TOWARD EFFECTIVE RELATIONAL FOUNDATION MODEL: A RETRIEVAL AND MEMORY BASED APPROACH

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational Foundation Models (RFMs) aim to learn generalizable representations across different relational databases, whose critical information is distributed across tabular entity attributes, primary–foreign key connections, and multi-hop relational contexts. While existing methods have achieved strong performance on relational databases, our empirical experiments reveal three fundamental limitations: ❶ Flattening relational databases obscures relational information, making primary–foreign key connections and relational context less explicit. ❷ Graph-based methods process relations indiscriminately, introducing task-irrelevant relational context which contributes little to target tasks. ❸ Existing pretraining objectives under-utilize relational information, limiting the models' ability to capture transferable relational information. To address these challenges, we propose **RaMA** (**R**etrieval **a**nd **M**emory **A**ssistant), an RFM that utilizes relational knowledge from records stored in a dynamically evolving external relational memory. ❶ *RaMA preserves relational information* through a relation-based model backbone and overall framework. ❷ *RaMA retrieves task-relevant relational information* through a dynamically evolving external relational memory, where task-aware queries retrieve records for adaptive fusion with local context and reward-guided feedback constantly refines memory effectiveness. ❸ *RaMA explicitly learns relational information* by jointly optimizing masked attribute reconstruction and masked foreign-key recovery. Extensive experiments on RelBench show that RaMA consistently outperforms baselines across classification, regression, and relation prediction, with up to **2.45%** AUROC improvement and significantly lower MAE. It further improves zero-shot transfer by up to **3.24%** on unseen databases, while retaining efficient adaptation with moderate time and GPU-memory costs.

1 INTRODUCTION

Relational databases support a wide range of real-world applications, including e-commerce, healthcare, academic systems, and social platforms. Unlike single-table data, their predictive information is distributed across entity attributes, primary–foreign key connections, historical interactions, and multi-hop relational contexts (Fey et al., 2024; Robinson et al., 2024; Dwivedi et al., 2025). Traditional tabular learning pipelines typically flatten multiple tables through joins, aggregation, or feature synthesis before applying tabular models (Kanter & Veeramachaneni, 2015; Fey et al., 2024), which requires task-specific preprocessing and may obscure the original relational information. Relational deep learning represents database rows as nodes and primary–foreign key references as typed edges, enabling graph models to directly capture cross-table dependencies (Fey et al., 2024; Wang et al., 2024; Chen et al., 2025; Dwivedi et al., 2026).

Recent advances motivate Relational Foundation Models (RFMs), which apply transferable relational learning through cross-database pre-training and adaptation (Wang et al., 2025; Ranjan et al., 2026; Hudovernik et al., 2026), while recent methods further explore relational in-context learning (Fey et al., 2025; Wang et al., 2026). Nevertheless, our empirical investigation reveals three limitations in existing relational learning paradigms: ❶ Flattening relational databases can obscure relational information, making primary–foreign key connections and relational contexts less explicit (Kanter & Veeramachaneni, 2015; Fey et al., 2024); ❷ graph-based relational models commonly

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

operate on generic neighborhoods or relational contexts, introducing task-irrelevant relational context that contributes little to target tasks (Chen et al., 2025; Dwivedi et al., 2026); and ❸ common RFM pre-training objectives primarily emphasize attribute- or token-level reconstruction, leaving relational structure less directly supervised (Wang et al., 2025; Ranjan et al., 2026).

To better understand and study these limitations, we conduct three empirical studies in Sec. 3, aiming to answer the following questions: **(Q1)** How to model relational databases? **(Q2)** How to represent relational graphs? and **(Q3)** How to take advantage of relational graphs? Our empirical results reveal three consistent principles: an effective RFM should preserve explicit relational structure, selectively exploit task-relevant relational information, and explicitly learn relational dependencies. These observations suggest that useful relational knowledge should remain accessible and reusable.

Motivated by these findings, we propose **RaMA** (**R**etrieval **a**nd **M**emory **A**ssistant), an RFM that utilizes reusable relational knowledge stored in a dynamically evolving external relational memory. ❶ *RaMA preserves relational information* by modeling relational databases as graphs and employing an atomic-route-oriented backbone to capture cross-table dependencies. ❷ *RaMA retrieves task-relevant relational information* through an external relational memory, where task- and relation-aware queries retrieve reusable records and adaptively fuse them with local context. A reward-guided mechanism further updates the memory according to the utility of retrieved records. ❸ *RaMA explicitly learns relational dependencies* by jointly optimizing masked attribute reconstruction and masked foreign-key recovery, enabling it to learn both entity semantics and relational structure.

We evaluate RaMA on 6 tasks from 5 datasets covering classification, regression, and relation prediction (Robinson et al., 2024). Extensive experiments in Sec. 5 demonstrate that RaMA consistently outperforms strong GNN, tabular foundation model, and existing RFM baselines, while having a strong ability to generalize to unseen databases and achieve favorable computational efficiency.

Our main contributions are summarized as follows: (1) *New Perspective.* We conducted empirical analysis to investigate effective relational foundation modeling from database modeling, task-relevant relation utilization, and relation-aware pre-training. (2) *Novel Approach.* We propose RaMA, which integrates relation-oriented modeling, dynamic external relational memory, task-relational retrieval, and relation-aware pre-training within a unified relational foundation model framework. (3) *SOTA Performance.* RaMA achieves the best performance across six downstream classification, regression, and relation-prediction tasks. On unseen databases, RaMA improves zero/few-shot performances by up to **3.72** percentage points.

2 Preliminaries

2.1 Notations and Problem Formulation

Consider a relational database $\mathcal{D} = \{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$, where each table $\mathcal{T}_k$ contains rows, attribute columns, and primary- or foreign-key columns. Following relational deep learning (Fey et al., 2024; Robinson et al., 2024; Gu et al., 2026), we transform $\mathcal{D}$ into a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi)$, where each row is represented as a node, each primary–foreign key reference induces an edge, and ϕ and ψ denote node and relation types. For temporal prediction, only records observed before the prediction time are included to avoid information leakage. This transformation preserves the database semantics while enabling relational representation learning over heterogeneous entities and interactions.

An atomic route ρ is a typed path induced by primary–foreign key structures and treated as a unified relational computation unit (Chen et al., 2025). We define each memory record as $e = (\rho, \mathbf{x}, \mathbf{m}, u)$, where $\mathbf{x}$ contains the attributes associated with the route, $\mathbf{m}$ records its table, relation, direction, and source information, and u denotes its utility. These records form a dynamically evolving memory $\mathcal{M} = \{e_i\}_{i=1}^{N}$, which stores reusable relational patterns discovered from historical databases.

Given a target entity v, a task τ, and its local relational context $\mathcal{G}_v$, RaMA retrieves task-compatible records $\mathcal{R}_{v,\tau}^{\mathrm{ret}}$ from $\mathcal{M}$ and predicts $\hat{y}_v = f_\theta\big(\mathcal{G}_v, \mathcal{R}_{v,\tau}^{\mathrm{ret}}\big)$.

Our objective is to learn transferable parameters θ and reusable relational information from source databases, and adapt them to unseen databases and tasks with limited supervision. Unlike conventional database-specific models, RaMA aims to decouple relational knowledge storage from task-specific prediction, enabling flexible transfer across heterogeneous relational environments.

2.2 Related Works

Tabular learning and foundation models. Traditional tabular prediction commonly relies on gradient-boosted trees (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018), while deep tabular models use attention and transferable feature representations (Huang et al., 2020; Gorishniy et al., 2021; Wang & Sun, 2022). Retrieval-augmented tabular learning further incorporates neighboring training instances into prediction (Gorishniy et al., 2024). Recent tabular foundation models, including TabPFN, TabICL, TabDPT, and UniTabE, learn reusable priors through synthetic tasks, real tables, or in-context learning (Hollmann et al., 2025; Qu et al., 2025; Ma et al., 2025; Yang et al., 2024). However, they operate on single table representations without directly modeling multi-table primary–foreign key structures.

Relational deep learning. Relational deep learning represents rows as nodes and key references as typed edges (Fey et al., 2024). RelBench and 4DBInfer provide standardized relational tasks and evaluation protocols (Robinson et al., 2024; Wang et al., 2024). RelGNN introduces composite message passing over atomic routes to model cross-table relationships, while RelGT models sampled relational subgraphs with transformers (Chen et al., 2025; Dwivedi et al., 2026). These methods preserve cross-table connectivity and aggregate multi-hop relational context under temporal constraints, but they are typically optimized for a particular database or downstream task rather than reusable cross-database transfer.

Relational foundation models. Griffin and Relational Transformer study cross-database relational pretraining and unified task modeling (Wang et al., 2025; Ranjan et al., 2026) with Griffin adopting graph-centric shared encoders and task decoders and Relational Transformer supporting zero-shot transfer through task-conditioned relational attention. KumoRFM and RDB-PFN explore relational in-context learning and synthetic relational priors (Fey et al., 2025; Wang et al., 2026). RAG-GFM further introduces external retrieval into graph foundation models (Yuan et al., 2026). Existing RFMs transfer knowledge through shared encoders, task representations, relational context, or synthetic priors, while RaMA instead maintains reusable attribute and typed-route records in an external relational memory and updates them according to downstream utility.

3 Empirical Analysis

We conduct three empirical analyses of database modeling, relational context selection, and relational pretraining. These studies motivate the design in RaMA with results in Figure 2.

3.1 Q1: How to Model Relational Databases?

We compare two relational database modeling paradigms: flattening multiple tables into task-specific aggregated features with TabDPT (Ma et al., 2025), and explicitly preserving the original primary–foreign key relationships as graphs with Griffin (Wang et al., 2025). Experiments are conducted on Rel-F1 Driver-DNF, Rel-Stack User-Badge, and Rel-Amazon User-Churn, covering relational databases of different scales and application domains.

Observation. The graph-based modeling method consistently outperforms flattened representations, improving AUROC by 1.7–2.0% across all three datasets while achieving an approximately 2.6–2.8× runtime speedup.

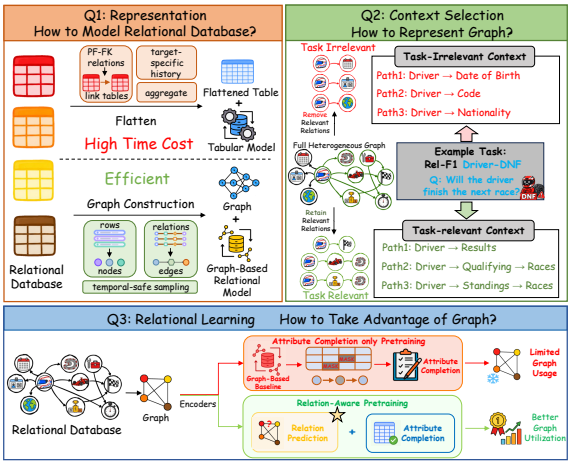


Figure 1: Empirical questions and design motivations for effective relational foundation modeling.

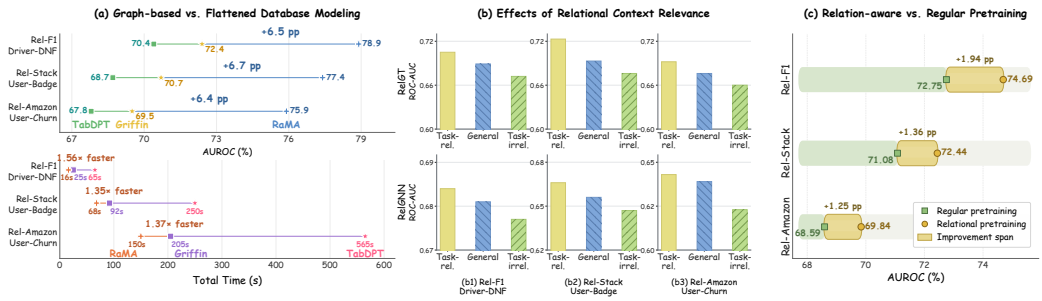


Figure 2: Empirical analyses of relational foundation modeling. **a:** graph-based versus flattened database modeling in predictive performance and running time. **b:** performance under task-relevant, general, and task-irrelevant relational contexts. **c:** relation-aware versus regular pretraining.

These results indicate that preserving relational structure as graphs provides a more favorable effectiveness–efficiency trade-off than task-specific flattening.

Conclusion & Key Insight. Flattening relies on task-specific joins and aggregation that can obscure relation types, directions, and cross-table dependencies. Graph representations retain these database semantics directly and avoid repeatedly reconstructing them for individual tasks. To preserve relational structure from the input stage, RaMA represents rows as nodes and primary–foreign key references as typed edges, and builds its relational backbone on this graph representation.

3.2 Q2: How to Represent Relational Graphs?

Graph construction preserves relational knowledge, but relations contribute unequally among tasks. A target entity may be linked to many historical interactions, only a subset of which is useful for the current task. We therefore compare *task-relevant*, *general*, and *task-irrelevant* relational contexts under similar context budgets on the same datasets with representative relational backbones.

Observation. Task-relevant contexts consistently achieve the best performance, whereas general and task-irrelevant contexts are weaker under matched context budgets. This demonstrates the importance of task compatibility when selecting relational information under a fixed context budget.

Conclusion & Key Insight. Uniformly processing all available relations can introduce irrelevant dependencies and weaken useful signals. Therefore, RaMA stores node attributes and atomic routes as reusable relational records and retrieves only a compact set of task-compatible records for each prediction. Because useful relations can change across tasks and training stages, predictive feedback is further used to refine which records remain valuable in memory.

3.3 Q3: How to Take Advantage of Relational Graphs?

Existing relational pretraining primarily reconstructs masked attributes, with relations serving mainly as information-propagation paths. However, the primary–foreign key structure also contains explicit information about how entities are connected. To test whether this structure should be learned directly, we augment Griffin with masked primary–foreign key recovery while retaining its original masked attribute reconstruction objective.

Observation. Relation-aware pretraining improves attribute-only pretraining by **1.2–1.9%** AUROC across the three datasets. The consistent gains indicate that explicitly predicting relational dependencies contributes transferable information beyond attribute reconstruction alone.

