RFMBench: Towards Principled Benchmarking of Relational Foundation Models

Zhikai Chen[1*], Han Xie[2], Jian Zhang[2], Haiyang Yu[2], Huzefa Rangwala[2], Xiang Song[2†]
[1]Michigan State University [2]Amazon
USA

Abstract

Relational foundation models (RFMs) have emerged as a promising paradigm for predictive tasks over relational databases (RDBs), enabling online predictions without task-specific training. Despite this potential, RFM research remains in an exploratory phase: model designs are fragmented, a performance gap persists between open-source and commercial implementations, and benchmarks lack the principled design to explain *when* and *why* certain architectural choices work better. To address these gaps, we introduce RFMBench, a principled benchmark for developing and evaluating RFMs. First, we propose a design framework that decomposes RFMs into encoder, backbone, and inference head, and identifies three core design dimensions—representation level (cell vs. row), relational weight sharing, and pre-training source—that govern model behavior. Second, we develop a codebase supporting multi-level sampling and modular model components, identify pitfalls in existing open-source RFMs, and propose remedies such as adapting tabular foundation models as inference heads and dual-stage in-context learning. Third, we establish a view-based evaluation framework that enables controlled analysis along axes such as context length and relational homophily. Experiments on RFMBench yield actionable insights: while existing open-source RFMs significantly underperform commercial ones, automated feature engineering combined with tabular foundation models offers a competitive alternative. On the modeling side, achieving strong RFMs requires balancing expressiveness against scalability. On the data side, the central challenge is pre-training on sufficiently diverse tasks so that a single model generalizes robustly—current RFMs often require task-specific checkpoint selection, lacking the cross-task stability that mature tabular foundation models excel at.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Information systems** → *Data management systems.*

Keywords

Relational Foundation Models, Relational Deep Learning

1 Introduction

Foundation models have shifted the learning paradigm from task-specific modeling to a unified approach: pre-train once on massive, diverse corpora, then adapt to downstream tasks [3, 4, 43]. Yet while text, image, and video have witnessed this paradigm shift, relational data—stored in RDBs that forms the backbone of enterprise systems—has largely been left behind. Without a foundation model for these data, practitioners still endure the cycle: manual pipeline configurations and task-specific training from scratch [16].

Relational Foundation Models (RFMs) [17] represent early attempts to close this gap, bringing the foundation model paradigm to relational data. Pre-trained on real-world and synthetic relational data, an RFM can generalize to unseen databases without task-specific architectural designs, demonstrating two advantages: (1) *online prediction*—generating predictions on new relational data without any tuning or training [17, 20], achieving competitive performance while eliminating task-specific engineering; and (2) *potential transferability*—may leverage pre-trained knowledge to surpass models trained from scratch after fine-tuning [17, 48] (negative transfer has also been observed in practice [41, 48]). The first capability is particularly transformative for business scenarios, as it eliminates substantial engineering effort and enables paradigms like RFM-as-a-Service [40], which can be seamlessly integrated into existing data agent pipelines [55].

Despite the emergence of RFMs, their design remains in an exploratory phase. For instance, an open question is whether relation-native models are necessary [13], or whether tabular foundation model (TFM) [24] augmented with heuristic structural embeddings or non-parametric feature aggregation suffices [11, 26, 48]. Even among relation-native approaches, it remains unclear whether models should operate at the cell or row level. Answering such questions requires us to revisit current evaluation practices. First, existing benchmarks focus on narrow task types—for instance, RelBench [42] emphasizes temporal event prediction, while masked cell prediction (inferring missing column values in non-temporal settings) remains underexplored—making it difficult to draw general conclusions about which RFM design is superior across diverse scenarios. Second, current evaluations primarily report performance numbers without extracting actionable insights; they reveal *what* performs better but not *why* or *when*, leaving practitioners unable to understand the trade-offs between different architectures. Specifically, we identify three broader benchmark-related issues: (1) *fragmented designs without a unifying framework*—RFM architectures vary widely, from linearizing relational data for tabular foundation models [24] to employing graph neural networks [48], yet existing benchmarks evaluate them as monolithic systems. Without a principled framework to decompose these architectures, key design dimensions—such as representation granularity (cell vs. row), weight sharing strategies, and context construction—remain poorly understood in terms of how they individually and jointly affect performance; (2) *gap between open-source and commercial models*—representative open-source RFMs exhibit significant limitations: Griffin [48] cannot perform online prediction, while RT [41] shows unstable performance, particularly on sparse relational structures. In contrast, commercial models like KumoRFM [17] achieve substantially better results, yet their technical details remain largely

* Part of this work was done during an internship at Amazon. Email: chenzh85@msu.edu.
†Corresponding author.

undisclosed. This gap hinders the research community. (3) *limited evaluation coverage*—existing benchmarks fall short in three aspects. First, task diversity is narrow: RelBench [42] focuses on temporal event prediction with predominantly binary classification, failing to expose limitations of models like RT that don't support multi-class tasks. Second, evaluation settings are designed for task-specific models rather than foundation models: context length—a critical factor for in-context learning [24]—is rarely considered as an evaluation axis. Third, evaluations yield limited insights into when different architectures excel. Despite research such as relational homophily is shown to correlate with GNN performance [1], its influence on RFM design choices remains unstudied.

To close these gaps, we introduce RFMBench, a benchmark designed for developing and evaluating RFMs. Our contributions are threefold: (1) *A modular RFM design framework*—we identify three core design dimensions that differentiate RFMs: *representation level* (cell vs. row), *relational weight sharing* (none, backbone-shared, or per-relation), and pre-training source. These dimensions provide a principled basis for comparing existing RFMs and guiding the design of new ones. (2) *An RFM component library*—we provide a framework with composable encoder-backbone-head architecture. Our unified sampler efficiently handles both cell-level relational sampling and row-level graph sampling. It addresses limitations of existing open-source models. As one example, we introduce dual-stage in-context learning (ICL), distinguishing *graph-level* context (labels from sampled neighbors) from *batch-level* context (cross-sample attention). We demonstrate that RT's degradation on sparse structures stems from a lack of batch-level ICL. (3) *A view-based evaluation framework*—Beyond fixed context and test splits, we introduce customizable *views*—controlled evaluation to probe specific model capabilities. Views can vary in context length or relational homophily, enabling fine-grained analysis of model behavior. Leveraging this framework, experiments reveal several findings: most existing open-source RFMs are not yet ready to handle diverse task types under proper foundation model evaluation protocols; meanwhile, non-parametric feature aggregation combined with tabular foundation models proves a strong competitor to relation-native designs, particularly when enriched with task-table features; and building a fully end-to-end RFM remains an open problem—current evidence suggests that pairing a relational backbone with a tabular foundation model demonstrates some promise at the current stage.

2 Preliminaries

Relational data. Relational data is a commonly adopted abstraction to describe interconnected entities and their relationships. They are often stored in an RDB, which can be formally defined as a pair $\mathcal{D} = (\mathcal{T}, \mathcal{L})$. Here, $\mathcal{T} = \{T_1, \ldots, T_n\}$ denotes a collection of tables, and $\mathcal{L} \subseteq \mathcal{T} \times \mathcal{T}$ represents the inter-table relationships. Every table $T_i \in \mathcal{T}$ contains a set of rows (also called entities) $\{v_1, \ldots, v_{m_i}\}$. The relationships between tables are defined by primary keys (PKs) and foreign keys (FKs): a PK p_v serves as a unique identifier for each row, whereas an FK references a row in a different table by pointing to its corresponding PK. Each row carries non-key attributes x_v, along with an optional timestamp t_v.

Predictive tasks over RDB. Given an RDB, business problems can be formulated as predictive tasks, such as predicting customer churn or forecasting product sales. We consider two types of predictive tasks. The first is the *temporal predictive task*, in which we predict future outcomes for target entities using historical data up to a given prediction time. The second is the *autocomplete task*, where we predict missing attribute values for target entities by leveraging the relational structure and available information across the entire database. Unlike temporal prediction, autocomplete tasks do not impose strict temporal cutoffs. We note that recommendation tasks are not the main focus of this work (as no promising foundation model paradigm has yet emerged for relational recommendation); we briefly discuss them in Section 6.

Relational Foundation Models. Relational Foundation Models (RFMs) are designed to make predictions over arbitrary RDBs without requiring task-specific training [17]. An RFM conducts in context learning as a function $\mathcal{M}_\theta : (\mathcal{D}, \Pi, C) \to \mathcal{Y}$, where θ denotes pre-trained parameters, $\mathcal{D}$ is an RDB (potentially unseen during pre-training), Π specifies a predictive task, and $C = \{(v_i, y_i)\}_{i=1}^k$ is a set of in-context examples. A related line of work considers extending Tabular Foundation Models (TFMs), which can generate predictions for tabular data in context [24], to handle relational data. Applying TFMs to RDBs requires converting relational structures to tabular features. Deep Feature Synthesis (DFS) [26] addresses this by converting relational context into a flattened tabular format by recursively applying mathematical aggregation primitives (such as sum, mean, and mode) along key relationships. However, it remains a question whether such flattened representations can achieve the effectiveness of an end-to-end RFM [17].

3 RFM: Design space and Taxonomy

RFMs differ substantially in how they encode tables, aggregate relational context, and perform inference. In this section, we first formalize the RFM pipeline as modular components, and then identify three core design dimensions that determine RFM behavior.

3.1 Design Space of RFMs

Before diving into specific architectural choices, we clarify what constitutes a valid RFM.

Baseline requirement for RFMs. Following the criteria established for TFMs [24], we consider a model to qualify as an RFM if *it can generate online predictions in context with quality on par with models trained from scratch*. This requirement naturally implies that an RFM must operate inductively on novel databases with unseen schemas and relational structures. We note that potential transferability—where fine-tuning an RFM surpasses training from scratch—is not included as a baseline requirement. As observed in both RFM [41, 48] and TFM [32] literature, transfer benefits are inconsistent and highly dependent on the alignment between pre-training data and downstream tasks. Based on this requirement, we note that works like Wu et al. [50], Wydmuch et al. [51], despite their usage of LLMs, do not qualify as RFMs because of task-specific GNN adapter [50] or head [51] training.

We then examine the various RFM designs. As shown in Table 1, RFM designs vary across three principal axes: how they encode relational structure, what backbone architecture they employ, and how they perform inference.

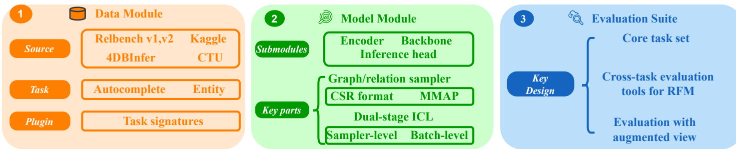


Figure 1: RFMBench provides diverse datasets, modular model implementations, and an evaluation suite with controlled augmentation tools.

