Task Scarcity and Label Leakage in Relational Transfer Learning

Francisco Galuppo Azevedo[1,2], **Clarissa Lima Loures**[1,2], **Denis Oliveira Correa**[1]
[1]Kunumi Institute, Belo Horizonte, Minas Gerais, Brazil
[2]Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, Brazil
{francisco,clarissa.loures,denis}@kunumi.com

Abstract

Training relational foundation models requires learning representations that transfer across tasks, yet available supervision is typically limited to a small number of prediction targets per database. This task scarcity causes learned representations to encode task-specific shortcuts that degrade transfer even within the same schema, a problem we call label leakage. We study this using K-Space, a modular architecture combining frozen pretrained tabular encoders with a lightweight message-passing core. To suppress leakage, we introduce a gradient projection method that removes label-predictive directions from representation updates. On RelBench, this improves within-dataset transfer by +0.145 AUROC on average, often recovering near single-task performance. Our results suggest that limited task diversity, not just limited data, constrains relational foundation models.

1 Introduction

Relational databases remain the dominant substrate for operational data. Building foundation models for this setting requires representations that transfer across tasks. Yet a typical relational database offers only a handful of supervised prediction tasks. This task scarcity is a data bottleneck distinct from sample scarcity: even with millions of rows, the number of distinct objectives available to shape representations is small.

When supervision comes from a single task, learned representations are free to encode any feature that predicts the label, including task-specific shortcuts that do not generalize. We call this label leakage: the representation absorbs label-predictive directions that help the training objective but hurt transfer. With abundant task diversity, conflicting gradients from different objectives would suppress such shortcuts. Under task scarcity, no such pressure exists. As we show empirically, within-dataset transfer can collapse to near-chance performance even when single-task accuracy is strong.

We study this problem within the Relational Deep Learning (RDL) framework, which constructs heterogeneous graphs from multi-table databases and learns directly over that structure Fey et al. (2023); Robinson et al. (2024). Our testbed is **K-Space**, a modular RDL architecture with a convolutional message-passing core that has linear edge complexity. Row-level representations come from a frozen pretrained tabular encoder, and predictions are made by a frozen in-context learning head Qu et al. (2025). This modularity isolates the relational backbone as the component where leakage can occur. To suppress it, we introduce a gradient projection method: at each training step, we remove from the main loss gradient any component that aligns with an adversarial label-prediction gradient, computed per sample at the representation level.

Contributions.

- We empirically demonstrate that task scarcity causes label leakage in relational learning: within-dataset transfer often collapses to near-chance performance despite strong single-task accuracy.

- We propose a sample-wise gradient projection method that suppresses label-predictive directions in the shared representation during training.
- We show that this intervention improves within-dataset transfer (+0.145 AUROC on average) but does not help and can hurt cross-dataset transfer, suggesting different mechanisms underlie the two settings.

2 Related Work

RDL and RelBench. Relational deep learning constructs heterogeneous graphs from multi-table databases and trains neural predictors over these structures Fey et al. (2023). RelBench provides standardized datasets and tasks for this setting, with multiple tasks per database Robinson et al. (2024). This enables evaluation of within-dataset transfer.

Relational transformers. Recent transformer-based approaches target strong single-task performance and transfer Dwivedi et al. (2025); Wang et al. (2025); Ranjan et al. (2025). RelGT achieves the highest reported results on RelBench under supervised training and serves as our primary baseline. Relational Transformer takes a different approach, using masked prediction to create self-supervised tasks from unlabeled data, sidestepping task scarcity. We explore a complementary direction: mitigating label leakage when only supervised tasks are available.

Pretrained tabular encoders and ICL heads. TabICL provides a scalable tabular ICL model whose embedding stages yield strong row representations Qu et al. (2025). TabPFN-style models illustrate that ICL heads can be effective predictors for tabular tasks, including variants that support regression Hollmann et al. (2022). Freezing both encoder and predictor isolates the relational backbone as the sole learnable component and the locus of potential label leakage.

Scalable message passing. SMPNN shows that deep message passing can be stabilized using transformer-like residual blocks while preserving linear edge complexity Sáez de Ocáriz Borde et al. (2024). K-Space adopts SMPNN as the relational building block and extends it to heterogeneous directed propagation with shared weights. A simple architecture allows transfer failures to be attributed to the representation rather than architectural mismatches.