Conclusion & Key Insight. Attribute reconstruction mainly teaches the model what information entities contain, whereas masked foreign-key recovery additionally teaches how entities are connected through explicit relational dependencies. Therefore, using relations only as message-passing channels does not fully exploit the structural information available during pretraining. This motivates RaMA to jointly optimize masked attribute reconstruction and masked foreign-key recovery, enabling the backbone to capture both transferable entity semantics and explicit relational dependencies within a unified relational pretraining objective.

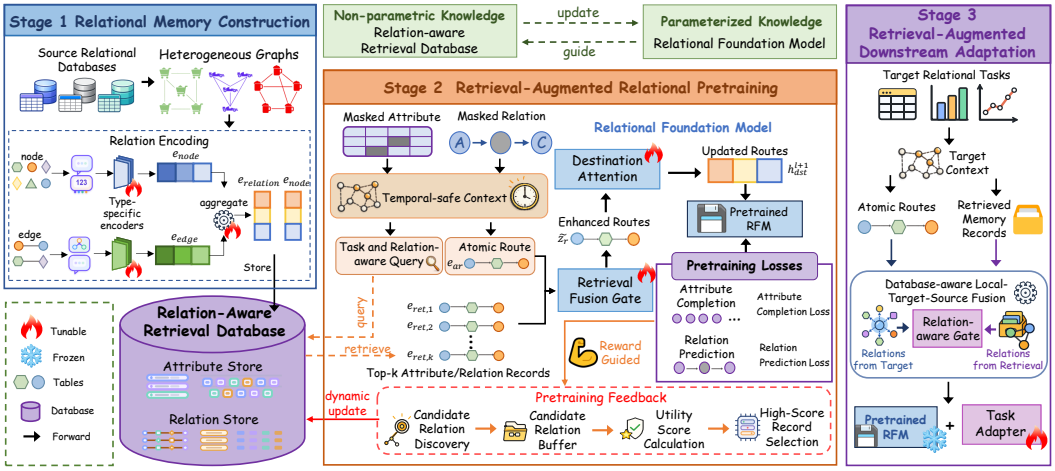


Figure 3: Overview of our proposed **RaMA** framework.

4 Methodology

4.1 Overview of RaMA Framework

Guided by Sec. 3, RaMA follows three design principles corresponding to Q1–Q3: preserving explicit relational structure, selecting task-compatible relational information, and explicitly learning relational dependencies.

As shown in Fig. 3, RaMA consists of three stages. Stage 1 constructs temporal heterogeneous graphs and organizes reusable entity- and atomic route into the Attribute Store and Relation Store. Stage 2 retrieves memory records for fusion with local atomic routes and jointly optimizes attribute and relation reconstruction, while training feedback updates the Relation Store. Stage 3 transfers the pre-trained backbone and memory to downstream relational tasks.

RaMA uses *atomic routes* (Chen et al., 2025) as basic units for cross-table computation, preserving the ordered relational structure along each route. The local representation for an ordered atomic route $\rho = (v_0, \tau_1, v_1, \ldots, \tau_L, v_L)$ with $L \leq 2$ is

$$\mathbf{e}_\rho^{\text{loc}} = f_{\text{route}}([\mathbf{h}_{v_0} \| \mathbf{e}_{\tau_1} \| \mathbf{h}_{v_1} \| \cdots \| \mathbf{e}_{\tau_L} \| \mathbf{h}_{v_L}]), \tag{1}$$

where $\mathbf{h}_v$ and $\mathbf{e}_\tau$ denote entity and relation representations. Retrieved records enhance these representations before destination attention aggregates atomic routes for each target entity. Using atomic routes for both local context and memory records aligns the backbone and external memory at the route level. The complete atomic-route aggregation procedure is provided in Appendix C.4.

4.2 Task-Relational Retrieval and Memory Update

Following the context analysis in Sec. 3.2, RaMA maintains an external relational memory that stores reusable relational information from source databases and retrieves a compact set of compatible records for each target route. The memory contains an *Attribute Store* for entity-level information and a *Relation Store* for complete atomic-route records. An atomic-route memory record is represented as $e_i = (\rho_i, \mathbf{x}_i, \mathbf{m}_i, u_i)$, where ρ_i specifies the route, $\mathbf{x}_i$ contains its visible attributes, $\mathbf{m}_i$ records relational and source metadata, and u_i denotes its running utility. Memory construction follows the temporal constraints of the source databases; the detailed rules are given in Appendix C.2.

RaMA retrieves records according to both semantic and structural compatibility. The semantic view represents visible entity attributes, while the structural view describes relation sequences and relational metadata. Given the current target, task $\mathcal{T}$, and route context, the query is projected into the two retrieval spaces and scores a candidate record by

$$s(\mathbf{q}, e_i) = w_{\text{sem}} \cos(\mathbf{q}_{\text{sem}}, \mathbf{z}_{i,\text{sem}}) + w_{\text{str}} \cos(\mathbf{q}_{\text{str}}, \mathbf{z}_{i,\text{str}}) + b(\mathbf{q}, \mathbf{m}_i). \tag{2}$$

The highest-scoring eligible records are aggregated by retrieval attention to produce a retrieved route representation $\mathbf{e}_\rho^{\text{ret}}$. This retrieval strategy supports cross-database transfer by filtering candidates before fusion and aligning retrieved context with both route structure and prediction semantics. The retrieved route representation is integrated with the target route through a learned vector gate:

$$\widetilde{\mathbf{e}}_\rho = \mathbf{g}_\rho \odot \mathbf{e}_\rho^{\text{loc}} + (1 - \mathbf{g}_\rho) \odot \mathbf{e}_\rho^{\text{ret}}. \tag{3}$$

The enhanced atomic routes are then aggregated by destination attention. During source pre-training, newly observed route records are maintained in a candidate buffer and evaluated using training feedback. For a retrieved record e_i, its utility combines retrieval compatibility $s(\mathbf{q}, e_i)$ and a usage-weighted contribution score of the batch-level learning feedback $\mathcal{A}_b(i)$:

$$U_b(i) = \alpha\, s(\mathbf{q}, e_i) + \beta\, \mathcal{A}_b(i). \tag{4}$$

Compatibility determines whether a record is relevant to the current route, whereas $\mathcal{A}_b(i)$ measures the batch-level benefits of retrieved memory according to the records' retrieval attention and fusion-gate usage. Candidate records are added to the Relation Store only after accumulating sufficient observations and positive utility, while persistently low-utility records are removed. Thus, memory growth is governed by predictive usefulness rather than retrieval frequency alone. The complete feedback attribution, admission, decay, and maintenance procedures are provided in Appendix C.5.

4.3 Relation-Aware Pre-training Objective

To jointly capture entity semantics and relational information, RaMA masks entity attributes and primary–foreign key relations and learns them jointly. The pretraining objective is

$$\mathcal{L}_{\text{pre}} = \mathcal{L}_{\text{attr}} + \lambda_{\text{rel}} \mathcal{L}_{\text{rel}}, \tag{5}$$

where $\mathcal{L}_{\text{attr}}$ reconstructs masked attributes and $\mathcal{L}_{\text{rel}}$ recovers masked foreign-key destinations under the corresponding source entity and relation type. During relation recovery, the masked connection is removed and candidates are restricted to temporally eligible entities. This joint objective complements the retrieval mechanism: the backbone learns transferable entity and relation representations, while the external memory keeps reusable route-level records accessible. Candidate construction, negative sampling, and type-specific reconstruction losses are detailed in Appendix C.6.

4.4 Theoretical Analysis

We theoretically characterize the two key designs motivated by Secs. 3.2 and 3.3, with detailed proofs provided in Appendix F.

Theorem 4.1 (Bounded Retrieval Contamination). For a query $\mathbf{q}$, let $\mathcal{C}^+$ and $\mathcal{C}^-$ contain n_+ task-compatible and n_- task-incompatible records. If $\min_{e_i \in \mathcal{C}^+} s(\mathbf{q}, e_i) - \max_{e_j \in \mathcal{C}^-} s(\mathbf{q}, e_j) \geq \gamma > 0$ and $\|\mathbf{z}_i\|_2 \leq B$, then softmax retrieval with temperature τ satisfies

$$\left\| \mathbf{e}_\rho^{\text{ret}} - \mathbf{e}_\rho^{\text{ret},+} \right\|_2 \leq 2B \frac{n_-}{n_- + n_+ \exp(\gamma/\tau)}, \tag{6}$$

where $\mathbf{e}_\rho^{\text{ret},+}$ uses only compatible records. The bound decreases exponentially with the compatibility margin γ/τ and becomes tighter as compatible records dominate the retrieved set. Hence, task-incompatible relational information is suppressed before fusion.

Proposition 4.2 (Relational Information Preservation). Let R denote the true masked foreign-key destination, C the observed relational context, and Z the representation used for relation recovery. For $\mathcal{L}_{\text{rel}} = \mathbb{E}[-\log q_\phi(R \mid Z, C)]$, we have

$$I(Z; R \mid C) \geq H(R \mid C) - \mathcal{L}_{\text{rel}}. \tag{7}$$

Since $H(R \mid C)$ is determined by the relational data distribution, reducing $\mathcal{L}_{\text{rel}}$ raises the lower bound on the relational information retained in Z. The relation-aware objective therefore encourages the learned representation to preserve foreign-key dependencies beyond attribute semantics.

Table 1: Comparison with baseline methods on relational downstream tasks.

Method	Classification (AUROC (%) ↑)		Regression (MAE ↓)		Relation Prediction (MAP (%)↑)	
	Driver-DNF	Studies-has-dmc	Review-rating	Attendance	Paper-Paper-Co	Site-Spon-Run
Heterogeneous GNN Models						
GraphSAGE	71.86 ±0.43	73.47 ±0.37	0.5713 ±0.0076	0.2716 ±0.0108	9.53 ±0.42	10.46 ±0.29
Graph Transformer	73.17 ±0.38	75.62 ±0.45	0.5937 ±0.0062	0.2541 ±0.0087	10.68 ±0.34	11.93 ±0.36
GraphMAE	72.21 ±0.31	76.14 ±0.41	0.5792 ±0.0054	0.2693 ±0.0095	10.37 ±0.46	10.71 ±0.27
Single-table Models						
XGBoost	65.77 ±0.26	70.48 ±0.30	0.5916 ±0.0046	0.2612 ±0.0061	N/A	N/A
LightGBM	68.86 ±0.29	71.13 ±0.34	0.5487 ±0.0041	0.2764 ±0.0052	N/A	N/A
TabICL v1.1	72.39 ±0.37	73.62 ±0.44	0.5358 ±0.0068	0.2595 ±0.0064	N/A	N/A
TabPFN v2.5	71.67 ±0.23	73.18 ±0.31	0.5564 ±0.0038	0.2547 ±0.0043	N/A	N/A
Relational Foundation Models						
Griffin	73.02 ±0.42	76.32 ±0.51	0.5240 ±0.0067	0.2573 ±0.0058	9.44 ±0.39	10.04 ±0.31
Relational Transformer	76.48 ±0.35	77.41 ±0.46	0.5272 ±0.0052	0.2309 ±0.0071	9.56 ±0.32	9.11 ±0.24
RDL-GNN	71.53 ±0.51	75.18 ±0.57	0.5586 ±0.0081	0.2620 ±0.0077	10.14 ±0.28	11.27 ±0.19
RDB-PFN	71.29 ±0.28	73.67 ±0.36	0.5492 ±0.0044	0.2467 ±0.0048	10.58 ±0.30	11.41 ±0.25
Rel-LLM	74.86 ±0.58	76.94 ±0.53	0.5387 ±0.0086	0.2572 ±0.0082	12.83 ±0.55	12.64 ±0.45
RelGT	71.91 ±0.33	75.63 ±0.42	0.5238 ±0.0049	0.2417 ±0.0059	12.26 ±0.37	13.38 ±0.29
Ours	**78.93 ±0.29**	**79.01 ±0.36**	**0.5113 ±0.0041**	**0.2205 ±0.0049**	**13.08 ±0.34**	**15.04 ±0.27**

5 EXPERIMENTS

To evaluate RaMA, we organize our experiments around five research questions: **Q1** (Effectiveness) Does RaMA outperform strong baselines on downstream relational prediction tasks? **Q2** (Generalization) Can the pre-trained RaMA generalize to unseen relational databases with limited task-specific supervision? **Q3** (Interpretability) What enables RaMA to effectively improve relational prediction? **Q4** (Robustness) Is RaMA robust to variations in pre-training scale and retrieval configurations? **Q5** (Efficiency) Does RaMA achieve a favorable performance–efficiency trade-off?

5.1 OVERALL EFFECTIVENESS (Q1)

As shown in Table 1, RaMA achieves the best performance across all six tasks spanning classification, regression, and relation prediction. The gains are consistently observed across different databases and tasks. In classification, RaMA improves over strong relational foundation model baselines on both Driver-DNF and Studies-has-dmc by up to **2.45** percentage points. For regression, it further reduces prediction errors on Rel-Amazon review-rating and Rel-Event user-attendance, while maintaining clear advantages on the two relation prediction tasks Rel-Arxiv paper-paper-cocitation and Rel-Trial site-sponsor-run by up to **1.66** percentage points. These results suggest that the improvements come from the explicitly preserved relational structure, retrieved task-compatible relational knowledge, and the useful records maintained in the external memory, which together provide stable benefits that transfer across various relational settings and prediction objectives.

5.2 GENERALIZATION TO UNSEEN DATABASES (Q2)

A key capability of relational foundation models is transferring knowledge beyond databases observed during pre-training. We evaluate this ability under two complementary settings. First, leave-one-database-out zero-shot evaluation excludes the target database from source pre-training and memory construction, then evaluates unseen tasks without target-label-based parameter updates. Results across classification, regression, and relation prediction are reported in Table 2. Second, we investigate few-shot adaptation and relational in-context learning on unseen tasks, where only 32, 64, or 128 labeled support examples are provided. Different from parameter updating based adaptation, RaMA performs in-context inference by retrieving and integrating relevant relational evidence from external memory. Results comparing adapter fine-tuning and in-context learning are shown in Table 3. RaMA consistently leads all six zero-shot tasks and all nine few-shot settings across different databases, with gains up to 3.72%. Detailed split construction, support-set rules, in-context learning protocols, and inference procedures are provided in Appendix E.2.

