Large-Scale Pretraining unlocks Few-Shot Prediction for Relational Data

Anonymous Authors[1]

Abstract

The ability to learn a new task from a few examples has been crucial to the success of foundation models such as large language models (LLMs). However, current foundation models for structured data like tables and relational databases still require tens of thousands of labeled examples to perform well on new tasks. Here we show that when pretrained at scale with the right recipe, Relational Transformers (RTs) can make state-of-the-art (SoTA) predictions with only hundreds of labels. This capability arises from the following ingredients: First, we assemble THE JOIN, the largest pretraining corpus of relational data to date, comprising 6k forecasting tasks across 650 real-world databases from diverse domains. Second, we develop a pretraining recipe that combines (1) mixed context sizes for multi-scale learning, (2) multi-cell masking for dense supervision, and (3) a novel random-walk-based retriever to efficiently gather relevant context. Third, we identify context ensembling and context tuning as complementary axes for scaling test-time compute to further improve few-shot performance. Pretrained with our recipe on THE JOIN, RT achieves parity with LLM Agent + TabICLv2 and RDBLearn + TabICLv2 pipelines with 32–23$\times$ fewer labels respectively, and even surpasses the prior SoTA of full task-specific training, on average nMAE for RelBench regression tasks. Context ensembling and tuning improves this further by up to 3%, and 4% respectively. Our ablations highlight the importance of schema semantics, multi-cell masking, and random-walk retrieval. Overall, our work paves the way for developing relational foundation models with strong few-shot capabilities.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

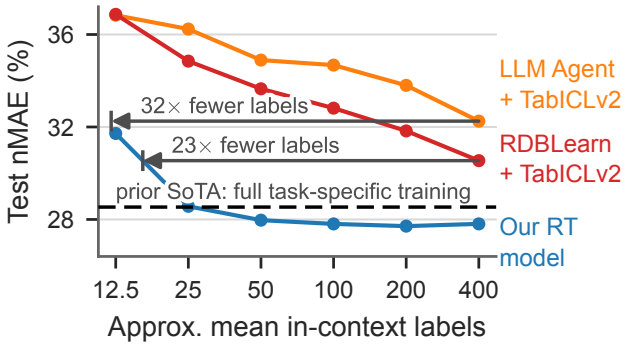


Figure 1. Z-score nMAE for in-context regression averaged over 9 tasks from RelBench, which was never trained on. For any point on the x-axis, all curves see the exact same set of labels, with label count $\lesssim$ the x-axis value on average (hence "Approx."). Our pretrained Relational Transformer (RT) matches the strongest in-context learning baselines using $\approx$ 23–32$\times$ fewer labels, and surpasses prior fully-supervised SoTA (RelGNN (Chen et al., 2025)).

1. Introduction

Foundation models have changed how machine learning systems are adapted to new tasks (Bommasani et al., 2022; Zhou et al., 2024). In language and vision, pretrained models can often solve new tasks from only a small number of examples provided in context (Brown et al., 2020; Alayrac et al., 2022). This few-shot paradigm is central to their practical value: users need not train a new model from scratch whenever the task, domain, or label distribution changes. Yet this capability remains largely out of reach for relational data, even though relational databases and data warehouses store much of the world's structured enterprise, scientific, and operational data (Vogel et al., 2022; Fey et al., 2024). Today, strong relational predictors typically require task-specific training (Robinson et al., 2024; Chen et al., 2025; Dwivedi et al., 2025a; Peleška & Šír, 2024), substantial labeled data (Fey et al., 2025; Hudovernik et al., 2026), or careful feature engineering (Chen & Guestrin, 2016; Lam et al., 2021; Shwartz-Ziv & Armon, 2022). This limits predictive modeling in exactly the settings where it would be most useful: when labels are scarce, tasks change frequently, or users need rapid ad hoc predictions.

Recent work has made substantial progress toward foundation models for relational prediction, but existing ap-

proaches remain mismatched to few-shot deployment. Relational deep learning models such as RelBench (Robinson et al., 2024), RelGNN (Chen et al., 2025), and the Relational Graph Transformer (Dwivedi et al., 2025a) operate directly on multi-table databases by representing rows as nodes and primary–foreign-key links as edges, but they are usually trained or fine-tuned for each schema and task. TabPFN (Hollmann et al., 2023; 2025) and TabICLv2 (QU et al., 2025; Qu et al., 2026) provide in-context learning for single-table prediction, and systems such as RDBLearn (Zhang et al., 2026) extend them to relational databases by aggregating relational neighborhoods into engineered features. This can be effective with many labeled rows, but it flattens the database and removes the explicit relational structure needed to reason over entities, links, and multi-table dependencies. More recent schema-agnostic relational foundation models, including Griffin (Wang et al., 2025), KumoRFM (Fey et al., 2025), KumoRFM-2 (Hudovernik et al., 2026), and the Relational Transformer (RT) (Ranjan et al., 2026), pretrain across databases and adapt to new schemas at inference time. However, these models have not yet been optimized for demonstrating few-shot behavior.

We argue that this label inefficiency is not primarily an architectural limitation, but a mismatch between existing relational foundation-model recipes and the demands of few-shot prediction. RT provides a natural starting point because it operates directly over database cells, schema metadata, and relational links. However, the original RT recipe was not designed to turn the architecture into a strong in-context learner. Three gaps are central. First, public relational pretraining data remains narrow, leaving rare-event, cold-start, and heavy-tailed regimes sparsely represented (Gu et al., 2026; Motl & Schulte, 2015). Second, existing masked-cell pretraining provides sparse supervision and trains at a single short context length, while few-shot inference relies on longer contexts containing many labeled examples. Third, standard context construction either ignores relational structure or expands local neighborhoods naively, without preferentially retrieving rows that are both relevant to the target and useful as in-context supervision.

In this paper, we show that a pretrained RT (Ranjan et al., 2026) can become a strong few-shot predictor when its data, training, retrieval, and test-time-compute recipe is designed for this regime. Our goal is not to introduce a new backbone architecture, but to identify the ingredients that turn an RT-style model into an effective in-context learner for relational databases. First, we assemble THE JOIN, the largest open pretraining corpus of relational data to date, comprising 6,255 forecasting tasks across 650 real-world databases from 63 domains, with deliberate coverage of rare-event, cold-start, and heavy-tailed regimes. Second, we introduce a pretraining recipe that aligns training with few-shot inference by combining mixed context sizes with multi-cell masking, which exposes the model to long contexts while providing dense supervision within each window. Third, we propose a random-walk-based context retriever that fills the context window with rows selected for both relational proximity to the target and likelihood of carrying useful label signal. Finally, we study context ensembling and context hyperparameter tuning as two complementary forms of test-time compute that improve predictions without changing the pretrained model.

Together, these ingredients yield strong few-shot prediction on relational data (Fig. 1). At an 8k-cell context, which admits roughly 360 in-context labeled task rows on average across the RelBench evaluation suite, our model matches RDBLearn + TabICLv2 and LLM Agent + TabICLv2 using 23× and 32× fewer in-context labels, respectively. The same model reaches the prior fully-supervised state-of-the-art on RelBench regression normalized mean absolute error (nMAE) with ∼24 in-context labels and surpasses it with ∼46, all without task-specific training. Context ensembling and context tuning provide further relative improvements at inference, while ablations show that schema semantics, multi-cell masking, and random-walk retrieval each contribute meaningfully.

Our results suggest that few-shot relational prediction can be unlocked not by changing the backbone alone, but by matching the entire foundation-model recipe to the few-shot regime: broad relational pretraining data, dense context-level supervision, retrieval that exposes useful labels and evidence, and scalable test-time compute. This is a step toward relational foundation models that work for databases the way language models work for text: pretrained once, reused across domains, and adapted to new tasks from only a small number of examples.

2. Background

2.1. Relational databases and relational structure

A *relational database* consists of multiple tables whose rows represent entities, events, or interactions. Each row is a collection of *cells*, one per column. Columns store either attributes, such as numerical, categorical, textual, or temporal values, or identifiers that define how rows are linked. *Primary keys* uniquely identify rows within a table, while *foreign keys* reference rows in other tables, inducing a multi-table relational structure (Fey et al., 2024; Robinson et al., 2024).

2.2. Relational Transformer

Tokenization and semantics The Relational Transformer (RT) (Ranjan et al., 2026) is the schema-agnostic architecture our work builds on. RT tokenizes relational databases

at the cell level: each database cell is a token whose embedding combines the cell value, datatype-specific information, and frozen text embeddings of the corresponding table and column names. This places feature cells, foreign-key cells, temporal cells, and target cells in a shared representation space, while allowing the model to be independent of a fixed schema.

Attention mechanism RT employs custom attention masks based on the relational structure of the cell tokens. Each transformer block applies three attention layers in sequence: *column attention*, where cells attend within the same column; *feature attention*, where cells attend within the same row; and *neighbor attention*, where cells attend across rows connected by primary–foreign-key links. This enables reasoning across rows, columns, and tables without flattening the database.

Masked token prediction RT expresses downstream tasks as masked-token prediction. Each task is associated with a *task table*, whose rows define prediction instances and whose target column stores the values to be predicted. A *label* is the value of this target column for a given task-table row. At inference time, this target cell is masked and predicted from the surrounding unmasked database context. RT was originally pretrained using single-cell masking at a fixed context length (Ranjan et al., 2026).

3. THE JOIN: A Large-Scale Pretraining Corpus of Relational Data

Existing collections of relational databases, such as RelBench (Robinson et al., 2024; Gu et al., 2026), comprise a limited set of datasets and manually curated predictive tasks. Although previous works have employed it for pretraining RT models (Ranjan et al., 2026; Kothapalli et al., 2026), the collection fails to capture the structural and distributional variety that real-world relational data exhibits. To address this issue, we introduce THE JOIN, a corpus comprising 650 relational databases across diverse domains (e-commerce, sports, media, finance, healthcare, etc.) and 6,255 forecasting tasks spanning various levels of difficulty, including predictive tasks for rare events, cold-start entities, and heavy-tailed targets. We follow a four-stage approach to curate the corpus: (1) collecting raw databases from the web, (2) standardisation of data types in tables, (3) task generation using LLMs, and (4) task filtering using an XGBoost baseline.

3.1. Collection

We collected the raw data from five disjoint source classes: (1) curated relational repositories (CTU, OGB, OpenAlex), (2) public APIs (US Treasury, SEC EDGAR, CFPB, FDA, World Bank, AACT, EIA), (3) domain portals (Lahman, ATP, StatsBomb, Cricsheet, OpenFlights, Stooq, MusicBrainz), (4) community dumps (Stack Exchange, GitHub-hosted datasets), and (5) benchmarks (Kaggle, Spider/BIRD). Every contributing source is either a bulk archive published under a permissive license or a public API whose published terms permit programmatic research use. The classes carry qualitatively different relational structures: CTU schemas are small, clean, and follow third-normal-form, whereas Stack Exchange dumps are wide and denormalised. As a preliminary quality filter, we only collect databases that have at least two tables joined by a foreign key, and at least one column that is or can be parsed as a timestamp. Table 2 lists the domains included in the corpus.

3.2. Standardization

The databases collected in their raw form are usually messy, with inconsistencies across column data types, timestamps, and primary-foreign key links being the major issues. For example: natively boolean columns are represented as strings `"true"/"false"`, dates are represented as ISO strings, and foreign keys exist only as a column-naming convention with no explicit declaration. To address such issues, we apply six idempotent detector-and-transformer fixes that recover the canonical relational structure: (1) *boolean coercion*, rewriting string and `uint8` columns whose values fall within a canonical truth-set as Arrow booleans; (2) *numeric coercion*, casting string-encoded numerics to integers or floats with an identifier blacklist guarding zip codes and ISBNs; (3) *foreign-key inference* by name match plus an $\geq 80\%$ value-containment test against parent primary keys; (4) *temporal normalization*, electing a single time column per table from regex, name-hint, and integer-year candidates; (5) *entity-axis subsampling* of oversized databases, retaining all activity rows for sampled entities to preserve their full activity histories; and (6) *source-overlap exclusion*, removing databases whose upstream source coincides exactly with that of a downstream evaluation benchmark.

3.3. Task generation

Audit and eligibility. After the collection and standardization stages, we audit the databases to generate forecasting tasks. Each database is first parsed into a typed schema graph with two named roles: an *entity table* (whose primary key is referenced by a foreign key (FK) from another table, identifying a long-lived object such as a customer, player, or hospital) and an *activity table* (with both a timestamp column and an FK to an entity table, so each row is a timestamped event attributable to an entity). A database is marked as eligible for task generation if it has at least one entity table, at least one activity table, ≥ 5 unique timestamps in that activity table, and ≥ 100 rows. The first two conditions ensure the schema can express a per-entity forecasting task. The timestamp and row count conditions allow us to

perform XGBoost-based filtering of the generated tasks for quality control.

Propose and generate. Proposing forecasting tasks for a database requires subject matter expertise and a scalable verification process to evaluate the correctness. We use an LLM (Claude Opus) for the former and a lightweight deterministic checker to satisfy the latter requirement. For every eligible database, the LLM is given the schema description in text and is asked to write a set of task proposals in JSON. Each proposal picks one of the 25 registered template families spanning eight reasoning regimes: *aggregation* (volume and intensity of activity), *threshold/existence* (gating a continuous signal at a decision boundary), *trend/change* (direction and magnitude of change), *behaviour/churn* (entity lifecycle), *ranking* (relative position within a peer group), *adoption/novelty* (first-time events), *streak* (resilience as a single metric), and *tail-event* (deliberately constructed templates for rare and extreme regimes such as cold-start, rare-positive, and heavy-tail targets). The checker then iterates through each proposal and drops them if: (1) a referenced column does not exist, (2) the entity column is not a foreign key, or (3) the requested pattern is not one of the registered templates. The LLM proposer is run multiple times per database, where each pass sees the tasks already accepted and is asked specifically for templates that have less coverage. The valid proposals are finally turned into concrete forecasting tasks. Comprehensive details are provided in Appendix C.4.

3.4. Task filtering

Finally, we score each task using an XGBoost model over window-aggregation features. We retain tasks that are either *learnable*, where XGBoost beats the lightweight historical baselines, or *random*-tier, where the signal is too weak for these baselines but may still be useful for stronger models. Details on the XGBoost baseline and full filtering rules are given in Appendix C.5.

4. Pretraining Recipe

Here we describe our key improvements to the pretraining recipe from the original RT paper (Ranjan et al., 2026) which enable few-shot predictions and effective context scaling.

Mixed context sizes The context size L is a key axis to scale prediction quality. However, training at large L is expensive. Further, the same pretrained model should be able to handle both small and large L at test-time, so that inference compute can be traded off against quality. Hence, we train at a mix of context sizes. For each training batch, we sample $L \sim \text{Unif}(\mathcal{L})$. We keep batch size B constant irrespective of L for stable optimization with shared optimizer state and learning rate. Since GPU memory scales as $L \times B$, we use gradient accumulation adaptively to prevent out-of-memory (OOM) errors at large L.

Random-walk retrieval (Ranjan et al., 2026) samples the context via a bounded-width breadth-first search (BFS) over the primary–foreign-key links, starting from the target row. As L grows, BFS exhausts the local neighborhood and samples rows less likely to be relevant. We use a 2-stage context sampling algorithm. In the first stage, we run W walks of length K starting from the target row and record the visit counts of task table rows. We sort the task rows by (timestamp, visit count) if a *recency bias* parameter ρ is set to 1, otherwise by visit count alone. In the second stage, we consume these rows in order as seed rows for a local BFS with local context size ℓ and width-bound w, continuing until the context size L is reached. The full procedure is given in § E (Algorithm 1). Thus, for a given L, 3 parameters (ℓ, w, ρ) can be tuned at test-time. To make the model robust to different context hyperparameters, and enable test-time context tuning, we sample $(\ell, b, \rho) \sim \mathcal{L}_\ell, \mathcal{B}, \mathcal{R}$. W and K exist to tractably approximate the full random-walk ordering, and are tuned for overall latency requirements, rather than for task-specific predictive quality. Unlike other retriever's such as vector similarity, our random-walk retrieval can be performed on-the-fly with no pre-computation, making it scalable for training on a large corpus of databases and tasks.

Multi-cell masking (Ranjan et al., 2026) masks only the target cell per item. As L grows, the supervision-to-compute ratio becomes increasingly unfavorable. Following prior work on masked pretraining (Dong et al., 2025), we additionally mask non-target cells with a hierarchical Bernoulli scheme: per training context i we draw a rate $p_i \sim \text{Unif}\big([0, p_{\max}]\big)$, then mask each non-target, non-missing cell independently with that rate $m_{i,j} \sim \text{Bernoulli}(p_i)$, i.e. cell j from context i is masked if $m_{i,j} = 1$ This enables learning relational structure at different scales (Dong et al., 2025) and improves the robustness of the model.

5. Pretraining Results

Pretraining details. In all our experiments we use an 85M-parameter RT model with 12 layers, 512 hidden size and 8 attention heads, incorporating the improvements in (Kothapalli et al., 2026) over the original architecture of (Ranjan et al., 2026). We sample the context size L from $\mathcal{L} = \{1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}\}$, the local context size ℓ from $\mathcal{L}_\ell = \{512, 1\text{k}, 2\text{k}\}$, the BFS width-bound w from $\mathcal{B} = \{16, 32, 64, 128\}$, and the recency parameter ρ from $\mathcal{R} = \{0, 1\}$. The random-walk retriever uses $W = 10{,}000$ walks of length $K = 20$; and multi-cell masking uses

$p_{\max} = 0.5$.

We pretrain for 100k optimizer steps at a global batch size of 1k contexts per step using Muon (Jordan et al., 2024) for 2D weight matrices and AdamW for the remaining parameters, with at a constant learning rate of 5×10^{-4} (RMS-matched for Muon as in (Liu et al., 2025)) after a linear warmup over the first 2000 steps. To compensate for a lack of decay phase (Tian et al., 2025), and following the recommendation in (Ranjan et al., 2024), we track the stochastic weight average (SWA) performance during training and select the best SWA checkpoint based on validation performance. Specifically, we use exponentially moving average (EMA) for SWA, with momentum 0.9995 which gives an effective window of ≈ 2000 steps, and find that it produces more stable validation curves with better results. Pretraining took 2 days on 32×H100 GPUs at BFloat16 precision. While our pretraining recipe is the same for both classification and regression, we pick checkpoints independently for best results.

Baselines. We compare against the strongest publicly available in-context learning (ICL) pipelines for relational prediction: **RDBLearn** (Zhang et al., 2026), which aggregates relational neighborhoods into a flat feature table and applies a tabular predictor, and an **LLM Agent** (Claude Code with Opus 4.6), which explores the database and writes SQL queries to perform feature-engineering against the database before applying a tabular predictor. We use two predictors: **TabICLv2**, an in-context tabular foundation model (QU et al., 2025; Qu et al., 2026), and a **Ridge** regressor/classifier from SciKit-Learn. We also compare with the **original RT model** (Ranjan et al., 2026) checkpoints.