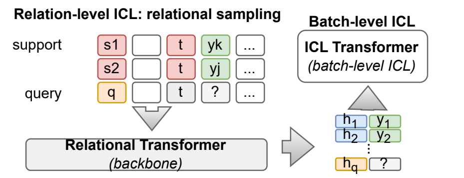


Figure 2: Dual-stage ICL with RT: 1. history labels from task tables provide relation-level labels, 2. seed nodes present batch-level labels for in-context learning.

Table 1: Taxonomy of RFM models across key design dimensions. Grey columns indicate the design dimensions we identify. RFMBench(*) indicates that for fair comparison, we pre-train these models using tasks from RFMBench. Griffin does not support online prediction and only supports few-shot fine-tuning. RDL-GNN and RelGT+TFM are included as training-from-scratch baselines for reference. We don't consider the performance of LLM-based models since they always achieve negative R^2 performance on regression tasks.

Model	Level	Rel. Weight Sharing	Pre-training Source	Encoder	Aggregation	Backbone	Inference Patterns
DFS+TabPFN	Cell	None	N/A (relation-level)	Tab encoder	Non-parametric	Transformer	Batch-level ICL
RT	Cell	None	RFMBench(*)	Cell-level encoder	Attention over set	Relational Transformer	Graph-level ICL
LLM-based	N/A	N/A (relation-level)	N/A (relation-level)	Text serialization	N/A	LLM	Batch + Graph ICL (text)
Griffin	Row	Backbone shared	RFMBench(*)	Tab encoder + metadata	MP	GNN	No ICL (*)
KumoRFM	Row	Backbone shared	Unknown	Tab encoder	MP + Attention over set	Rel GT	Batch + Graph ICL
RT+TFM (ours)	Cell	None	RFMBench(*)	Cell-level encoder	Attention over set	Relational transformer	Batch + Graph ICL
Griffin+TFM (ours)	Row	Backbone shared	RFMBench(*)	Tab encoder + metadata	MP	NBFNet	Batch + Graph ICL
RDL-GNN	Row	Per-relation	N/A	Tab encoder	MP	GNN	Task-specific head
RelGT	Row	Backbone shared	N/A	Tab encoder	MP + Attention over set	RelGT	Task-specific head

Encoder. The encoder transforms rows or cells into embeddings. Despite surface-level differences—table encoders [17, 42], metadata-augmented encoders [48], and cell-level encoders [41]—most encoders share a similar core: MLPs for numerical features and inductive encoders (e.g., text encoders) for categorical ones. Encoder choice has a limited impact on performance across existing RDB benchmarks, as can be seen from Appendix H, and random features sometimes achieve comparable results.

Backbone Architecture. The backbone aggregates cross-table information into a representation through *context sampling* and *context processing*. For context sampling, three paradigms exist: (1) *non-parametric aggregation* [11, 26], where DFS-style methods traverse PK-FK links and apply type-aware primitives to flatten relational context into a single table; (2) *cell-level relation sampling* [41], where a time-aware traversal collects neighboring cells from selected rows into a sequence; and (3) *row-level graph sampling* [17, 42, 48], where the RDB is treated as a heterogeneous graph and a neighborhood sampler extracts subgraphs around target entities. For context processing, the choice depends on the sampling strategy: non-parametric aggregation feeds the flattened table to a TFM; cell-level sampling uses a *relational transformer* [41] with specialized attention masks, extracting predictions directly from target cell embeddings; graph sampling supports two views: (i) *graph view*, using GNNs like GraphSAGE [42] or Griffin [48]; and (ii) *set view*, using graph transformers like RelGT [12] that serialize the subgraph into a sequence and inject relational structure through positional encodings.

Inference Head. After obtaining representations, an inference head generates final predictions. Task-specific models retrain this head for each new task, but RFMs require a unified head that generalizes across tasks and supports in-context learning (ICL). For TFMs, ICL operates at a single level: each batch is partitioned into support and query sets, with the support set providing labeled context [24, 33]. As shown in Figure 2, relational data introduces a second context level: during sampling, the sampler may traverse back to the task table and utilize historical labels—we term this *relation-level ICL*, in contrast to the *batch-level ICL* used in TFMs. Among open-source models, only RT [41] supports in-context inference, but considers only relation-level ICL; KumoRFM [17] discusses an idea similar to dual-level ICL, yet releases no code and even no technical details.

3.2 Core Design Dimensions of RFMs

The encoder-backbone-head decomposition provides a modular view of RFM architectures, but analyzing how individual components influence performance remains challenging due to the large number of interacting design choices. We therefore seek to identify latent design dimensions that better explain model behavior. Inspired by Relatron [1], which observes that DFS paired with different tabular models exhibits similar behavior, as do RDL methods with different GNNs, we hypothesize that certain design choices may dominate others in shaping performance patterns. To investigate this, we conduct preliminary experiments comparing representative methods across driver-dnf and user-badge. As shown in Table 2, we observe two phenomena. First, a *grouping* effect emerges; different models may share similar performance characteristics: RT and DFS+TabPFN behave similarly despite different backbones, as do SAGE and NBFNet. Second, these groups exhibit different trade-offs: one achieves lower validation scores but generalizes better on the test set, while the other shows the opposite pattern. Third, the two groups scale differently with context: DFS+TabPFN excels in low-data regimes, whereas SAGE wins as more context is provided.

Table 2: (a) Grouping and generalization gap: on rel-f1/driver-dnf, SAGE and NBFNet achieve higher validation scores but generalize worse to test, while RT and DFS+TabPFN show the opposite pattern. (b) Context scaling: on rel-stack/user-badge, DFS+TabPFN excels in low-data regimes while SAGE catches up given more context.

(a) Grouping and Generalization Gap (rel-f1 / driver-dnf)			
Group	**Method**	**Val AUROC**	**Test AUROC**
A	RT	71.74	**76.94**
	DFS+TabPFN	71.33	76.90
B	SAGE	77.65	73.62
	NBFNet	**80.02**	73.87
(b) Context Scaling (rel-stack / user-badge)			
Group	**Method**	**Test AUROC@100**	**Test AUROC@10000**
A	DFS+TabPFN	**77.62**	84.61
B	SAGE	75.39	**86.35**

Comparing the grouped methods in Table 1, we extract three design dimensions that distinguish them (highlighted in grey): (1) *representation level*—whether the model operates at cell level or row level; (2) *relational weight sharing*—whether model weights are shared across relation types (none, backbone-shared, or per-relation); and (3) *pre-training source*—what data the model is pre-trained on (e.g., tabular data, relational data, or text). We systematically validate whether these dimensions can predict model performance across diverse tasks in Section 5.

4 The RFMBench Benchmark

Section 3 formalizes the RFM pipeline and identifies three core design dimensions that may influence model behavior. Yet systematically investigating these dimensions remains difficult due to fragmented codebases, inconsistent evaluation protocols, and limited tools for controlled analysis. To bridge this gap, we introduce **RFMBench**, an open-source benchmark comprising: (1) diverse **datasets** and **tasks** from multiple domains and scales; (2) reference **model implementations** with modular building blocks for rapid prototyping; and (3) a comprehensive **evaluation suite** with controlled augmentation tools to generate dataset views with varying properties. Figure 1 provides an overview of RFMBench. We now describe each component in detail.

4.1 Datasets and Tasks

RFMBench supports data from four sources: *RelBench* [42], *CTU Prague Repository* [35], *4DBInfer* [46], and *Kaggle*. It considers *autocomplete tasks* and *entity-level prediction tasks* as discussed in Section 2. Both formats support binary and multi-class classification, as well as regression. Evaluating RFMs on every task is time-consuming, and not all tasks are equally informative for assessing RFM capabilities. Examining tasks across different domains reveals recurring patterns: some tasks are dominated by tabular features; some tasks suffer from temporal leakage, where future information inadvertently influences predictions [41]; and some tasks are simply too easy, with near-perfect accuracy achievable by trivial baselines [46]. Evaluating on such tasks provides little insight into an RFM's capability. RFMBench extracts a *core evaluation set*—representative tasks that quickly assess an RFM (Figure 3). As shown in Figure 3, we select 12 representative tasks as the core evaluation suite, spanning diverse task characteristics: driver-dnf and driver-position (rel-f1), study-outcome and study-adverse (rel-trial), user-badge (rel-stack), author-category and author-publication (rel-arxiv), user-visits and ad-ctr (rel-avito), charge and prepay (dbinfer-seznam), and repeater (dbinfer-avs). Detailed license and selection process are provided in Appendix F.

Task signatures. To understand model behaviors on core test set, we characterize each task via four signatures: (1) *relational homophily* H_{rel}, extending graph homophily to RDBs [1]; (2) *task size*, the number of training samples; (3) *feature strength*, the performance ratio between GNN with features and GNN without features, indicating how much tabular signal dominates; and (4) *relation strength*, the performance ratio between 2-hop DFS and 1-hop DFS, measuring the usefulness of relational context. Task signatures help us understand the correlation between task properties and the performance gap associated with design choices.

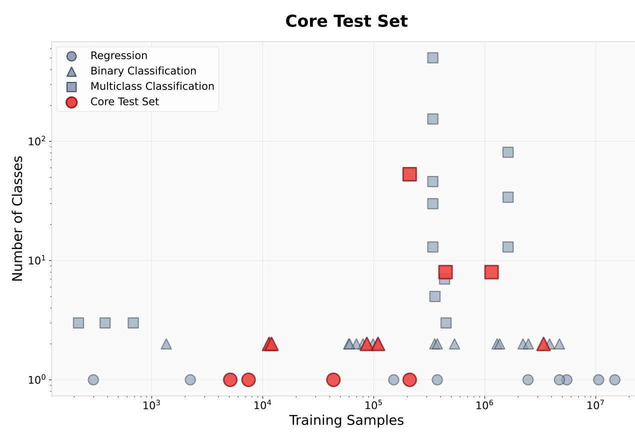


Figure 3: Distribution of our core evaluation set across task types and sizes. The selected 12 tasks span binary and multi-class classification and regression, with varying dataset scales and the ability to differentiate RFM models.

4.2 Models and Modular Building Blocks

Beyond data, a key challenge is that existing RFM codebases are fragmented and difficult to use: Griffin [48] relies on a deprecated test version of DGL [47], while PyG-based frameworks [18, 19] are heavy and difficult to customize (for example, to support the hop-based encoding of RelGT [12]). To address this, we extend RT's Rust-implemented sampler [41] to support both relational and graph sampling, and reimplement Griffin and RT within a unified framework using a PyTorch Lightning [14] wrapper for automatic multi-GPU training. For DFS, we provide a more scalable implementation with automatic batching and SQL query optimization.