Table 2: Leave-one-database-out zero-shot generalization on unseen databases and tasks.

Held-out Database	Unseen Task	Task Type	Metric	Griffin	RT	RaMA
Rel-F1	`driver-top3`	Classification	AUROC ↑	82.52 ±0.41	84.23 ±0.36	**87.47 ±0.31**
Rel-Stack	`user-engagement`	Classification	AUROC ↑	71.67 ±0.45	75.18 ±0.38	**76.22 ±0.33**
Rel-Amazon	`review-rating`	Regression	MAE ↓	0.6328 ±0.0061	0.5737 ±0.0054	**0.5384 ±0.0046**
Rel-Event	`user-attendance`	Regression	MAE ↓	0.3517 ±0.0092	0.3289 ±0.0075	**0.3040 ±0.0068**
Rel-Arxiv	`paper-paper-co`	Relation Prediction	MAP ↑	8.74 ±0.46	9.17 ±0.39	**9.82 ±0.34**
Rel-Trial	`site-sponsor-run`	Relation Prediction	MAP ↑	9.52 ±0.37	9.97 ±0.38	**10.63 ±0.29**

Table 3: Few-shot adaptation and in-context learning performance on unseen relational databases. All tasks are evaluated using AUROC (↑).

Method	Rel-F1 Driver-DNF			Rel-Stack User-Badge			Rel-Amazon User-Churn		
	32-shot	64-shot	128-shot	32-shot	64-shot	128-shot	32-shot	64-shot	128-shot
Griffin(FT)	64.87 ±0.42	66.63 ±0.37	68.14 ±0.31	75.38 ±0.46	77.16 ±0.39	78.93 ±0.34	62.89 ±0.41	64.17 ±0.36	65.62 ±0.33
RT(FT)	71.37 ±0.35	72.14 ±0.32	72.76 ±0.29	80.69 ±0.38	81.95 ±0.34	83.31 ±0.30	64.32 ±0.36	65.16 ±0.31	65.94 ±0.28
RelGT(FT)	56.38 ±0.53	59.67 ±0.47	62.89 ±0.42	58.21 ±0.56	62.38 ±0.49	67.14 ±0.43	53.86 ±0.51	57.19 ±0.45	60.37 ±0.40
RDB-PFN (ICL)	68.37 ±0.44	69.32 ±0.39	70.00 ±0.35	75.61 ±0.47	77.29 ±0.41	74.64 ±0.43	55.93 ±0.48	57.91 ±0.42	60.07 ±0.38
RaMA(ICL)	71.18 ±0.33	72.64 ±0.29	73.97 ±0.26	79.86 ±0.35	81.42 ±0.31	82.79 ±0.27	62.37 ±0.34	64.28 ±0.30	65.96 ±0.27
RaMA(FT)	**73.39 ±0.28**	**74.87 ±0.25**	**76.48 ±0.22**	**82.13 ±0.30**	**84.17 ±0.27**	**85.94 ±0.24**	**64.89 ±0.29**	**66.95 ±0.26**	**68.82 ±0.23**

5.3 Component Analysis and Retrieval Behavior (Q3)

Table 4: Ablation study of key components in RaMA on different tasks.

Model Variant	Driver-DNF (↑)	Driver-Position (↓)	Paper-Paper-Co (↑)
Full RaMA Model	**78.93 ±0.29**	**3.1082 ±0.0341**	**13.08 ±0.34**
Backbone Architecture			
Heterogeneous GNN	70.19 ±0.47	4.2274 ±0.0628	9.61 ±0.45
Memory Construction			
w/o Memory	72.27 ±0.41	3.5638 ±0.0517	9.85 ±0.39
Node-only Record Construction	74.89 ±0.38	3.3351 ±0.0462	9.67 ±0.42
Initialization w/o Candidate Buffer	76.98 ±0.34	3.2456 ±0.0415	10.82 ±0.36
Memory Update & Pre-training Objective			
Static Memory	72.21 ±0.43	3.2910 ±0.0448	11.67 ±0.38
Loss-based Only Reward	77.08 ±0.31	3.1814 ±0.0387	11.73 ±0.35
w/o Relational Loss	77.65 ±0.33	3.2016 ±0.0402	10.45 ±0.41

To answer Q3, we examine which components drive RaMA's performance gains and the interpretability of its retrieval module.

As shown in Table 4, replacing the relation-oriented backbone with a heterogeneous GNN causes the largest overall degradation, highlighting the importance of explicitly preserving route-level cross-table semantics. Removing the external relational memory also consistently reduces performance. Moreover, constructing memory records from nodes alone performs worse than preserving complete relational routes, indicating that RaMA benefits from retrieving structured relational records rather than isolated entity representations.

The memory maintenance mechanism is also important. Static memory, loss-only utility estimation, and removing the candidate buffer all degrade performance, suggesting that the benefit of RaMA comes from both external relational memory itself and continuously identifying and maintaining high-utility relational records. While the removal of relational pre-training objective consistently weakens downstream performance, confirming the importance of explicitly learning relational dependencies.

As Figure 4 shows, strong responses frequently appear between matched target and retrieved routes, while selective cross-relation interactions are also retained. This indicates that RaMA is able to distinguish retrieved routes according to their compatibility with the current target relation instead of simply aggregating additional context.

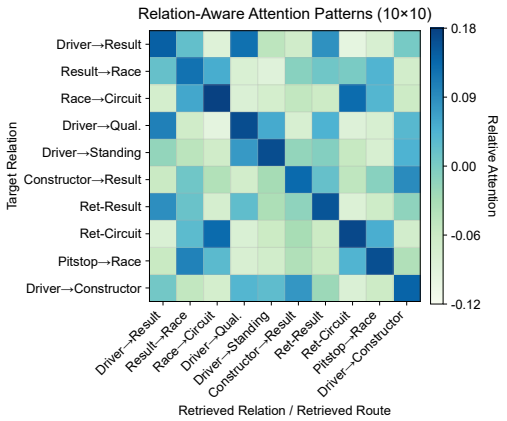


Figure 4: Relation-aware retrieval behavior on Rel-F1. Values indicate relative attention scores.

5.4 Robustness to Scale and Retrieval Choices (Q4)

We examine RaMA under changes in pre-training scale and retrieval hyperparameters. As shown in Figure 5, performance improves consistently as source pre-training data increase, with larger gains at smaller scales and gradual saturation at larger scales. We further vary memory size and retrieval top-k on Driver-DNF. RaMA maintains strong performance across a relatively broad range of configurations when memory capacity is sufficient and retrieval depth is moderate. Performance declines with limited memory capacity or overly large top-k, reflecting reduced relational coverage in the former case and weaker retrieval selectivity in the latter. Overall, RaMA remains robust without relying on a narrow retrieval configuration.

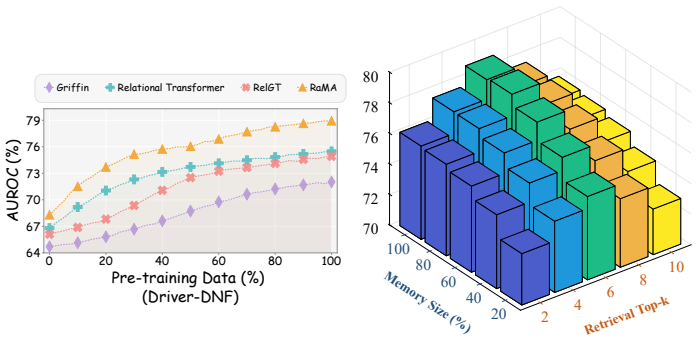


Figure 5: Robustness to pre-training scale **(left)** and retrieval configuration **(right)**.

5.5 Computational Efficiency (Q5)

We further evaluate RaMA in terms of downstream adaptation time and peak GPU memory. As shown in Figure 6, RaMA achieves the strongest performance while maintaining moderate adaptation time, yielding a favorable performance–time trade-off. Its downstream adaptation also requires relatively low GPU memory, while model-specific pre-training remains within a practical memory range. The higher memory demand is mainly concentrated in the pre-training stage rather than downstream task adaptation. Taken together, these results show that the retrieval and external-memory components introduce controlled computational overhead while preserving efficient downstream adaptation. Additional training-time, memory, and computational-cost analyses are provided in Appendix F.5.

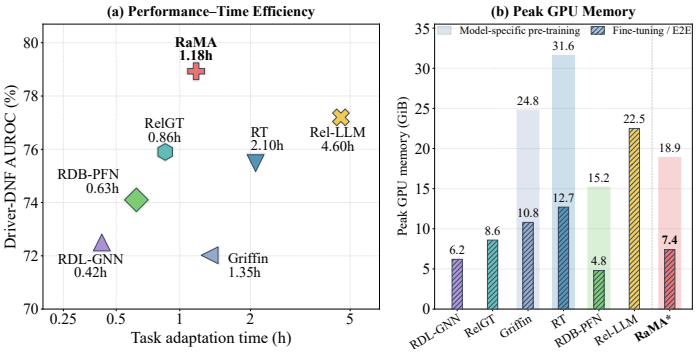


Figure 6: Computational efficiency in adaptation time **(left)** and peak GPU memory **(right)**.

6 Conclusion

In this work, we study how to build effective Relational Foundation Models from three complementary perspectives: preserving relational structure, selecting task-relevant relational information, and explicitly learning relational dependencies. Guided by our empirical analysis, we propose RaMA, a retrieval- and memory-augmented RFM that combines graph-based relational modeling with a dynamically evolving external relational memory. RaMA preserves explicit cross-table dependencies, retrieves task-compatible relational information from an external relational memory, and adaptively integrates it with local database context. Meanwhile, joint masked attribute reconstruction and foreign-key recovery enable the model to explicitly capture both entity semantics and relational dependencies during pre-training. Extensive experiments across diverse RelBench databases and prediction tasks demonstrate that RaMA consistently outperforms strong tabular, graph-based, and relational baselines. Moreover, RaMA exhibits strong generalization to unseen databases and tasks, robust performance under varying retrieval configurations, and favorable computational efficiency, providing an effective framework for transferable relational learning.

Reproducibility Statement

We provide the implementation details of RaMA in the main text and appendix. All experiments use RelBench and Relbenchv2 datasets which are publicly available. Code and configurations of RaMA will be released for reproduction.

AI Use Statement

Generative AI tools were not used in the preparation of this manuscript. All technical contents, method details, and experimental results were designed and verified by the authors.

References

Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite message passing for relational deep learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 8296–8312, 2025.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.

Vijay Prakash Dwivedi, Charilaos I. Kanatsoulis, Shenyang Huang, and Jure Leskovec. Relational deep learning: Challenges, foundations and next-generation architectures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 5999–6009, 2025. doi: 10.1145/3711896.3736558.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. In *The Fourteenth International Conference on Learning Representations*, 2026.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Position: Relational deep learning—graph representation learning on relational databases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 13592–13607, 2024.

Matthias Fey, Vid Kocijan, Federico López, Jan Eric Lenssen, and Jure Leskovec. KumoRFM: A foundation model for in-context learning on relational data. Technical report, Kumo AI, 2025. URL `https://kumo.ai/research/kumo_relational_foundation_model.pdf`.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, volume 34, pp. 18932–18943, 2021.

Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. TabR: Tabular deep learning meets nearest neighbors. In *The Twelfth International Conference on Learning Representations*, 2024.

Justin Gu, Rishabh Ranjan, Charilaos Kanatsoulis, Haiming Tang, Martin Jurkovic, Valter Hudovernik, Mark Znidar, Pranshu Chaturvedi, Parth Shroff, Fengyu Li, and Jure Leskovec. Relbench v2: A large-scale benchmark and repository for relational data. *arXiv preprint arXiv:2602.12606*, 2026.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 3929–3938, 2020.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025. doi: 10.1038/s41586-024-08328-6.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Valter Hudovernik, Federico López, Vid Kocijan, Akihiro Nitta, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. KumoRFM-2: Scaling foundation models for relational learning. *arXiv preprint arXiv:2604.12596*, 2026.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics*, pp. 1–10, 2015. doi: 10.1109/DSAA.2015.7344858.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, pp. 3146–3154, 2017.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Alex Labach, Hamidreza Kamkari, Jesse C. Cresswell, Keyvan Golestan, Guangwei Yu, Anthony L. Caterini, and Maksims Volkovs. TabDPT: Scaling tabular foundation models on real data. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31, pp. 6638–6648, 2018.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 50817–50847, 2025.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos I. Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. In *The Fourteenth International Conference on Learning Representations*, 2026.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. RelBench: A benchmark for deep learning on relational databases. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0672.

Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Ning Li, Jianheng Tang, Yanlin Zhang, Zizhao Zhang, Zunyao Mao, Yakun Song, Yanbo Wang, Jiahang Li, Han Zhang, Guang Yang, Xiao Qin, Chuan Lei, Muhan Zhang, Weinan Zhang, Christos Faloutsos, and Zheng Zhang. 4DBInfer: A 4d benchmarking toolbox for graph-centric predictive modeling on rdbs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 27236–27273, 2024. doi: 10.52202/079017-0856.

Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a graph-centric relational database foundation model. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 64604–64627, 2025.

Yanbo Wang, Jiaxuan You, Chuan Shi, and Muhan Zhang. Relational in-context learning via synthetic pre-training with structural prior. *arXiv preprint arXiv:2603.03805*, 2026.

Zifeng Wang and Jimeng Sun. TransTab: Learning transferable tabular transformers across tables. In *Advances in Neural Information Processing Systems*, volume 35, pp. 2902–2915, 2022.

Yazheng Yang, Yuqi Wang, Guang Liu, Ledell Wu, and Qi Liu. UniTabE: A universal pretraining protocol for tabular foundation model in data science. In *The Twelfth International Conference on Learning Representations*, 2024.

Haonan Yuan, Qingyun Sun, Jiacheng Tao, Xingcheng Fu, and Jianxin Li. RAG-GFM: Overcoming in-memory bottlenecks in graph foundation models via retrieval-augmented generation. In *Proceedings of the ACM Web Conference 2026*, pp. 626–637, 2026. doi: 10.1145/3774904.3792139.