Experimental setup. We evaluate on RelBench (Robinson et al., 2024), which comprises 21 forecasting tasks across 7 databases (9 regression and 12 binary classification tasks). All databases used at evaluation are held out from pretraining. We use Z-score normalized mean absolute error (nMAE, lower is better) for regression and AUROC (higher is better) for classification, averaging each metric uniformly across tasks in its category.

For every task we sweep the test-time context size $L \in \{256, 512, 1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}\}$ cells for RT, holding all other context hyperparameters fixed, ensuring that the 256-cell context is a subset of the 512-cell context and so on. For a fair label-matched evaluation of the other baselines, we extract the task rows from the context prepared for RT, but do not further constrain the feature extraction or processing from the source database. Any checkpoint selection or hyperparameter tuning is done on the validation split, and all numbers are reported on the test split.

Fig. 2 shows the results. Since we sweep over RT context sizes, and task table rows are encountered organically during context sampling, the exact number of labels differs between tasks and is generally not a nice round number. Hence we take the mean over tasks, and plot against a conservative upper-bound on the x-axis.

Observations. (1) Our model dominates every baseline at every label count on both metrics. It is up to 18% and 3% better than the second strongest baseline on regression and classification respectively. (2) The gap with the TabICLv2 baselines is highest at lower label counts. (3) Pretrained RT (prior work) is the worst of all our baselines, and is unable to learn from more labels with increasing context size. The gap between our RT model and prior RT model reflects the impact of our data and pretraining recipe in enabling strong few-shot predictions from RT.

6. Test-Time Compute Scaling

We identify two complementary ways to improve prediction quality by scaling test-time compute: context ensembling and context tuning. Here we evaluate their effects through controlled experiments.

6.1. Context ensembling

Both the random walk retrieval and the bounded-width local BFS components of the context sampler are stochastic. Running with different random seeds yields different contexts, which are nevertheless equally viable. Here we ensemble the predictions on contexts sampled with different random seeds, but the same context hyperparameters, by simply averaging them, and evaluate against different context sizes as before. Fig. 3 shows the results, with up to 16 sized ensembles.

Observations. (1) Context ensembling improves both metrics at every context size, up to 3.2% on regression and 4.3% on classification. (2) Ensembling is complementary to context size: 16 context seeds at $L = 512$ lead to better classification results than 1 context seed at $L = 8\text{k}$, despite the former taking less inference compute (as RT is quadratic in L).

6.2. Context tuning

The default model uses a single shared (ℓ, b, ρ) = (local context size, BFS width, recency bias) for evaluation on all RelBench tasks. Here we tune these parameters separately for each task on its respective validation set. Intuitively, different tasks might prioritize width vs depth of BFS, local vs global neighborhoods, recent vs long-term patterns differently. The full protocol is described in § G.8, where we also report the best context parameters per task. Fig. 4 compares this per-task tuned model against the shared-config

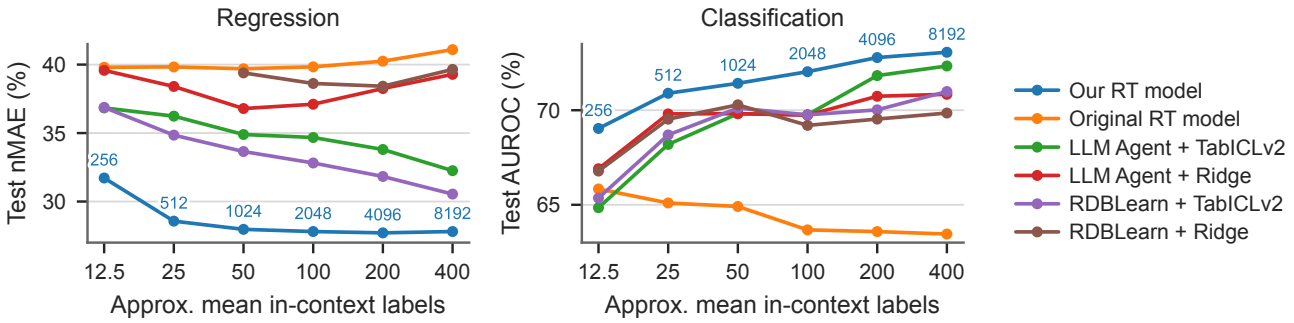


Figure 2. Few-shot relational prediction vs. in-context-learning baselines on RelBench. The x-axis reports the mean number of in-context labeled task rows delivered to our model; markers on the RT curve are annotated with the underlying context size in cells (baselines flatten relational neighborhoods into engineered features and consume the same labels differently). Our model (blue) dominates every baseline at every L, by up to 18% relative nMAE on regression and 3% relative AUROC on classification over the next-best baseline. "Original RT model" is from (Ranjan et al., 2026).

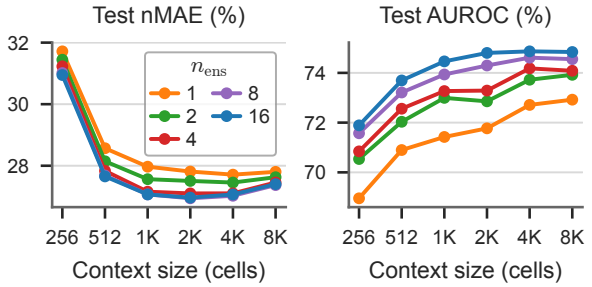


Figure 3. Context ensembling at test time. Each curve averages predictions across contexts that share hyperparameters but use different sampler seeds. Ensembling improves both metrics at every L, by up to 3.2% relative nMAE on regression and 4.3% relative AUROC on classification. Per-task curves are in § G.6.

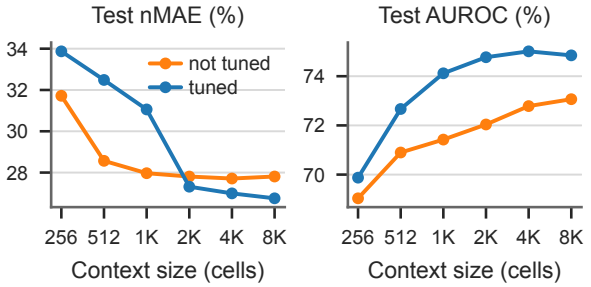


Figure 4. Context tuning at test time. The shared default fixes (ℓ, b, ρ) across tasks; per-task tuning picks the val-best (ℓ, b, ρ) per task at $L = 8$k. Tuning improves classification at every L (+1.2% to +3.8% AUROC), and regression once L is large enough to host the chosen ℓ (−3.8% nMAE at $L = 8$k). Per-task curves are in § G.7.

default across different context sizes, despite only tuning for $L = 8$k context size.

Observations. (1) At the tuning point of $L = 8$k, per-task tuning improves both metrics, by 3.8% relative on nMAE and 2.4% relative on AUROC. (2) The classification gain holds across the entire context sweep (+1.2% to +3.8% relative AUROC at every L), but the regression gain only materializes once the context window is large enough to host the selected ℓ: at $L \leq 1$k, several tasks pick an ℓ larger than L and tuning hurts regression by up to 13.7% relative nMAE. (ℓ, b) chosen at $L = 8$k therefore does not transfer unconditionally to short contexts. (3) Classification tasks are relatively robust to over-large ℓ (their context windows still admit useful structure even when ℓ exceeds L), while regression tasks are not.

7. Ablation Studies

Here we try to isolate the impact of a few important components of our model and recipe. For computational feasibility, the pretraining ablations are performed at smaller scales (pretraining up to 4k context size, up to 32k steps) than the results in prior sections.

7.1. Alternatives to random-walk retrieval

We compare our random-walk retrieval (+ local BFS) (§ 4) against five alternatives (see Figure 5): uniformly random task row selection; BFS of width 32 and 256 rooted at the target (original from (Ranjan et al., 2026)); and two vector-similarity retrievers that score task rows based on RDBLearn features or RT embeddings. The third panel additionally plots the mean number of in-context labels that each retriever delivers as L grows.

Observations. (1) Random-walk retrieval is the strongest variant on regression at every context size, with nMAE falling from 31.7% at $L = 256$ to 27.8% at $L = 8$k; on classification it leads at $L = 512$ and $L = 8$k and trails the best alternative by at most 0.5 AUROC at inter-

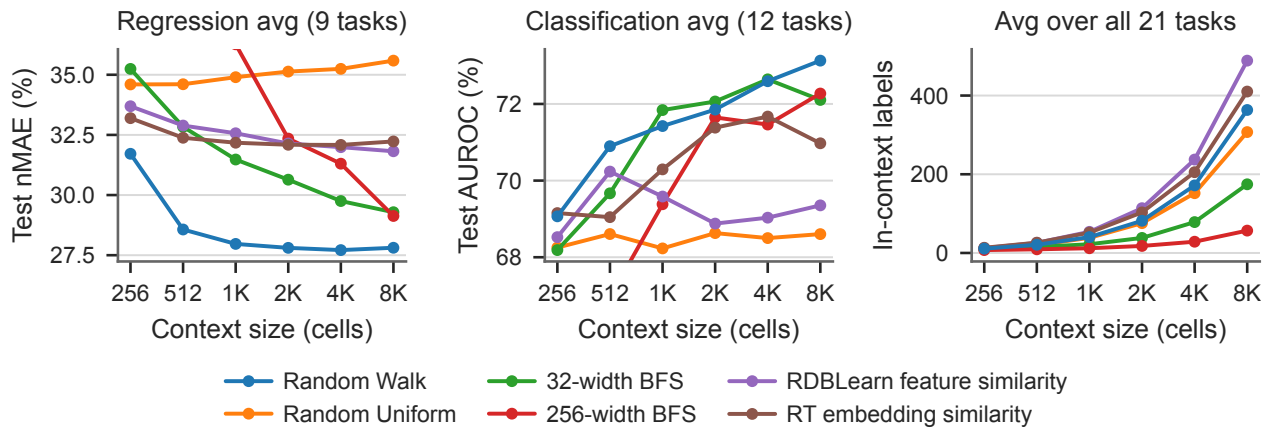


Figure 5. Context-retrieval ablation. Left and middle panels report test nMAE and AUROC averaged over RelBench tasks; right panel reports the mean number of in-context labeled task rows each retriever delivers as L grows. Random-walk retrieval (blue) is the strongest variant on regression at every L and on classification at $L = 8$k. Higher label yield alone does not suffice: RDBLearn-feature similarity delivers more labels yet trails on accuracy ($\approx$4 nMAE behind random walk at $L = 8$k). The 256-width BFS curve at small L is clipped for readability.

mediate sizes. (2) Random-walk's regression lead over the next-best retriever peaks at intermediate context sizes (≈ 3.8 nMAE at $L = 512$) and narrows to ≈ 1.3 nMAE at $L = 8$k as wide BFS eventually exhausts the relevant subgraph and closes the gap. By contrast, the lead over the structurally-naive retrievers (uniform random, semantic similarity) widens with L, reaching $\approx$8 and $\approx$4 nMAE respectively at $L = 8$k: these alternatives stall while random walk and BFS keep improving. (3) Label yield alone does not explain retriever quality: RDBLearn-feature similarity delivers more in-context labels than random-walk at every $L \geq 512$ (e.g. 464 vs. 385 at $L = 8$k on regression) yet trails it by $\approx$4 nMAE—random walk wins because it surfaces rows that are both topologically close to the target and label-bearing, not just one or the other. (4) BFS of width 32 underfills label yield at small L (17 vs. 24 in-context labels at $L = 512$) and still trails random walk by 1.5 nMAE at $L = 8$k; width 256 is sharply worse at small L because its very wide initial layer dilutes the budget with low-relevance rows (its small-L outliers are clipped for readability but the line is still drawn). (5) Uniform random selection is essentially flat in both metrics across L (slowly worsening by $\approx$1 nMAE on regression), so as the structured retrievers improve its rank slips from middle-of-the-pack at $L = 256$ to worst-on-regression at $L = 8$k.

7.2. Multi-cell masking rate

Fig. 6 sweeps the upper bound $p_{\max}$ of the hierarchical Bernoulli masking rate of § 4. Setting $p_{\max} = 0$ recovers the single-cell masking of the original RT paper (Ranjan et al., 2026) and supervises only the task target; $p_{\max} = 0.25$ and $p_{\max} = 0.5$ mask progressively denser subsets of non-target cells.

Observations. (1) Higher masking rates reduce regression error monotonically at every context size, with the single-cell variant ($p_{\max} = 0$) consistently weakest—at $L = 8$k it lags $p_{\max} = 0.5$ by 4.2% relative nMAE. This supports our hypothesis that dense per-window supervision is what makes large-context pretraining data-efficient. (2) On classification, the three rates are nearly tied at small contexts and separate cleanly at $L \geq 1$k; at $L = 8$k, $p_{\max} = 0.5$ leads $p_{\max} = 0$ by 3.1% relative AUROC. (3) This ablation validates the cell-level tokenization design of RT. Row-level tokenized models like TabICLv2 (QU et al., 2025; Qu et al., 2026) and Griffin (Wang et al., 2025) do not benefit from multi-cell masking as straightforwardly as RT can.

7.3. Schema semantics

RT consumes language model (LM) embeddings of table/-column names in addition to cell values. Fig. 7 shows a comparison with and without such semantics, where semantics are removed by shuffling table/column names.

Observations. (1) Schema semantics produce a consistent improvement on both metrics across the entire context sweep—6.7%–9.1% relative nMAE on regression and 2.6%–4.1% relative AUROC on classification at every L—indicating that schema text supplies information the model cannot recover from structural signal alone. (2) The gap is roughly stable across context sizes rather than collapsing as more in-context labels enter, so schema text is genuinely complementary to in-context examples and not just a prior the examples eventually replace. This confirms that the

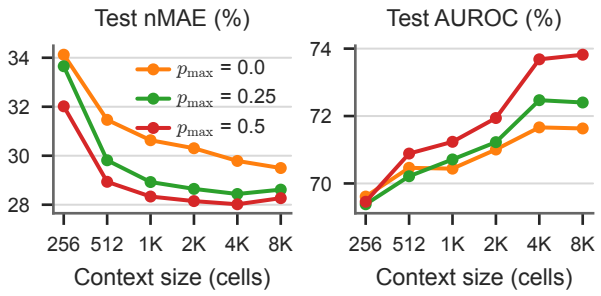


Figure 6. Multi-cell masking ablation. $p_{\max}$ is the upper bound on the masking rate; $p_{\max} = 0$ recovers the single-cell masking of the original RT (Ranjan et al., 2026). At $L = 8$k, $p_{\max} = 0.5$ leads $p_{\max} = 0$ by 4.2% nMAE and 3.1% AUROC.

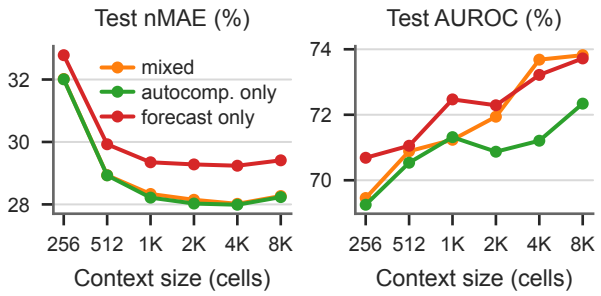


Figure 8. Pretraining task-mix ablation. Each single-task variant specializes to one metric. Only mixed is competitive on both.

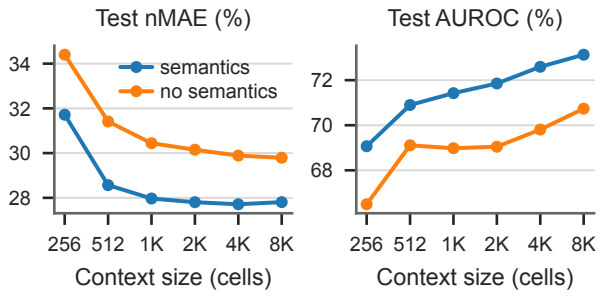


Figure 7. Schema-semantics ablation. Shuffling table and column names at evaluation degrades both metrics across the full context sweep by 6.7%–9.1% nMAE and 2.6%–4.1% AUROC. Semantics remain useful even with many labels.

Configuration	nMAE(%)↓	AUROC(%)↑
Prior RT (Ranjan et al., 2026)	41.1 (+13.3)	63.4 (−9.7)
Ours, full recipe	**27.8**	**73.1**
Ablations		
− schema semantics	29.8 (+2.0)	70.7 (−2.4)
− multi-cell masking	29.5 (+1.7)	71.6 (−1.5)
− random-walk retrieval	29.1 (+1.3)	72.3 (−0.8)
− mixed pretraining	28.2 (+0.4)	72.3 (−0.8)
Test-time compute additions		
+ per-task context tuning	26.8 (−1.0)	74.9 (+1.8)
+ 16-seed context ensembling	27.4 (−0.4)	74.8 (+1.7)

Table 1. Recipe summary on RelBench at $L = 8$k. nMAE averages 9 regression tasks; AUROC averages 12 classification tasks.

model picks up world-knowledge and semantics when pretrained on a large-scale corpus of real data. This opportunity is lost by models like TabICLv2 (QU et al., 2025; Qu et al., 2026), TabPFN (Hollmann et al., 2023; 2025) and PluRel (Kothapalli et al., 2026), which pretrain primarily on synthetic data, which explains the lack of few-shot abilities in these models.

7.4. Pretraining task mix

Our pretraining corpus contains two families of cell-prediction tasks: *autocompletion* tasks, which mask random cells in ordinary database rows, and *forecasting* tasks, which mask cells in specially constructed forecasting task tables. Our default mixes both. Fig. 8 compares the mixed objective against ablations that pretrain on either family of tasks alone.

Observations. (1) Each variant specializes to its own family of downstream metrics: forecasting-only is consistently worse on regression (by 2.3%–4.2% relative nMAE at every L); autocompletion-only is essentially tied with mixed on regression but trails it on classification by up to 3.5% relative AUROC at large L. Only the mixed objective is competitive on both. (2) The autocompletion-vs.-mixed classification gap emerges only at contexts $L \geq 2$k. At small contexts the two are within 0.5% relative AUROC, while at $L = 4$k mixed leads by 3.5% relative AUROC. In essence, forecasting-style supervision allows exploitation of additional in-context labels for classification.

8. Conclusion

We introduced a few-shot pretrained Relational Transformer that converts the cell-level backbone of (Ranjan et al., 2026) into a model capable of strong predictions from hundreds of in-context labels rather than tens of thousands of task-specific examples. The model pairs THE JOIN, a corpus of 6,255 forecasting tasks across 650 real-world databases with a pretraining recipe that combines multi-scale context sampling, multi-cell masking, and random-walk context retrieval. On RelBench, our model outperforms every in-context learning baseline we evaluated at every context size. As summarized in Tab. 1, our ablations show the role of various components in this, and our test-time experiments highlight avenues to improve prediction quality further. We discuss limitations and future work in Appendix A.