We further provide modular components that can be freely composed. Based on our sampling module, we implement three inductive encoders from Section 3.1: TabEncoder, MetadataEncoder, and CellEncoder. For backbones, we provide vanilla Transformer [45], Relational Transformer [41], Graph Transformer [12], Griffin [48], NBFNet [56], and GraphSAGE [22]. For inference heads, we implement transformer-based in-batch ICL, either a pre-trained one [24] or a training-from-scratch one.

Uplifting a task-specific model to an RFM. To demonstrate the flexibility of our modular design, we showcase how to transform a familiar task-specific model into an RFM. The transformation involves two main steps. First, make the model inductive by replacing transductive components with inductive ones. For example, for RDL-GNN from Robinson et al. [42], this requires replacing both the transductive ResNet feature encoder and heterogeneous GNN with an inductive feature encoder that views categorical columns as text columns, and an inductive GNN that uses type embeddings (e.g., text embedding of table names) and shares model weight across different relation types. Second, an inductive model is used as a feature extractor, treating its output as tabular data that can be fed into a tabular foundation model (TFM) such as TabPFN for online predictions. As shown in Figure 2, we illustrate this approach with RT + TabPFN, where a pre-trained RT serves as the encoder-backbone to generate relational representations for each entity, while a pre-trained TFM (e.g., TabPFN) treats these representations as a batch of tabular features and performs in-context inference. This hybrid architecture addresses a key limitation of RT: RT relies solely on graph-level ICL, where in-context labels must appear within the sampled subgraph—a requirement that fails on sparse RDBs or autocomplete tasks. By delegating inference to a TFM, RT+TFM can leverage support-query splits within each batch, enabling effective predictions even when few labeled neighbors exist in the relational context. We put more details on pre-training an RT or Griffin model across tasks with variable categories in Appendix G.2.

4.3 Evaluation Suite

The third component of RFMBench is a controlled evaluation suite. We adopt a *view-based* evaluation framework, where each view comprises N_{view} samples drawn from the original task's evaluation set. Views can be constructed randomly or through custom selection rules. One example selection rule groups test samples by *relational homophily*—a measure of label consistency across relational neighbors. Following Relatron [1], relational homophily quantifies whether connected entities tend to share similar labels: high homophily indicates that neighbors provide reinforcing signals for prediction, while low homophily (heterophily) means neighbor labels may conflict with the target. Selecting views with varying homophily enables us to tune the "difficulty" of the task, which may also shed light on different RFM's preference for specific subgroups. Since RFMs perform in-context learning, *context length* N_{context} is another critical evaluation dimension. Context length determines how much relational information and how many in-context examples are available during inference—RFMs with different architectures may scale differently as context grows. In total, each model is evaluated K times across different random seeds, context lengths, and view selection rules.

5 Experimental Results

Building upon RFMBench, we design a series of experiments to address the core question of this paper: which RFM is better suited for what kind of task, and why? First, we evaluate the overall performance of RFMs on the core evaluation suite (RQ1). Second, we combine evaluation results with core design dimensions and task signatures to understand which RFMs perform better and how task characteristics drive this (RQ2). Third, we move from cross-task to intra-task analysis, evaluating model performance across different view functions (RQ3). Finally, we examine pre-training design choices (RQ4).

5.1 RQ1: Overall Performance Comparison

Experimental setup. Our core evaluation suite comprises three task types: original RelBench-style tasks, autocomplete tasks, and many-class tasks (e.g., author-category with 53 classes). The candidate models span three categories: (1) DFS combined with TFMs (TabPFN, MITRA [54]), (2) a commercial RFM (KumoRFM), and (3) open-source RFMs (Griffin, RT), along with uplifted variants (RT+TabPFN, Griffin+TabPFN) and task-specific baselines trained from scratch. We also include RealMLP [25] as a non-foundational baseline to test whether full training data access benefits DFS-based approaches.

For DFS models, we select the hop count with the best validation performance (2–3 hops in most cases), using uni-directional DFS despite bidirectional DFS potentially improving performance (see Section 5.2) at the cost of significantly longer runtime. For KumoRFM, we use the official API with batch size 1000 and the "best" run mode. For RT, we adopt both checkpoints from the original paper and checkpoints pre-trained with our new designs and tasks. The original paper uses task-specific checkpoints, violating the foundation model paradigm; we instead use the "pretrain_rel-f1_driver-dnf" checkpoint as a unified backbone, noting that it is pre-trained on all RelBench tasks except rel-f1, creating overlap with the evaluation set (we name it RT (official)). For RT pre-trained ourselves (used in RT + TabPFN), we adopt a prototype-based loss on a unified set of 30 tasks. As an ablation study, we also pre-train RT (binary) with the original head and only binary classification, together with regression tasks. Griffin is pre-trained using the original loss function and head designed by Wang et al. [48]. Additional details are in Appendix G. Following Table 4, we highlight key observations:

(1) **Overall performance.** KumoRFM achieves the best overall performance, though its improvement over DFS+TabPFN is marginal. Among training-from-scratch (TFS) baselines, cell-level RT clearly outperforms other counterparts. It should be noted that although RT (TFS) can achieve good results, it needs lots of hyper-parameter tuning (at least 30 trials for each task), while foundation models like RT + TabPFN is based on default parameters.
(2) **Context quality over quantity.** A common belief is that in-context learning is bottlenecked by context length. However, DFS+RealMLP with access to the full training set rarely surpasses DFS+TabPFN, suggesting that context quality matters more than quantity.

Table 3: Cross-task analysis on design dimensions. ***Upper*****: effect of in-context label attention on user-badge (AUROC).** ***Lower*****: weight-sharing strategies on driver-dnf (AUROC).**

RT	RT w/o ICL labels	DFS+TabPFN	Bidir. DFS+TabPFN
0.885	0.837	0.846	0.863

	Griffin	RDL-GNN	RelGT	DFS+TabPFN	RT
Sharing	Backbone+text	None	Backbone+ID	Shared	Shared
Val	0.775	0.783	0.680	0.713	0.717
Test	0.745	0.741	0.759	0.769	0.769

(3) **Revisiting reported RT performance.** Ranjan et al. [41] report promising zero-shot results, but RT actually performs graph-level in-context learning rather than true zero-shot inference. Moreover, the reported results use task-specific checkpoints; switching to a unified backbone leads to noticeable degradation (see RT (official) vs. RT (binary)).

(4) **Benefits of integrating TFMs.** Adopting a TFM as the prediction head generally improves performance. RT+TabPFN significantly outperforms the pre-trained RT (official) and RT (binary), making it a viable model for practical applications. Griffin+TabPFN enables Griffin to conduct in-context learning, though its performance remains inferior to RT+TabPFN.

5.2 RQ2: Cross-Task Analysis of Design Dimensions

To understand *why* certain models outperform others, we analyze model performance along two core design dimensions from Section 3: **level** and **relational weight sharing**. We focus on models without RDB-level pre-training to isolate architectural effects, and filter to tasks with significant performance gaps (excluding study-outcome, user-visits, and avs). To characterize tasks, we define four task signatures: *relational homophily* [1], *relational strength* (2-hop vs. 1-hop DFS+TabPFN), *feature strength* (RDL-GNN with original vs. random features), and *label strength* (RT with vs. without in-context label attention). Details are in Appendix H. Key observations:

(1) **Cell-level models perform better when column–label correlations are strong.** Cell-level models capture finer-grained relational structure than row-level models, which compress entire rows into single embeddings. A motivating example is the two tasks from Seznam, which exhibit high feature strength, indicating a strong correlation between certain columns and prediction labels. Cell-level RT, which attends to in-context labels at the cell level, achieves strong performance on these tasks. However, this fine-grained modeling comes at the cost of higher computation than row-level alternatives.

(2) **Weight sharing trades expressiveness for generalization.** Weight-sharing models achieve better test performance relative to their validation scores compared to non-weight-sharing models (Table 3, lower). The reported performance reflects the best validation configuration from a hyperparameter sweep; the key observation is that weight-sharing models are more reliable at finding configurations that generalize well to the test set. Conversely, non-weight-sharing models are more expressive when relational signals are abundant and mixed. For example, on user-badge, which has the lowest relational homophily (−0.076), RDL-GNN outperforms DFS+TabPFN by a notable margin. However, this expressiveness gap can be mitigated by *including the task table in database schemas*, allowing models to attend to historical labels—analogous to the label embedding technique in Lin et al. [30]—thereby transforming feature-only smoothing into label-conditioned relational reasoning. For DFS, we can run bidirectional DFS to achieve a similar effect.

5.3 RQ3: Intra-Task Analysis Across View Functions

The cross-task analysis in RQ2 reveals how design dimensions correlate with performance across tasks. Practitioners also care about performance on specific subgroups within a task—for example, predicting churn for specific users. We thus introduce view functions (Section 4) to analyze group-level behaviors, which also serve to verify whether the design-dimension insights from RQ2 hold at a finer granularity. Here, we consider two view functions: context length, applied to the context set, and relational homophily, applied to the query set.

Experimental setup. For the context length view function, we vary the size of the task table across six levels: 64, 128, 256, 512, 1,024, and 10,000. Note that there are two notions of context length in RFMs: (1) the size of the task table from which context samples are drawn, and (2) the number of context samples fed into the model at each inference call. Here, we control the former, which upper-bounds the latter. For the relational homophily view function, we partition the query set into 5 groups based on sample relational homophily and compute group-level metrics within each group. As shown in Figure 4, we highlight the main observations:

(1) **KumoRFM and RDL-GNN benefits more from additional context.** Compared to DFS+TabPFN, KumoRFM and RDL-GNN tend to underperform when the context length is small, but exhibit steeper performance gains as more context is provided (Figure 4, left, top row). This is likely attributable to their weight-sharing mechanism, which requires a sufficient amount of context to be fully leveraged.

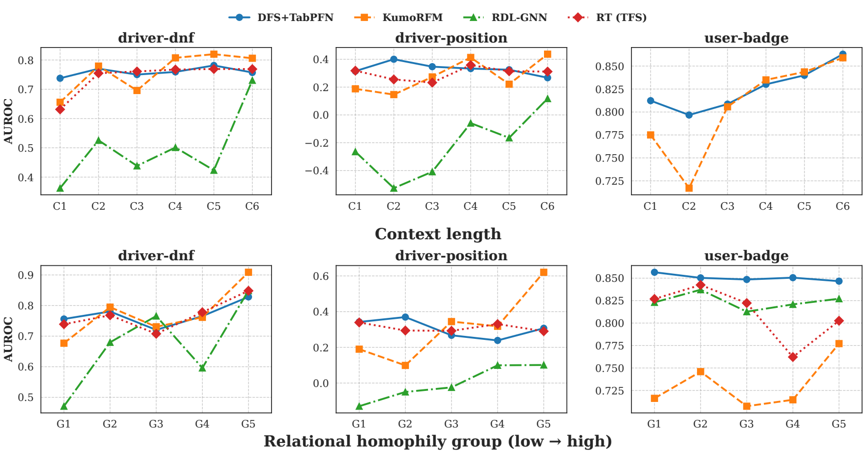
(2) **Relational homophily drives KumoRFM's advantage.** DFS exhibits relatively stable performance across homophily subgroups, whereas KumoRFM achieves notably stronger results in high-homophily groups (Figure 4, left, bottom row). Furthermore, comparing group-level metrics to the global-level metric—for example, on user-badge—reveals that group-level metrics are substantially lower. This suggests that KumoRFM's prediction mechanism heavily relies on *cross-group* discrimination for classification tasks: much of its global performance stems from separating groups with different label distributions rather than distinguishing samples within homogeneous groups.