Adversarial methods for representation learning. Adversarial training can remove unwanted information from representations. Domain-adversarial neural networks (DANN) use gradient reversal to learn domain-invariant features Ganin & Lempitsky (2015), but we found this unstable for removing task-specific label information. Instead, inspired by Muon Jordan et al. (2024), we subtract the adversarial gradient's component from the main gradient.

3 Architecture

We now describe the K-Space architecture and our gradient projection implementation. It has three blocks: a frozen table encoder, a relational convolutional core, and an ICL predictor head, see Figure 1.

Frozen table encoder. Each node begins as a row in some table. We compute row embeddings using the frozen embedding stages of TabICL Qu et al. (2025). To these, we concatenate temporal features (index plus periodic encodings) and random walk positional encodings (RWPE). Unlike the original RWPE formulation Dwivedi et al. (2021), which uses matrix exponentiation, we compute encodings via explicit random walks that cannot traverse future edges. The result is projected to the model width, with table type injected via RoPE Su et al. (2021).

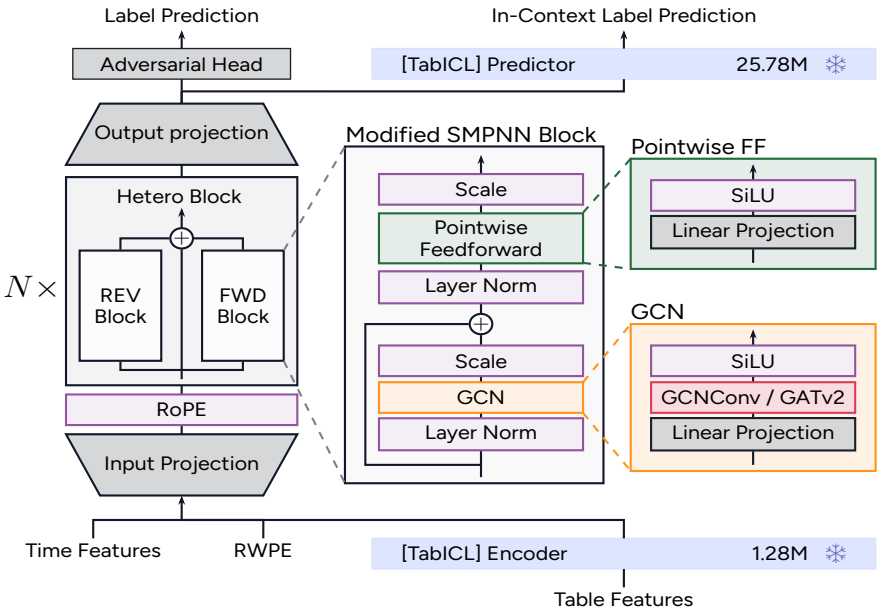

Figure 1: Schematic of the proposed architecture. A TabICL encoder (column embedder and row interactor) encodes table features; its output is concatenated with RWPE and time features and passed through an input projection and per-table-type RoPE into a stack of N Hetero (SMPNN) blocks with reversible (REV) and forward (FWD) paths. Zoomed views show the SMPNN block and its GCN and pointwise feedforward sub-blocks. An output projection feeds two heads: an adversarial MLP head and a TabICL predictor head.

Relational convolution. We build on SMPNN Sáez de Ocáriz Borde et al. (2024). Each layer uses shared parameters across relation types and separates incoming from outgoing propagation. For each relation type r, we compute an update from incoming neighbors and an update from outgoing neighbors, aggregate across r, and apply a residual update. The full procedure is given in Algorithm 1 (Appendix A).

ICL predictor head We predict binary labels via a pretrained frozen ICL head: each mini-batch splits labeled instances into support (embedding, label) pairs and query embeddings. The head consumes support context and outputs query probabilities. Our experiments focus on classification due to the TabICL predictor interface Qu et al. (2025).

Label-leakage suppression All reported K-Space results use the same architecture; we vary only whether training includes an adversarial label-prediction head. The motivation is to reduce direct label information in representations, which can hinder transfer when tasks change.