References

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Binkowski, M., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 23716–23736, 2022.

Beaini, D., Huang, S., Cunha, J. A., Li, Z., Moisescu-Pareja, G., Dymov, O., Maddrell-Mander, S., McLean, C., Wenkel, F., Müller, L., et al. Towards foundational models for molecular learning on large-scale multi-task datasets. In *The Twelfth International Conference on Learning Representations*, 2024.

Bechler-Speicher, M., Gottlieb, Y., Isakov, A., Abensur, D., Tavory, A., Haimovich, D., Guy, I., and Weinsberg, U. Billion-scale graph foundation models, 2026.

Bommasani, R. et al. On the opportunities and risks of foundation models, 2022. URL https://arxiv.org/abs/2108.07258.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Cantador, I., Brusilovsky, P., and Kuflik, T. Second workshop on information heterogeneity and fusion in recommender systems (hetrec2011). In *Proceedings of the fifth ACM conference on Recommender systems*, pp. 387–388, 2011.

Chen, T. and Guestrin, C. 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.

Chen, T., Kanatsoulis, C., and Leskovec, J. RelGNN: Composite message passing for relational deep learning. In *Forty-second International Conference on Machine Learning*, 2025.

Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. Fma: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference*, 2017.

Dong, M., Wang, L., and Kluger, Y. Understanding and enhancing mask-based pretraining towards universal representations. In *Advances in Neural Information Processing Systems*, 2025.

Dwivedi, V. P., Jaladi, S., Shen, Y., López, F., Kanatsoulis, C. I., Puri, R., Fey, M., and Leskovec, J. Relational graph transformer, 2025a. URL https://arxiv.org/abs/2505.10960.

Dwivedi, V. P., Kanatsoulis, C., Huang, S., and Leskovec, J. 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, 2025b.

ENCODE Project Consortium, a. et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57, 2012.

Fey, M., Hu, W., Huang, K., Lenssen, J. E., Ranjan, R., Robinson, J., Ying, R., You, J., and Leskovec, J. Position: Relational deep learning - graph representation learning on relational databases. In *Forty-first International Conference on Machine Learning*, 2024.

Fey, M., Kocijan, V., Lopez, F., Lenssen, J. E., and Leskovec, J. Kumorfm: A foundation model for in-context learning on relational data, 2025.

Galkin, M., Yuan, X., Mostafa, H., Tang, J., and Zhu, Z. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.

Gu, J., Ranjan, R., Kanatsoulis, C., Tang, H., Jurkovic, M., Hudovernik, V., Znidar, M., Chaturvedi, P., Shroff, P., Li, F., et al. Relbench v2: A large-scale benchmark and repository for relational data. *arXiv preprint arXiv:2602.12606*, 2026.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark:

Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Huang, Q., Ren, H., Chen, P., Kržmanc, G., Zeng, D., Liang, P. S., and Leskovec, J. Prodigy: Enabling in-context learning over graphs. *Advances in Neural Information Processing Systems*, 36:16302–16317, 2023.

Hudovernik, V., López, F., Kocijan, V., Nitta, A., Lenssen, J. E., Leskovec, J., and Fey, M. KumoRFM-2: Scaling foundation models for relational learning, 2026. URL https://arxiv.org/abs/2604.12596.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Kanatsoulis, C. I., Choi, E., Jegelka, S., Leskovec, J., and Ribeiro, A. Learning efficient positional encodings with graph neural networks. In *The Thirteenth International Conference on Learning Representations*, 2025.

Kim, M. J., Grinsztajn, L., and Varoquaux, G. Carte: Pretraining and transfer for tabular learning. In *International Conference on Machine Learning*, pp. 23843–23866. PMLR, 2024.

Kothapalli, V., Ranjan, R., Hudovernik, V., Dwivedi, V. P., Hoffart, J., Guestrin, C., and Leskovec, J. Plurel: Synthetic data unlocks scaling laws for relational foundation models. *arXiv preprint arXiv:2602.04029*, 2026.

Lam, H. T., Buesser, B., Min, H., Minh, T. N., Wistuba, M., Khurana, U., Bramble, G., Salonidis, T., Wang, D., and Samulowitz, H. Automated data science for relational data. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2689–2692. IEEE, 2021.

Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., SU, H., SUO, Z., Gao, H., Hu, W., Yin, P., Zhong, V., Xiong, C., Sun, R., Liu, Q., Wang, S., and Yu, T. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*, 2025.

Leskovec, J. and Sosič, R. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., Chen, Y., Zheng, H., Liu, Y., Liu, S., Yin, B., He, W., Zhu, H., Wang, Y., Wang, J., Dong, M., Zhang, Z., Kang, Y., Zhang, H., Xu, X., Zhang, Y., Wu, Y., Zhou, X., and Yang, Z. Muon is scalable for LLM training, 2025. URL https://arxiv.org/abs/2502.16982.

Motl, J. and Schulte, O. The ctu prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015.

Nivre, J., De Marneffe, M.-C., Ginter, F., Hajic, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F., and Zeman, D. Universal dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the twelfth language resources and evaluation conference*, pp. 4034–4043, 2020.

Peleška, J. and Šír, G. Transformers meet relational databases, 2024. URL https://arxiv.org/abs/2412.05218.

Priem, J., Piwowar, H., and Orr, R. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint arXiv:2205.01833*, 2022.

QU, J., Holzmüller, D., Varoquaux, G., and Morvan, M. L. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025.

Qu, J., Holzmüller, D., Varoquaux, G., and Morvan, M. L. TabICLv2: A better, faster, scalable, and open tabular foundation model, 2026.

Ranjan, R., Garg, S., Raman, M., Guestrin, C., and Lipton, Z. Post-hoc reversal: Are we selecting models prematurely? *Advances in Neural Information Processing Systems*, 37: 91460–91491, 2024.

Ranjan, R., Hudovernik, V., Znidar, M., Kanatsoulis, C. I., Upendra, R. R., Mohammadi, M., Meyer, J., Palczewski, T., Guestrin, C., and Leskovec, J. Relational transformer: Toward zero-shot foundation models for relational data. In *The Fourteenth International Conference on Learning Representations*, 2026.

Robinson, J., Ranjan, R., Hu, W., Huang, K., Han, J., Dobles, A., Fey, M., Lenssen, J. E., Yuan, Y., Zhang, Z., et al. Relbench: A benchmark for deep learning on relational databases. *Advances in Neural Information Processing Systems*, 37:21330–21341, 2024.

Shwartz-Ziv, R. and Armon, A. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

Spinaci, M., Polewczyk, M., Hoffart, J., Kohler, M. C., Thelin, S., and Klein, T. Portal: Scalable tabular foundation

models via content-specific tokenization, 2024. URL https://arxiv.org/abs/2410.13516.

Swartz, A. Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.

Tasneem, A., Aberle, L., Ananth, H., Chakraborty, S., Chiswell, K., McCourt, B. J., and Pietrobon, R. The database for aggregate analysis of clinicaltrials. gov (aact) and subsequent regrouping by clinical specialty. *PloS one*, 7(3):e33677, 2012.

Tian, C., Wang, J., Zhao, Q., Chen, K., Liu, J., Liu, Z., Mao, J., Zhao, W. X., Zhang, Z., and Zhou, J. WSM: Decay-free learning rate schedule via checkpoint merging for LLM pre-training, 2025. URL https://arxiv.org/abs/2507.17634.

Vanschoren, J., Van Rijn, J. N., Bischl, B., and Torgo, L. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Vogel, L., Hilprecht, B., and Binnig, C. Towards foundation models for relational databases [vision paper]. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

Wang, M., Gan, Q., Wipf, D., Cai, Z., Li, N., Tang, J., Zhang, Y., Zhang, Z., Mao, Z., Song, Y., et al. 4dbinfer: A 4d benchmarking toolbox for graph-centric predictive modeling on rdbs. *Advances in Neural Information Processing Systems*, 37:27236–27273, 2024.

Wang, Y., Wang, X., Gan, Q., Wang, M., Yang, Q., Wipf, D., and Zhang, M. Griffin: Towards a graph-centric relational database foundation model. In *Forty-second International Conference on Machine Learning*, 2025.

Wu, F., Dwivedi, V. P., and Leskovec, J. Large language models are good relational learners, 2025. URL https://arxiv.org/abs/2506.05725.

Wydmuch, M., Łukasz Borchmann, and Graliński, F. Tackling prediction tasks in relational databases with llms, 2024. URL https://arxiv.org/abs/2411.11829.

Xia, J., Zhao, C., Hu, B., Gao, Z., Tan, C., Liu, Y., Li, S., and Li, S. Z. Mole-bert: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2023.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Zhang, Y., Xu, L., Gan, Q., Wipf, D., and Wang, M. RDBLearn: Simple in-context prediction over relational databases, 2026. URL https://arxiv.org/abs/2602.18495.

Zhao, J., Zhu, Z., Galkin, M., Mostafa, H., Bronstein, M. M., and Tang, J. Fully-inductive node classification on arbitrary graphs. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Zhao, Q., Ren, W., Li, T., Liu, H., He, X., and Xu, X. Graphgpt: Generative pre-trained graph eulerian transformer. In *Forty-second International Conference on Machine Learning*, 2025b.

Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., Zhang, K., Ji, C., Yan, Q., He, L., et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *International Journal of Machine Learning and Cybernetics*, pp. 1–65, 2024.

A. Limitations and Future Work

Our evaluation centers on the 21 forecasting tasks of RelBench, which leaves open how the recipe transfers to the harder distributional clusters that THE JOIN covers but no public benchmark exposes. We expect the advantage to grow in those regimes but do not measure it here. The model handles a single task per inference call, whereas production workloads often predict many targets jointly. We do not study fine-tuning on top of the few-shot model, which is the natural next step when more labels are available.

We highlight three directions as most promising for future work. First, further scaling both the corpus and the model. Our classification AUROC continues to improve at $L = 8{,}192$ and our regression nMAE plateaus around $L \approx 512$, suggesting that classification will benefit most from longer contexts while regression likely needs stronger models or richer pretraining to improve further. Second, integrating the few-shot pretrained Relational Transformer with downstream tooling, including supervised fine-tuning, multi-task prediction heads, and SQL-style query interfaces. Third, extending the recipe beyond entity-level forecasting to link prediction and cross-database transfer, both of which the cell-level masked objective should accommodate without architectural changes.

B. Related Work

Relational deep learning and benchmarks. Relational deep learning (RDL) is a machine learning paradigm that operates directly on relational databases, avoiding the need to manually flatten a database into a single feature table (Fey et al., 2024). A common formulation represents a database as a heterogeneous relational entity graph: rows are nodes, tables define node types, and primary–foreign-key links define typed edges (Robinson et al., 2024). Graph neural networks can then propagate information across related rows and tables, enabling end-to-end prediction over multi-table data (Robinson et al., 2024; Chen et al., 2025). Recent work has improved the efficiency and expressiveness of relational message passing (Chen et al., 2025), and transformer-based relational architectures have also been proposed (Peleška & Šír, 2024; Dwivedi et al., 2025a). (Kanatsoulis et al., 2025) introduces an efficient positional encoding framework based on lightweight message-passing operations, providing structural features that have been shown to improve relational deep learning models. Broader surveys and position papers frame these models as part of a larger agenda for relational deep learning (Dwivedi et al., 2025b).

Benchmarking has been central to this progress. RelBench (Robinson et al., 2024) introduced a standardized suite of relational prediction tasks, while 4DBInfer (Wang et al., 2024) and RelBenchV2 (Gu et al., 2026) expand the scale and diversity of relational benchmarks, including broader task families and database domains. These benchmarks make it possible to compare relational models under realistic multi-table and temporal constraints.

Relational foundation models. Recent work has begun to develop foundation models specifically for relational data. These models seek to transfer across schemas and tasks rather than training a separate predictor for every database. KumoRFM and KumoRFM-2 study in-context learning for relational data at scale (Fey et al., 2025; Hudovernik et al., 2026). Griffin develops a graph-centric relational foundation model for transfer across relational databases (Wang et al., 2025). Beyond the relational setting, recent graph foundation models scale these ideas across heterogeneous graphs at billion-parameter scale (Bechler-Speicher et al., 2026). The Relational Transformer (RT) takes a complementary cell-level approach: database cells are treated as tokens, schema semantics are included in the input, and relational attention masks encode primary–foreign-key structure (Ranjan et al., 2026). PLUREL (Kothapalli et al., 2026) enables data RT pretraining with synthetic data. It synthesizes multi-table databases from scratch, including schemas, primary–foreign-key links, and feature distributions, enabling controlled studies of synthetic data scaling for relational foundation models.

Our work builds directly on RT because its cell-level formulation is well suited to few-shot relational prediction. Labeled task rows, unlabeled rows, schema metadata, and linked evidence can all appear in the same context window, making masked cell prediction a natural interface for both pretraining and inference. However, the original RT work primarily establishes a schema-agnostic architecture; it does not develop a pretraining and inference recipe specifically designed to unlock few-shot learning. We therefore use RT as the backbone and study how to make it effective when only a small number of labeled rows are available.

Tabular foundation models and relational tabularization. Tabular foundation models aim to bring pretraining and in-context learning to single-table prediction. TabPFN and its successors show that transformers can perform strong few-shot or in-context prediction when labeled rows from a tabular task are provided in the context (Hollmann et al., 2023; 2025). More recent models, including TabICLv2 and other large-scale tabular foundation models, extend this paradigm to larger

datasets, richer feature spaces, and broader tabular distributions (QU et al., 2025; Spinaci et al., 2024; Kim et al., 2024). These methods provide strong evidence that structured prediction can benefit from foundation-model-style training.

Relational databases, however, contain structure that is not present in ordinary single tables. Predictive evidence may lie several joins away from the target row, and the relevant signal may involve temporal histories, linked entities, or multi-hop dependencies. One way to use tabular foundation models in this setting is to first transform the database into a single feature table. RDBLearn follows this strategy by aggregating relational neighborhoods into engineered features and then applying a tabular in-context learner (Zhang et al., 2026). This approach is effective and provides a strong baseline, but the flattening step changes the input representation: the model no longer reasons directly over cells, schemas, timestamps, primary–foreign-key links, or multi-table structure.

Large language models for relational data. Another line of work applies large language models to structured and relational data by serializing tables or database neighborhoods into text (Wydmuch et al., 2024). This has the advantage of using powerful pretrained language models without designing a specialized architecture. However, serialization can be inefficient and brittle. Relational structure must be expressed as text, long database contexts can exceed the model context window, and the resulting input format may differ substantially from the data distribution seen during language-model pretraining. Hybrid approaches that combine graph or relational encoders with LLMs can partially address these issues, but they often still depend on task-specific components or limited context construction (Wu et al., 2025).

Graph foundation models. Several papers have led the path towards graph foundation models. MoleBERT (Xia et al., 2023) pretrains GNNs with masked atom prediction and triplet-based contrastive learning for node- and graph-level drug-discovery tasks. (Beaini et al., 2024) further show that molecular pretraining benefits from scale, using large multi-task datasets with billions of labels and demonstrating that combining quantum and biological supervision improves low-resource performance.

Recent work has extended pretraining graph models beyond molecules. PRODIGY (Huang et al., 2023) introduces prompt-based graph representations for in-context learning. GraphAny (Zhao et al., 2025a) studies zero-shot node classification across graphs with disjoint feature and label spaces, using LinearGNN ensembles and inductive attention inspired by linear least-squares estimation. ULTRA (Galkin et al., 2024) learns transferable relational representations for zero-shot reasoning over unseen knowledge graphs. GraphGPT (Zhao et al., 2025b) converts graphs into reversible token sequences using Eulerian paths, enabling scalable generative Transformer pretraining for graph-structured data. Most recently, (Bechler-Speicher et al., 2026) scales the graph foundation model paradigm to heterogeneous billion-scale graphs, proposing a billion-parameter Transformer architecture and a recipe for link prediction pretraining.

C. Additional Details on THE JOIN

C.1. Agentic execution

Constructing THE JOIN corpus manually would require months of human engineering, particularly for web collection as each source presents different download mechanisms, file formats, schema conventions, parsing failures, and data-quality issues. We therefore used Claude Code agents as the execution layer for corpus construction. The agents identify and inspect candidate sources, implement and debug domain-specific collectors, adapt standardization routines to newly observed edge cases, run the pipeline, inspect logs and intermediate artefacts, and iterate until each stage reaches its completion criteria. The agents were used for engineering and orchestration, while task validation, scoring, filtering, baseline computation, loadability checks, and provenance tracking are performed by deterministic scripts based on human-defined criteria.

C.2. Phase 1: Collection

Collecting from heterogeneous sources. The first phase builds a raw relational snapshot from a broad set of public data sources. This is difficult to automate with a single scraper because different sources expose data in very different ways: as SQL dumps, SQLite databases, CSV bundles, parquet or JSON exports, archive mirrors, or paginated APIs. We therefore use source-specific collectors. Each collector downloads the data, parses it in its native format, recovers table and key metadata when available, and converts the result into a common parquet-based representation.

Respecting upstream constraints. The collectors are designed to be reproducible while remaining respectful of the sources. API-based collectors follow the documented rate limits and include the required identifying information. Bulk-download sources are handled at the archive level, and for sources that publish periodic snapshots, the refresh cadence of our corpus is limited by the cadence of the upstream release.

Normalizing the collected data. Before any database is passed to later stages, the collector performs basic normalization. For example, (1) primary and foreign keys are converted to integer indices, (2) timestamps and booleans are stored in consistent types, and (3) mixed-type columns are either normalized or rejected. We also store primary-key, foreign-key, and time-column metadata alongside each Parquet table, so that downstream stages do not rely on source-specific assumptions. Table 2 lists the domains included in the corpus.

C.3. Phase 2: Standardization

Creating a common representation. After collection, each database is standardized into a uniform relational format. The purpose of this stage is to make heterogeneous sources loadable and comparable, without altering their semantic content. Each fix is implemented as an idempotent detector–transformer pass over the collected parquet files: a pass first checks whether the target invariant already holds, and only rewrites the data when needed.

Recovering relational structure. Many public datasets contain useful structure that is implicit rather than declared. We therefore recover primary keys, foreign keys, time columns, boolean fields, and numeric columns when they can be inferred reliably. These repairs are conservative: candidate keys must satisfy uniqueness checks, foreign keys must pass containment checks, and identifier-like fields such as zip codes, ISBNs, phone numbers, and barcodes are not coerced into numeric values.

Normalizing common type issues. The standardization pass resolves common inconsistencies across sources, such as booleans stored as strings, dates stored in mixed textual formats, year columns stored as integers, and numeric values stored as strings. Ambiguous cases are left unchanged or flagged rather than silently modified. This keeps the downstream interface simple while avoiding brittle source-specific assumptions.