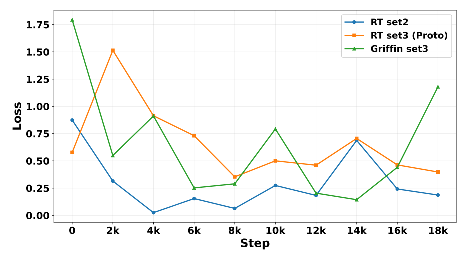
5.4 RQ4: Design Choices in Pre-training RFM

We then turn to the pre-training aspect and examine design choices in pre-training RFMs. Revisiting Table 4 and Fey et al. [17], we observe that (1) different choices of model architecture and pre-training data affect pre-training effectiveness, and (2) for RT and KumoRFM, certain tasks exhibit better performance than the same

Table 4: Results across 12 tasks spanning 7 databases. NS denotes unsupported configurations; Unstable indicates the method fails to produce meaningful results. RT does not support multiclass tasks. DFS+MITRA is limited to at most 10 classes. Griffin (TFS) and RelGT (TFS) require prohibitive resources (>1TB intermediate storage) for the avs database using the original code; the pre-trained Griffin uses the RFMBench version which avoids this issue. We rank methods per task using competition ranking, treating two methods as tied if their uncertainty intervals overlap. Methods marked NS/Unstable are always assigned the worst rank for that task.

	RelBench-style Entity prediction								*Autocomplete*			*Entity (Manyclass)*	
Task	driver-d	driver-p	study-o	study-a	user-b	author-p	user-v	ad-ctr	charge	repeater	prepay	author-c	Avg Rank
Database	f1	f1	trial	trial	stack	arxiv	avito	avito	seznam	avs	seznam	arxiv	
Metric	AUROC	R²	AUROC	R²	AUROC	R²	AUROC	R²	Acc	AUROC	Acc	Macro F1	
Foundation Models													
DFS+TabPFN	0.769 ± 0.000	0.307 ± 0.000	**0.722 ± 0.000**	0.350 ± 0.000	0.846 ± 0.011	**0.275 ± 0.000**	0.657 ± 0.006	0.114 ± 0.000	0.736 ± 0.004	0.549 ± 0.006	0.792 ± 0.003	0.317 ± 0.005	3.33
DFS+MITRA	0.771 ± 0.000	0.308 ± 0.001	0.673 ± 0.003	Unstable	0.842 ± 0.007	0.086 ± 0.040	0.645 ± 0.006	0.131 ± 0.000	0.714 ± 0.005	0.541 ± 0.004	0.788 ± 0.005	NS	6.00
DFS+RealMLP	0.770 ± 0.000	0.316 ± 0.000	0.719 ± 0.000	0.189 ± 0.000	0.842 ± 0.013	-0.037 ± 0.073	0.636 ± 0.005	0.105 ± 0.000	0.726 ± 0.003	0.516 ± 0.002	0.765 ± 0.004	0.192 ± 0.004	5.08
RT (official)	0.793 ± 0.000	**0.465 ± 0.000**	0.540 ± 0.000	-0.072 ± 0.000	0.861 ± 0.000	0.070 ± 0.000	0.630 ± 0.000	0.002 ± 0.000	NS	0.475 ± 0.000	NS	NS	9.17
RT (binary)	0.748 ± 0.000	-0.112 ± 0.000	0.495 ± 0.000	-0.022 ± 0.000	0.726 ± 0.013	0.089 ± 0.009	0.554 ± 0.007	-0.796 ± 0.000	NS	0.504 ± 0.004	NS	NS	10.83
KumoRFM	**0.811 ± 0.000**	0.439 ± 0.000	0.700 ± 0.000	0.350 ± 0.000	0.856 ± 0.011	0.133 ± 0.009	0.632 ± 0.003	0.134 ± 0.000	**0.892 ± 0.003**	**0.593 ± 0.003**	0.674 ± 0.003	0.260 ± 0.006	**3.17**
RT (ours) + TabPFN	0.772 ± 0.000	0.273 ± 0.000	0.683 ± 0.007	0.214 ± 0.000	0.855 ± 0.015	0.234 ± 0.016	0.636 ± 0.007	-0.001 ± 0.000	0.838 ± 0.007	0.542 ± 0.000	**0.919 ± 0.005**	0.124 ± 0.006	4.50
Griffin + TabPFN	0.738 ± 0.000	0.304 ± 0.000	0.625 ± 0.000	0.132 ± 0.000	0.787 ± 0.009	-0.738 ± 0.025	0.627 ± 0.005	-0.038 ± 0.000	0.683 ± 0.003	0.508 ± 0.005	0.826 ± 0.002	0.010 ± 0.003	8.42
Training from Scratch													
RT (TFS)	0.769 ± 0.005	0.337 ± 0.000	0.686 ± 0.006	**0.413 ± 0.000**	0.885 ± 0.007	0.218 ± 0.008	0.650 ± 0.006	-0.015 ± 0.000	0.860 ± 0.000	0.559 ± 0.000	0.918 ± 0.000	**0.345 ± 0.006**	3.25
Griffin (TFS)	0.745 ± 0.005	0.299 ± 0.008	0.689 ± 0.006	0.112 ± 0.006	0.870 ± 0.007	0.229 ± 0.011	0.650 ± 0.008	0.059 ± 0.013	0.790 ± 0.005	NS	0.729 ± 0.005	0.024 ± 0.000	6.42
RDL-GNN (TFS)	0.741 ± 0.000	0.102 ± 0.000	0.693 ± 0.000	0.171 ± 0.000	**0.889 ± 0.007**	0.157 ± 0.010	0.655 ± 0.006	0.136 ± 0.000	0.678 ± 0.003	0.563 ± 0.005	0.651 ± 0.007	0.294 ± 0.006	5.50
RelGT (TFS)	0.759 ± 0.005	0.124 ± 0.008	0.686 ± 0.006	0.170 ± 0.006	0.863 ± 0.007	0.265 ± 0.009	**0.668 ± 0.008**	**0.184 ± 0.013**	0.712 ± 0.004	NS	0.718 ± 0.003	0.143 ± 0.005	5.83





Step	2k	4k	10k	18k	20k
driver-position (R^2)					
RT Set A	0.34	0.31	0.35	0.32	0.32
RT Set B (Proto)	0.27	0.23	0.27	0.33	0.32
Griffin Set B	0.15	0.26	0.30	0.30	0.30
user-badge (AUROC)					
RT Set A	0.85	0.85	0.85	0.84	0.84
RT Set B (Proto)	0.86	0.85	0.84	0.84	0.84
Griffin Set B	0.72	0.75	0.77	0.78	0.79

Figure 4: RQ3 and RQ4 results. *Left*: intra-task analysis via view functions on three tasks (driver-dnf, driver-position, user-badge). *Top row*: performance under varying context length. *Bottom row*: performance across relational homophily subgroups. No TFS result for user-badge because of label imbalance. $C1$ to $C6$ represent various context lengths ranging from 64 to 10000. $G1$ to $G5$ represent five groups of test samples based on relational homophily, with homophily increasing from small to large. *Right*: trend of average pre-training loss on sampled pre-training batches across various steps and downstream task performance; all models use TabPFN head.

backbone trained from scratch, indicating potential transferability. This goes beyond TabPFN-style pre-training, which primarily reduces prediction variance and the need for hyperparameter tuning [36]. We thus study two perspectives: the design of pre-training architecture and data, and the source of pre-training benefits.

Comparison of pre-training architectures and data. Pre-training an RFM requires training both the relational backbone and the (in context) inference head. There are three potential choices: (1) train both together end-to-end; (2) train the backbone with a head-free objective first, and then train the head on top of the pre-trained backbone; (3) pre-train the backbone and use an existing ICL head, like TabPFN. We start with the first two choices; however, for the first choice, keeping both relational and batch-level inference contexts greatly limits scalability and is not suitable for large-scale pre-training. For the second choice, we first pre-train an RT on rel-f1, then use the pre-trained backbone to further train a transformer head. However, performance is only around 50 auroc, which is nearly random guessing. Compared with the last choice, we find the last choice much more effective. To study backbone, data, and pre-training objective, we further consider the following configurations: (1) RT pre-trained on Set A (binary and regression tasks only) with its original readout head; (2) RT pre-trained on Set B (with an inductive multiclass head), using either a prototype-based loss or the original loss; and (3) Griffin pre-trained on Set B. Details of pre-training sets, configurations, and full results are provided in Appendix G.2. As shown in Figure 4, three observations emerge:

(1) **Lower training loss does not reliably predict downstream improvement.** Training loss reduction correlates with downstream performance gains only during the initial few steps; beyond that, downstream performance plateaus and fluctuates regardless of continued loss decrease.

Table 5: RT performance on rel-f1 tasks when pre-trained on individual source databases. Bold indicates best per task.

Source DB	E-commerce	rel-hm	rel-stack	rel-avito	rel-event
driver-dnf (AUC)	0.776	0.756	0.610	**0.757**	**0.765**
driver-top3 (AUC)	0.810	0.736	**0.873**	**0.850**	0.844
driver-position (R^2)	0.315	**0.467**	−0.236	0.136	0.330

(2) **RT with prototype-based loss achieves best performance.** Among the pre-training configurations, RT pre-trained with the prototype-based loss consistently outperforms alternatives. Furthermore, RT generally achieves better downstream performance than Griffin when both are combined with TabPFN as the prediction head.

(3) **TabPFN head reduces checkpoint sensitivity.** Different tasks may still favor different checkpoints, but with the TabPFN head, the performance gap across checkpoints becomes much smaller compared to using the original RT inference head.

Source of pre-training transfer. We observe that transfer occurs for RT and KumoRFM specifically on the rel-f1 database. To investigate, we use RT as an anchor model and adopt a pre-train-on-one-test-on-another strategy: pre-training on individual RelBench source databases and evaluating on rel-f1 tasks. Here, we use the original RT architecture.