APPENDIX

APPENDIX OVERVIEW

In the Appendix, we provide additional details organized as follows:

1. Appendix A: Datasets in Details.
2. Appendix B: Detailed Related Work.
3. Appendix C: Methodology in Details.
4. Appendix D: Empirical Studies in Details.
5. Appendix E: Experiment and Implementation in Details.
6. Appendix F: Theoretical Proofs in Details.
7. Appendix G: Limitations and Broader Impacts.
8. Appendix H: Details of Baseline Hyperparameters.

A DATASETS IN DETAILS

A.1 OVERVIEW OF RELBENCH

We evaluate RaMA on RelBench v1 and v2, a large-scale benchmark designed for machine learning over relational databases. Unlike conventional tabular benchmarks that flatten multiple tables into independent feature vectors, RelBench preserves relational structures among entities by maintaining the primary-key and foreign-key relationships across tables. Such a design enables models to directly exploit the structural dependencies and rich contextual information contained in relational databases.

RelBench consists of realistic relational databases collected from diverse real-world domains, including e-commerce, social networks, healthcare, scientific publications, enterprise systems, and sports. Each database contains multiple interconnected tables, where rows correspond to entities and relational links define interactions among entities. The benchmark provides standardized task definitions, temporal data splits, and evaluation protocols, enabling fair comparisons among relational learning methods.

Following previous relational foundation models, we use the relational databases in RelBench as the evaluation platform for RaMA. The benchmark contains three categories of downstream tasks: entity classification, entity regression, and recommendation. These tasks evaluate whether models can effectively learn relational representations and utilize multi-table dependencies for prediction.

A.2 RELATIONAL DATABASE FORMULATION

A relational database is represented as a collection of tables $\mathcal{R} = \{T_1, T_2, ..., T_n\}$, where each table contains multiple entities described by attributes. Primary keys uniquely identify entities, while foreign keys establish relationships between different tables.

Following the relational deep learning paradigm, each database can be naturally transformed into a heterogeneous relational graph: $G = (V, E, \phi, \psi, \tau)$, where nodes V represent entities from database tables, edges E represent primary-key–foreign-key relationships, ϕ denotes node types, ψ represents relation types, and τ captures temporal information when available.

For each downstream task, RelBench defines a target entity and prediction objective. Given a seed entity, models are required to predict its target attribute by leveraging both entity-level features and relational information propagated through connected tables.

A.3 Dataset Statistics

RelBench v2 extends the original RelBench benchmark to 11 relational databases spanning diverse real-world domains, including e-commerce, social networks, scientific publications, enterprise systems, consumer platforms, healthcare, and sports. In our experiments, we excluded `rel-mimic` because access to the underlying MIMIC-IV clinical data requires credentialed authorization through PhysioNet, Table 5 summarizes the datasets used in our experiments.

Table 5: Statistics of the RelBench v2 datasets used in our experiments. `rel-mimic` is excluded because it requires credentialed access to the underlying MIMIC-IV data.

Dataset	Domain	#Tables	#Rows	#Columns	Task Types	#Tasks
rel-amazon	E-commerce	3	15,000,713	15	Cls/Reg/Rec	8
rel-avito	Online Marketplace	8	20,679,117	42	Cls/Reg/Rec	6
rel-event	Social Network	5	41,328,337	128	Cls/Reg	6
rel-f1	Sports Analytics	9	74,063	67	Cls/Reg/Rec	6
rel-hm	E-commerce	3	16,664,809	37	Cls/Reg/Rec	4
rel-stack	Social Network	7	4,247,264	52	Cls/Reg/Rec	6
rel-trial	Healthcare	15	5,434,924	140	Cls/Reg/Rec	9
rel-arxiv	Academic Network	6	2,146,112	21	Cls/Reg/Rec	4
rel-salt	Enterprise ERP	4	4,257,145	31	Cls	8
rel-ratebeer	Consumer Platform	13	13,787,005	221	Cls/Reg/Rec	8

A.4 Dataset Description

rel-amazon. The rel-amazon dataset represents an e-commerce relational database, containing information about users, products, reviews, and user-item interactions. It supports various predictive tasks, including user churn, item churn, lifetime value estimation, and recommendation.

rel-avito. The rel-avito dataset is collected from an online marketplace platform. It contains relational information among users, advertisements, search queries, and contextual features. It includes tasks related to click-through rate prediction, user behavior modeling, and recommendation.

rel-event. The rel-event dataset describes user behaviors in an event recommendation platform. It models interactions among users, events, and social contexts, and evaluates models on user engagement and attendance prediction.

rel-f1. The rel-f1 dataset contains historical Formula 1 racing information, including drivers, constructors, circuits, races, and performance information. It supports prediction tasks involving driver performance, race outcomes, and recommendation over racing entities.

rel-hm. The rel-hm dataset is constructed from the H&M retail platform and models customer-product interactions. It contains customer profiles, product attributes, and transaction histories for tasks such as churn prediction, sales estimation, and recommendation.

rel-stack. The rel-stack dataset is derived from Stack Exchange communities, including users, posts, comments, votes, and badge records. It focuses on modeling user activity and engagement patterns.

rel-trial. The rel-trial dataset is based on clinical trial records and contains information about studies, sponsors, interventions, and outcomes. It supports prediction tasks related to clinical trial success and study characteristics.

rel-arxiv. The rel-arxiv dataset models a scholarly publication network containing papers, authors, categories, and citation relationships. It captures the evolution of scientific research through paper citations and author-paper connections, providing a large-scale relational benchmark for academic network analysis.

rel-salt. The rel-salt dataset is derived from the Sales Autocompletion Linked Business Tables (SALT) database released by SAP AI Research. It represents enterprise resource planning workflows, including sales documents, customers, addresses, shipping information, and payment-related attributes. The dataset enables relational prediction in realistic business process scenarios.

rel-ratebeer. The rel-ratebeer dataset contains consumer interactions collected from the RateBeer platform. It models relationships among users, beers, places, brewers, ratings, and reviews, providing rich relational signals for user preference modeling, churn prediction, and recommendation.

Overall, these datasets cover a broad spectrum of real-world relational applications, including e-commerce, online communities, scientific research, healthcare, enterprise systems, and recommendation platforms. Their diverse schemas, heterogeneous entity relationships, and temporal interaction patterns provide a challenging benchmark for evaluating the ability of relational foundation models to learn transferable structural knowledge across databases and downstream tasks.

A.5 Data Splitting and Evaluation

Following the official RelBench protocol, we adopt temporal splits rather than random splits to avoid information leakage from future interactions. Specifically, training, validation, and test samples are separated according to timestamps, which better reflects realistic deployment scenarios where models must make predictions using only historically available information.

For classification tasks, we report Area Under the Receiver Operating Characteristic Curve (AUROC). For regression tasks, we use Mean Absolute Error (MAE). For recommendation tasks, RelBench provides ranking-based evaluation metrics. In our experiments, we follow the corresponding official evaluation protocols for each downstream task.

B Related Work in Details

B.1 Tabular Learning and Foundation Models

Tabular predictors operate on a feature vector for each prediction entity. For relational databases, these vectors can be constructed by joining tables and aggregating related records. Deep Feature Synthesis automates this process by composing transformations and aggregations along database relationships (Kanter & Veeramachaneni, 2015). The resulting features can be used by gradient-boosted trees such as XGBoost, LightGBM, and CatBoost (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). Such pipelines encode relational information through the chosen features; the predictor's access to individual related records and their connectivity depends on what this preprocessing retains.

Tabular foundation models address transfer across feature spaces and tasks through several forms of pre-training. TransTab encodes column names and cell values to support learning across tables with different columns (Wang & Sun, 2022). UniTabE uses a shared representation of table elements and prompt-based pre-training and fine-tuning (Yang et al., 2024). TabPFN learns predictive behavior from synthetic datasets, conditioning on labeled examples at inference time (Hollmann et al., 2025). TabICL also uses synthetic pre-training, with column-then-row attention producing row embeddings for in-context prediction (Qu et al., 2025). TabDPT combines self-supervised learning on real tables with retrieval-based context selection (Ma et al., 2025). Applying these models to multi-table tasks requires an input construction that makes cross-table information available to the predictor.

B.2 Relational Deep Learning

Relational deep learning represents database rows as nodes and primary–foreign key references as edges, combining attribute encoders with graph-based computation (Fey et al., 2024). RelBench provides relational databases, prediction tasks, and temporal evaluation protocols (Robinson et al., 2024). The 4DBInfer toolbox studies database datasets, tasks, graph construction strategies, and predictive models within a common evaluation framework (Wang et al., 2024).

RelGNN develops composite message passing over atomic routes, accounting for the bridge and hub structures through which databases represent many-to-many relationships (Chen et al., 2025). Its attention mechanism weights messages associated with these routes. RelGT tokenizes entities using attributes, type, hop distance, time, and local structure, and combines attention over sampled subgraphs with attention to learnable global centroids (Dwivedi et al., 2026). Following RelGNN's atomic-route formulation, RaMA uses routes as units of computation and memory retrieval.

B.3 Relational Foundation Models

Cross-database pre-training requires encoders and prediction interfaces that can accommodate changes in schema, value types, and task definitions. Griffin shares input encoders and task decoders across databases, uses cross-attention to aggregate cells within rows, and aggregates messages within relation types before combining them (Wang et al., 2025). Its training pipeline includes masked-cell pre-training on single tables, joint supervised training on tabular and relational tasks, and downstream fine-tuning. Relational Transformer represents tasks through tables and tokenizes cells together with table and column metadata. It uses masked-token pre-training and attention over columns, rows, and key-linked records to support zero-shot prediction (Ranjan et al., 2026).

KumoRFM extends in-context learning to relational data by constructing temporally consistent entity subgraphs and labeled context examples (Fey et al., 2025). KumoRFM-2 incorporates task information early in its computation and pre-trains on synthetic and real data across row, column, foreign-key, and cross-sample dimensions (Hudovernik et al., 2026). RDB-PFN approaches transfer through a synthetic relational prior that generates schemas and dependencies across tables. Its implementation applies DFS to obtain tabular inputs for in-context prediction (Wang et al., 2026).

B.4 Retrieval-Augmented Learning and External Relational Memory

Retrieval-augmented models combine learned parameters with information selected from an external collection. REALM trains document retrieval using masked language modeling feedback (Guu et al., 2020), while RAG conditions sequence generation on passages retrieved from a dense index (Lewis et al., 2020). In tabular prediction, TabR retrieves training examples and uses their features and labels through a learned nearest-neighbor component (Gorishniy et al., 2024).

RAG-GFM applies external retrieval to graph foundation modeling, building semantic and structural stores from prefix-structured text and centrality-based motifs. It trains with dual-view alignment and augments downstream support instances with retrieved records (Yuan et al., 2026). RaMA differs by constructing database-structured attribute and typed route records, retrieving them according to the prediction context, and dynamically maintaining external relational memory during pre-training.

C Methodology in Details

C.1 Record Representation

Each memory record is $e = (\rho, \mathbf{x}, \mathbf{m}, u)$, where ρ specifies participating entities and ordered, typed connections: (v_s, τ, v_d) for one hop, with an intermediate entity and second relation type for two hops. The field $\mathbf{x}$ contains visible attributes with column and value types; $\mathbf{m}$ stores table and relation types, traversal directions, source database, partition, and temporal information; and u is the utility for admission and maintenance.

C.2 Memory Construction and Encoding

Memory uses only source-training records within their temporal cutoffs. Sampling across tables and temporal buckets, with bounded one-/two-hop routes and capacity budgets, limits dominance by high-degree entities and frequent relation types. Traversal includes forward and explicitly defined reverse primary–foreign key relations. Algorithm 1 gives the construction procedure; fingerprints combine source database, participating rows, relation sequence, and reference time.

Type-specific encoders map heterogeneous attributes into a shared representation space. Numerical features are normalized using statistics computed from the training split, categorical features are mapped through a training vocabulary with an unknown-value token, and text fields are represented using frozen GloVe embeddings. Table, column, relation-type, and traversal-direction information is encoded separately from entity attributes; raw row identifiers are not used as transferable features.

For a one-hop atomic route, we use the route encoder defined in Eq.1. For a two-hop route $\rho = (v_0, \tau_1, v_1, \tau_2, v_2)$, we extend the encoder to include the intermediate entity and ordered relation sequence:

$$\mathbf{e}_\rho^{\text{loc}} = \phi_\rho(\mathbf{h}_{v_0}, \mathbf{e}_{\tau_1}, \mathbf{h}_{v_1}, \mathbf{e}_{\tau_2}, \mathbf{h}_{v_2}) \,. \tag{8}$$

For retrieval, semantic index vectors encode visible attributes, whereas structural index vectors encode relation sequences and metadata. Both are normalized before indexing.

Algorithm 1: External Relational Memory Construction

Input: Source training databases $\mathcal{D}_{\mathrm{src}}$; visibility rules $\mathcal{V}$; sampling and capacity budgets $\mathcal{B}$; record encoders.
Output: $\mathcal{M}_{\mathrm{attr}}, \mathcal{M}_{\mathrm{rel}}$, and retrieval indices $\mathcal{I}_{\mathrm{sem}}, \mathcal{I}_{\mathrm{str}}$.