Controlling scale and leakage. For databases that are too large for repeated downstream processing, we apply deterministic entity-axis subsampling: entities are sampled with a fixed seed, all associated activity rows are retained, and keys are reindexed to preserve relational consistency. We also exclude databases that exactly overlap with established benchmark sources, using source identity rather than schema similarity as the criterion.

Preserving auditability. All standardization fixes are performed by mechanical scripts, and each transformation is governed by explicit rules. As a result, downstream task generation and evaluation operate on a consistent representation, while every repair remains reproducible and inspectable.

C.4. Phase 3: Task proposal and validation

Proposing tasks from schema context. For each eligible database, the proposer receives a structured summary of the database, including its domain, tables, row counts, key relationships, time columns, column types, basic column statistics, and a small number of example rows from activity tables. This context is wrapped in a fixed prompt that specifies the allowed task templates, the required JSON format, and the validation rules. The proposer also receives the names of previously generated tasks for that database, which helps reduce duplicates across passes.

Constraining the proposal space. Task proposals are restricted to a registered set of templates, listed in Table 3. This keeps the generated tasks interpretable and mechanically checkable, while still allowing broad coverage across domains. Proposals that refer to unknown templates are rejected before further processing.

Validating mechanically. The LLM proposes candidate tasks, but acceptance is decided by a deterministic validator. The validator checks that referenced tables and columns exist, entity columns are valid foreign keys, numeric fields are used where required, generated target names do not collide with existing columns, task names follow the required format, and binary thresholds avoid trivially imbalanced labels. Rejected proposals are logged with their failure reason and are not written to the released task set.

Improving diversity across passes. Each database is processed through multiple proposal passes. Early passes tend to recover natural or obvious tasks for the schema, while later passes explicitly target under-represented templates and harder prediction settings, such as rare positives, cold-start behavior, heavy-tailed regression, ranking, and derived-ratio tasks. This multi-pass strategy increases task diversity while preserving the same mechanical validation standard for every accepted proposal.

Keeping generation auditable. The proposal stage uses an LLM to suggest candidate tasks, but the final task set is determined by explicit templates and deterministic checks. This separation allows the pipeline to benefit from the LLM's

Table 2. Per-domain corpus breakdown across all 63 collected domains, grouped by italic subject-area headers and sorted by descending DB count. *DBs*: raw count; *Tasks*: count after eligibility, audit, and quality filtering (dash = none yielded); *Tabl./Cols/Rows/Span*: per-domain medians of structural shape.

Domain	Source	DBs	Tasks	Tabl.	Cols	Rows	Span
Commerce & Online Platforms							
tech community	Stack Exchange dumps	76	2,277	7	51	874K	13.2y
recommendations	Public e-commerce dumps	36	1,217	3	16	617K	16.5y
prediction market	PredictIt / Polymarket	2	63	4	26	617K	1.8y
ecommerce	Public e-commerce dumps	2	59	4	13	5.6M	159d
consumer product	Curated CPG datasets	1	52	4	17	349K	6.0y
food reviews	Yelp / OpenTable	1	11	3	13	679K	19.4y
real estate	Public listing dumps	1	—	2	4	383K	—
Sports, Culture & Media							
motorsport	Ergast / FIA archives	9	—	2	19	4.8K	24.0y
sports	Lahman / ATP / StatsBomb / Cricsheet	6	164	5	49	315K	23.9y
food	USDA / Open Food Facts / RateBeer	5	158	3	15	1.7M	13.6y
music	MusicBrainz / Million Song / Last.fm	3	35	4	14	143K	31.7y
museum	Museum collection APIs	3	25	3	40	353K	143.2y
gaming	Game telemetry dumps	3	21	3	19	3.8K	3.4y
cultural	Cultural-heritage portals	2	—	3	16	83K	—
cultural surveys	ICPSR / GESIS	2	—	5	33	684K	16.5y
podcast	Podcast directory APIs	1	29	3	7	554K	210d
history	Historical archives	1	—	3	15	2.2K	1.9y
Government, Finance & Energy							
government us	US Treasury / SEC EDGAR / CFPB	11	121	2	54	748K	3.6y
energy	EIA / OPSD	6	73	3	158	217K	59.7y
international	World Bank / IMF	4	47	3	18	46K	32.5y
finance	Stooq / public market dumps	2	59	2	14	159K	17.8y
politics	FEC / OpenSecrets	2	32	2	6	2.5M	—
eia forms	EIA Form-861 / 923	2	6	3	141	41K	70.5y
civic	Civic data portals	1	29	4	19	501K	46d
energy opsd	Open Power System Data	1	17	2	5	602K	2.8y
crypto	CoinGecko / blockchain dumps	1	—	3	31	20K	56.3y
economics	FRED / OECD	1	—	3	16	99K	23.0y
energy infrastructure	FERC / pipeline registries	1	—	4	22	1.1K	69.5y
labor	BLS / OECD	1	—	2	12	300K	—
legal	CourtListener / SCOTUS	1	—	4	20	933K	6.7y
patent	USPTO / PatentsView	1	—	4	25	125K	18.0y
uk	UK gov data portals	1	—	4	31	40K	—
Health & Public Services							
clinical	ClinicalTrials.gov / AACT	5	27	3	29	150K	22.0y
biology	NCBI / UniProt	4	—	2	21	122K	31.1y
healthcare quality	CMS Hospital Quality	2	37	4	125	248K	4.7y
education	NCES	1	43	7	56	11M	3.9y
public health	CDC / WHO	1	12	3	34	53K	—
transport	GTFS / NYC TLC	1	8	7	52	585K	—
agriculture	USDA / FAO	1	—	3	41	1.4M	—
biology advanced	ENCODE / GTEx	1	—	3	21	9.7M	—
disaster	FEMA / EM-DAT	1	—	2	12	6.2K	72.9y
skills	O*NET / job-posting dumps	1	—	10	94	354K	19.7y
Scientific & Reference							
text2sql	Spider / BIRD benchmarks	260	487	5	25	79.5	12.0y
ctu	CTU Prague Repository	118	407	4	33	29K	17.6y
temporal network	SNAP / KONECT	12	457	2	6	579K	3.3y
network	SNAP / KONECT (raw)	10	—	2	3	1.0M	12.6y
academic	OpenAlex / DBLP	7	92	4	27	224K	6.9y
geospatial	OpenStreetMap / GADM	6	98	3	13	3.5M	1.7y
graph learning	OGB / DGL benchmarks	5	—	3	8	1.2M	49.0y
fivethirtyeight	FiveThirtyEight datasets	4	—	2	45	1.5K	39.3y
ecology	GBIF / iNaturalist	3	34	3	47	110K	—
linguistics	LDC / UD treebanks	3	—	4	30	32K	—
security	MITRE / CVE feeds	3	—	5	36	79K	8.5y
astronomy	Sloan / Gaia	2	—	2	269	355K	—
geonames	GeoNames	1	30	3	43	34K	20.2y
forecasting	M-competitions / Rossmann	1	23	3	11	302K	1.3y
hydrology	USGS / NOAA	1	5	3	15	4.4K	364d
biodiversity	GBIF / IUCN	1	—	3	32	21K	305d
environment	EPA / Copernicus	1	—	3	69	85K	—
network infra	RIPE / CAIDA	1	—	5	22	2.0K	2.2y
ocean	NOAA / Argo	1	—	2	22	2.1K	45d
storm	NOAA Storm Events	1	—	3	44	125K	364d
threat intel	AlienVault / VirusTotal	1	—	2	10	281K	7.8y
Total		**650**	**6,255**				

ability to identify plausible prediction problems from heterogeneous schemas, while keeping task inclusion reproducible, inspectable, and independent of unverified free-form judgments.

```
Database: civic/nyc_311
Total: 4 tables, 501223 rows
Entity tables: ['agencies', 'locations', 'complaint_types']

Schema:
  agencies (15 rows pkey=agency_id):
    columns: agency(string), agency_name(string), agency_id(Int64, range=[0..14])
  locations (270 rows pkey=location_id):
    columns: borough(string), incident_zip(Int64, range=[8854..92108]), location_id(Int64, range=[0..269])
  complaints (500000 rows pkey=unique_key time_col=created_date
              fks={'complaint_type_id': 'complaint_types', 'agency_id': 'agencies', 'location_id': 'locations'}):
    columns: unique_key(Int64), complaint_type_id(Int64), agency_id(Int64), location_id(Int64),
             created_date(datetime64[us, UTC]), closed_date(datetime64[us, UTC]),
             resolution_hours(Float64, mean=49.6, range=[0..1105]), status(string),
             community_board(string), open_data_channel_type(string)
  complaint_types (938 rows pkey=complaint_type_id):
    columns: complaint_type(string), descriptor(string), complaint_type_id(Int64, range=[0..937])

Activity tables (have time_col + FK to entity):
  complaints: time_col=created_date, entity_fks={'complaint_type_id': 'complaint_types',
              'agency_id': 'agencies', 'location_id': 'locations'}, 500000 rows
    sample: [{'unique_key': 0, 'agency_id': 0, 'created_date': '2026-04-14 02:33:29+00:00',
              'resolution_hours': None, 'status': 'Open', 'community_board': '15 BROOKLYN', ...}, ...]

Existing tasks for this DB (do NOT duplicate these):
  predict-agency-avg-resolution-hours, predict-location-max-resolution-hours,
  predict-agency-slow-resolution, predict-complaint-type-active,
  predict-agency-resolution-trend, ... and 24 more

TASK: Propose up to 15 high-quality forecasting tasks that a domain expert would find meaningful.
Each task predicts a future value for an entity over a time window.

PRIORITY: Our existing task suite is heavy on aggregate_numeric, threshold_binary, existence_binary,
count_events, churn_activity. Use at least 40% of your proposals from these under-represented
patterns (where applicable): time_until_next (REG), streak_length (REG), rank_change (CLF),
rank_regression (REG), peer_outperform (CLF), new_category_adoption (CLF), will_repeat_last (CLF),
trend_reversal (CLF), volatility_numeric (REG), concentration (REG), derived_ratio (REG).

If the schema supports a `group_col` (race/week/country) -> propose rank_change + rank_regression.
If there's a natural `peer_col` (category/country/segment) -> propose peer_outperform. If there's
a `category_col` (product/type/status) -> propose new_category_adoption + will_repeat_last.

ALLOWED PATTERNS:
  - aggregate_numeric (REG): mean/sum/max of numeric col next window. Needs `source_col`,
    `aggregation` in [mean, sum, max, min].
  - threshold_binary (CLF): aggregate exceeds threshold. Needs `source_col`, `aggregation`,
    `threshold`, `compare` in [gt, gte, lt, lte].
  - existence_binary (CLF): entity appears in event_table next window. Needs `event_table`,
    `event_entity_col`, `event_time_col`.
  ... [22 more templates omitted]
  - heavy_tail_regression (REG, HARD TIER): loglp(agg(source_col)) per entity next window.
    Entity pool = ALL historical entities. Matches rel-amazon/item-ltv, rel-hm/item-sales.

HARD REQUIREMENTS (validator rejects violations):
  1. entity_col MUST be a foreign key in the activity_table.
  2. source_col / numerator_col / denominator_col MUST be numeric columns in the activity_table.
  3. target_col must be short and not collide with an existing column.
  4. task_name must be kebab-case starting with "predict-".
  5. For derived_ratio: pick MEANINGFUL pairs (batting average, completion rate, margins).
  6. For threshold_binary: thresholds near the median of the column.
  7. Prefer real-world interpretability over mechanical column selection.

Return ONLY a JSON array of task objects:
[ { "task_name": "predict-...", "task_type": "REGRESSION" | "BINARY_CLASSIFICATION",
    "pattern": "aggregate_numeric" | ..., "entity_table": "...", "entity_col": "...",
    "activity_table": "...", "time_col": "...", "target_col": "...", "target_description": "...",
    ... template-specific fields ..., "skip_cols": [], "tables_used": [...],
    "timedelta_multiplier": 1 } ]
```

Figure 9. Full proposer prompt for `civic/nyc_311` (one representative database). The schema, FK map, sample rows, and existing-task list are populated per database; the `ALLOWED PATTERNS`, `HARD REQUIREMENTS`, and JSON contract blocks are constant across databases (the `ALLOWED PATTERNS` block in particular is abbreviated here — the full version enumerates all 25 templates with their required fields).

```
{
  "task_name": "predict-agency-avg-resolution-hours",
  "task_type": "REGRESSION",
  "pattern": "aggregate_numeric",
  "entity_table": "agencies",
  "entity_col": "agency_id",
  "activity_table": "complaints",
  "time_col": "created_date",
  "target_col": "avg_resolution_hours",
  "target_description": "Predict the mean resolution time (hours) for complaints handled by an agency in the next
     window.",
  "source_col": "resolution_hours",
  "aggregation": "mean",
  "tables_used": ["agencies", "complaints"],
  "timedelta_multiplier": 1
}
```

Figure 10. First accepted proposal for `civic/nyc_311`, returned by the Claude Opus proposer in response to the prompt of Figure 9. The proposal passes the mechanical validator (the `entity_col` is a declared FK in the `activity_table`, `source_col` is a numeric column of the same table, `target_col` does not collide with any existing column, and the `pattern` is registered) and is forwarded to the generator for instantiation.

Table 3. The 25 task-template families instantiated by the corpus, organised under italic signal-type headers. Each row gives the prediction target as the model sees it, paired with one concrete domain example. R: regression; C: binary classification.

Template	Type	Prediction target (example)
Aggregation — volume / intensity of activity		
`aggregate_numeric`	R	mean / sum of a numeric column over the next window *(e.g. a player's home runs next season)*
`count_events`	R	number of events for the entity in the next window *(e.g. orders per customer next month)*
`distinct_count`	R	distinct values of a column the entity touches *(e.g. distinct co-authors next year)*
`concentration`	R	entity's share of total activity *(e.g. seller's share of category sales)*
Threshold / existence — continuous signal at a decision boundary		
`threshold_binary`	C	whether an aggregate exceeds a threshold *(e.g. revenue >$1M next quarter)*
`existence_binary`	C	whether the entity appears in the event table next window *(e.g. user posts at least once)*
`top_k_binary`	C	whether the entity is in the top $K\%$ on a metric *(e.g. artist enters top 1% of plays)*
Trend / change — direction and magnitude of change		
`trend_binary`	C	whether a metric rises relative to the current window *(e.g. weekly volume rises)*
`trend_reversal`	C	whether the direction of change flips between windows *(e.g. sales reverse from rising to falling)*
`change_numeric`	R	numeric delta from current to next window *(e.g. change in monthly energy consumption)*
`volatility_numeric`	R	standard deviation of a metric in the next window *(e.g. next-week price volatility)*
Behaviour / churn — entity lifecycle		
`churn_activity`	C	entity has *no* activity in the next window *(e.g. subscriber cancels next month)*
`return_binary`	C	dormant entity reactivates next window *(e.g. dormant Stack Exchange user posts again)*
`time_until_next`	R	days until the entity's next event *(e.g. days until a player's next ATP match)*
`will_repeat_last`	C	next event's category equals the most-recent one *(e.g. customer re-buys the same product)*
`first_event_binary`	C	entity experiences a first occurrence next window *(e.g. first purchase by a new account)*
Ranking — relative position within a peer group		
`rank_change`	C	entity's rank within its group improves by $\geq K$ *(e.g. driver climbs ≥5 places)*
`rank_regression`	R	numeric rank within the group ($1 =$ best) *(e.g. predict the driver's next-race finish position)*
`peer_outperform`	C	entity beats the mean of its peer group *(e.g. artist beats peer-group median plays)*
Adoption / novelty — first-time events		
`new_category_adoption`	C	entity touches a category value it never has before *(e.g. first purchase from a new product category)*
`derived_ratio`	R	ratio of two numeric columns over the next window *(e.g. new-to-returning customer ratio)*
Streak — resilience as a single metric		
`streak_length`	R	longest run of consecutive active sub-periods *(e.g. consecutive seasons a player remains active)*
Tail-event (hard-tier) — rare and extreme regimes		
`cold_start_binary`	C	newly-appearing entity has repeat activity in its first window *(e.g. new user posts within 24h of joining)*
`rare_positive_binary`	C	rare event ($\leq 10\%$ positive rate) occurs for the entity *(e.g. rare adverse-quality hospital event)*
`heavy_tail_regression`	R	$\log_1$p of an aggregate; preserves heavy upper tail *(e.g. $\log_1 p$ of next-month spend, large spenders preserved)*

Table 4. Fingerprint clusters used to characterize retained tasks. Rules are evaluated in priority order so that the rarest applicable characteristic determines the assigned cluster.

Cluster	Rule
A rare-event cold-start	Classification positive rate < 0.10 and cold-start rate ≥ 0.30
B inverse imbalance	Classification positive rate > 0.85
C full cold-start	Cold-start rate ≥ 0.90
D zero-inflated regression	Regression target zero-rate ≥ 0.50
E heavy-tail regression	Regression target skewness > 5 or $\log(1 + \text{std}) > 1.0$
F long-horizon churn	Pattern is one of `churn_activity`, `return_binary`, or `time_until_next`, with timedelta multiplier ≥ 3
G standard	Residual category

C.5. Phase 4: Baselines and quality filtering

Measuring basic task signal. After task generation, each task is evaluated with a small set of baselines. We first compute two cheap historical baselines: an entity-mean predictor, which uses each entity's average training label, and a last-value predictor, which uses the entity's most recent observed label when available. These baselines are inexpensive and provide a simple check for whether the task contains recoverable temporal or entity-level signal.

Training a lightweight ML baseline. We also train a fixed XGBoost baseline for each task. For classification tasks we use `XGBClassifier`, and for regression tasks we use `XGBRegressor`, with `n_estimators=100`, `max_depth=6`, `learning_rate=0.1`, `tree_method="hist"`, and `n_jobs=2`. Features are simple window aggregations over the activity table: for the 20 most populous numeric columns, we compute `count`, `mean`, `sum`, `max`, and `min` within a lookback window of length $5 \times$ `timedelta` ending at the prediction time. To keep the pass scalable, training is capped at 5,000 rows per task and evaluation at 2,000 rows per task. This baseline is not intended to be a strong relational model and is only intended to provide a consistent and reproducible reference point for filtering tasks across domains.

Filtering and difficulty assignment. The quality filter removes tasks that are unlikely to be useful for downstream evaluation, such as tasks that are nearly trivial, exhibit severe distribution drift, or fail during baseline fitting. Tasks with non-trivial baseline signal are retained, and we also retain non-trivial tasks for which XGBoost does not recover signal, since stronger relational models may still exploit structure missed by the lightweight baseline. Retained tasks are assigned a coarse difficulty level from their XGBoost score: easy, medium, or hard.