As shown in Table 5, certain single-database sources yield transfer on specific tasks (e.g., rel-stack → driver-top3, rel-hm → driver-position), confirming that structurally related pre-training data can provide meaningful transfer. However, such transfer is highly task-specific: improving one downstream task often comes at the cost of substantial degradation on others, even within the same database. This explains why, in practice, RT still relies on task-specific checkpoint selection rather than a single unified model. One indication of this experiment is that purely zero-shot transferability for RFM may not be feasible, since different tasks require different relational inductive biases. To achieve positive transfer for most tasks, we probably need to adapt the backbone model based on task-related data.

6 Discussions

We further discuss related aspects not covered in the main text.

The Mystery of Fine-tuning. KumoRFM exhibits two notable properties: (1) online prediction on par with training-from-scratch models, and (2) fine-tuning that substantially outperforms training-from-scratch on certain tasks. Our analysis explains the first property through RDB-level pre-training combined with dual-stage ICL, but the mechanism behind the second remains opaque due to undisclosed technical details. For link prediction, KumoRFM ensembles a shallow embedding module during fine-tuning, similar to ContextGNN [53], yet it is unclear how to replicate these gains for node-level prediction tasks. For example, on rel-f1, it achieves an AUC near 1.0 after fine-tuning, surpassing other models by a large margin. We experiment with fine-tuning pre-trained RT and Griffin using multiple strategies—vanilla fine-tuning and LoRA, with backbone weights either frozen or trainable—but observe no improvements over training-from-scratch or ICL baselines, consistent with Ranjan et al. [41]. This raises an open question: under what conditions can fine-tuning RFMs yield gains, and how should such strategies be designed?

RFM for Link Prediction Tasks. KumoRFM also reports results on link prediction tasks, but its fine-tuning approach simply adopts ContextGNN [53] and achieves comparable performance. Moreover, unlike entity-level prediction, in-context learning for link prediction still lags behind training-from-scratch baselines. Given this limited effectiveness and the orthogonal technical requirements compared to entity-level RFMs, we defer link prediction to future iterations of RFMBench.

7 Core Related Works

We discuss the most closely related work here; a broader survey is deferred to Appendix E. **Relational foundation models** are still in an early exploration stage, with varying definitions and objectives across works. Griffin [48] follows a traditional pretrain-finetune paradigm; RT [41] investigates task-specific transfer; Google's graph foundation model [20] targets zero-shot generalization but is limited to specialized tasks such as anomaly detection; and several efforts explore LLM-based approaches for RDB prediction [50, 51]. KumoRFM [17] demonstrates strong online prediction performance and provides a more unified RFM definition akin to tabular foundation models like TabPFN [24]; RFMBench follows this direction by emphasizing in-context learning and tuning-free evaluation as core RFM capabilities. **Benchmarking ML on RDBs.** The CTU Prague Repository [35] is the first benchmark for this setting, but many of its tasks are either too easy or suffer from leakage. RelBench [42] and 4DBInfer [46] try to address these issues by designing high-quality tasks: RelBench introduces an SQL-based task definition framework, and Redelex [37] provides utilities for converting CTU datasets into the RelBench format. However, none of these benchmarks is specifically designed to evaluate foundation model capabilities. Related efforts such as AutoG [9] and RDB2G-Bench [10] focus on graph construction strategies rather than RFM evaluation.

8 Conclusion

We introduce RFMBench, an open-source benchmark for developing and evaluating Relational Foundation Models, featuring a design-space taxonomy, a modular model library with dual-stage in-context learning, and a view-based evaluation framework for principled analysis. Empirical exploration in this paper inspires us to explore several future directions: 1. better pre-training data and objective design, especially synthetic relational data, which is essential to improve model performance and enhance cross-task stability, 2. better RFM architecture design, especially the combination of relation-level representation generation and batch-level inference, which is essential to build a scalable RFM.

References

[1] Anonymous. 2026. Relatron: Automating Relational Machine Learning over Relational Databases. In *Submitted to The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=59avbH4HnU under review.

[2] Maya Bechler-Speicher, Yoel Gottlieb, Andrey Isakov, David Abensur, Ami Tavory, Daniel Haimovich, Ido Guy, and Udi Weinsberg. 2026. Billion-Scale Graph Foundation Models. *arXiv preprint arXiv:2602.04768* (2026).

[3] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon,

Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the Opportunities and Risks of Foundation Models. *ArXiv* (2021). https://crfm.stanford.edu/assets/report.pdf

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[5] Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. 2025. RelGNN: Composite Message Passing for Relational Deep Learning. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=XXh3zmw2Uy

[6] Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2024. Exploring the potential of large language models (llms) in learning on graphs. *ACM SIGKDD Explorations Newsletter* 25, 2 (2024), 42–61.

[7] Zhikai Chen, Haitao Mao, Jingzhe Liu, Yu Song, Bingheng Li, Wei Jin, Bahare Fatemi, Anton Tsitsulin, Bryan Perozzi, Hui Liu, and Jiliang Tang. 2024. Text-space Graph Foundation Models: Comprehensive Benchmarks and New Insights. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=Q2sDuwtutB

[8] Zhikai Chen, Haitao Mao, Hongzhi Wen, Haoyu Han, Wei Jin, Haiyang Zhang, Hui Liu, and Jiliang Tang. 2024. Label-free Node Classification on Graphs with Large Language Models (LLMs). In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=hESD2NJFg8

[9] Zhikai Chen, Han Xie, Jian Zhang, Xiang song, Jiliang Tang, Huzefa Rangwala, and George Karypis. 2025. AutoG: Towards automatic graph construction from tabular data. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=hovDbX4Gh6

[10] Dongwon Choi, Sunwoo Kim, Juyeon Kim, Kyungho Kim, Geon Lee, Shinhwan Kang, Myunghwan Kim, and Kijung Shin. 2025. RDB2G-Bench: A Comprehensive Benchmark for Automatic Graph Modeling of Relational Databases. In *NeurIPS*.

[11] Code17 GmbH. 2024. New Algorithms for Relational Learning. getML Blog. https://getml.com/latest/blog/new-algorithms-relational-learning/ Accessed: 2025-12-08.

[12] Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. 2025. Relational Graph Transformer. *arXiv preprint arXiv:2505.10960* (2025).

[13] Dmitry Eremeev, Gleb Bazhenov, Oleg Platonov, Artem Babenko, and Liudmila Prokhorenkova. 2025. Turning tabular foundation models into graph foundation models. *arXiv preprint arXiv:2508.20906* (2025).

[14] William Falcon and The PyTorch Lightning team. 2019. *PyTorch Lightning*. doi:10.5281/zenodo.3828935

[15] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a Graph: Encoding Graphs for Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=IuXR1CCrSi

[16] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. 2024. Position: Relational Deep Learning - Graph Representation Learning on Relational Databases. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 13592–13607. https://proceedings.mlr.press/v235/fey24a.html

[17] Matthias Fey, Vid Kocijan, Federico Lopez, Jan Eric Lenssen, and Jure Leskovec. [n. d.]. KumoRFM: A Foundation Model for In-Context Learning on Relational Data. ([n. d.]).

[18] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[19] Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanovič, Ramona Bendias, Barghi Alexandria, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan E. Lenssen, and Jure Leskovec. 2025. PyG 2.0: Scalable Learning on Real World Graphs. In *Temporal Graph Learning Workshop @ KDD*.

[20] Michael Galkin and Pramod Doguparty. 2025. Graph Foundation Models for Relational Data. Google Research Blog. https://research.google/blog/graph-foundation-models-for-relational-data/

[21] Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. 2024. Towards Foundation Models for Knowledge Graph Reasoning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=jVEoydFOl9

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[23] Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. Harnessing Explanations: LLM-to-LM Interpreter for Enhanced Text-Attributed Graph Representation Learning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=RXFVcynVe1

[24] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. 2025. Accurate predictions on small data with a tabular foundation model. *Nature* (09 01 2025). doi:10.1038/s41586-024-08328-6

[25] David Holzmüller, Léo Grinsztajn, and Ingo Steinwart. 2024. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. *Advances in Neural Information Processing Systems* 37 (2024), 26577–26658.

[26] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.

[27] Lecheng Kong, Jiarui Feng, Hao Liu, Chengsong Huang, Jiaxin Huang, Yixin Chen, and Muhan Zhang. 2025. GOFA: A Generative One-For-All Model for Joint Graph Language Modeling. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=mIjblC9hfm

[28] Divyansha Lachi, Mahmoud Mohammadi, Joe Meyer, Vinam Arora, Shivashriganesh P Mahato, Tom Palczewski, and Eva L Dyer. [n. d.]. Learning from Multi-Table Relational Data with the Relational Graph Perceiver. In *EurIPS 2025 Workshop: AI for Tabular Data*.

[29] Ning Li, Kounianhua Du, Han Zhang, Quan Gan, Minjie Wang, David Wipf, and Weinan Zhang. 2025. Synthesize, Retrieve, and Propagate: A Unified Predictive Modeling Framework for Relational Databases. *arXiv preprint arXiv:2508.08327* (2025).

[30] Junhong Lin, Xiaojie Guo, Shuaicheng Zhang, Yada Zhu, and Julian Shun. 2025. When Heterophily Meets Heterogeneity: Challenges and a New Large-Scale Graph Benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 5607–5618. doi:10.1145/3711896.3737421

[31] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2024. One For All: Towards Training One Graph Model For All Classification Tasks. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=4IT2pgc9v6

[32] Junwei Ma, Nour Shaheen, Alex Labach, Amine Mhedhbi, Frank Hutter, Anthony L Caterini, and Valentin Thomas. 2025. Generalization Can Emerge in Tabular Foundation Models From a Single Table. *arXiv preprint arXiv:2511.09665* (2025).

[33] Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Alex Labach, Jesse C. Cresswell, Keyvan Golestan, Guangwei Yu, Anthony L. Caterini, and Maksims Volkovs. 2025. TabDPT: Scaling Tabular Foundation Models on Real Data. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=pIZxEOZCId

[34] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. 2024. Position: Graph Foundation Models Are Already Here. In *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=Edz0QXKKAo

[35] Jan Motl and Oliver Schulte. 2024. The CTU Prague Relational Learning Repository. arXiv:1511.03086 [cs.LG] https://arxiv.org/abs/1511.03086

[36] Thomas Nagler. 2023. Statistical foundations of prior-data fitted networks. In *International Conference on Machine Learning*. PMLR, 25660–25676.

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

[38] Jakub Peleška and Gustav Šír. 2025. Tabular Transformers Meet Relational Databases. *ACM Trans. Intell. Syst. Technol.* 16, 5, Article 115 (Sept. 2025), 24 pages. doi:10.1145/3749991

[39] Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862* (2024).