$\mathcal{M}_{\mathrm{attr}} \leftarrow \varnothing$; $\mathcal{M}_{\mathrm{rel}} \leftarrow \varnothing$.
for each $\mathcal{D} \in \mathcal{D}_{\mathrm{src}}$ **do**
  $\mathcal{F} \leftarrow \varnothing$; $c \leftarrow \mathrm{InitializeCounts}(\mathcal{B})$.
  $\mathcal{S} \leftarrow \mathrm{SampleRoots}(\mathcal{D}, \mathcal{B})$.
  for each $(v, t) \in \mathcal{S}$ **do**
    $\mathcal{G}_{v,t} \leftarrow \mathrm{VisibleContext}(\mathcal{D}, v, t, \mathcal{V})$.
    $\mathcal{M}_{\mathrm{attr}} \leftarrow \mathcal{M}_{\mathrm{attr}} \cup \mathrm{AttributeRecords}(\mathcal{G}_{v,t})$.
    $\mathcal{P} \leftarrow \mathrm{EnumerateRoutes}(\mathcal{G}_{v,t}, \mathcal{B})$.
    for each $\rho \in \mathcal{P}$ **do**
      $e \leftarrow \mathrm{ConstructRecord}(\rho, \mathcal{G}_{v,t}, \mathcal{B})$.
      if $e = \varnothing$ **then continue**.
      $f \leftarrow \mathrm{Fingerprint}(e)$.
      if $f \in \mathcal{F}$ **then continue**.
      if $\neg\, \mathrm{Valid}(e, \mathcal{V})$ **then continue**.
      if $\neg\, \mathrm{WithinCapacity}(e, c, \mathcal{B})$ **then continue**.
      $\mathcal{M}_{\mathrm{rel}} \leftarrow \mathcal{M}_{\mathrm{rel}} \cup \{e\}$.
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$; $c \leftarrow \mathrm{UpdateCounts}(c, e)$.
    end for
  end for
end for
$\mathcal{M} \leftarrow \mathcal{M}_{\mathrm{attr}} \cup \mathcal{M}_{\mathrm{rel}}$.
$(Z_{\mathrm{sem}}, Z_{\mathrm{str}}) \leftarrow \mathrm{EncodeAndNormalize}(\mathcal{M})$.
$\mathcal{I}_{\mathrm{sem}} \leftarrow \mathrm{BuildIndex}(Z_{\mathrm{sem}}, \mathcal{M})$.
$\mathcal{I}_{\mathrm{str}} \leftarrow \mathrm{BuildIndex}(Z_{\mathrm{str}}, \mathcal{M})$.
return $\mathcal{M}_{\mathrm{attr}}, \mathcal{M}_{\mathrm{rel}}, \mathcal{I}_{\mathrm{sem}}, \mathcal{I}_{\mathrm{str}}$.

C.3 Task- and Relation-Aware Retrieval

The task representation used in the retrieval query encodes the prediction type, target table or column, value type, and relevant relation specification. Separate projections map the query into the semantic and structural index spaces.

Retrieval is restricted to records permitted by the source, partition, and temporal constraints. During pre-training, records revealing the masked attribute or relation are excluded. The semantic and structural indices retrieve candidate records separately, whose deduplicated union is scored by

$$\begin{aligned} s(\mathbf{q}, \mathbf{e}_m) = w_{\mathrm{sem}} \cos(\mathbf{q}^{\mathrm{sem}}, \mathbf{z}_m^{\mathrm{sem}}) + w_{\mathrm{str}} \cos(\mathbf{q}^{\mathrm{str}}, \mathbf{z}_m^{\mathrm{str}}) \\ + b_{\mathrm{time}}(\mathbf{q}, m) + b_{\mathrm{meta}}(\mathbf{q}, m). \end{aligned} \tag{9}$$

The weights w_{sem} and w_{str} are learned through a softplus parameterization. The bias terms b_{time} and b_{meta} encode temporal, source, table, relation, route-length, and retrieval-rank compatibility features through lightweight MLPs. The top-k candidates under this score form $\mathcal{R}_k(\mathbf{q})$.

For a nonempty retrieved set $\mathcal{R}_k(\mathbf{q})$, the attention weights and aggregated representation are

$$\begin{aligned} a_m &= \frac{\exp(s(\mathbf{q}, \mathbf{e}_m)/T)}{\sum_{j \in \mathcal{R}_k(\mathbf{q})} \exp(s(\mathbf{q}, \mathbf{e}_j)/T)}, \\ \mathbf{e}_r^{\mathrm{ret}} &= \sum_{m \in \mathcal{R}_k(\mathbf{q})} a_m \phi_e(\mathbf{e}_m), \end{aligned} \tag{10}$$

where $T > 0$ is the aggregation temperature and ϕ_e maps retrieved record representations into the route representation space. Gradients propagate through the scores and weighted aggregation of selected records, but not through candidate lookup or the discrete top-k selection.

C.4 Route Fusion and Destination Update

Retrieved and local route representations are fused using the relation-aware gate defined in Eq. 3. If no eligible record is retrieved, the local representation is retained: $\widetilde{\mathbf{e}}_r = \mathbf{e}_r^{\mathrm{loc}}$.

For the set $\mathcal{P}(v_d)$ of routes ending at v_d, route messages and their attention weights are computed as

$$\begin{aligned} \mathbf{m}_r &= \phi_{\mathrm{msg}}([\widetilde{\mathbf{e}}_r \| \mathbf{h}_{v_d} \| \mathbf{e}_{\mathcal{T}}]) \,, \\ \omega_r &= \frac{\exp(\phi_{\mathrm{att}}([\mathbf{h}_{v_d} \| \mathbf{m}_r \| \mathbf{e}_{\mathcal{T}}]))}{\sum_{r' \in \mathcal{P}(v_d)} \exp(\phi_{\mathrm{att}}([\mathbf{h}_{v_d} \| \mathbf{m}_{r'} \| \mathbf{e}_{\mathcal{T}}]))}. \end{aligned} \tag{11}$$

The destination representation is updated by

$$\mathbf{h}'_{v_d} = \mathrm{Norm}\left(\mathbf{h}_{v_d} + \eta \sum_{r \in \mathcal{P}(v_d)} \omega_r \mathbf{m}_r \right), \tag{12}$$

where ϕ_{msg} and ϕ_{att} are shared across routes. The residual coefficient is parameterized as $\eta = \sigma(\gamma)$ with learnable scalar γ, and Norm applies layer normalization followed by L_2 normalization. When $\mathcal{P}(v_d)$ is empty, the update is skipped and $\mathbf{h}'_{v_d} = \mathbf{h}_{v_d}$.

For downstream prediction, the same retrieval-and-fusion interface provides the target representation to a task-specific classification, regression, or relation-prediction module. Evaluation protocols are specified in Appendix E.1.

C.5 Reward-Guided Memory Update

A candidate buffer accumulates visible relation records during source pre-training, associating repeated observations by fingerprint.

Learning feedback. At scheduled feedback steps, the model evaluates the same batch with and without source memory using the same parameter state; the additional forward pass does not contribute gradients. The reference loss is

$$\mathcal{L}_b^{\mathrm{ref}} = \begin{cases} \mathcal{L}_b^{\mathrm{without}}, & \text{at scheduled feedback steps,} \\ \overline{\mathcal{L}}_{b-1}, & \text{otherwise,} \end{cases} \tag{13}$$

where $\overline{\mathcal{L}}_{b-1}$ is an exponential moving average of preceding training losses. Normalized feedback is

$$G_b = \mathrm{clip}_{[-1,1]}\left(\frac{\mathcal{L}_b^{\mathrm{ref}} - \mathcal{L}_b^{\mathrm{with}}}{|\mathcal{L}_b^{\mathrm{ref}}| + \epsilon} \right), \qquad \epsilon > 0. \tag{14}$$

The moving-average reference is updated after feedback computation:

$$\overline{\mathcal{L}}_b = \nu \overline{\mathcal{L}}_{b-1} + (1-\nu)\mathcal{L}_b^{\mathrm{with}}, \qquad 0 \le \nu < 1. \tag{15}$$

If no reference is available, G_b is set to zero and the moving average is initialized with the current loss. For a record m retrieved for route r, its usage weight is

$$c_{rm} = a_{rm} \frac{1}{d} \sum_{\ell=1}^{d} (1 - g_{r,\ell}), \tag{16}$$

where a_{rm} is its retrieval attention weight and $1 - g_{r,\ell}$ is the retrieved-branch contribution of the fusion gate in Eq. 3. For a record used by multiple routes in a batch, the usage weights are averaged over its retrieval occurrences to obtain $c_{bm} \in [0, 1]$. Its usage-weighted contribution score is

$$\mathcal{A}_b(m) = c_{bm} G_b. \tag{17}$$

Utility and candidate admission. The utility follows the two-term definition in Eq. 4:

$$U_b(m) = \alpha \, s(\mathbf{q}, e_m) + \beta \, \mathcal{A}_b(m). \tag{18}$$

When a record is retrieved multiple times in a batch, its relevance scores are averaged over those occurrences. For existing memory records, observed utilities update a running estimate:

$$\overline{U}_b(m) = \mu \overline{U}_{\text{prev}}(m) + (1-\mu) U_b(m), \qquad 0 \le \mu < 1. \tag{19}$$

Because new candidates have not yet participated in active retrieval, their provisional utility is initialized from batch-level feedback rather than record-specific loss gain. We use the mean utility of retrieved records when available, and zero otherwise.

For each candidate, the buffer retains the maximum provisional utility observed so far, together with its observation count n_m and positive and negative feedback counts n_m^+ and n_m^-. Zero-valued feedback contributes to n_m but not to n_m^+ or n_m^-. A candidate is eligible for admission when

$$n_m \ge n_{\min}, \qquad u_m^{\text{cand}} \ge u_{\min}, \qquad \frac{n_m^+}{\max(1, n_m^+ + n_m^-)} \ge p_{\min}. \tag{20}$$

Memory maintenance. Algorithm 2 applies the admission criterion in Eq. 20, ranking eligible candidates by accumulated utility and support. Duplicate matching uses fingerprints or representation similarity. Stale records are those not selected for the configured interval; their utility decays by $0 < \delta < 1$. Memory maintenance enforces the capacity budget while preserving relational coverage and protecting newly added records. Records with sufficient observations and persistently low utility may also be removed. Thresholds and intervals appear in Table 7.

Algorithm 2: Reward-Guided Memory Maintenance

Input: Relation Store $\mathcal{M}_{\text{rel}}$; candidate buffer $\mathcal{C}$; accumulated statistics; step b; admission and maintenance settings.
Output: Updated store, buffer, and retrieval indices.

$\mathcal{A} \leftarrow \{m \in \mathcal{C} : \text{Valid}(m) \wedge \text{Eligible}(m)\}$.
$\mathcal{A} \leftarrow \text{SelectCandidates}(\mathcal{A}, K_{\text{ins}})$.
for each $m \in \mathcal{A}$ **do**
  $m' \leftarrow \text{FindDuplicate}(m, \mathcal{M}_{\text{rel}})$.
  if $m' \neq \varnothing$ **then**
    $m' \leftarrow \text{UpdateUtilityAndProvenance}(m', m)$.
  else
    $\mathcal{M}_{\text{rel}} \leftarrow \mathcal{M}_{\text{rel}} \cup \{m\}$.
    $\text{InitializeProtection}(m, b)$.
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \{m\}$.
end for
for each $m \in \mathcal{M}_{\text{rel}}$ **do**
  if $\text{Stale}(m, b)$ **then** $\overline{U}(m) \leftarrow \delta \overline{U}(m)$.
end for
$\mathcal{E} \leftarrow \text{SelectEvictions}(\mathcal{M}_{\text{rel}}, b)$.
$\mathcal{M}_{\text{rel}} \leftarrow \mathcal{M}_{\text{rel}} \setminus \mathcal{E}$.
$\mathcal{I} \leftarrow \text{SynchronizeIndices}(\mathcal{M}_{\text{rel}})$.
return $\mathcal{M}_{\text{rel}}, \mathcal{C}, \mathcal{I}$.

C.6 Pre-training Objectives

Masked attribute completion. An observed attribute value x_m is randomly selected and masked in the input. The model reconstructs it from the visible relational context and retrieved records. The target value is also withheld from any local or retrieved record that directly reveals it. Primary-key identifiers are excluded from attribute reconstruction targets, and missing values do not contribute reconstruction loss.

Discrete attributes are reconstructed with cross-entropy losses, whereas continuous attributes use Smooth-L_1 loss with a unit transition threshold after normalization using source-training statistics. Reconstruction losses are averaged within each attribute type and then across the types present in the batch to obtain $\mathcal{L}_{\text{attr}}$.

Masked foreign-key recovery. For a positive relation (v_s, τ, v_d), the target connection and its reverse edge are removed from the input context. Retrieved records that directly reveal the same connection are also excluded.

Negative destinations are sampled from the destination table specified by τ, subject to temporal eligibility. Known positive destinations, identified using permitted training information, are excluded from the negative set $\mathcal{V}_{\text{neg}}$. The candidate set is

$$\mathcal{V}_{\text{cand}} = \{v_d\} \cup \mathcal{V}_{\text{neg}}. \tag{21}$$

Using the relation prediction scores normalized over $\mathcal{V}_{\text{cand}}$, the loss for an individual relation is

$$\ell_{\text{rel}}(v_s, \tau, v_d) = -\log p(v_d \mid v_s, \tau), \tag{22}$$

where the probability is normalized over $\mathcal{V}_{\text{cand}}$. The batch loss $\mathcal{L}_{\text{rel}}$ is the mean over eligible relation examples. The number of sampled negatives is specified in Table 8.

Joint optimization. The two objectives are combined using Eq. 5. Objective sampling controls the frequency of attribute- and relation-level examples, whereas λ_{rel} controls their relative loss contribution. The corresponding settings are reported in Table 8.

Algorithm 3 summarizes the training schedule.

Algorithm 3: Retrieval-Augmented Relational Pre-training

Input: Source training data; initialized memory $\mathcal{M}$; parameters θ; intervals $J_{\text{fb}}, J_{\text{mem}}$; loss and optimization settings.
Output: Selected parameters and associated memory state.

$\mathcal{C} \leftarrow \varnothing$; $\overline{\mathcal{L}} \leftarrow$ undefined.
for $b = 1, \ldots, B$ **do**
  Sample and mask a source batch $\mathcal{B}_b$; forward with memory to obtain predictions, retrieved records $\mathcal{R}$, and candidates $\mathcal{A}$.
  Compute $\mathcal{L}_b^{\text{with}}$ using the joint objective in Eq. 5.
  Every J_{fb} steps, evaluate the same batch without source memory, at the same parameters and without gradients; otherwise use the preceding loss average as reference (Eq. 13).
  Compute G_b (Eq. 14); take a gradient-clipped optimizer step on $\mathcal{L}_b^{\text{with}}$.
  Accumulate record feedback from $\mathcal{R}$, update candidate statistics from $\mathcal{A}$, and update the loss average as defined in Appendix C.5.
  Every J_{mem} steps, update the Relation Store, buffer, and indices using Algorithm 2.
end for
return the source-validation-selected checkpoint and its associated memory state.