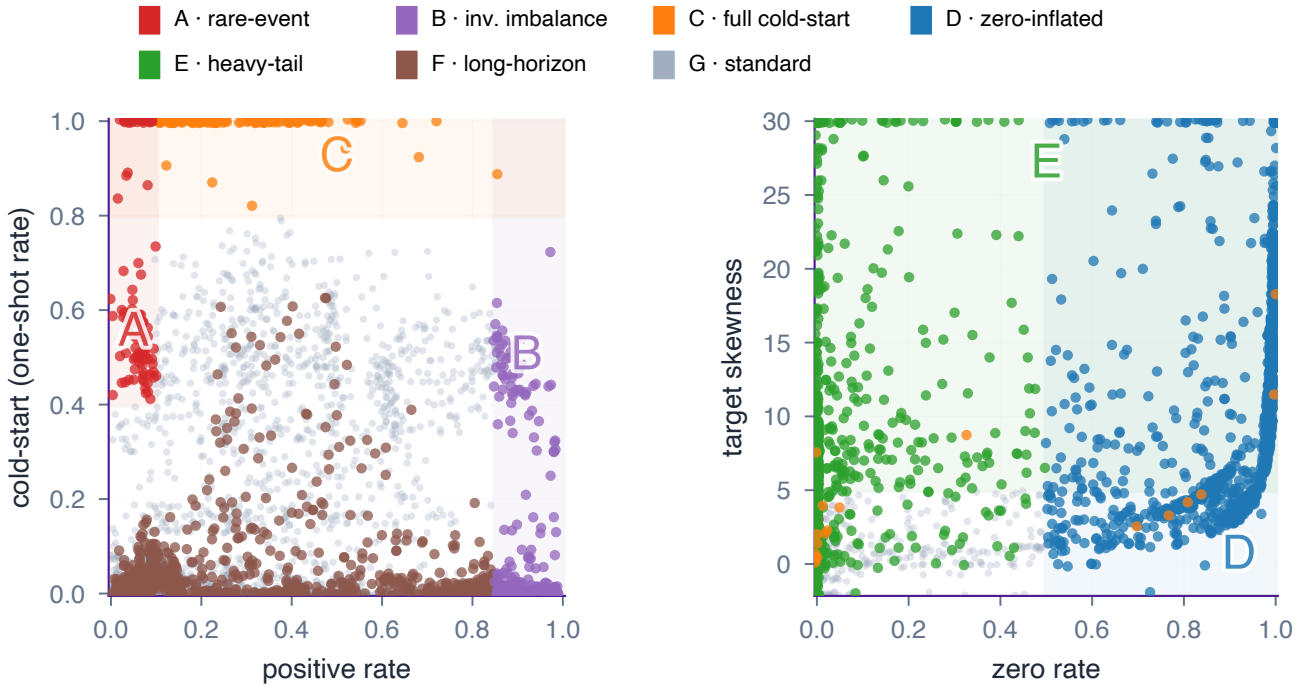
Tagging task fingerprints. Each retained task is assigned a coarse fingerprint describing its label distribution and prediction setting. These tags distinguish qualitatively different regimes, such as rare-event cold start, inverse imbalance, full cold start, zero-inflated regression, heavy-tailed regression, long-horizon churn, and standard tasks. Tasks generated with an explicit fingerprint tag keep that tag; otherwise, the tagger assigns one using the deterministic rules in Table 4. Figure 11 visualizes the resulting clusters.

Preparing the release format. After loadability validation, each task is converted into the release format. The train, validation, and test splits are concatenated in chronological order into a single time-ordered training file, while preserving the schema metadata required by downstream loaders. This ensures that released tasks expose a consistent relational interface and that broken splits do not propagate into the final snapshot.

C.6. Asset licenses, Citations, and Compliance

Compliance practice. Every contributing source is either (a) a bulk archive that the upstream provider publishes for download under an explicit licence, or (b) a public API whose documented terms permit programmatic research use. We do not crawl arbitrary websites, do not bypass paywalls, do not ignore `robots.txt`, and do not scrape sources whose terms of service forbid automation. Every collector identifies itself with an academic User-Agent and throttles to the rate documented by the provider. The per-source licences recorded in the released metadata are summarised below.

Data sources. Tables 5–8 list every contributing data source, with the version we used, the provider's published licence, and a citation. Sources we accessed through bulk archive downloads are marked *archive*; sources accessed through documented public APIs are marked *API*.



(a) Classification tasks ($n = 2{,}857$). *(b)* Regression tasks ($n = 3{,}398$).

Figure 11. Where every task sits in the metric pairs that gate the fingerprint clusters. Shared cluster legend on top. Each dot is one task, coloured by its cluster; outlined boxes mark the rule boundaries and the cluster letter anchors each box. Cluster counts: A 93 (1.5%), B 215 (3.4%), C 165 (2.6%), D 1,323 (21.2%), E 995 (15.9%), F 892 (14.3%), G 2,572 (41.1%). Cluster F (long-horizon churn) is template-defined and G (standard) is the residual category, so neither carves out a metric zone — both appear in the dot population only.

Table 5. Curated relational repositories.

Source	Access	Licence	Reference
CTU Prague Relational Repository	archive	research-use, attribution	(Motl & Schulte, 2015), https://relational.fel.cvut.cz/
Open Graph Benchmark (OGB)	archive	MIT	(Hu et al., 2020), https://ogb.stanford.edu/
OpenAlex	API	CC0 1.0	(Priem et al., 2022), https://openalex.org/

Table 6. Public APIs (US federal works are public domain under 17 USC §105).

Source	Access	Licence	Reference
US Treasury (Fiscal Data)	API	public domain	https://fiscaldata.treasury.gov/
SEC EDGAR	API	public domain	https://www.sec.gov/edgar
CFPB Consumer Complaint DB	API	public domain	https://www.consumerfinance.gov/data-research/
US FDA (openFDA)	API	public domain	https://open.fda.gov/
USPTO PatentsView	API	public domain	https://patentsview.org/
EIA (Form-861/923, Bulk)	API	public domain	https://www.eia.gov/
NCES (Common Core)	archive	public domain	https://nces.ed.gov/
CMS Hospital Quality	API	public domain	https://data.cms.gov/
CDC / WHO Public Health	API	public domain / CC-BY-SA 3.0 IGO	https://data.cdc.gov/, https://www.who.int/data
ClinicalTrials.gov / AACT	archive	public domain	(Tasneem et al., 2012), https://aact.ctti-clinicaltrials.org/
World Bank Open Data	API	CC-BY 4.0	https://data.worldbank.org/
IMF Data Mapper	API	CC-BY 4.0	https://www.imf.org/en/Data
FEC / OpenSecrets	API / archive	public domain / OpenSecrets ToS (research use)	https://www.fec.gov/data/, https://www.opensecrets.org/
USDA / FAO	archive	public domain / CC-BY 4.0	https://www.usda.gov/, https://www.fao.org/
GBIF, iNaturalist	API	CC0 / CC-BY / CC-BY-NC	https://www.gbif.org/, https://www.inaturalist.org/

Table 7. Domain portals and community data dumps.

Source	Access	Licence	Reference
Lahman Baseball Database	archive	CC-BY-SA 3.0	http://www.seanlahman.com/baseball-archive/
ATP Tennis (Sackmann)	archive	CC-BY-NC-SA 4.0	https://github.com/JeffSackmann/tennis_atp
StatsBomb Open Data	archive	StatsBomb Open Data licence (research/non-commercial)	https://github.com/statsbomb/open-data
Cricsheet	archive	ODC-BY 1.0	https://cricsheet.org/
OpenFlights	archive	ODC-BY 1.0	https://openflights.org/data.html
Stooq (markets)	archive	free, non-commercial use	https://stooq.com/
MusicBrainz, ListenBrainz	archive (Postgres dumps)	CC0 (data)	(Swartz, 2002), https://musicbrainz.org/doc/MusicBrainz_Database
Last.fm (HetRec 2011)	archive	research-use	(Cantador et al., 2011), https://grouplens.org/datasets/hetrec-2011/
Million Song Dataset / FMA	archive	CC-BY-NC-SA 4.0 (FMA)	(Defferrard et al., 2017), https://github.com/mdeff/fma
Stack Exchange Data Dump	archive	CC-BY-SA 4.0	https://archive.org/details/stackexchange
GTFS feeds (transit)	archive	per-agency (mostly CC-BY)	https://gtfs.org/
NYC TLC trip records	archive	NYC OpenData (public)	https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
Yelp Open Dataset / OpenTable	archive	Yelp Dataset Agreement (academic)	https://www.yelp.com/dataset
Wikidata, Europeana	API / archive	CC0 / CC-BY-CC0 (mixed per record)	https://www.wikidata.org/, https://www.europeana.eu/

Table 8. Tabular and text-to-SQL benchmarks; specialised data dumps.

Source	Access	Licence	Reference
Spider 1.0	archive	CC-BY-SA 4.0	(Yu et al., 2018), https://yale-lily.github.io/spider
BIRD	archive	CC-BY-SA 4.0	(Li et al., 2023), https://bird-bench.github.io/
LiveSQLBench	archive	CC-BY-SA 4.0	https://huggingface.co/datasets/birdsql/livesqlbench-base-lite-sqlite
Spider 2.0	archive	CC-BY-SA 4.0	(Lei et al., 2025), https://github.com/xlang-ai/Spider2
SNAP / KONECT (graph data)	archive	per-dataset (CC-BY)	(Leskovec & Sosič, 2016), https://snap.stanford.edu/data/
M-competitions / Rossmann	archive	per-competition (Kaggle ToS)	https://www.kaggle.com/
Kaggle / OpenML (general)	API	per-dataset (varies)	(Vanschoren et al., 2014), https://www.kaggle.com/datasets, https://www.openml.org/
NOAA, NASA, USGS, FEMA (storms, oceans, hydrology, disasters)	archive / API	public domain	https://www.noaa.gov/, https://www.nasa.gov/, https://www.usgs.gov/, https://www.fema.gov/
ENCODE, GTEx, NCBI / UniProt	archive	ENCODE Data Use Policy / dbGaP-style restricted; CC-BY (UniProt); public-use (NCBI)	(ENCODE Project Consortium et al., 2012), https://www.encodeproject.org/, https://www.uniprot.org/
LDC / Universal Dependencies treebanks	archive	per-corpus (CC-BY-SA)	(Nivre et al., 2020), https://universaldependencies.org/

Table 9. Binary classification forecasting tasks in RelBench.

Dataset	Task	Entity	Prediction target
rel-amazon	user-churn	User	Whether the user does not review any product in the next 3 months.
rel-amazon	item-churn	Item	Whether the item receives no reviews in the next 3 months.
rel-avito	user-visits	User	Whether the user visits more than one ad in the next 4 days.
rel-avito	user-clicks	User	Whether the user clicks more than one ad in the next 4 days.
rel-f1	driver-dnf	Driver	Whether the driver does not finish a race in the next month.
rel-f1	driver-top3	Driver	Whether the driver qualifies in the top 3 for a race in the next month.
rel-hm	user-churn	User	Whether the customer makes no transactions in the next week.
rel-stack	user-engagement	User	Whether the user makes any votes, posts, or comments in the next 3 months.
rel-stack	user-badge	User	Whether the user receives a new badge in the next 3 months.
rel-trial	study-outcome	Study	Whether the trial achieves its primary outcome in the next year.
rel-event	user-repeat	User	Whether a previously active user attends another event in the next 7 days.
rel-event	user-ignore	User	Whether the user ignores more than two event invitations in the next 7 days.

Released-corpus licence. We intend to release the corpus under **CC-BY-SA 4.0**, the most permissive licence compatible with the strictest contributing source (Stack Exchange, Spider, BIRD, and Lahman are all CC-BY-SA). Per-database metadata records (i) the upstream source identifier, (ii) the upstream licence string verbatim, and (iii) any usage restrictions inherited from the source (e.g. non-commercial-only for ATP-Tennis, StatsBomb, Yelp, FMA, and ListenBrainz; research-use-only for the CTU repository). Downstream users redistributing derivative works must honour those per-source restrictions in addition to the aggregated CC-BY-SA 4.0 licence on the corpus assembly.

Excluded sources. Per the source-overlap exclusion fix (Appendix C.3), we explicitly remove databases whose upstream source coincides with that of a downstream evaluation: `recommendations/amazon_books` (rel-amazon), `clinical/aact_clinicaltrials` (rel-trial), and `ctu/ergastf1` (rel-f1).

PII. The corpus contains no personally-identifying information beyond what each upstream source has already published under its own licence. Where source data uses pseudonymised identifiers (e.g. Stack Exchange user IDs, Lahman player IDs), we preserve those identifiers verbatim and do not attempt to de-anonymise or join across sources.

D. RelBench Forecasting Tasks

Binary classification tasks. The binary classification tasks in RelBench ask whether a future event or condition will occur for a given entity. They cover churn, future engagement, ad-click behavior, race outcomes, clinical outcomes, and event attendance behavior. See Table 9 for task descriptions.

Regression tasks. The regression tasks ask for a continuous or count-valued future outcome. They cover lifetime value,

Table 10. Regression forecasting tasks in RelBench.

Dataset	Task	Entity	Prediction target
rel-amazon	user-ltv	User	Value of the products the user buys and reviews in the next 3 months.
rel-amazon	item-ltv	Item	Value of the purchases and reviews the item receives in the next 3 months.
rel-avito	ad-ctr	Ad	Click-through rate for the ad over the next 4 days.
rel-f1	driver-position	Driver	Average finishing position of the driver across races in the next 2 months.
rel-hm	item-sales	Item	Total sales for the article in the next week.
rel-stack	post-votes	Post	Number of votes the post receives in the next 3 months.
rel-trial	study-adverse	Study	Number of affected patients with severe adverse events in the next year.
rel-trial	site-success	Site	Success rate of a trial site in the next year.
rel-event	user-attendance	User	Number of events the user responds yes or maybe to in the next 7 days.

Algorithm 1 Random-walk context construction (one task item).

Require: target row $v^\star$ in table τ with timestamp $t^\star$; database $\mathcal{D}$; total cell budget L; per-seed cell budget ℓ; BFS width b; recency flag $\rho \in \{0, 1\}$; walks W; walk length K.
Ensure: cell sequence $\mathbf{x}$ with $|\mathbf{x}| \leq L$.

```
Stage 1: score same-table candidates by random-walk visits.
score[u] ← 0 for every row u.
for w = 1 to W do
    u ← v*.
    for k = 1 to K do
        Move u to a uniformly random PK/FK neighbor whose timestamp is ≤ t*; if no such neighbor exists, end this
        walk.
        if u ≠ v* and u is in table τ then
            score[u] ← score[u] + 1.
        end if
    end for
end for
Stage 2: launch BFS from the top-scoring candidates.
Order same-table rows by score descending; if ρ = 1, break ties by timestamp descending.
x ← empty cell sequence.
for each row s in this order do
    if |x| ≥ L then
        break.
    end if
    Append BFSCollect(D, s, ℓ, b, t*, L − |x|) to x.
end for
return x.
```

item sales, click-through rate, future votes, race position, adverse-event counts, site success, and event attendance. See Table 10 for task descriptions.

E. Random-Walk Context Construction

This appendix gives the full procedure for context construction summarized in § 4. For one task item, we score same-table candidate rows by random-walk visit counts (Stage 1), then consume the top-scoring candidates in order as launch points for the depth-bounded BFS cell collector BFSCollect of the original RT, which we reuse unchanged (Stage 2). Walks step uniformly at random along PK and FK edges, restricted to neighbor rows whose timestamp does not exceed the target's timestamp $t^\star$; this enforces temporal validity at the walk level and avoids leaking future information into the context. Only visits to non-source rows in the target's table τ contribute to the score, so the resulting ordering ranks rows in the same table as the target by their personalized random-walk proximity to it.

F. Per-Task Baseline Curves

Figs. 12 and 13 disaggregate the aggregate baselines comparison of Fig. 2 into one panel per RelBench task, plotting test MAE for the 9 regression tasks and test AUROC for the 12 classification tasks against inference-time context size. The six methods, the context grid, and the evaluation protocol match § 5; per-task y-axes are auto-scaled. For readability, panel y-limits are set ignoring *RDBLearn + Ridge*, which is degenerate on a few tasks; its outlier points may extend outside the visible range, while the line itself is still drawn.

At the largest context, our model is the strongest or near-strongest method on most tasks; its aggregate dominance in Fig. 2 comes from a small number of wide wins outweighing narrow losses elsewhere. The widest wins are on tasks with heavy-tailed targets (*rel-stack/post-votes*, *rel-trial/site-success*, *rel-amazon/user-ltv*) where flat-feature pipelines collapse; the few tasks where a baseline edges ahead (*rel-avito/ad-ctr*, *rel-hm/item-sales*, *rel-amazon/item-churn*) are typically narrow losses to a single feature-engineering pipeline rather than systematic gaps.

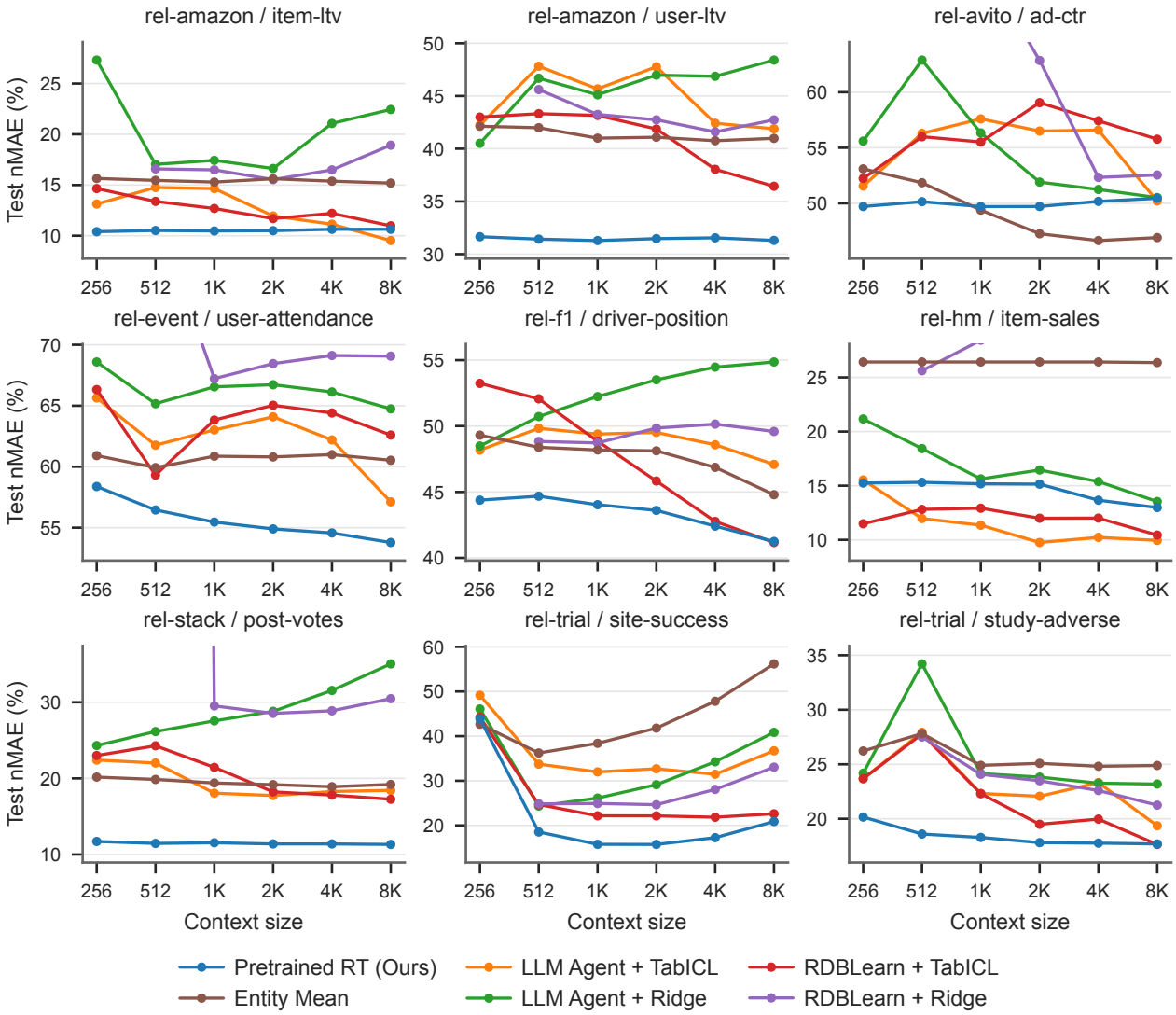


Figure 12. Per-task test MAE versus inference-time context size on the nine RelBench regression tasks. Lower is better.