[40] Josh Przybylko, Matthias Fey, Blaž Stojanovič, Sally Liu, and Jure Leskovec. 2025. Bringing AI Predictions to Agents - Announcing KumoRFM Support for Model Context Protocol (MCP). Kumo AI Blog. https://kumo.ai/company/news/kumorfm-mcp/

[41] Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. 2025. Relational Transformer: Toward Zero-Shot Foundation Models for Relational Data. *arXiv preprint arXiv:2510.06377* (2025).

[42] Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. 2024. RelBench: A Benchmark for Deep Learning on Relational Databases. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=WEFxOm3Aez

[43] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[44] Quang Truong, Zhikai Chen, Mingxuan Ju, Tong Zhao, Neil Shah, and Jiliang Tang. 2025. A Pre-training Framework for Relational Data with Information-theoretic Principles. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=xNUNxRj2vJ

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[46] Minjie Wang, Quan Gan, David Wipf, Zheng Zhang, Christos Faloutsos, Weinan Zhang, Muhan Zhang, Zhenkun Cai, Jiahang Li, Zunyao Mao, Yakun Song, Jianheng Tang, Yanlin Zhang, Guang Yang, Chuan Lei, Xiao Qin, Ning Li, Han Zhang, Yanbo Wang, and Zizhao Zhang. 2024. 4DBInfer: A 4D Benchmarking Toolbox for Graph-Centric Predictive Modeling on RDBs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=YXXmIHJQBN

[47] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[48] Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. 2025. Griffin: Towards a Graph-Centric Relational Database Foundation Model. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=TxeCxVb3cL

[49] Zehong Wang, Zheyuan Liu, Tianyi Ma, Jiazheng Li, Zheyuan Zhang, Xingbo Fu, Yiyang Li, Zhengqing Yuan, Wei Song, Yijun Ma, et al. 2025. Graph Foundation Models: A Comprehensive Survey. *arXiv preprint arXiv:2505.15116* (2025).

[50] Fang Wu, Vijay Prakash Dwivedi, and Jure Leskovec. 2025. Large Language Models are Good Relational Learners. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 7835–7854. doi:10.18653/v1/2025.acl-long.386

[51] Marek Wydmuch, Łukasz Borchmann, and Filip Graliński. 2024. Tackling prediction tasks in relational databases with LLMs. *arXiv preprint arXiv:2411.11829* (2024).

[52] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021.

[53] Yiwen Yuan, Zecheng Zhang, Xinwei He, Akihiro Nitta, Weihua Hu, Manan Shah, Blaž Stojanovič, Shenyang Huang, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. 2025. ContextGNN: Beyond Two-Tower Recommendation Systems. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=nzOD1we8Z4

[54] Xiyuan Zhang, Danielle C. Maddix, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael W. Mahoney, Cuixiong Hu, Huzefa Rangwala, George Karypis, and Bernie Wang. 2025. Mitra: Mixed Synthetic Priors for Enhancing Tabular Foundation Models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=t8YRsWY6HM

[55] Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, et al. 2025. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *arXiv preprint arXiv:2510.23587* (2025).

[56] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in neural information processing systems* 34 (2021), 29476–29490.

A Reproducibility Statement

Due to submission policy, we are unable to include source code at this stage. To facilitate reproducibility, we provide detailed descriptions of all model configurations, training procedures, and evaluation protocols in the following appendix sections. We will release the full codebase after it passes internal review. Appendix D provides an overview of the RFMBench framework design. Appendix G details model configurations, hyperparameters, and data processing for all baselines and evaluated systems. Appendix G.2 describes the pre-training sets, checkpoint-level training procedures, inference protocol, and full results for our own pre-trained models. Appendix H formally defines the task features used for fine-grained analysis.

B Impact Statement

RFMBench aims to strengthen the methodological foundations of relational foundation model research in three ways. First, it promotes fair comparison across studies by standardizing task formats, evaluation metrics, and data splits that are currently defined ad hoc in individual papers. Second, it moves beyond single-number leaderboard rankings by providing controlled evaluation views that reveal how model performance varies with structural properties such as relational homophily, feature informativeness, and training set scale. Third, it lowers the engineering barrier for studying RFMs by offering shared abstractions for context construction, feature extraction, and inference, allowing researchers to focus on modeling rather than pipeline reimplementation. On the practical side, the benchmark can inform both academic investigation—for example, clarifying when relational inductive biases are necessary versus when DFS combined with tabular foundation models suffices—and industrial prototyping, such as assessing the viability of tuning-free prediction on previously unseen database schemas.

C Ethics Statement

RFMBench is a benchmark suite for predictive modeling over relational databases. It aggregates tasks and datasets from existing public sources—including RelBench, the CTU Prague Relational Learning Repository, 4DBInfer, and selected Kaggle competitions—rather than collecting new data from human subjects. Because the benchmark builds on third-party datasets, their respective licenses and terms of use govern how data may be accessed and redistributed. To respect these terms, RFMBench does not redistribute raw data; instead, it provides download scripts and manifests that fetch datasets from the original hosts and generate pre-processed versions locally. A per-dataset summary documenting source locations, licenses, access requirements, and redistribution status is provided in Appendix F (Table 6).

D Overview of RFMBench framework design

The framework is organized around three core modules. (1) Data Interface: RFMBench provides three complementary data access paradigms — a unified relational/graph sampling module built on Rustler [41] (a high-performance Rust engine extending RT's random-walk sampler with Polars, PyO3, and memory-mapped I/O) that supports temporal-aware BFS walks and graph-level sampling; a DFS-based feature engineering interface with optimized query processing and bidirectional aggregation support; and a legacy PyG/DGL-compatible interface for training GNN baselines. (2) Model Module: On the modeling side, RFMBench features a sparse encoder tailored to the new sampling module, a broad suite of backbone architectures including Relational Transformer (RT), relational graph transformers, and various GNN variants (e.g., GraphSAGE), as well as flexible inference heads spanning ICL-based transformers in both pre-trained and non-pretrained configurations. (3) Evaluation Module: RFMBench introduces a view-based evaluation protocol that decouples data context from model assessment through configurable view selection strategies (e.g., homophily, degree, temporal), complemented by task signatures that characterize dataset properties such as sparsity, class balance, and temporal dynamics to enable fine-grained analysis of model strengths across heterogeneous relational tasks.

E Related Works

Relational Deep Learning. Standard benchmarks such as RelBench [42] and DBInfer [46] formalize predictive tasks over multi-table relational databases and establish GNN-based pipelines as the dominant paradigm: tabular features are projected into a shared latent space via type-specific encoders, then aggregated through temporal-aware message passing along primary key–foreign key edges. Several lines of work extend this basic recipe. Chen et al. [5] enrich the message passing function with composite (higher-order) routes that capture multi-hop relational patterns in a single aggregation step. Lachi et al. [28] replace standard GNN layers with a Perceiver-style cross-attention bottleneck, compressing heterogeneous node and edge representations into a fixed set of latent tokens while incorporating a temporal subgraph sampler for long-range context. Li et al. [29] augment the GNN pipeline with a retrieval module that discovers structurally similar rows across tables via BM25, capturing implicit dependencies beyond explicit foreign key links. Yuan et al. [53] adapts GNN-based relational learning specifically for recommendation by combining an ID-based GNN [56] with shallow embedding-based retrieval through a path-based routing mechanism. Orthogonal to GNN-based designs, transformer architectures have also been explored for relational data. Peleška and Šír [38] model relational databases as nested hypergraphs and apply tabular transformer encoders at both the attribute level and the cross-table level, preserving fine-grained column representations that standard GNNs collapse into row-level embeddings. Dwivedi et al. [12] propose a graph transformer backbone operating directly over relational graph structures, though it currently demands significantly more compute than GNN methods for comparable gains. Finally, Wu et al. [50] and Wydmuch et al. [51] investigate large language models for relational prediction; however, these methods exhibit a substantially lower performance-to-resource ratio and may introduce confounding factors from potential overlap between pre-training corpora and downstream evaluation data. Moving beyond task-specific training, several works aim to build foundation models for relational data. Tabular foundation models such as TabPFN [24] demonstrate that pre-trained in-context learners can match or exceed traditional supervised methods on single-table tasks, motivating their extension to the relational setting. RT [41] tokenizes individual database cells and applies a relational attention mechanism structured along column, row, and foreign key axes, enabling graph-level in context learning across unseen schemas. Griffin [48] adopts a cross-table attention module that mimics deep feature synthesis aggregation, but its performance does not consistently surpass task-specific GNN baselines. KumoRFM [17] combines a graph transformer backbone with in-context learning capabilities and achieves strong predictive performance, though its architecture and training procedure remain undisclosed. Beyond architecture design, how to pre-train effectively on heterogeneous relational tasks is itself an open problem. Truong et al. [44] provide an information-theoretic analysis showing that task-aware pre-training objectives—which explicitly account for the diversity of possible relational prediction tasks—retain more relevant signals than task-agnostic self-supervised losses, and propose a pre-training framework based on set-based aggregation over schema traversal graphs that consistently improves downstream transfer within GNN-based pipelines.

Graph Foundation Models. Graph foundation models [34, 49] seek to pre-train a single model across heterogeneous graph domains so that it transfers to unseen graph types without task-specific retraining. One-For-All [31] unifies diverse graph domains by converting node features into textual descriptions and training a shared GNN over the resulting text-attributed graphs, though Chen et al. [7] show through systematic benchmarking that text-space unification alone does not yield reliable cross-domain transferability even under co-training. GOFA [27] interleaves GNN layers with a frozen LLM backbone to handle both predictive and generative graph tasks jointly, but finds that multi-task training can degrade predictive accuracy relative to specialized models. In the knowledge graph setting, ULTRA [21] achieves strong zero-shot link prediction on unseen knowledge graphs by learning transferable relational representations through a dual-level NBFNet conditioned on local graph structure; Knowledge graph shares similar formats compared to the metadata of relational databases, however, the underlying data of relational databases share much more complex structures, so directly applying such foundation model can't achieve good performance of RDB link prediction tasks. More recently, Bechler-Speicher et al. [2] demonstrate that transformer-based graph foundation models can scale to billions of nodes for classification and link-level tasks, though their generalizability across structurally diverse domains remains to be further investigated. A separate line of work investigates LLMs as graph learners—either by encoding graph structure as text for direct LLM reasoning [15, 39], using LLM-generated annotations to supervise GNNs [8], distilling LLM explanations into smaller encoders [23], or studying hybrid LLM–GNN pipelines [6]. These approaches primarily target text-attributed graphs and tend to lag behind graph-native architectures on structural tasks, with performance often depending on overlap between LLM pre-training knowledge and the downstream domain.

We refer interested readers to Wang et al. [49] for a comprehensive survey of graph foundation models.

F Dataset Card

Table 6 summarizes the provenance, licensing, and access conditions for every dataset included in RFMBench. Datasets are drawn from three repository families: RelBench, 4DBInfer, and the CTU Prague Relational Learning Repository. Access ranges from fully open to credentialed (e.g., MIMIC-III) or competition-gated (e.g., Kaggle). Users should verify that their intended use complies with the terms listed before downloading or redistributing any data.