Here, $\mathcal{R}$ includes retrieval scores, attention weights, and gate usage, and $\mathcal{A}$ contains newly observed relation records.

D Empirical Studies in Details

This section provides implementation details for the empirical studies introduced in Sec. 3. We focus on three aspects that are omitted from the main text for brevity: the construction of flattened and graph-based database representations, the construction and sampling of different relational contexts, and the implementation of the relation-aware Griffin variant used to study relational pre-training. Unless otherwise specified, all data processing follows the same temporal cutoffs as the corresponding downstream tasks, such that only information available before the prediction timestamp is accessible to the model.

D.1 Q1: How to Model Relational Databases?

Flattened representations. We construct task-specific flattened tables using two complementary pipelines: a manually designed pipeline and an agent-based pipeline implemented with the single-table flattening skills of RDBLearn. Both pipelines transform the original multi-table relational database into a single table in which each row corresponds to one prediction instance.

For the *manual flattening* pipeline, we first identify the target entity and prediction timestamp from the task definition. We then inspect the database schema and manually determine relational paths that may provide historical information for the target. Tables connected to the target entity through primary–foreign key relations are joined or aggregated according to their relational semantics. One-to-one or many-to-one relations are directly mapped to target-level attributes when appropriate, whereas one-to-many historical relations are summarized using aggregation operators. Numerical columns are summarized using statistics such as mean, minimum, maximum, standard deviation, sum, and count, while categorical columns are represented using frequency- or count-based statistics. For temporal tables, all aggregations are restricted to records whose timestamps precede the prediction timestamp. This process produces a fixed-dimensional feature vector for every prediction instance.

In addition to manual feature construction, we employ the single-table flattening skills provided by *RDBLearn* to construct flattened tables in an agent-based manner. Given the database schema, task specification, target entity, and temporal constraint, the agent analyzes candidate relational paths, identifies tables and attributes that can be connected to the target entity, and generates the corresponding join and aggregation operations automatically. The resulting relational information is converted into a task-specific single table in the same form as the manually flattened representation. Importantly, the agent operates only on the schema and data available before the prediction time and is not provided with validation or test labels when constructing features.

Using both manually designed and agent-generated flattened tables allows us to reduce the dependence of the empirical conclusion on a particular flattening implementation. The former represents conventional expert-driven feature engineering, whereas the latter represents an automated schema-aware flattening procedure.

Graph representations. For the graph-based setting, we directly preserve the relational structure of the original database. Each row in a relational table is represented as a node whose type is determined by its source table, while each primary–foreign key reference is converted into a typed edge between the corresponding row nodes. Node attributes are constructed from the non-key columns of each row, while primary and foreign keys are represented through graph connectivity rather than treated as ordinary predictive features.

For every prediction instance, the target row is used as the root node and its historical relational neighborhood is extracted to form the model input. We enforce the same temporal constraint as in the flattened setting: nodes and relations that become available after the prediction timestamp are excluded from the graph. Therefore, the flattened and graph-based pipelines have access to the same historical database state, and differ primarily in whether multi-table information is compressed into a single feature vector or retained explicitly as relational structure.

For the comparison in Q1, we evaluate the resulting representations on Rel-F1 Driver-DNF, Rel-Stack User-Badge, and Rel-Amazon User-Churn using AUROC. We additionally record the end-to-end preprocessing and training cost of each pipeline. For flattened representations, this includes relational path processing and feature aggregation, while for graph representations it includes graph construction, temporal filtering, and subgraph sampling.

D.2 Q2: How to Represent Relational Graphs?

The second empirical study examines how the composition of relational context affects downstream prediction. Starting from the full temporally valid graph described above, we construct three types of contexts: *task-relevant context*, *task-irrelevant context*, and *general context*. The construction procedure is illustrated in Fig. 7.

Task-relevant context. For each downstream task, we manually inspect its prediction target together with the database schema and identify relation types whose historical records have a direct semantic connection to the target. We then retain the corresponding relation types and relational paths and instantiate them as a task-specific subgraph around each target entity.

For example, a relation is considered task-relevant when it connects the target entity to historical states, interactions, or outcomes that can reasonably provide evidence for the prediction target. The

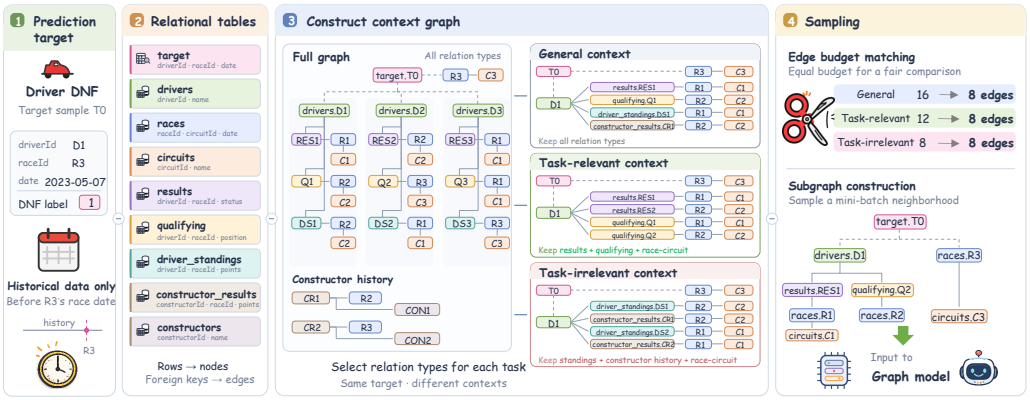


Figure 7: Detailed construction of relational contexts used in the empirical study. Starting from the temporally valid relational graph, we construct task-relevant, task-irrelevant, and randomly sampled contexts under a matched context budget.

selection is performed at the level of relation types and schema paths rather than individual labels or test instances, so the same context construction rule is applied consistently to all examples of a task.

Task-irrelevant context. We additionally identify relational information that is valid and available at prediction time but is not directly associated with the prediction objective. These relations are also manually selected at the schema level and converted into subgraphs using the same construction procedure as the task-relevant context. We refer to these subgraphs as task-irrelevant contexts.

The purpose of this setting is not to introduce artificial noise, but to distinguish between relational information that is structurally connected to the target entity and relational information that is specifically informative for the current task. Consequently, task-relevant and task-irrelevant contexts may both originate from legitimate historical relations in the same database, while differing in their semantic relevance to the target prediction.

General context. As a third reference, we construct random relational contexts without using task-specific relation selection. Given a target entity, we randomly sample historical subgraphs from the temporally valid relational neighborhood. Because the result of a single random sample may depend strongly on the sampled nodes and edges, we repeat random subgraph sampling multiple times using different random seeds. Model performance is evaluated separately on these samples and the resulting scores are averaged to obtain the reported random-context performance. This reduces the variance caused by an individual random subgraph realization.

Context-size matching. A direct comparison between the three contexts can otherwise be confounded by the amount of information exposed to the model. For example, a context with more nodes or edges may benefit simply from a larger input rather than from more informative relational structure. We therefore explicitly control the context budget across all settings.

For each task, the task-relevant, task-irrelevant, and general contexts are sampled to the same predefined graph budget. In practice, we control the number of retained relational records and edges and apply the same sampling depth and input-size constraint whenever applicable. If a candidate context contains more relational records than the budget, it is subsampled before being passed to the model. Contexts containing fewer records are not supplemented with additional relation types from another category.

As a result, models compared in this study are trained with approximately the same amount of relational evidence and under the same optimization setting. The comparison therefore primarily reflects differences in the *composition and relevance* of relational context rather than differences in graph size or training-data volume.

D.3 Q3: How to Take Advantage of Relational Graphs?

The third empirical study modifies Griffin to examine whether explicitly learning primary–foreign key dependencies during pre-training provides additional transferable relational knowledge. We keep the Griffin architecture, input encoder, message-passing backbone, and downstream fine-tuning procedure unchanged, and modify only its pre-training objective.

Original Griffin objective. Griffin performs completion pre-training by randomly masking a cell and predicting its representation from the remaining attributes of the row. For a row $\mathbf{x}_i$ with masked attribute j, let $\mathbf{h}_{i,j}$ denote the representation predicted by Griffin and $\mathbf{e}_{i,j}$ denote the encoded representation of the original attribute value. The original completion objective can be written as

$$\mathcal{L}_{\text{comp}} = 1 - \cos\left(\mathbf{h}_{i,j}, \mathbf{e}_{i,j}\right). \tag{23}$$

This objective encourages the model to reconstruct attribute information, but it does not directly require the model to recover the relational links connecting rows across tables.

Relation-aware pre-training. To explicitly introduce relational supervision, we augment the original completion objective with a primary–foreign key recovery objective. During pre-training, a subset of observed relational links is masked. Given two row representations $\mathbf{h}_u$ and $\mathbf{h}_v$ associated with a candidate relation type r, the model predicts whether the two rows are connected by the corresponding primary–foreign key relation.

For every masked positive relation (u, v, r), we construct negative pairs by replacing one endpoint with type-compatible rows that are not connected by relation r. A relation-specific scoring function is applied to each candidate pair,

$$s_{uv}^{(r)} = \text{MLP}_r\left([\mathbf{h}_u \| \mathbf{h}_v]\right), \tag{24}$$

where $\|$ denotes concatenation. The relational objective is then defined using binary cross-entropy over positive and negative relation pairs,

$$\mathcal{L}_{\text{rel}} = -\frac{1}{|\mathcal{P}| + |\mathcal{N}|}\left[\sum_{(u,v,r)\in\mathcal{P}} \log \sigma(s_{uv}^{(r)}) + \sum_{(u,v,r)\in\mathcal{N}} \log\left(1 - \sigma(s_{uv}^{(r)})\right)\right], \tag{25}$$

where $\mathcal{P}$ and $\mathcal{N}$ denote the positive and negative relation sets, respectively.

The resulting relation-aware pre-training objective is

$$\mathcal{L}_{\text{pre}} = \mathcal{L}_{\text{comp}} + \lambda_{\text{rel}}\mathcal{L}_{\text{rel}}, \tag{26}$$

where λ_{rel} controls the contribution of relational reconstruction. This modification leaves Griffin's original attribute completion objective intact and introduces relational learning as an additional source of supervision.

Controlled comparison. To isolate the effect of the relational objective, we construct two Griffin variants. *Original Griffin* is trained using the unchanged completion pre-training objective, whereas *Relation-aware Griffin* is trained on the same pre-training databases with the additional relational loss above. Both variants use the same model architecture, parameter initialization scheme, data splits, subgraph sampler, optimization schedule, and downstream fine-tuning protocol. The additional relation prediction head is used only during pre-training and is discarded for downstream prediction.

After pre-training, the two checkpoints are independently fine-tuned on the same downstream training sets and evaluated on Rel-F1 Driver-DNF, Rel-Stack User-Badge, and Rel-Amazon User-Churn using AUROC. Consequently, the performance difference between the two variants primarily measures the effect of explicitly modeling relational dependencies during pre-training, rather than changes in model capacity, downstream supervision, or relational input structure.

E Experiment and Implementation in Details

E.1 Evaluation Protocols

Supervised adaptation. For supervised downstream tasks, RaMA uses the target training split to optimize the task-adaptation modules, and checkpoints are selected using the target validation split. Each evaluation uses the memory state associated with its selected pre-trained checkpoint, and query outcomes are withheld from model inputs.

E.2 Zero-shot, Few-shot, and In-context Evaluation

Leave-one-database-out zero-shot evaluation. For each unseen database, all of its records are excluded from pre-training, candidate collection, memory construction, memory updates, and source-side model selection. The remaining databases provide the complete pre-training and the fixed memory used at evaluation time. The left-out database contributes only the visible attributes and relations needed to build temporally valid local contexts. Zero-shot prediction uses the source-selected checkpoint and its memory without fitting or selecting parameters from target labels.

Few-shot fine-tuning. For each unseen target task, the support set contains 32, 64, or 128 labeled examples drawn from its permitted training split. These examples optimize only the task-adaptation modules; checkpoint selection follows the target validation protocol when available. The support set is disjoint from the evaluation queries, and all methods compared at the same shot count use the same sampled examples.

In-context learning. For each target task, we use the same 32, 64, or 128 labeled support examples as in the corresponding few-shot setting. Each support example is independently encoded by the frozen RaMA backbone using its own temporally valid relational context, with its target field excluded from the encoder input. The labeled support context is kept separate from the source memory. Model parameters and source memory remain fixed, and no gradient update is performed.

E.3 Reference Configuration

Tables 6–8 provide reference settings for RaMA, assembled from compatible training and memory configurations.

The attribute-to-relation sampling ratio is normalized over the two pre-training objectives. During few-shot fine-tuning, the training batch size is capped by the number of available support examples. The validation patience in Table 8 applies to supervised adaptation with a target validation split.

The implementation uses PyTorch, PyTorch Geometric, and PyTorch Frame, with datasets, tasks, and evaluation protocols from RelBench. Retrieval uses inner-product search over L2-normalized index vectors, with direct scoring for small local sets and FAISS IndexFlatIP for source memory.

E.4 Ablation and Sensitivity Studies

Component ablations. We conduct component ablations over the relational backbone, memory construction and maintenance, and the relation-aware pre-training objective. Each variant modifies the corresponding component while keeping the remaining settings unchanged, allowing us to isolate its contribution to RaMA. Table 9 summarizes the specific changes applied in each ablation.

Retrieval attention across additional datasets. Figure 8 extends the retrieval visualization in Fig. 4 to Rel-Avito, Rel-Stack, and Rel-Event. Rows denote target relation types and columns denote retrieved atomic-route categories. Let A_{ij} be the average retrieval attention assigned by queries of target relation i to retrieved category j. For a dataset with C retrieved categories, we report

$$H_{ij} = A_{ij} - \frac{1}{C} \sum_{j'=1}^{C} A_{ij'}. \tag{27}$$

Positive and negative values indicate above- and below-average attention within each target relation, respectively. The centering is used only for visualization and does not affect retrieval or fusion.

Table 6: Reference configuration for encoding, local context, and retrieval.

