exploiting all laplacian eigenvectors for node classification with graph transformers. In *New Perspectives in Graph Machine Learning*. 7

[24] Jakub Peleška and Gustav Šír. Redelex: A framework for relational deep learning exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 438–456. Springer, 2025. 7

[25] 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

[26] 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

[27] 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

[28] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022. 7

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

[30] 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

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

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

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

[34] 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. 9

[35] Qiheng Mao, Zemin Liu, Chenghao Liu, and Jianling Sun. Hinormer: Representation learning on heterogeneous information networks with graph transformer. In *Proceedings of the ACM web conference 2023*, pages 599–610, 2023. 9

[36] 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. 9

[37] 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-modal 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, 14].

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.

In practice, the edge-sorting step required for sampling is performed once during preprocessing. During training and inference, sampling reduces to simple lookups within these pre-sorted adjacency lists, making its computational cost negligible compared to attention operations.

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. All experiments were conducted on a single NVIDIA B200 GPU.

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

A.3 Additional Results

A.3.1 Performance on Relbench regression tasks

As shown in Table 6, RGP consistently outperforms baseline methods across all evaluated datasets, with particularly strong performance on the Avito dataset.

Table 6: Results on regression tasks for Relbench: We report Mean Absolute Error (MAE, lower is better). Best values are shown in **bold**.

Dataset	Task	RDL	HGT	HGT+PE	RelGT	**RGP (ours)**
F1	driver-position	4.022	4.1598	4.2358	3.9170	**3.8702**
avito	ad-ctr	0.0410	0.0441	0.0494	0.0345	**0.01166**
hm	item-sales	0.0560	0.0655	0.0641	0.0536	**0.05269**

A.3.2 Effect of Latent Token Count

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 4, 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.

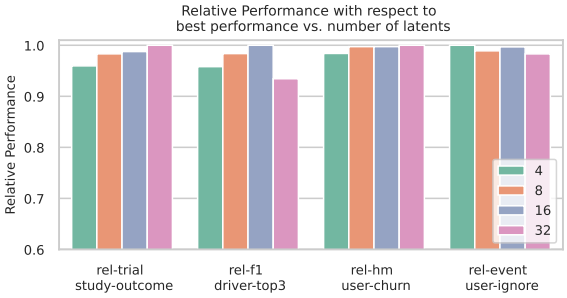

Figure 4: Effect of number of latent tokens on model performance across four representative tasks. For each task, we normalize results with respect to the best-performing configuration to compute relative performance.

A.3.3 Multi-Task Results

We compare RGP's performance under single-task and multi-task training across all datasets to assess the scalability of the proposed decoder. In the multi-task setup, a single model is trained jointly on all tasks within each dataset using task-conditioned supervision via text-based task queries and label embeddings.

As shown in Figure 5, RGP maintains comparable or improved performance under multi-task training in most datasets. While the *event* dataset benefits from joint training, the *f1* dataset shows a slight drop due to task imbalance. Overall, these results demonstrate that RGP's decoder supports efficient and generalizable multi-task learning across diverse relational tasks.

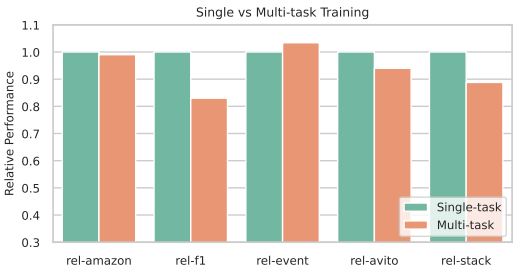

Figure 5: Relative performance of multi-task vs. single-task training across datasets: The Y-axis shows the average multi-task performance normalized with respect to the average single-task performance across all tasks within each dataset.