Let $\mathbf{z}$ denote the query representation after the GNN and before the ICL head. We train with the main ICL loss ℓ_{main} on query predictions. We optionally add an adversarial head trained to predict the label from $\mathbf{z}$ alone. When enabled, we compute per-query gradients w.r.t. $\mathbf{z}$ and remove components of $\nabla_{\mathbf{z}}\ell_{\text{main}}$ that align with $\nabla_{\mathbf{z}}\ell_{\text{adv}}$ via sample-wise projection, as in Figure 2. For efficiency, we apply this transformation at the representation level and backpropagate the averaged transformed gradient through the backbone.

4 Experiments

We ask two questions. First, does task scarcity cause label leakage that harms transfer? Second, does gradient projection reduce it? We evaluate on binary classification tasks from RelBench Robinson et al. (2024) and report AUROC. We compare against RelGT Dwivedi et al. (2025), a strong transformer-based baseline.

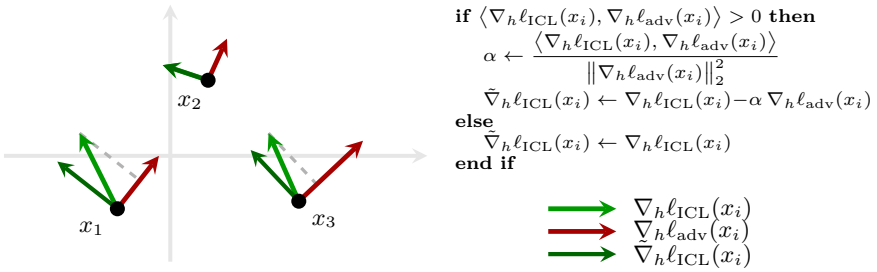

Figure 2: Geometry of per-sample gradients in the representation space of h, the shared representation just before the two heads. For each sample x_i, the light green arrow is the in-context learning gradient $\nabla_h \ell_{\text{ICL}}(x_i)$, the red arrow is the adversary gradient $\nabla_h \ell_{\text{adv}}(x_i)$, and the dark green arrow is the refined gradient $\tilde{\nabla}_h \ell_{\text{ICL}}(x_i)$ after subtracting its projection onto $\nabla_h \ell_{\text{adv}}(x_i)$ (dashed gray segment).

To isolate the effect of task scarcity, we evaluate under four supervision regimes:

- **ST (Single-task):** Train and evaluate on the same task. This is the standard setting and our upper bound.
- **WD (Within-dataset):** Train on a different task from the same database. Same schema, different labels. This is our main test for label leakage.
- **CD (Cross-dataset):** Train on all tasks except those from the target database. Different schema entirely.
- **ALL (All-task):** Train on all tasks jointly. Tests multi-task interference.

Two datasets (`rel-hm` and `rel-trial`) have a single task, so WD cannot be computed for these. Additionally, `rel-event` was excluded from ALL due to memory constraints, and ALL was trained for only 3 epochs (versus up to 20 for other regimes) due to time constraints, so ALL results may underestimate the potential of multi-task training.

For each regime, we report two K-Space variants: **Base** (without adversarial head) and **Adv.** (with adversarial head and gradient projection). Both use identical architecture; only the training objective differs. Hyperparameter selection is described in Appendix B.

5 Results

The main results are shown in Table 1.

Single-task. Under single-task supervision (ST), K-Space Base achieves an average AUROC of 0.737, compared to 0.767 for RelGT. The gap is expected given our modular constraints: frozen encoder, frozen predictor, and a simple message-passing backbone. Despite this, K-Space matches or exceeds RelGT on some tasks (e.g., rel-event/user-ignore: 0.852 vs. 0.816). The adversarial variant slightly underperforms Base in ST (0.722 vs. 0.737), suggesting that removing label-predictive directions can reduce accuracy when transfer is not required.

Within-dataset. Within-dataset transfer (WD) is where gradient projection has the largest effect. Average AUROC increases from 0.486 (Base) to 0.631 (Adv.), a +0.145 improvement. In several cases, Base collapses toward random while Adv. recovers to near-ST performance (e.g., rel-f1/driver-top3: 0.219 $\rightarrow$ 0.829). This supports our hypothesis that the backbone encodes task-specific shortcuts that do not transfer, and that projection substantially reduces this failure mode.