Parameter	Value
Encoding	
Hidden dimension	128
Semantic index dimension	128
Structural index dimension	128
Relational encoding layers	2
Maximum atomic-route length	2 edges
Dropout rate	0.1
Text embeddings	Frozen GloVe, 300 dimensions
Metadata hash vocabulary size	65,536
Local Context	
Neighbors sampled at each hop	128, 64
Temporal neighbor sampling	Uniform over eligible history
Maximum local routes per example	8
Retrieval	
Candidates from each semantic index	16
Candidates from each structural index	16
Selected local records	4
Selected source records	4
Aggregation temperature T	0.1
Similarity for index search	Normalized inner product

Table 7: Reference memory construction and maintenance settings. Memory and buffer capacities count records.

Parameter	Value
Construction	
Maximum sampled roots per table	1,024
Temporal sampling buckets	8
Maximum routes per root	16
Maximum rows per hop	128
Maximum rows per record	256
Initial record capacity per database	8,192
Record capacity per root table	4,096
Record capacity per route	2,048
Construction seed	42
Feedback and Admission	
Feedback evaluation interval J_{fb}	200 optimization steps
Relevance weight α	0.2
Learning-gain weight β	1.0
Loss moving-average coefficient ν	0.95
Utility moving-average coefficient μ	0.95
Candidate-buffer capacity	512
Minimum candidate observations $n_{\min}$	5
Minimum positive-feedback fraction $p_{\min}$	0.6
Minimum provisional utility $u_{\min}$	0.01
Maximum candidates considered per update K_{ins}	8
Maintenance	
Relation-memory capacity	8,192
Maintenance interval J_{mem}	50 optimization steps
Stale-record threshold	1,000 optimization steps
Utility decay factor δ	0.995
Near-duplicate similarity threshold	0.995

Table 8: Reference optimization and evaluation settings.

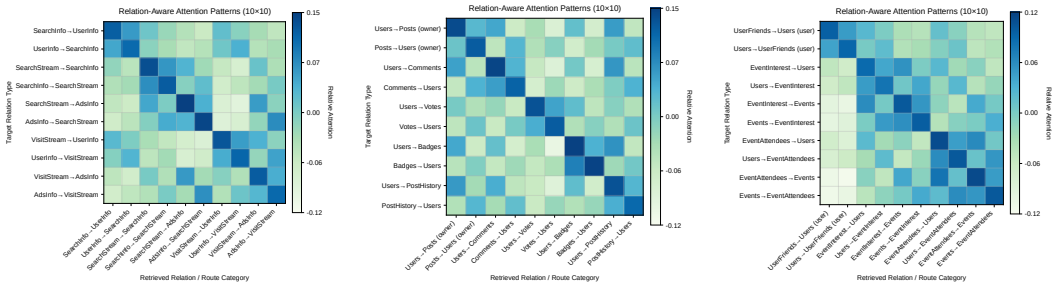
Parameter	Value
Shared Settings	
Optimizer	AdamW
Weight decay	0.01
Gradient clipping norm	1.0
Random seeds	17, 42, 73
Pre-training	
Training epochs	20
Training batch size	64
Validation batch size	128
Learning rate	3×10^{-4}
Attribute-to-relation sampling ratio	$5:4$
Relation loss weight λ_{rel}	1.0
Negative destinations per positive relation	16
Maximum categorical vocabulary size	4,096
Minimum category frequency	2
Task Adaptation	
Maximum training epochs	20
Training batch size	512
Evaluation batch size	1,024
Task-adapter learning rate	5×10^{-4}
Supervised validation patience	5 epochs
Relation-Ranking Evaluation	
Destinations per scoring block	4,096
Queries per scoring block	256

Table 9: Component changes in the ablation study.

Variant	Modification
Heterogeneous GNN	Replace the relation-oriented backbone with a heterogeneous GNN.
w/o Memory	Disable source-memory retrieval and fusion; preserve the target relational context.
Node-only Record Construction	Use entity-attribute records as retrieval units in place of route records.
Initialization w/o Candidate Buffer	Directly admit initial candidate records to the active memory, bypassing the candidate buffer.
Static Memory	Retain the initial records and disable reward-guided content updates.
Loss-based Only Reward	Set $\alpha = 0$ in the two-term utility, retaining the attributed learning-gain term.
w/o Relational Loss	Set $\lambda_{\text{rel}} = 0$, retaining attribute completion and the atomic-route relational backbone.

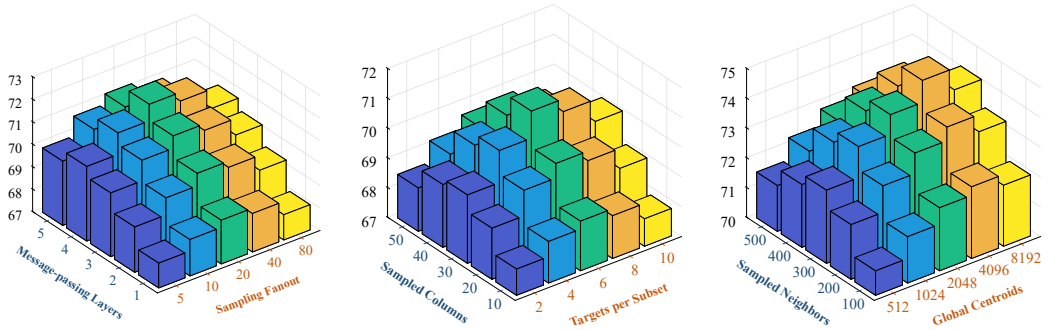
Hyperparameter sensitivity of representative RFMs. We further examine representative RFM baselines under their model-specific hyperparameters on Driver-DNF. For each model, we vary two key hyperparameters while keeping the remaining configuration fixed. We examine the number of layers and neighbor fanout for Griffin, the number of columns and targets for RDB-PFN, and the number of neighbors and global centroids for RelGT. Figure 9 shows that the evaluated models generally exhibit stable performance around their default configurations, while overly small or large settings lead to moderate degradation.

Memory capacity and retrieval depth. We jointly vary source-memory capacity and retrieval top-k while keeping the local context and other retrieval settings fixed. Memory capacity is expressed as a fraction of the full source memory. The reported sweep uses memory fractions $\{20, 40, 60, 80, 100\}\%$ and $k \in \{2, 4, 6, 8, 10\}$.



(a) Rel-Avito / User-Clicks (b) Rel-Stack / User-Badge (c) Rel-Event / User-Attendance

Figure 8: Retrieval attention heatmaps on three RelBench datasets. Values are row-centered relative attention scores.



(a) Griffin (b) RDB-PFN (c) RelGT

Figure 9: Hyperparameter sensitivity of representative relational foundation models on Driver-DNF.

F Theoretical Proofs in Details

We provide detailed proofs for the theoretical results in Sec. 4.4, followed by two additional propositions on temporal admissibility and gated fusion.

F.1 Proof of Bounded Retrieval Contamination

We first prove the result in Sec. 4.4. Consider a fixed query $\mathbf{q}$ and the records participating in the softmax retrieval aggregation. Let $\mathcal{C}^+$ and $\mathcal{C}^-$ denote the task-compatible and task-incompatible subsets, with $n_+ = |\mathcal{C}^+|$ and $n_- = |\mathcal{C}^-|$. For each record e_i, let $\mathbf{z}_i = \phi_e(e_i)$ denote its representation in the route space and $s_i = s(\mathbf{q}, e_i)$ its retrieval score. Assume $\|\mathbf{z}_i\|_2 \le B$ and

$$\min_{e_i \in \mathcal{C}^+} s_i - \max_{e_j \in \mathcal{C}^-} s_j \ge \gamma > 0. \tag{28}$$

Define the softmax normalizers over the compatible and incompatible subsets as

$$Z_+ = \sum_{e_i \in \mathcal{C}^+} \exp(s_i/\tau), \qquad Z_- = \sum_{e_j \in \mathcal{C}^-} \exp(s_j/\tau). \tag{29}$$

The total attention mass assigned to incompatible records is therefore

$$p_- = \frac{Z_-}{Z_+ + Z_-}. \tag{30}$$

Let $s_-^{\max} = \max_{e_j \in \mathcal{C}^-} s_j$. By Eq. 28, every compatible record satisfies $s_i \ge s_-^{\max} + \gamma$. Hence,

$$Z_+ \ge n_+ \exp\left(\frac{s_-^{\max} + \gamma}{\tau}\right), \qquad Z_- \le n_- \exp\left(\frac{s_-^{\max}}{\tau}\right). \tag{31}$$

Substituting these bounds into Eq. 30 gives

$$p_- \leq \frac{n_-}{n_- + n_+ \exp(\gamma/\tau)}. \tag{32}$$

Let $\mathbf{e}_\rho^{\mathrm{ret},+}$ and $\mathbf{e}_\rho^{\mathrm{ret},-}$ denote the softmax-normalized aggregations within $\mathcal{C}^+$ and $\mathcal{C}^-$, respectively. The full retrieved representation can be written as

$$\mathbf{e}_\rho^{\mathrm{ret}} = (1 - p_-)\mathbf{e}_\rho^{\mathrm{ret},+} + p_- \mathbf{e}_\rho^{\mathrm{ret},-}. \tag{33}$$

Because both subset representations are convex combinations of vectors satisfying $\|\mathbf{z}_i\|_2 \leq B$, we have $\|\mathbf{e}_\rho^{\mathrm{ret},+}\|_2 \leq B$ and $\|\mathbf{e}_\rho^{\mathrm{ret},-}\|_2 \leq B$. Therefore,

$$\begin{aligned} \left\|\mathbf{e}_\rho^{\mathrm{ret}} - \mathbf{e}_\rho^{\mathrm{ret},+}\right\|_2 &= p_- \left\|\mathbf{e}_\rho^{\mathrm{ret},-} - \mathbf{e}_\rho^{\mathrm{ret},+}\right\|_2 \\ &\leq 2Bp_- \\ &\leq 2B\frac{n_-}{n_- + n_+ \exp(\gamma/\tau)}. \end{aligned} \tag{34}$$

Thus, the contribution of task-incompatible records decreases exponentially with the compatibility margin γ/τ. If $\mathcal{C}^-$ is empty, the deviation is identically zero.

F.2 Proof of Relational Information Preservation

We next prove Proposition 4.2 in Sec. 4.4. Let R denote the true masked foreign-key destination within the candidate set, C the observed relational context, and Z the representation used for relation recovery. Let $p(R \mid Z, C)$ denote the true conditional distribution and $q_\phi(R \mid Z, C)$ the model prediction.

The relation recovery loss is the expected conditional cross-entropy

$$\mathcal{L}_{\mathrm{rel}} = \mathbb{E}[-\log q_\phi(R \mid Z, C)]. \tag{35}$$

Using the standard cross-entropy decomposition,

$$\begin{aligned} \mathcal{L}_{\mathrm{rel}} &= H(R \mid Z, C) + \mathbb{E}_{Z,C}\left[D_{\mathrm{KL}}\left(p(\cdot \mid Z, C) \| q_\phi(\cdot \mid Z, C)\right)\right] \\ &\geq H(R \mid Z, C), \end{aligned} \tag{36}$$

where the inequality follows from the non-negativity of KL divergence.

Therefore,

$$\begin{aligned} I(Z; R \mid C) &= H(R \mid C) - H(R \mid Z, C) \\ &\geq H(R \mid C) - \mathcal{L}_{\mathrm{rel}}. \end{aligned} \tag{37}$$

Since $H(R \mid C)$ is fixed, minimizing $\mathcal{L}_{\mathrm{rel}}$ increases this lower bound, encouraging Z to retain relational information about R conditioned on C.

F.3 Temporal Admissibility

Proposition. If graph construction, memory construction, retrieval, and masking use only information permitted before prediction time t, then all representations produced by RaMA depend only on temporally admissible information.

Proof. All inputs to encoding and retrieval satisfy the temporal and masking constraints defined in Appendix C.3. Subsequent scoring, fusion, attention, and aggregation are deterministic functions of these inputs and model parameters; therefore, no future or withheld target information can enter the resulting representations.

F.4 Bounded Gated Fusion

Proposition. Let $\mathbf{x} = \mathbf{e}_r^{\mathrm{loc}}$ and $\mathbf{y} = \mathbf{e}_r^{\mathrm{ret}}$ denote the local and retrieved route representations in Eq. 3. Write their fusion as $F(\mathbf{x}, \mathbf{y}) = \mathbf{g} \odot \mathbf{x} + (1 - \mathbf{g}) \odot \mathbf{y}$, where $\mathbf{g} = \sigma(A([\mathbf{x} \| \mathbf{y}]))$. For every coordinate j, F_j lies between x_j and y_j. If $\|\mathbf{x}\|_\infty, \|\mathbf{y}\|_\infty \leq B$, then $\|F(\mathbf{x}, \mathbf{y})\|_\infty \leq B$. For fixed $\mathbf{x}$ and network parameters, if A is L_A-Lipschitz in its second input under the infinity norm, then

$$\|F(\mathbf{x}, \mathbf{y}') - F(\mathbf{x}, \mathbf{y})\|_\infty \leq \left(1 + \frac{L_A}{4}\|\mathbf{x} - \mathbf{y}\|_\infty\right)\|\mathbf{y}' - \mathbf{y}\|_\infty. \tag{38}$$

Proof. The sigmoid gives $0 \le g_j \le 1$, so F_j is a convex combination of x_j and y_j, proving the first two statements. Let $\mathbf{g}' = \sigma(A([\mathbf{x}\|\mathbf{y}']))$. Since the derivative of the sigmoid is at most $1/4$,

$$\|\mathbf{g}' - \mathbf{g}\|_\infty \le \frac{L_A}{4}\|\mathbf{y}' - \mathbf{y}\|_\infty.$$

Expanding the difference gives

$$F(\mathbf{x}, \mathbf{y}') - F(\mathbf{x}, \mathbf{y}) = (1 - \mathbf{g}') \odot (\mathbf{y}' - \mathbf{y}) + (\mathbf{g}' - \mathbf{g}) \odot (\mathbf{x} - \mathbf{y}). \tag{39}$$

The triangle inequality and $\|1 - \mathbf{g}'\|_\infty \le 1$ yield Eq. 38.

F.5 Computational Complexity

Let N be the number of indexed source records, d the representation dimension, Q the number of retrieval queries, C the number of merged candidates per query, k the number of selected records, and P the number of local route instances. In our implementation, $Q = P$.