Selection of core evaluation set and pre-training data. We select a small group of representative evaluation tasks for quickly evaluating RFM. The selection process is as follows: 1. First, we refer to Peleška and Šír [37] to identify CTU tasks where different baselines may present over 50% performance gap, or achieving near-perfect accuracy. These signals indicate the corresponding task may have label leakage or other problems. We find that in general, CTU tasks are of low quality and not suitable for evaluation. RelBench and DBInfer have already extracted most high-quality tasks from CTU. We thus select some tasks from them only for pre-training. 2. For Relbench and DBInfer, we generally observe no clear data problem. To enhance the diversity of evaluation, we first select all available multiclass tasks from them considering their limited availability. Then, we assign rel-event to pre-training set since its test set is claimed to have leakage problem [41]. Our next step is to select some representative tasks and filter highly similar ones. Based on the Graphgym similarity [52] used in Relatron [1], we filter rel-hm, since user-churn share a high similarity with user-badge and user-engagement. We select user-badge in the core test set. Post-votes and item-sales are not adopted since nearly all models achieve similar performance on these tasks. Similarly, we select driver-dnf but not driver-top3, and user-visits but not user-clicks because of their similarity. Site-success is not used since we observe some models presenting numerical instability on this task. For DBInfer, we select repeater since other tasks are CTR-related problems, which is closer to recommendation scenarios.

G Model and Evaluation Details

This section provides additional details on the model and evaluation procedures used throughout our experiments.

G.1 Model and Evaluation Details

DFS+TFM Models. For DFS-based approaches (TabPFN, MITRA, RealMLP), we perform feature extraction using deep feature synthesis (DFS) with hop counts $h \in 2, 3$, selecting the hop count that achieves the best validation performance per task. In most cases, h=2 yields optimal performance. We extend the standard DFS with a bidirectional engine that allows table revisitation along relational paths (e.g., $A \to B \to A$). However, in practice, we typically use unidirectional DFS pipeline since bidirectional DFS is too slow. The aggregation primitives include max, min, mean, count, mode, sum, and std for numerical columns. Text columns are first encoded via text embeddings, then reduced to 3 dimensions using PCA before entering the DFS pipeline. A time cutoff is applied to feature preprocessing stage to make sure there's no leakage from future timestamps. TabPFN use up to 10,000 randomly sampled examples as context. For TabPFN, we use version v2 with 8 ensemble members; for tasks with more than 10 classes, a ManyClassClassifier wrapper decomposes the problem into multiple 10-class subproblems. MITRA is limited to at most 10 output classes due to its fixed architecture. RealMLP is fitted from scratch using pre-tuned default hyperparameters from pytabkit, with median imputation for missing values.

KumoRFM. We access KumoRFM through its official API. For all experiments, we use a batch size of 1000 and the "best" run mode, which allows the model to optimize its internal configurations for each task.

Griffin. We re-implement Griffin following the original architecture: a sparse feature encoder maps each cell value to a d-dimensional embedding using type-specific encoders (linear projections for numeric, text, datetime, and boolean columns), with text columns encoded via all-MiniLM-L12-v2 (384-dimensional). Cell embeddings within each row are aggregated via attention-weighted pooling to produce node representations. The backbone consists of 4 layers of Relational Message Passing Neural Networks (RMPNN), each performing two-level aggregation—mean within each (source, relation-type) group, then max across relation types—with gated residual connections.

RT. We follow the original Relational Transformer (RT) architecture: each cell in a relational sequence is encoded via type-specific linear projections (number, text, datetime, boolean), with text embeddings from all-MiniLM-L12-v2 (384-dimensional). Each block applies four structured masked attention operations in sequence—column, feature, neighbor, and full—followed by a feed-forward network, all with RMSNorm. Input sequences are constructed via BFS-based random walks with a sequence length of 1024 and a maximum BFS width of 256.

Training-from-Scratch Baselines. For training from scratch baselines, we adapt the settings from original papers Dwivedi et al. [12], Ranjan et al. [41], Robinson et al. [42], Wang et al. [48]'s configurations. For RDL-GNN, the only difference is that for entity prediction tasks, we include task table in the schema.

Data-related Details. For most tasks from the core evaluation set, we just follow the relbench setting. For Seznam autocomplete tasks (dbinfer-seznam/charge and dbinfer-seznam/prepay), we report results under history-aware setting. In history-aware setting, we keep historical `sluzba` values in the context to reflect realistic usage where past interactions are observable, and we ensure that the query target itself remains masked. This is the setting adopted by the original DBInfer paper. For RT+TabPFN, we additionally include an ICL-style summary of visible target-column occurrences in the context (excluding the masked query) to make sure it gets reasonable performance.

G.2 Our Pre-trained Models

Pre-training. For foundation models that we pre-train ourselves, we use the following two sets of tasks:

(1) Pretrain set A: E-commerce (3 binary cls, 3 reg); rel-hm (1 binary cls, 2 reg); rel-ratebeer (3 binary cls, 2 reg); rel-event (4 binary cls, 2 reg); dbinfer-outbrain-small (1 binary cls); dbinfer-retailrocket (1 binary cls).

Table 6: Dataset licensing and access summary.

Dataset	Source	Original Data Source	License	Access	Usage Restrictions
rel-stack	RelBench	Stack Exchange Data Dump	CC BY-SA 4.0	Open	Attribution, ShareAlike
rel-trial	RelBench	AACT (ClinicalTrials.gov)	Not specified	Open	Check AACT terms
rel-f1	RelBench	F1DB (Formula 1 racing data)	CC-BY-4.0	Open	Attribution required
rel-hm	RelBench	H&M Kaggle Challenge	Kaggle License	Restricted	Non-commercial & academic only
rel-event	RelBench	Event Recommendation Dataset	Restricted	Permission	Upon authors' agreement
rel-avito	RelBench	Avito Context Ad Clicks (Kaggle)	Kaggle Competition	Kaggle	Per Kaggle terms
rel-arxiv	RelBench	arXiv	CC0 1.0 (metadata)	Open	Link back to arXiv; no article redistribution
rel-mimic	RelBench	MIMIC-III (PhysioNet)	PhysioNet CHDL 1.5.0	Credentialed	CITI training + DUA; research only
rel-salt	RelBench	SAP SALT (Hugging Face)	CC-BY-NC-SA-4.0	Open	Non-commercial; Attribution; ShareAlike
AVS	4DBInfer	Acquire Valued Shoppers (Kaggle)	Kaggle Competition	Kaggle	Per Kaggle terms
Outbrain	4DBInfer	Outbrain Click Prediction (Kaggle)	Kaggle Competition	Kaggle	Per Kaggle terms
Diginetica	4DBInfer	CIKM Cup 2016	Competition License	Registration	Via cikm2016.cs.iupui.edu
RetailRocket	4DBInfer	Kaggle	Kaggle Dataset	Kaggle	Per Kaggle terms
MAG	4DBInfer	Microsoft Academic Graph	ODC-BY	Open	Attribution required
Seznam	4DBInfer	CTU Prague Repository	Academic/Research	Open	Research purposes
StackExchange	4DBInfer	Stack Exchange Data Explorer	CC BY-SA 4.0	Open	Attribution, ShareAlike
accidents	CTU Prague	National Inst. Statistics Belgium	Public Data	Open	Research use
legalActs	CTU Prague	datahub.io (Bulgarian court decisions)	Public Open Data	Open	Research use
Dallas	CTU Prague	Dallas Police Department	US Gov. Public Data	Open	Research use
NCAA	CTU Prague	Kaggle (2015 NCAA Basketball)	Kaggle License	Kaggle	Per Kaggle terms

(2) Pretrain set B: same as set A plus ctu-legalacts (1 multiclass); rel-salt (5 multiclass); ctu-accidents (1 multiclass); ctu-financial (1 multiclass).

The main difference between the two sets is the addition of multiclass tasks in set B. To pre-train RT on pretrain set A, we can directly use the original binary-type and number-type readouts for classification and regression tasks. For pretrain set B, we need to augment RT with an inductive multiclass readout for the multiclass tasks. Here, we adopt a learnable embedding with 60 MAX_CLASSES for multiclass tasks. For Griffin, we directly adopt the original text-embedding-based readouts for classification tasks and the original float-type decoder for regression tasks. Detailed introduction of the pre-training procedure is provided as follows.

(1) ckpt1 (Set A): RT backbone with original RT readout, i.e., native boolean/number decoding behavior. The backbone uses $d_{\text{model}} = 512$, 6 transformer blocks, 8 attention heads, and $d_{\text{ff}} = 2048$. Pre-training runs for 20k steps with learning rate 10^{-4}, weight decay 0.01, batch size 16, and sequence length 1024.

(2) ckpt2 (Set A): same pre-trained backbone as ckpt1 (identical architecture and pre-training), but evaluated with TabPFN as the downstream inference head.

(3) ckpt3 (Set B): pre-trains an RT backbone on Set B using an episodic, prototype-based objective designed to align the representation space for label-efficient transfer across heterogeneous relational tasks. Rather than relying on a fixed task-specific classifier during pre-training, it constructs few-shot episodes in which each task is presented as an N-way classification problem: support examples for each class are embedded, class prototypes are formed by aggregating support representations, and query examples are trained to match the correct prototype. This episodic formulation encourages the backbone to learn features that are directly usable under downstream few-shot-style heads (including TabPFN), while reducing reliance on fragile, task-local output layers. To further mitigate leakage and better reflect realistic inductive settings, episodes are built with temporal ordering when timestamps exist and the target column is removed from the context so that prediction must be driven by relational evidence rather than direct label exposure. In addition, the pre-training uses a dual-head design to jointly cover regression and classification: continuous targets are learned via a regression branch, while discretized/binned supervision provides complementary prototype-based signals. The RT backbone shares the same architecture as ckpt1. Pre-training runs for 20k steps with learning rate 3×10^{-4}, weight decay 0.1, warmup steps 2000, batch size 32, and sequence length 1024. We use 64 maximum classes with 8 support and 8 query examples per class, enforce temporally ordered episodes with leakage control, and balance the dual-head losses with equal weighting.

(4) ckpt5 (Set B): Griffin backbone pre-trained as a shared relational encoder, then evaluated via TabPFN over extracted row embeddings. The Griffin model uses $d_{\text{model}} = 128$, 3 layers, dropout 0.1, and bidirectional mode. Pre-training runs for 20k steps with learning rate 3×10^{-5}, no weight decay, warmup steps 2000, batch size 16, and 100 maximum classes.