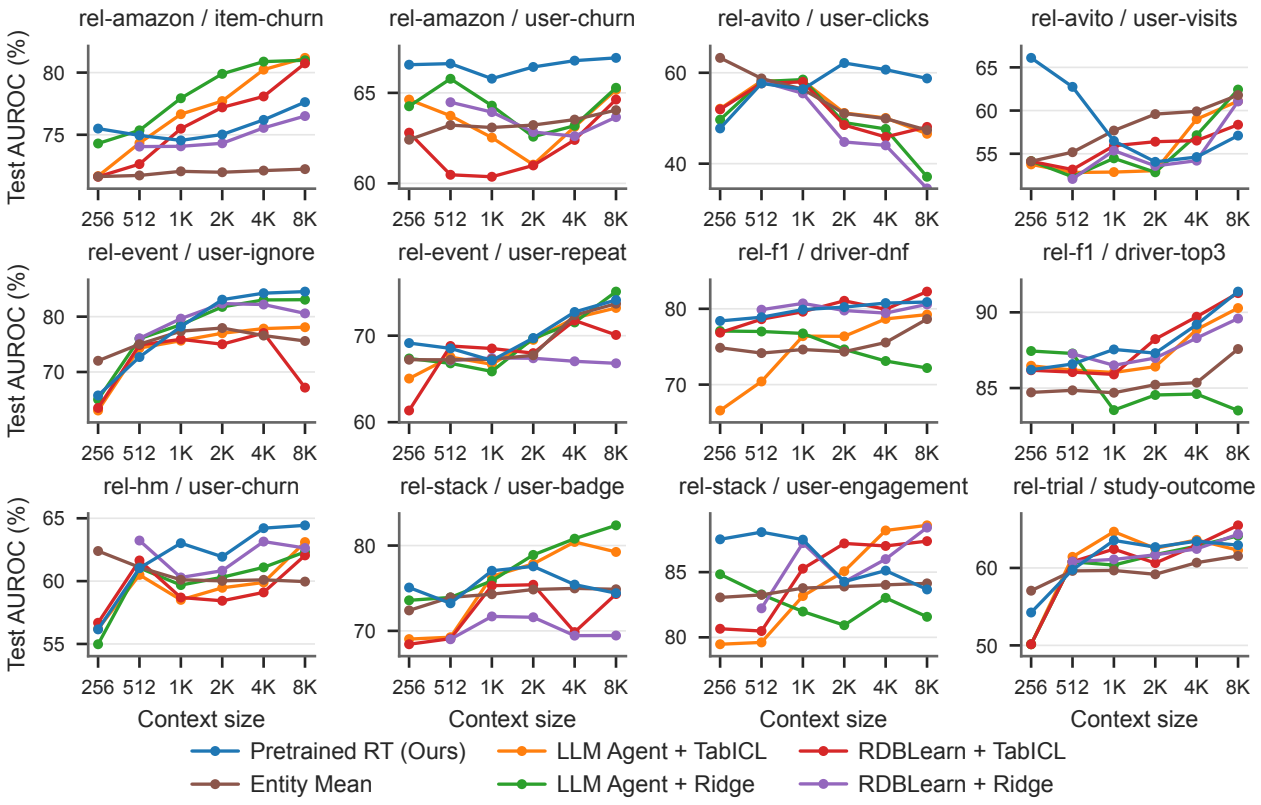


Figure 13. Per-task test AUROC versus inference-time context size on the twelve RelBench classification tasks. Higher is better.

G. Per-Task Ablation Curves

This appendix disaggregates each ablation in § 7 into per-task panels, one per RelBench task. The methods, context grid, and evaluation protocol match the corresponding aggregate figure in the main text; per-panel y-axes are auto-scaled so the visual ranking of methods within a task is preserved even where its absolute scale differs.

G.1. Context retrieval (per task)

Per-task counterpart of Fig. 5. The aggregate random-walk advantage on regression is concentrated on a few tasks with rich graph structure—most prominently *rel-trial/site-success*, where the next-best retriever is visibly far behind at $L = 8\text{k}$, and *rel-stack/post-votes*. On the remaining tasks the retrievers cluster more tightly, with random-walk competitive but not always the best. The 256-width BFS variant is a strong outlier at small L on several tasks; it is excluded from per-panel y-limit computation so the other curves remain readable, but its line is still drawn.

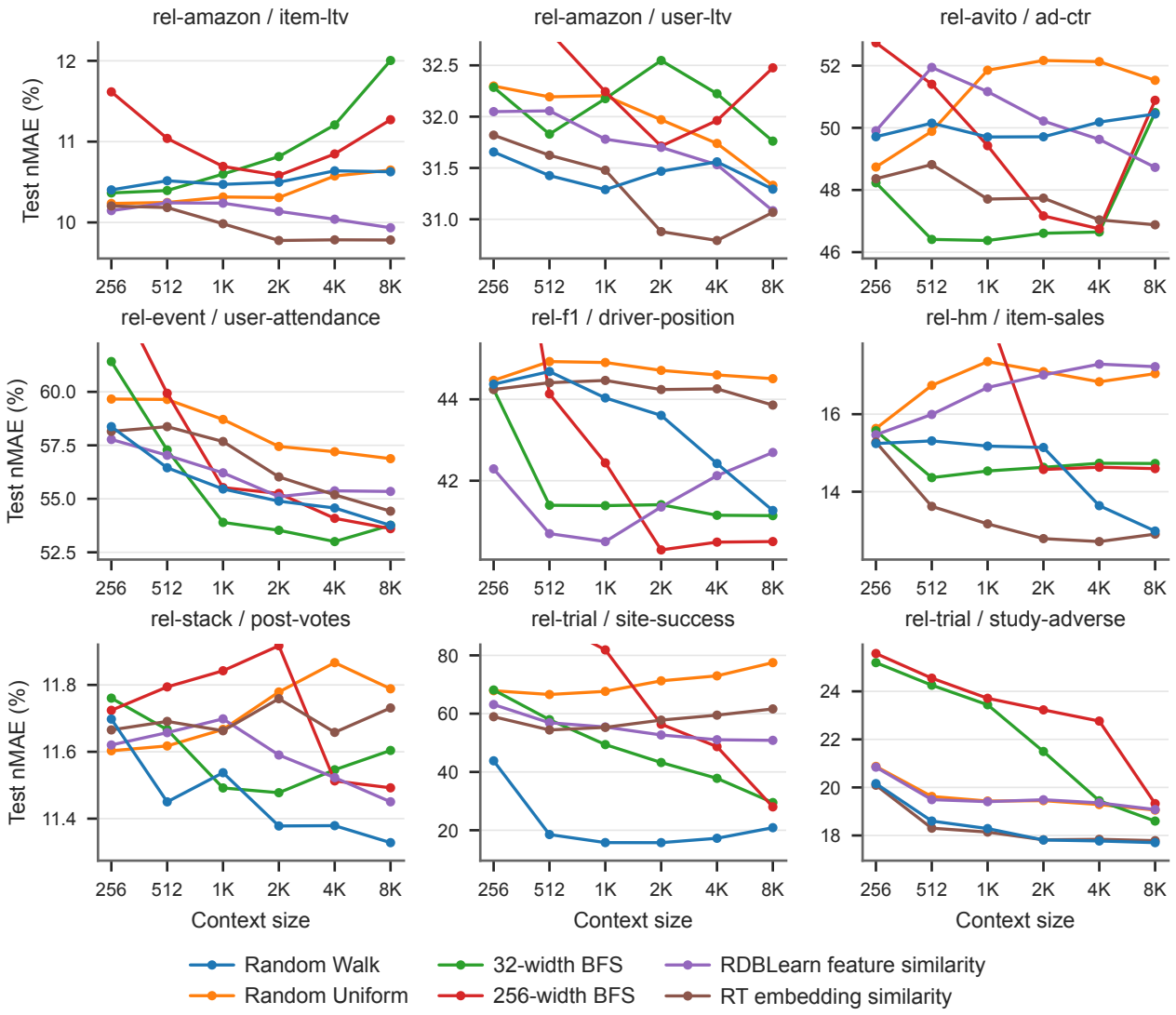


Figure 14. Per-task test MAE for context-retrieval ablations on the nine RelBench regression tasks.

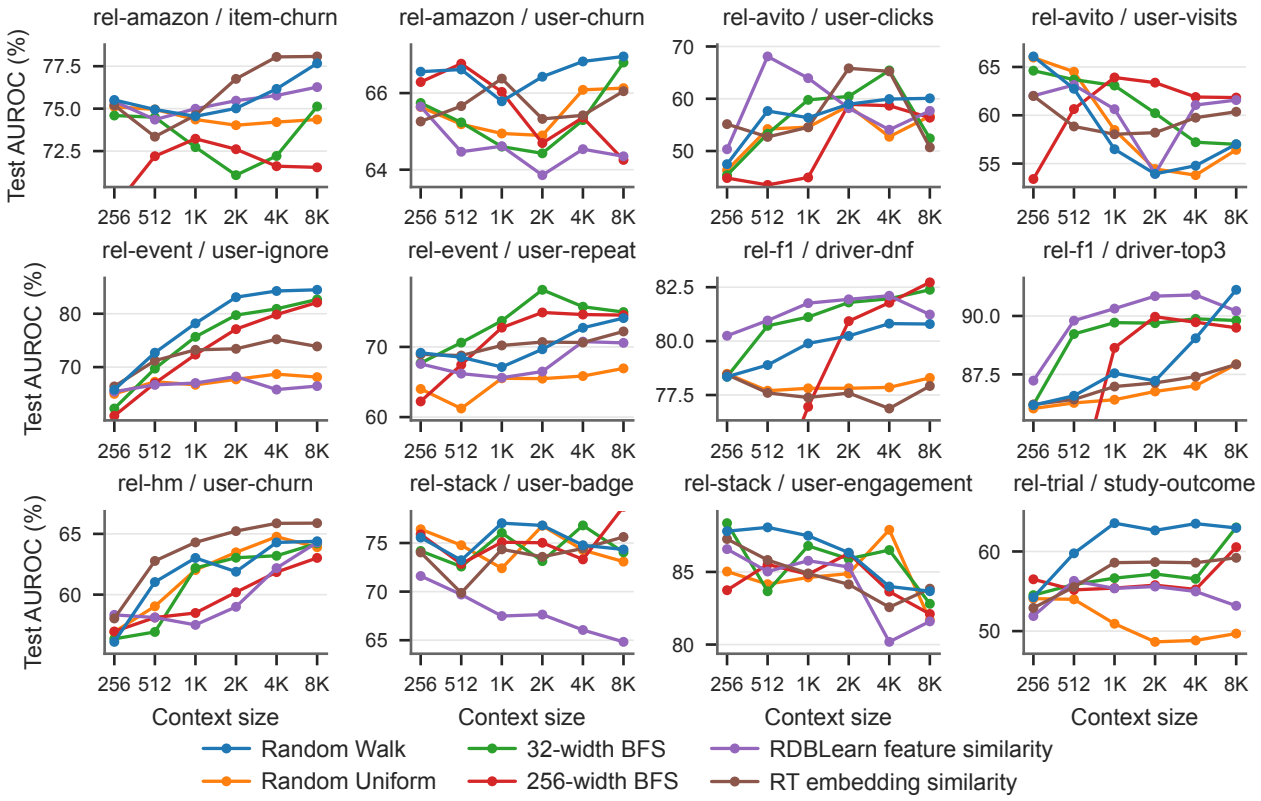


Figure 15. Per-task test AUROC for context-retrieval ablations on the twelve RelBench classification tasks.

G.2. In-context labels per task

Per-task counterpart of the in-context labels panel of Fig. 5. The mean number of in-context labels delivered by each retriever depends on both the entity-table density and the training-table cadence, so absolute label counts vary widely across tasks; the per-retriever ordering, however, is consistent with the aggregate.

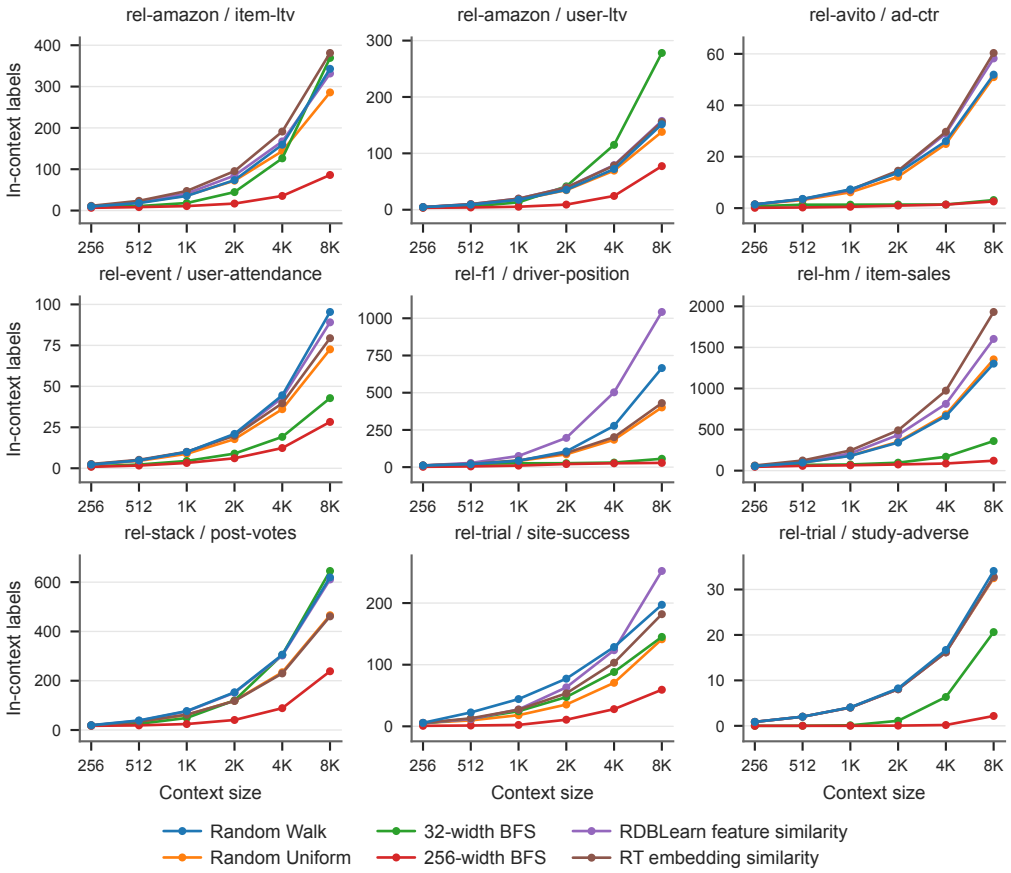


Figure 16. Per-task mean in-context labels for context-retrieval ablations on the nine RelBench regression tasks.

G.3. Multi-cell masking rate (per task)

Per-task counterpart of Fig. 6. Higher masking rates (denser per-window supervision) help essentially every regression task across the context sweep, in line with the aggregate result. On classification, the three rates are close on most tasks; the gap between $p_{\max} = 0.5$ and $p_{\max} = 0$ is largest on the longer context regime.

G.4. Schema semantics (per task)

Per-task counterpart of Fig. 7. Schema semantics help on nearly every task, with the gap most pronounced where column meanings disambiguate otherwise structurally identical features.

G.5. Pretraining task mix (per task)

Per-task counterpart of Fig. 8. The mixed objective inherits the strengths of both single-task variants on almost every task: autocompletion-only is competitive on regression but under-performs on classification at large L, and forecasting-only exhibits the reverse pattern.