Cross-dataset. In cross-dataset transfer (CD), the adversarial variant performs worse than Base (0.544 vs. 0.611). One interpretation is that projected-out directions include

Dataset	Task	RelGT	K-Space							
			ST		WD		CD		ALL	
			Base	Adv.	Base	Adv.	Base	Adv.	Base	Adv.
rel-f1	driver-dnf	0.759	0.735	0.729	0.368	0.702	0.645	0.447	0.609	0.446
	driver-top3	0.835	0.818	0.647	0.219	0.829	0.724	0.683	0.528	0.790
rel-avito	user-visits	0.668	0.645	0.646	0.401	0.602	0.517	0.421	0.583	0.523
	user-clicks	0.683	0.646	0.646	0.378	0.635	0.577	0.416	0.572	0.536
rel-event	user-repeat	0.761	0.735	0.736	0.298	0.264	—	—	—	—
	user-ignore	0.816	0.852	0.858	0.498	0.504	—	—	—	—
rel-stack	user-engagement	0.905	0.836	0.844	0.809	0.799	0.647	0.739	0.818	0.818
	user-badge	0.863	0.833	0.833	0.501	0.610	0.751	0.698	0.845	0.835
rel-amazon	user-churn	0.704	0.642	0.644	0.631	0.626	0.624	0.513	0.636	0.633
	item-churn	0.826	0.770	0.778	0.755	0.740	0.549	0.511	0.766	0.773
rel-hm	user-churn	0.693	0.674	0.673	—	—	0.546	0.496	0.602	0.622
rel-trial	study-outcome	0.686	0.651	0.627	—	—	0.528	0.515	0.523	0.528
Average		0.767	0.737	0.722	0.486	0.631	0.611	0.544	0.648	0.650

Table 1: **RelBench AUROC.** ST, WD, CD, ALL are the supervision regimes. Within each regime, the left column is the base model, and the right column (Adv.) uses the adversarial head with sample-wise gradient projection. RelGT is the baseline.

features useful for cross-schema generalization. Another is that projection reduces effective gradient signal in settings where the model already struggles. Either way, the current method improves task transfer within a dataset but not dataset transfer across schemas.

Multi-task. Training on all tasks (ALL) yields similar performance for both variants (Base: 0.648, Adv.: 0.650) and underperforms ST on average. Per-task outcomes vary substantially, consistent with negative transfer under heterogeneous supervision.

6 Discussion

Our experiments reveal a clear pattern: under task scarcity, learned representations encode label-predictive shortcuts that do not transfer. Within-dataset transfer collapses to near-chance for the base model, despite strong single-task performance. Gradient projection recovers much of this gap, improving within-dataset transfer by +0.145 AUROC on average. However, the intervention does not help cross-dataset transfer and may even hurt it. Within-dataset and cross-dataset transfer fail differently. The former is dominated by task-specific leakage, while the latter likely involves schema and domain mismatch that our method does not address.

These findings have implications for relational foundation models. Task diversity during training matters more than expected. Self-supervised objectives, such as the masked prediction used by Relational Transformer, offer one way to create abundant tasks from unlabeled data. Explicit leakage suppression offers an alternative approach when training is limited to supervised tasks.

Our study has limitations that suggest future directions: results are classification-only (a regression head such as TabPFN Hollmann et al. (2022) could be substituted), cross-dataset transfer remains weak (requiring methods beyond gradient projection), and we only evaluate on RelBench. More broadly, understanding when task scarcity harms relational representations remains open.

References

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. 2021.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. 2025.

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. 2023. doi: 10.48550/arXiv.2312.04615.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. 2015.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. 2022. ICLR 2023.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. 2025.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. 2025.

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. 2024. doi: 10.48550/arXiv.2407.20060.

Haitz Sáez de Ocáriz Borde, Artem Lukoianov, Anastasis Kratsios, Michael Bronstein, and Xiaowen Dong. Scalable message passing neural networks: No need for attention in large graph representation learning. 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. 2021.

Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a graph-centric relational database foundation model. 2025.

A Architecture Details

Require: Node features x, edge index dictionary $\mathcal{E}$
Initialize aggregate vector $A \leftarrow 0$ (same shape as x)
Initialize count vector $C \leftarrow 0$
for each edge type t in $\mathcal{E}$ **do**
    Retrieve edge index $E \leftarrow \mathcal{E}[t]$
    Determine destination nodes $D \leftarrow \text{unique}(E_{\text{dst}})$
    Update count: $C \leftarrow C + \mathbf{1}_D$
    Select block:
    $B \leftarrow \begin{cases} \text{block}_{\text{rev}}, & \text{if } t \text{ is reversed} \\ \text{block}_{\text{orig}}, & \text{otherwise} \end{cases}$
    Compute message: $\text{msg} \leftarrow B(x, E)$
    Accumulate: $A \leftarrow A + \text{msg} \odot \mathbf{1}_D$
end for
return $x + \dfrac{A}{\max(C, 1)}$

Algorithm 1: Forward Pass of Hetero Block

B Hyperparameter Search Details

We performed sequential hyperparameter optimization to determine the optimal architecture configuration. At each stage, we tuned one hyperparameter while fixing all previously optimized values. We optimized six parameters in order: (1) network depth (number of graph convolutional layers), (2) hidden dimension size for all convolutional layers, (3) neighborhood sampling strategy (number of neighbors sampled at each layer), (4) number of multi-head attention heads, (5) inclusion of temporal features, and (6) dimension of Random Walk Positional Encoding (RWPE) features.

Table 2 summarizes the optimization process across these six stages and results for each hyperparameter search are provided in sequence. We evaluated each configuration on two datasets: `rel-f1-driver-dnf` and `rel-avito-user-clicks`, in the WD setting with gradient projection, reporting average AUROC scores. The sequential search improved performance from 0.470 to 0.669.

Table 2: Sequential Hyperparameter Optimization Overview

Stage	Parameter	Optimal Value	Avg F1 Score
1	Number of Layers	3	0.581
2	Hidden Dimension	256	0.590
3	Neighborhood Size	[16, 8, 4]	0.611
4	Attention Heads	2	0.623
5	Temporal Encoding	Enabled	0.625
6	RWPE	32	0.669

Table 3: Effect of Layer Depth on Model Performance

Dataset	Number of Layers					
	1	2	3	4	5	6
rel-f1-driver-dnf	0.519	0.588	0.587	0.527	0.541	0.553
rel-avito-user-clicks	0.420	0.427	0.575	0.575	0.428	0.577
Average	0.470	0.507	**0.581**	0.551	0.485	0.565

Table 4: Effect of Hidden Dimension on Model Performance

Dataset	Hidden Dimension		
	128	256	512
rel-f1-driver-dnf	0.587	0.584	0.586
rel-avito-user-clicks	0.575	0.597	0.587
Average	0.581	**0.590**	0.587

Table 5: Effect of Neighborhood Configuration on Model Performance

Dataset	Neighborhood Configuration		
	[32, 8, 2]	[16, 8, 4]	[8, 8, 8]
rel-f1-driver-dnf	0.622	0.628	0.584
rel-avito-user-clicks	0.583	0.594	0.597
Average	0.602	**0.611**	0.590

Table 6: Effect of Attention Heads on Model Performance

Dataset	Number of Attention Heads		
	0	1	2
rel-f1-driver-dnf	0.628	0.547	0.623
rel-avito-user-clicks	0.594	0.621	0.624
Average	0.611	0.584	**0.623**

Table 7: Effect of Temporal Encoding on Model Performance

Dataset	Temporal Encoding	
	Disabled	Enabled
rel-f1-driver-dnf	0.623	0.628
rel-avito-user-clicks	0.624	0.621
Average	0.623	**0.625**

Table 8: Effect of Random Walk Positional Encoding (RWPE) on Model Performance

Dataset	RWPE Dimension	
	0 (Disabled)	32
rel-f1-driver-dnf	0.628	0.702
rel-avito-user-clicks	0.621	0.635
Average	0.625	**0.669**