(5) ckpt6 (Set B): pre-trains an RT backbone on Set B with a readout design intended to better match the heterogeneity of relational tasks while keeping a single shared backbone. During pre-training, classification and regression are routed through different, specialized decoding paths: classification is handled by a GriffinHead-style mechanism that is naturally compatible with variable label spaces, while regression uses a dedicated float decoder. The RT backbone shares the same architecture as ckpt1. Pre-training runs for 20k steps with learning rate 10^{-4}, weight decay 0.01, batch size 16, sequence length 1024, and 60 maximum classes.

(6) ckpt7 (Set B): pre-trains an RT backbone on Set B using a unified classification interface that treats all classification tasks (binary and multiclass) through a single fixed-size class table. The core idea is to standardize the classification supervision into a shared K-way head with masking, where K is an upper bound on the number of classes observed during pre-training. Regression remains a separate numeric decoding path, so the model learns to support both discrete and continuous targets during the same pre-training run. The backbone architecture and training schedule match ckpt6. After pre-training, we discard the pre-training heads and use TabPFN on top of extracted representations for evaluation.

Inference. For all checkpoints except ckpt1 (which uses the original RT readout), we perform inference by first extracting pre-trained relational representations from the frozen encoder—RT produces target-cell hidden states, while Griffin yields row embeddings. These embeddings are then partitioned into support (training) and query (test) pools, with strict-mode controls applied for leakage-sensitive tasks. Finally, we fit TabPFN on the support embeddings and predict on the query embeddings, using the classifier variant for classification (with the ManyClass extension for large-class settings) and the regressor variant for regression. The resulting predictor is a two-stage composition: a frozen relational encoder for representation learning, followed by TabPFN for task-level decision mapping.

Full Results. Full checkpoint-level results are provided in Table 7. In general, we find that RT pre-trained with prototype-based loss and RT pre-trained with MAX_CLASSES head achieve the best performance; we choose the first one in the main text.

H Task Features

To enable principled analysis of model behavior across diverse relational tasks, we introduce a set of *task features* (or *task signatures*) that characterize the intrinsic properties of each task. We design task features to capture three complementary aspects of relational prediction tasks: *relational structure* (how informative are neighbor labels), *scale* (how much training data is available), and *signal decomposition* (relative contributions of features vs. relational context). Table 8 summarizes the computed signatures for our core evaluation set. Below, we formally define each task feature.

Relational Homophily (H_{rel}). Let a task be defined on a target entity table (node type) $\tau^\star$ with target column y. We view the relational database as a typed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \rho)$ and denote the set of *returning 2-hop metapaths* that start and end at $\tau^\star$ by

$$\mathcal{P}_2(\tau^\star) = \left\{ p = (r_1, r_2) : \tau^\star \xrightarrow{r_1} \cdot \xrightarrow{r_2} \tau^\star \right\}.$$

For each $p \in \mathcal{P}_2(\tau^\star)$, let $\mathcal{E}_p \subseteq \mathcal{V}_{\tau^\star} \times \mathcal{V}_{\tau^\star}$ be the induced directed edge multiset between task nodes following p. Let $\mathcal{L} \subseteq \mathcal{V}_{\tau^\star}$ be the labeled task nodes used for signature computation (in the repo, train+val are concatenated). These quantify *relational label homophily*: whether labels are more consistent across relational neighbors than what would be expected from degree-/class-prior effects alone. In the repo, for each $p \in \mathcal{P}_2(\tau^\star)$, a per-metapath *degree-adjusted homophily* score $h_{\text{adj}}(p)$ is computed and then aggregated across metapaths. For classification, each labeled node $u \in \mathcal{L}$ is represented by a (possibly soft) label vector $P_u \in \Delta^{C-1}$ (one-hot if labels are hard). Define the edge-level expected label agreement on p as

$$h_{\text{edge}}(p) = \frac{1}{|\mathcal{E}_p^{\text{lab}}|} \sum_{(u,v) \in \mathcal{E}_p^{\text{lab}}} \langle P_u, P_v \rangle, \quad \mathcal{E}_p^{\text{lab}} = \{(u,v) \in \mathcal{E}_p : u, v \in \mathcal{L}\}.$$

Let $\deg_p(v)$ be the number of valid incoming edges to v under p (counting only neighbors used in $\mathcal{E}_p^{\text{lab}}$), and define the degree-weighted class mass

$$m_c(p) = \sum_{v \in \mathcal{L}} P_{v,c} \deg_p(v), \qquad M(p) = |\mathcal{E}_p^{\text{lab}}| = \sum_{v \in \mathcal{L}} \deg_p(v).$$

The repo's degree-adjustment baseline is

$$\text{adjust}(p) = \sum_{c=1}^{C} \left(\frac{m_c(p)}{M(p)} \right)^2,$$

and the adjusted homophily is

$$h_{\text{adj}}(p) = \frac{h_{\text{edge}}(p) - \text{adjust}(p)}{1 - \text{adjust}(p) + \varepsilon},$$

with a small ε for numerical stability.

For regression tasks, we use a correlation-based homophily. Define the edge-level correlation on metapath p as

$$h_{\text{edge}}(p) = \text{Corr}\left(\{y_u\}_{(u,v) \in \mathcal{E}_p^{\text{lab}}}, \{y_v\}_{(u,v) \in \mathcal{E}_p^{\text{lab}}} \right),$$

where y_u, y_v are the continuous target values. Since the baseline under independence is 0, we set $h_{\text{adj}}(p) = h_{\text{edge}}(p)$ directly for regression.

Only metapaths with nontrivial support are kept: let $\mathcal{P}_2^{\text{valid}}(\tau^\star)$ be those p for which the computation yields a finite value and at least one labeled node has at least one labeled neighbor (the repo uses a sparsity/count filter). Then the reported aggregates are

$$H_{\text{rel\_mean}} = \frac{1}{|\mathcal{P}_2^{\text{valid}}|} \sum_{p \in \mathcal{P}_2^{\text{valid}}} h_{\text{adj}}(p),$$

$$H_{\text{rel\_min}} = \min_{p \in \mathcal{P}_2^{\text{valid}}} h_{\text{adj}}(p),$$

$$H_{\text{rel\_max}} = \max_{p \in \mathcal{P}_2^{\text{valid}}} h_{\text{adj}}(p).$$

Training Set Size. The natural logarithm of the number of training rows, serving as a scale-compressed proxy for task size.

Feature Strength. The headroom-normalized gain from enabling node/edge features in a GNN:

$$\text{FeatStr} = \frac{s_{\text{feat}} - s_{\text{no-feat}}}{1 - s_{\text{no-feat}} + \varepsilon},$$

where s_{feat} and $s_{\text{no-feat}}$ are GNN performance with and without features. We use ROC-AUC for binary classification, accuracy for multiclass, and normalized R^2 for regression. Values > 0 indicate features help; values < 0 indicate features hurt.

Relation Strength. The headroom-normalized gain from using longer-range relational context:

$$\text{RelStr} = \frac{s_{\text{2hop}} - s_{\text{1hop}}}{1 - s_{\text{1hop}} + \varepsilon},$$

where s_{2hop} and s_{1hop} are TabPFN performance using 2-hop and 1-hop DFS features. Values > 0 indicate multi-hop relations provide additional signal.

Table 7: Checkpoint-level performance across 12 tasks. NS denotes unsupported or unavailable.

	Original RelBench-style Tasks								*Autocomplete*			*Manyclass*
Task	driver-d	driver-p	study-o	study-a	user-b	author-p	user-v	ad-ctr	charge	repeater	prepay	author-c
Database	f1	f1	trial	trial	stack	arxiv	avito	avito	seznam	avs	seznam	arxiv
Metric	AUROC	R^2	AUROC	R^2	AUROC	R^2	AUROC	R^2	Acc	AUROC	Acc	Macro F1
ckpt1_rt_setA_orig_readout	0.760	-0.033	0.509	-0.008	0.516	0.045	0.554	-1.101	NS	0.497	NS	NS
ckpt2_rt_setA_tabpfn	0.782	0.339	0.681	0.275	0.845	0.222	0.627	-0.025	0.830	0.552	0.897	0.093
ckpt3_rt_setB_proto_tabpfn	0.772	0.272	0.683	0.214	0.855	0.234	0.635	-0.001	0.838	0.542	0.919	0.124
ckpt5_griffin_setB_tabpfn	0.738	0.304	0.625	0.132	0.787	-0.738	0.627	-0.038	0.683	0.508	0.826	0.010
ckpt6_rt_setB_griffinhead_tabpfn	0.776	0.356	0.651	0.203	0.832	0.238	0.642	-0.043	0.790	0.530	0.871	0.071
ckpt7_rt_setB_maxclasses_allclf_tabpfn	0.775	0.340	0.709	0.220	0.830	0.230	0.666	0.054	0.851	0.551	0.924	0.003

Table 8: Task signatures for the core evaluation set. H_{rel} denotes relational homophily (mean, min, max across relations). Feature strength, relation strength, and RT relative strength are headroom-normalized gains defined below.

Database	Task	#Classes	$H_{\text{rel}}^{\text{mean}}$	$H_{\text{rel}}^{\text{min}}$	$H_{\text{rel}}^{\text{max}}$	$\log N_{\text{train}}$	Feat. Str.	Rel. Str.	RT Rel. Str.
rel-f1	driver-dnf	2	0.06	−0.14	0.31	9.34	0.04	0.28	0.03
rel-f1	driver-position	–	0.48	0.13	1.00	8.92	0.00	0.33	0.18
rel-trial	study-outcome	2	0.07	0.01	0.11	9.39	0.00	0.07	0.00
rel-trial	study-adverse	–	0.97	0.80	1.00	10.68	0.00	0.24	0.11
rel-stack	user-badge	2	−0.08	−0.96	0.36	15.04	0.02	0.58	0.02
rel-arxiv	author-category	53	0.57	0.57	0.57	12.26	−0.06	0.45	4.38
rel-arxiv	author-publication	–	0.71	0.42	1.00	12.26	0.00	0.27	0.15
rel-avito	user-visits	2	0.17	−0.01	0.49	11.37	0.02	0.27	0.02
rel-avito	ad-ctr	–	0.83	0.61	1.00	8.54	0.00	0.05	0.00
rel-avs	repeater	2	0.04	0.03	0.06	11.60	0.05	0.07	0.00
dbinfer-seznam	charge	8	0.58	0.58	0.58	13.00	0.45	0.44	0.25
dbinfer-seznam	prepay	8	0.63	0.63	0.63	13.96	0.48	0.58	0.28

RT Relative Strength. The relative improvement from including task-table rows in RT's context:

$$\text{RTRelStr} = \frac{s_{\text{with}} - s_{\text{skip}}}{s_{\text{skip}} + \varepsilon},$$

where s_{with} and s_{skip} are RT performance with and without task-table context. Values > 0 indicate task-table labels provide useful in-context signal; large values can occur when s_{skip} is small.