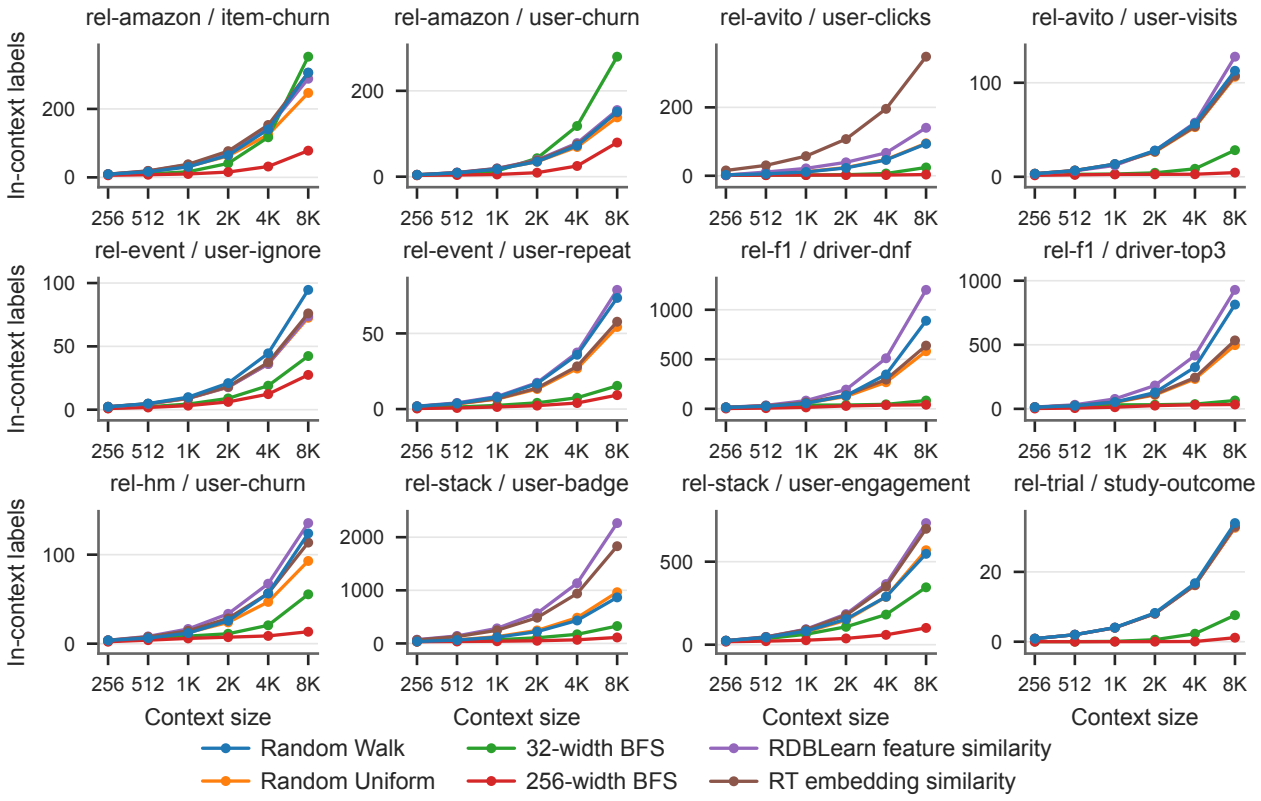


Figure 17. Per-task mean in-context labels for context-retrieval ablations on the twelve RelBench classification tasks.

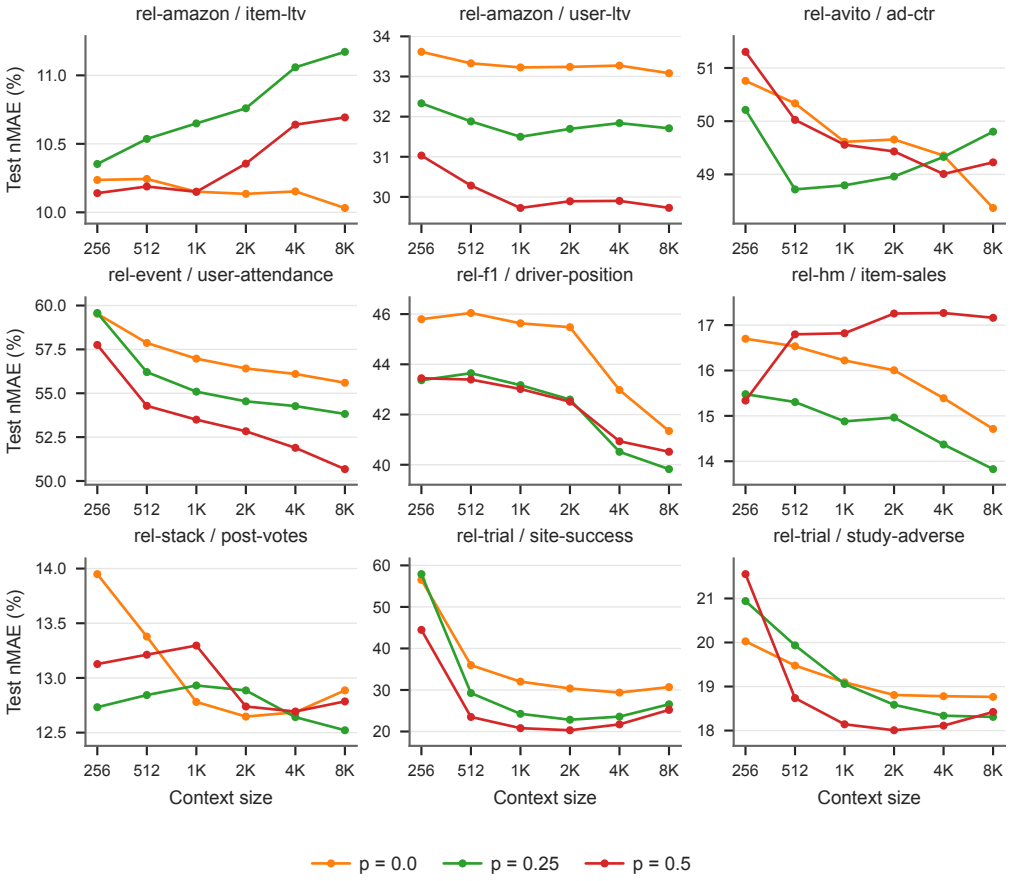


Figure 18. Per-task test MAE for multi-cell masking ablations on the nine RelBench regression tasks.

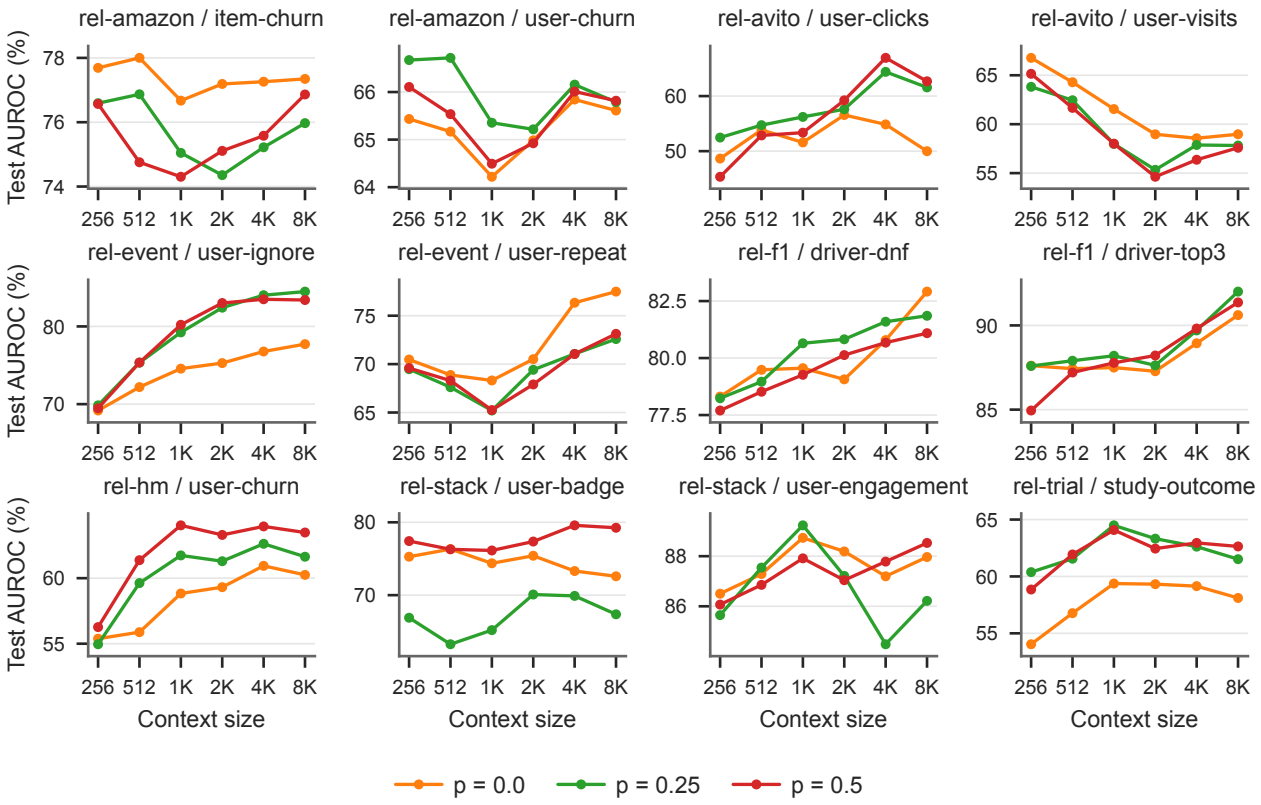


Figure 19. Per-task test AUROC for multi-cell masking ablations on the twelve RelBench classification tasks.

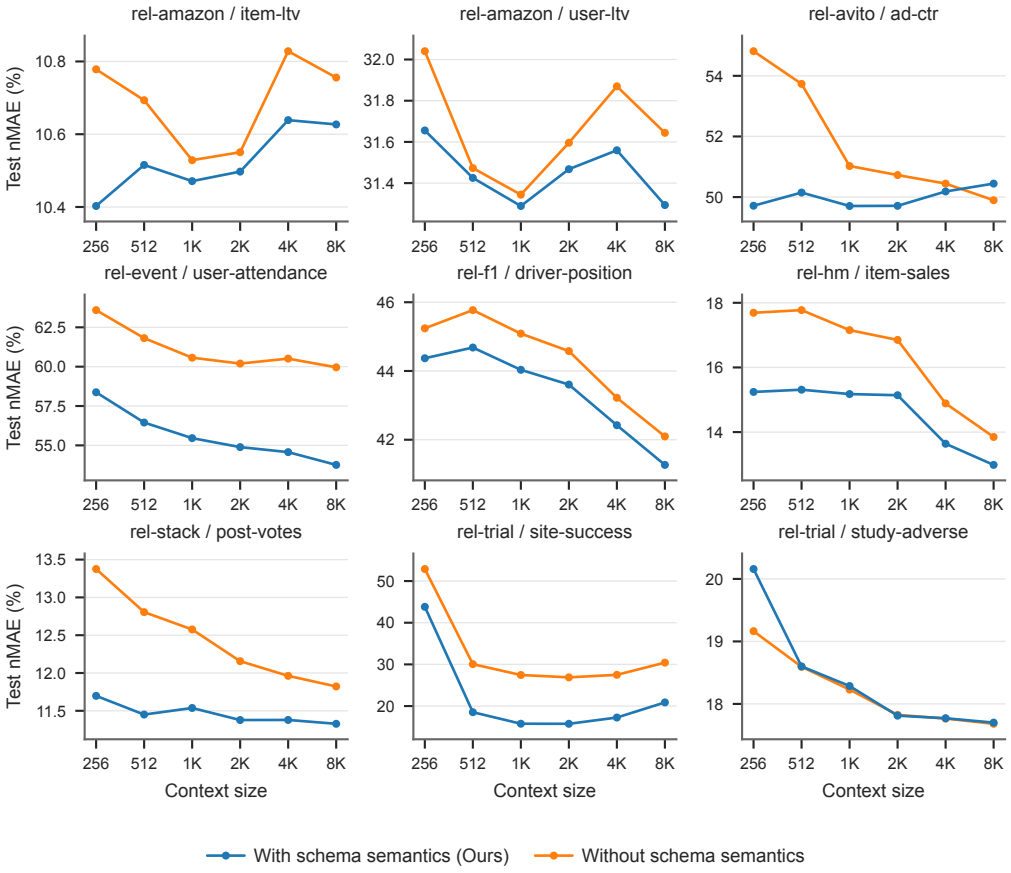


Figure 20. Per-task test MAE for the schema-semantics ablation on the nine RelBench regression tasks.

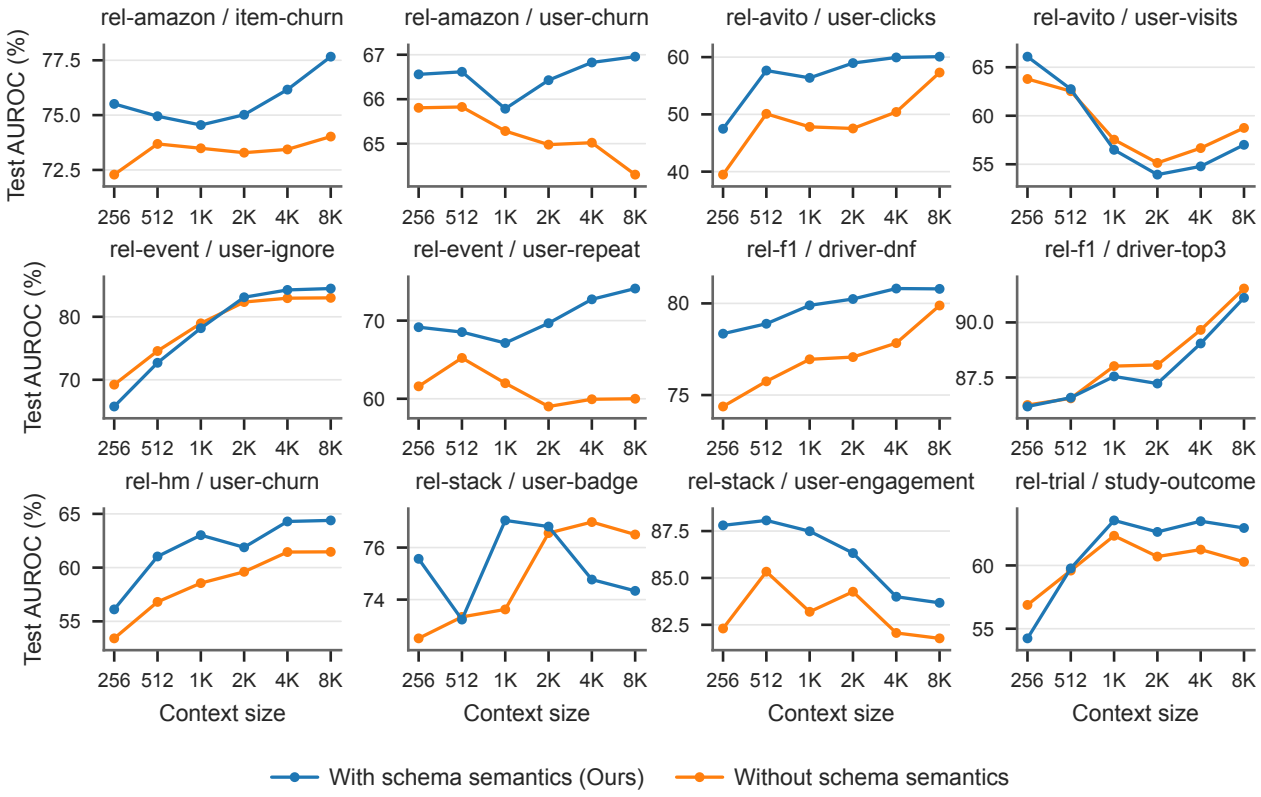


Figure 21. Per-task test AUROC for the schema-semantics ablation on the twelve RelBench classification tasks.

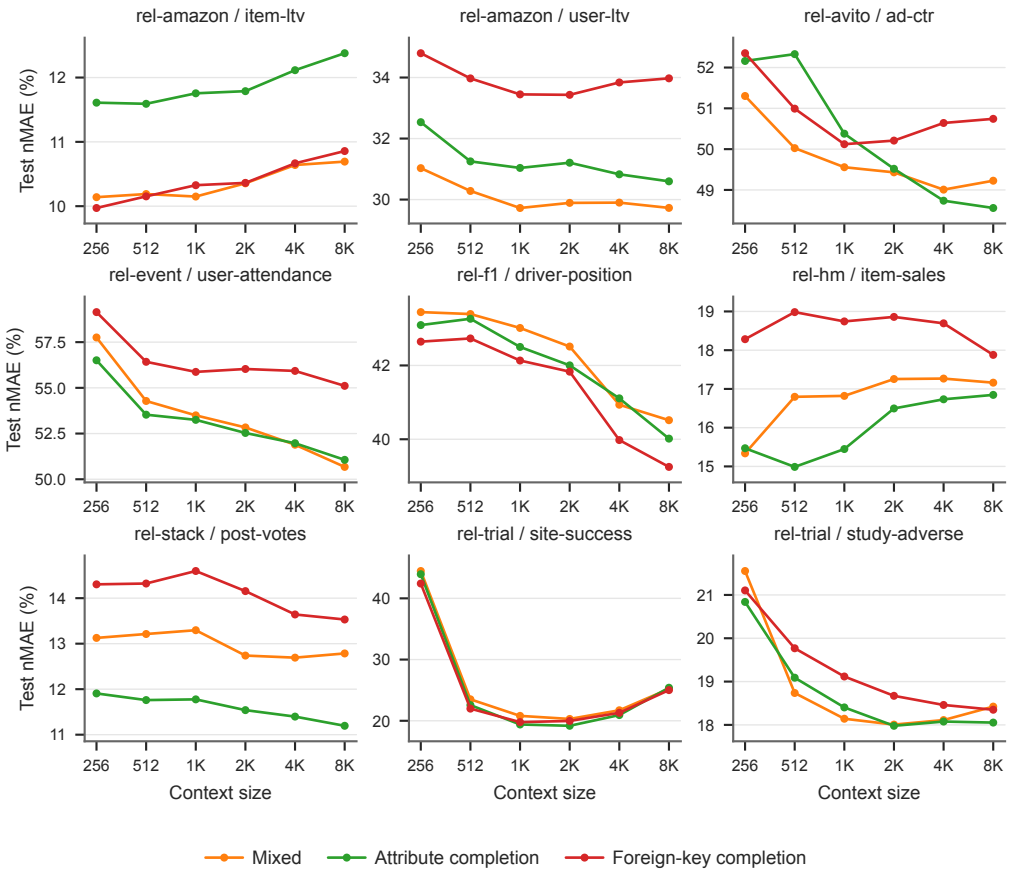


Figure 22. Per-task test MAE for the pretraining-task-mix ablation on the nine RelBench regression tasks.