Retrieval and relational computation. Exact retrieval over normalized source indices costs $O(QNd)$ in the worst case. Reranking the merged candidates costs $O(QCd)$, followed by $O(QC \log C)$ sorting. Projection and aggregation of the selected records cost $O(Qkd^2)$ and $O(Qkd)$, respectively. Route encoding, fusion, and destination attention contribute $O(Pd^2)$, in addition to the sampled graph encoding cost T_{graph}.

Feedback and storage. The memory-free reference forward pass performed every J_{fb} steps adds an average per-step computational cost of $1/J_{\text{fb}}$ of a memory-free forward computation. The two source indices require $O(Nd)$ storage; candidate representations require $O(QCd)$ workspace, while exhaustive score materialization requires $O(QN)$ workspace and can be reduced through blockwise processing.

G Limitations and Broader Impacts

Scope of transfer. Transfer depends on whether source memory covers target-relevant attribute semantics and relational patterns. Schema mismatches, ambiguous metadata, and unseen or rare relation compositions may reduce retrieval relevance. Larger relational corpora and more adaptive memory construction could improve robustness across broader schema distributions.

Memory and learning feedback. Memory effectiveness is constrained by storage capacity and retrieval depth. Utility estimates rely on batch-level feedback and jointly retrieved records, making credit assignment imperfect and potentially underrepresenting rare relations. The current update mechanism also does not explicitly model the long-term effects of accumulated knowledge. More advanced continual-learning strategies may improve the stability of memory updates.

Data quality and interpretation. Temporal validity depends on reliable key relationships and timestamps. Noisy schemas, incomplete foreign-key constraints, and inconsistent data maintenance may therefore affect model reliability. Moreover, provenance and attention weights improve inspectability but do not constitute mechanistic explanations of individual predictions.

Broader impacts. Cross-database memory reuse must respect source-data permissions and access restrictions, since stored attributes and relationships may reveal sensitive information even without direct identifiers. Deployment in high-impact domains also requires careful evaluation of prediction errors, privacy, fairness, and transparent usage policies.

H Details of Baseline Hyperparameters

H.1 Hyperparameters for Graph-based Baselines

To ensure a fair comparison with graph-based relational learning methods, we summarize the hyperparameters in our experiments used for graph-based relational models including GraphSAGE, Graph Transformer, GraphMAE, and RDL-GNN. Their model dimensions, neighborhood sampling strategies, optimization parameters, and training settings are reported below.

Table 10: Reference hyperparameters for heterogeneous graph baselines.

Hyperparameter	Value	Description
GraphSAGE		
Hidden dimension	128	Hidden size of node representations.
Message-passing layers	2	Number of GraphSAGE layers.
Neighbor fanout	$[128, 64]$	Sampled neighbors at the first and second hop.
Aggregator	mean	Neighborhood aggregation operator.
Dropout	0.1	Dropout applied to hidden representations.
Activation	ReLU	Non-linear activation between GraphSAGE layers.
Batch size	512	Training mini-batch size.
Learning rate	1×10^{-3}	Learning rate for downstream training.
Weight decay	1×10^{-4}	L2 regularization coefficient.
Optimizer	Adam	Optimizer used for supervised training.
Graph Transformer		
Hidden dimension	128	Transformer node-state dimension.
Attention heads	4	Number of multi-head attention heads.
Transformer layers	2	Number of graph-transformer blocks.
Feed-forward dimension	256	Hidden dimension of the transformer feed-forward sublayer.
Neighbor fanout	$[128, 64]$	Two-hop temporal-safe neighborhood sampling.
Attention dropout	0.1	Dropout applied to attention weights.
Hidden dropout	0.1	Dropout applied to hidden representations.
Activation	GELU	Activation used in the feed-forward sublayer.
Residual connections	enabled	Residual connections around attention and feed-forward blocks.
Normalization	LayerNorm	Normalization used in transformer blocks.
Learning rate	3×10^{-4}	AdamW learning rate.
Weight decay	1×10^{-4}	L2 regularization coefficient.
Batch size	256	Training mini-batch size.
Mask token	learned	Learnable token replacing masked node features.
Relation encoding	attention bias	Relation-type information injected into attention scores.
GraphMAE		
Hidden dimension	512	Latent encoder/decoder dimension.
Encoder / decoder	GAT / GAT	Graph encoder and reconstruction decoder.
Encoder / decoder layers	2 / 1	Number of graph layers in the encoder and decoder.
Encoder / decoder heads	4 / 1	Number of attention heads in the encoder and decoder.
Mask token	learned	Learnable token replacing masked node features.
Mask rate	0.5	Fraction of node features masked for pre-training.
Replace rate	0.05	Fraction of masked features replaced by noisy features.
Input / attention dropout	0.2 / 0.1	Input-feature and attention dropout rates.
Activation	PReLU	Non-linear activation in the graph encoder.
Reconstruction loss	SCE	Scaled cosine error used for masked feature reconstruction.
SCE exponent α	2	Exponent controlling the scaled cosine reconstruction loss.
Learning rate	10^{-3}	Adam learning rate.
Weight decay	2×10^{-4}	L2 regularization coefficient.
Optimizer / schedule	Adam / cosine	Optimizer and learning-rate decay.
RDL-GNN		
Hidden dimension	128	Hidden dimension in the RelBench heterogeneous GNN.
GNN layers	2	Message-passing depth.
Neighbor fanout	$[128, 64]$	Fanout obtained from base neighbor count 128 over two hops.
Aggregation	sum	Cross-relation aggregation operator.
Activation	ReLU	Non-linear activation between GNN layers.
Prediction head	2-layer MLP	Task-specific prediction module on the target representation.
Batch size	512	Training mini-batch size.
Learning rate	5×10^{-3} (entity); 10^{-3} (relation)	Learning rates for entity-level and relation-prediction tasks.
Temporal sampling	uniform	Sampling among temporally eligible historical neighbors.
Optimizer	Adam	Optimizer used for supervised training.
Relation transformations	shared	Shared message transformation across relation types.

H.2 HYPERPARAMETERS FOR SINGLE-TABLE BASELINES

We report the hyperparameter configurations used for the single-table baselines in our experiments, including gradient boosting models and tabular foundation models. These configurations cover tree-based optimization parameters as well as ensemble and inference settings for TabICL and TabPFN.

Table 11: Reference hyperparameters for single-table baselines.

Hyperparameter	Value	Description
XGBoost		
Booster	`gbtree`	Tree-based gradient boosting.
Boosting rounds	500	Maximum number of trees.
Learning rate	0.05	Shrinkage factor for each boosting step.
Maximum depth	6	Maximum tree depth.
Minimum child weight	1	Minimum Hessian mass required in a child.
Minimum split loss	0.0	Minimum loss reduction required for a split (`gamma`).
Subsample	0.8	Fraction of training rows sampled per tree.
Column subsample	0.8	Fraction of features sampled per tree.
L1 regularization	0.0	Regularization coefficient `reg_alpha`.
L2 regularization	1.0	Regularization coefficient `reg_lambda`.
Tree construction	`hist`	Histogram-based tree construction algorithm.
Early-stopping patience	50 rounds	Stop when validation performance does not improve.
Maximum bins	256	Maximum number of histogram bins used by the `hist` tree method.
Column subsample per level	1.0	Fraction of available features sampled at each tree level.
Column subsample per node	1.0	Fraction of available features sampled for each split.
Positive-class weight	1.0	Relative weight assigned to positive examples in binary classification.
Feature shifting	`shuffle`	Feature permutation strategy across ensemble members.
Class shifting	`shuffle`	Class permutation strategy across ensemble members.
LightGBM		
Boosting type	`gbdt`	Standard gradient boosting decision trees.
Boosting rounds	500	Maximum boosting iterations.
Learning rate	0.05	Shrinkage factor.
Number of leaves	31	Maximum leaves in each tree.
Maximum depth	-1	No explicit depth limit.
Minimum data in leaf	20	Minimum examples per leaf.
Maximum bins	255	Maximum number of discrete bins for continuous features.
Feature fraction	0.8	Fraction of features sampled per tree.
Bagging fraction / frequency	0.8 / 1	Row subsampling fraction and frequency.
Minimum split gain	0.0	Minimum gain required to perform a split.
L1 regularization	0.0	Leaf-score L1 regularization coefficient.
L2 regularization	1.0	Leaf-score L2 regularization coefficient.
Early-stopping patience	50 rounds	Validation-based early stopping.
Minimum Hessian in leaf	10^{-3}	Minimum sum of Hessians required in each leaf.
Minimum data per bin	3	Minimum number of examples assigned to a histogram bin.
Samples for bin construction	200,000	Maximum number of samples used to construct histogram bins.
TabICL v1.1		
Ensemble members	32	Number of transformed in-context ensemble views.
Normalization methods	none, power	Input normalization ensemble.
Feature shuffling	Latin	Latin-square feature permutations.
Class shifting	enabled	Cyclic class-label shifts for ensembling.
Outlier threshold	4.0	Z-score clipping threshold.
Softmax temperature	0.9	Temperature for predictive logits.
Average logits	true	Average ensemble logits rather than probabilities.
Hierarchical classification	enabled	Enable hierarchical decomposition for many-class tasks.
Inference batch size	8	Number of ensemble members processed jointly.
Automatic mixed precision	enabled	Use mixed-precision inference on compatible accelerators.
TabPFN v2.5		
Ensemble members	8	Number of estimators used by the v2.5 preset.
Automatic ensemble scaling	enabled	Increase ensemble size when needed for feature coverage.
Softmax temperature	0.9	Classification temperature in the v2.5 preset.
Inference precision	`auto`	Automatically select an appropriate inference precision.
Maximum feature	500	Maximum number of features processed by each ensemble estimator.
Categorical threshold	30	Maximum number of unique numerical values for categorical inference.
Numerical threshold	4	Minimum number of unique values for treating a feature as numerical.
Minimum samples for categorical inference	100	Minimum dataset size required for conducting automatic categorical-feature inference.
Classification outlier threshold	12.0	Standard-deviation threshold used for outlier removal in classification.

H.3 HYPERPARAMETERS FOR RELATIONAL BASELINES

Finally, we report the hyperparameter configurations of existing relational foundation models. These settings include architecture parameters, pre-training and fine-tuning strategies, temporal sampling schemes, and optimization details for Griffin, Relational Transformer, RDB-PFN, Rel-LLM, and RelGT, ensuring reproducible comparisons with state-of-the-art relational learning approaches.

Table 12: Reference hyperparameters for relational foundation-model baselines.

Hyperparameter	Value	Description
	Griffin	
Hidden dimension	512	Shared representation dimension.
Message-passing layers	4	Number of relational message-passing layers.
Sampling hops / fanout	2 / 20	Downstream neighborhood depth and per-hop fanout.
Batch size	256	Fine-tuning batch size.
Learning rate	3×10^{-4}	Downstream fine-tuning learning rate.
Weight decay	2×10^{-4}	Fine-tuning L2 regularization.
Maximum epochs / patience	50 / 15	Training budget and early-stopping patience.
Reverse edges	enabled	Include reverse relational edges.
	Relational Transformer (RT)	
Transformer layers	12	Number of relational-transformer blocks.
Hidden dimension	256	Token hidden dimension.
Attention heads	8	Number of attention heads per layer.
MLP hidden dimension	1024	Gated SiLU feed-forward hidden size.
Context length	1024	Number of relational tokens per training context.
Pre-training steps / batch	50k / 256	Pre-training optimization steps and batch size.
Pre-training LR / weight decay	10^{-3} / 0.1	Peak AdamW learning rate and weight decay.
Warmup / decay	20% / linear	Linear warmup followed by linear decay to zero.
Fine-tuning steps / LR	33k / 10^{-4}	Downstream fine-tuning schedule.
Fine-tuning weight decay	0	No weight decay during fine-tuning.
	RDB-PFN	
Embedding dimension	96	Transformer embedding size.
Attention heads	4	Number of self-attention heads.
MLP hidden dimension	192	Feed-forward hidden dimension.
Transformer layers	6	Number of model layers.
Relational pre-training steps	2.2M	Number of stage-2 RDB pre-training steps.
Relational pre-training batch	1	Batch size in the released RDB pre-training config.
Relational pre-training LR	10^{-3}	Learning rate in the released RDB pre-training config.
Sampled columns	30	Columns sampled per relational synthetic subset.
Targets per subset	6	Prediction targets drawn from each subset.
Single-table initialization	75k steps, batch 8, LR 5×10^{-4}	Stage-1 initialization before relational pre-training.
	Rel-LLM	
Base LLM	Llama 3.2-1B	Language-model backbone reported in the paper.
LLM hidden dimension	2048	Hidden dimension of the Llama backbone.
GNN layers	2	Graph-encoder depth.
Graph hidden dimension	128	Output dimension of the graph encoder.
Dropout search	$\{0.1, 0.2, 0.3, 0.4, 0.6, 0.8\}$	Candidate GNN dropout values.
Learning-rate search	$\{10^{-4}, 5 \times 10^{-5}, 10^{-6}\}$	Candidate optimization rates.
Batch-size search	$\{16, 32, 128, 256, 512\}$	Candidate mini-batch sizes.
Temporal strategy	uniform or last	Candidate temporal neighborhood strategies.
Weight decay	0 or 1.5×10^{-4}	Adam or AdamW depending on selected value.
Focal-loss γ	2.0	Focusing parameter for classification.
Fine-tuning strategy	frozen or LoRA	LLM adaptation choices considered by Rel-LLM.
	RelGT	
Hidden dimension	512	Entity/token representation width.
Transformer layers	1	Default local RelGT depth in the released node-task script.
Attention heads	4	Number of attention heads.
Sampled neighbors	300	Maximum local sampled nodes.
Global centroids	4096	Number of learnable global centroid tokens.
Feed-forward / attention dropout	0.1 / 0.1	Dropout in feed-forward and attention blocks.
Learning rate	10^{-4}	Training learning rate.
Weight decay	10^{-5}	L2 regularization coefficient.
Batch size	512	Default mini-batch size.
Warmup steps	1000	Learning-rate warmup steps.
Epochs	10	Default maximum training epochs.
Temporal sampling	uniform	Temporal sampling strategy.