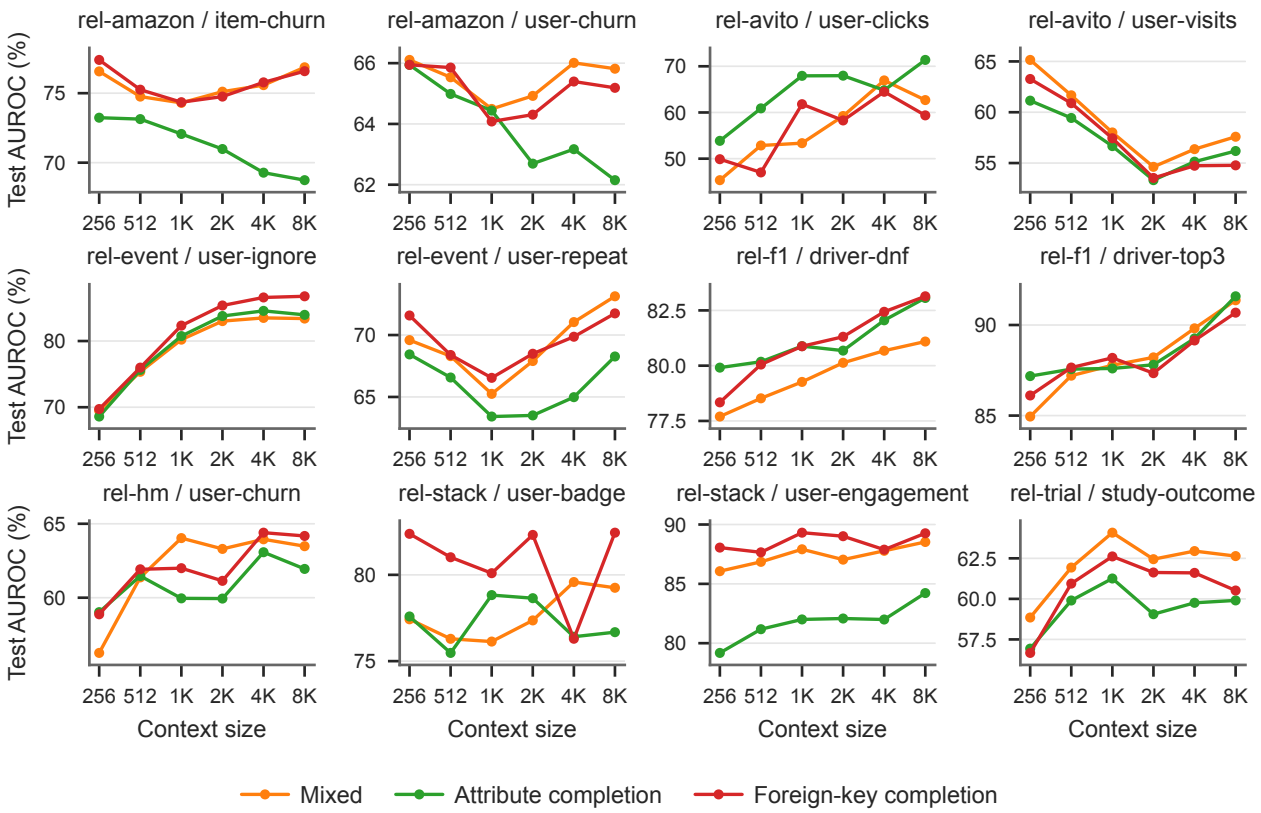


Figure 23. Per-task test AUROC for the pretraining-task-mix ablation on the twelve RelBench classification tasks.

G.6. Context ensembling (per task)

Per-task counterpart of Fig. 3. Increasing the number of context-shuffle seeds n_{ens} helps on essentially every task on classification and on most tasks on regression; a handful of regression tasks are insensitive to ensembling, in line with their flatter aggregate curves.

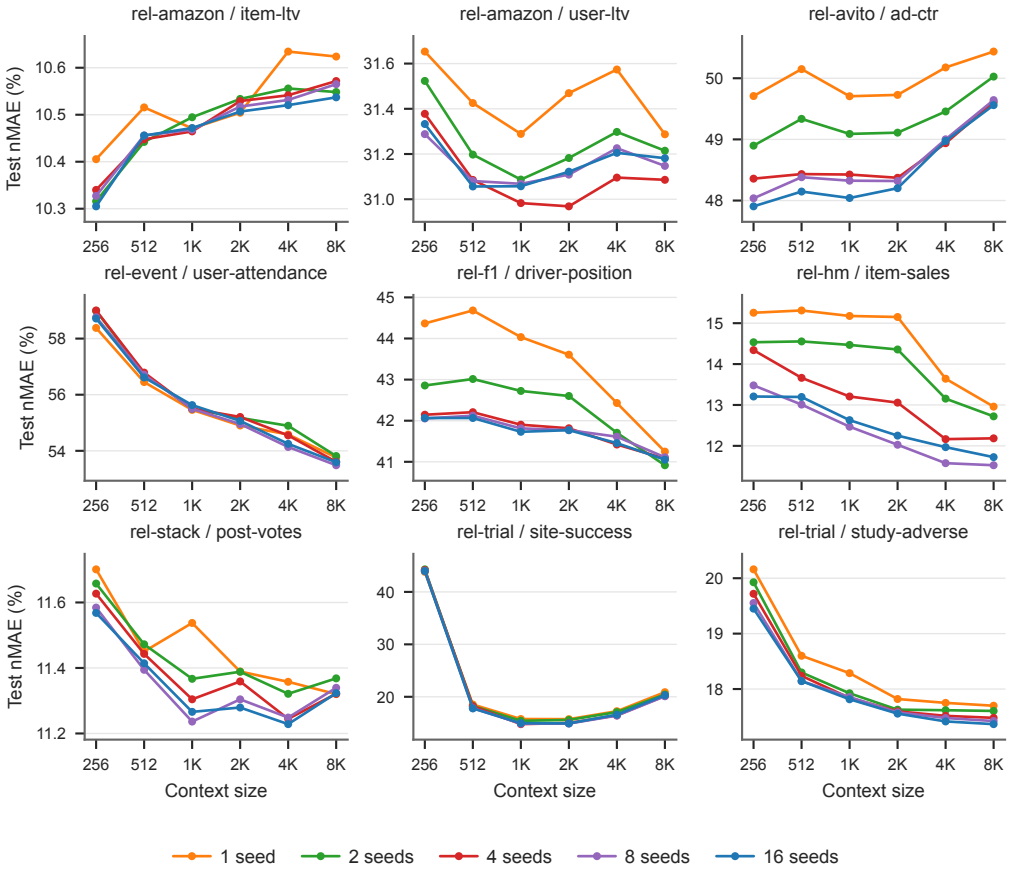


Figure 24. Per-task test nMAE for context-shuffle ensembling on the nine RelBench regression tasks.

G.7. Context hyperparameter tuning (per task)

Per-task counterpart of Fig. 4. The selected per-task (ℓ, b) are reported in § G.8; the curves below trace the resulting test metric across the full context grid. The mixed picture matches the aggregate: regression tasks that pick $\ell > 256$ pay a penalty at $L \leq 1\text{k}$ but recover at large L, while most classification tasks improve uniformly.

G.8. Context hyperparameter selection (per task)

Each task's reported metrics are obtained via a two-stage protocol. In the first stage, a grid search is conducted over the per-seed cell budget $\ell \in \mathcal{L}_\ell$, the BFS width $b \in \mathcal{B}$, and the recency flag $\rho \in \mathcal{R}$ at a fixed global context length of $L = 8\text{k}$, restricted to the validation split. For each of the thirty-two (ℓ, b, ρ) configurations, four context-shuffle seeds are evaluated and the per-row predictions are averaged across seeds prior to computing the metric (mean absolute error for regression tasks; ROC-AUC for classification tasks). The configuration that minimizes the validation MAE (resp. maximizes the validation AUC) is selected per task, with ties resolved by preferring the smaller ℓ, then the smaller b, and finally $\rho = 1$ over $\rho = 0$. In the second stage, the selected (ℓ, b, ρ) is evaluated on the held-out test split using sixteen context-shuffle seeds, again ensembled by averaging predictions before scoring. Table 11 reports the validation and test metrics so obtained at $L = 8\text{k}$, together with the selected (ℓ, b, ρ).

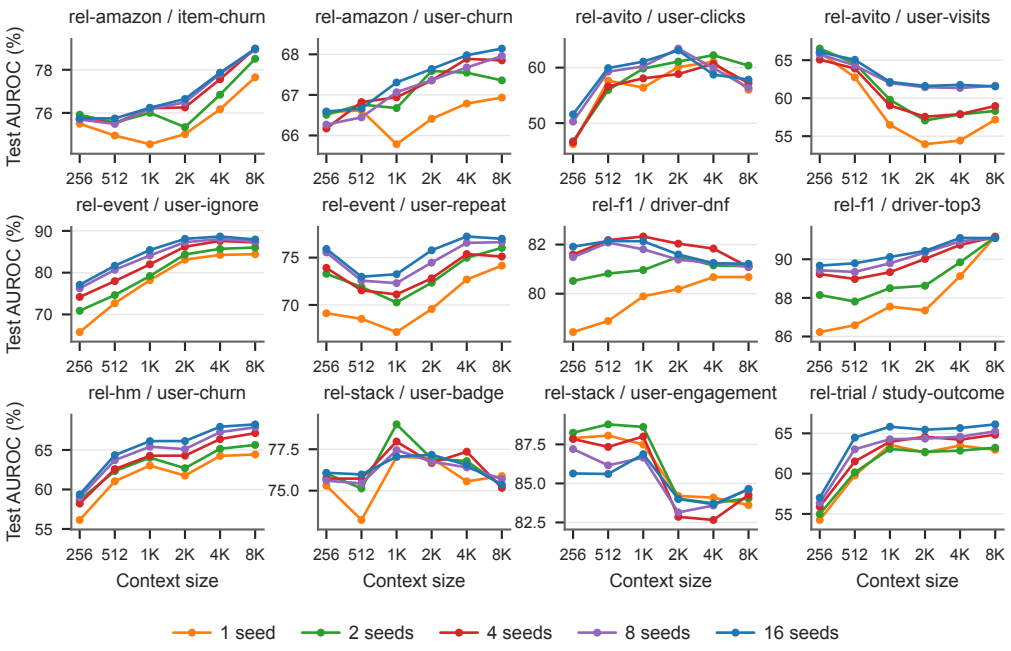


Figure 25. Per-task test AUROC for context-shuffle ensembling on the twelve RelBench classification tasks.

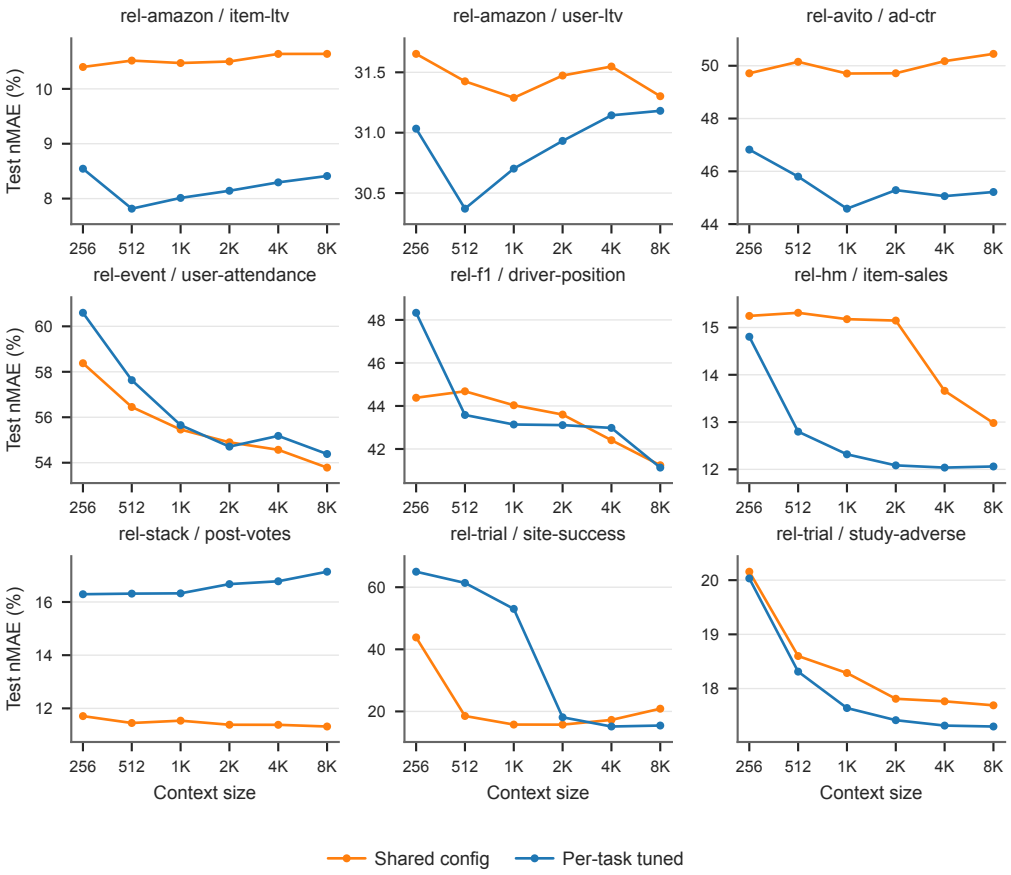


Figure 26. Per-task test nMAE for context-hyperparameter tuning on the nine RelBench regression tasks.

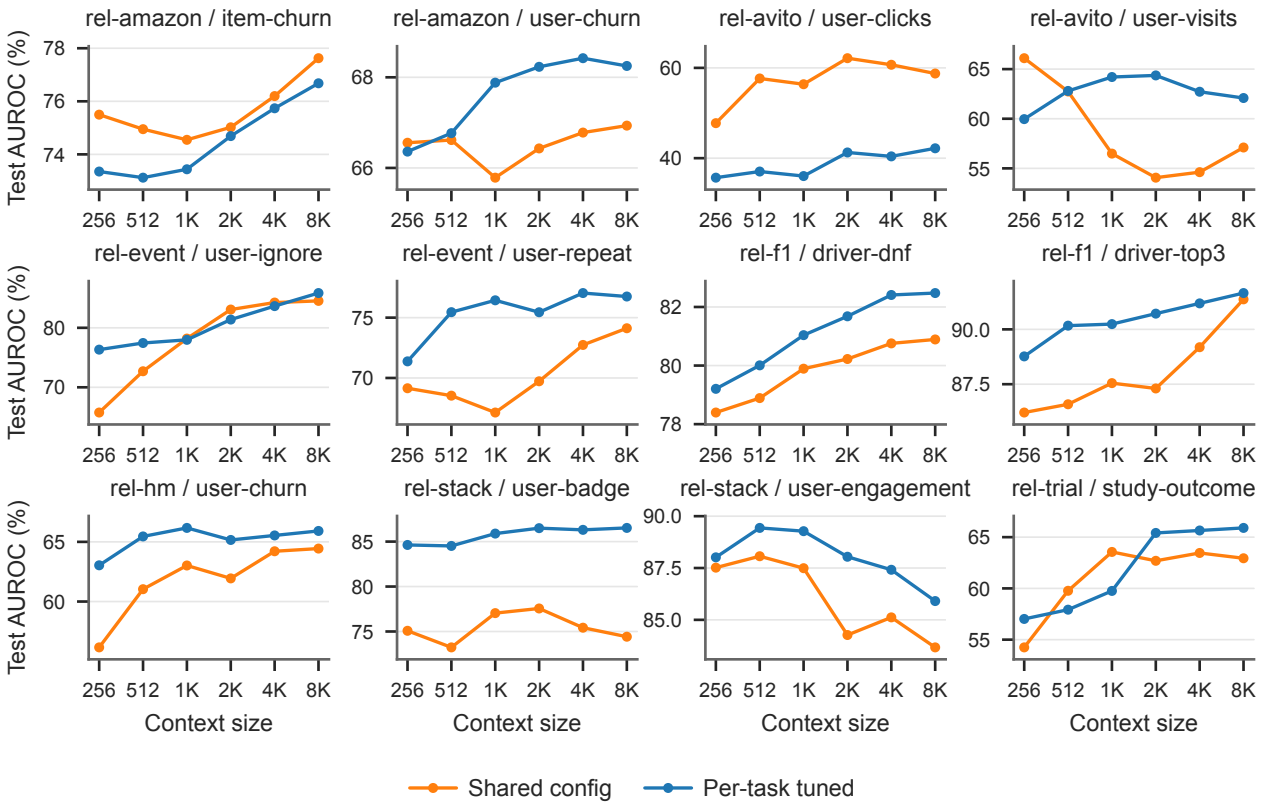


Figure 27. Per-task test AUROC for context-hyperparameter tuning on the twelve RelBench classification tasks.

Table 11. Per-task val and test metrics at $L = 8$k using the val-best (ℓ, b, ρ) per task. Columns `lcs`, `bw`, `pl` report the selected ℓ, b, and ρ ($\mathtt{T} = 1$, $\mathtt{F} = 0$). Test numbers ensemble over 16 context seeds; val numbers ensemble over 4. Lower is better for MAE, higher is better for AUC.

DB	Task	lcs	bw	pl	Val MAE	Test MAE
rel-amazon	item-ltv	512	32	F	0.0998	0.1026
rel-amazon	user-ltv	256	32	F	0.2422	0.3018
rel-avito	ad-ctr	1k	128	T	0.3978	0.4842
rel-event	user-attendance	2k	256	F	0.5808	0.5433
rel-f1	driver-position	512	256	T	0.4231	0.4114
rel-hm	item-sales	256	32	T	0.1576	0.1173
rel-stack	post-votes	1k	256	F	0.1358	0.1145
rel-trial	site-success	1k	256	T	0.1740	0.1592
rel-trial	study-adverse	256	256	F	0.1817	0.1735
Mean					**0.2659**	**0.2675**

DB	Task	lcs	bw	pl	Val AUC	Test AUC
rel-amazon	item-churn	256	32	T	0.8059	0.7898
rel-amazon	user-churn	512	32	T	0.6883	0.6873
rel-avito	user-clicks	2k	128	T	0.6325	0.5914
rel-avito	user-visits	2k	256	T	0.5707	0.6196
rel-event	user-ignore	256	32	F	0.8670	0.8558
rel-event	user-repeat	1k	128	T	0.7241	0.7680
rel-f1	driver-dnf	2k	256	T	0.7573	0.8249
rel-f1	driver-top3	512	64	T	0.8791	0.9170
rel-hm	user-churn	256	64	F	0.6984	0.6713
rel-stack	user-badge	256	32	T	0.9124	0.7525
rel-stack	user-engagement	2k	128	T	0.9105	0.8394
rel-trial	study-outcome	256	32	F	0.6220	0.6646
Mean					**0.7557**	**0.7485**

Table 12. Per-task test metrics versus RDBLearn (Zhang et al., 2026), as reported in the KumoRFM-2 paper (Hudovernik et al., 2026). RT uses the val-best $(\mathrm{lcs}, \mathrm{bw})$ at $\mathrm{ctx} = 8\mathrm{k}$ ensembled over 16 context seeds (Table 11); RDBLearn is trained on up to 10k labels per task. Regression MAE is divided by the train-target standard deviation so the per-task arithmetic mean is meaningful. All metrics shown in percent. Best per row in bold; lower is better for nMAE, higher is better for AUROC.

DB	Task	RT (ours, $\sim$ 400 labels)	RDBLearn (10k labels)
Regression (nMAE %, lower is better)			
rel-amazon	item-ltv	10.26	**8.22**
rel-amazon	user-ltv	30.18	**25.28**
rel-avito	ad-ctr	48.42	**35.53**
rel-event	user-attendance	54.33	**30.97**
rel-f1	driver-position	**41.14**	54.58
rel-hm	item-sales	**11.73**	12.92
rel-stack	post-votes	**11.45**	13.32
rel-trial	site-success	**15.92**	89.10
rel-trial	study-adverse	17.35	**12.95**
Mean		**26.75**	31.43
Classification (AUROC %, higher is better)			
rel-amazon	item-churn	78.98	**82.07**
rel-amazon	user-churn	**68.73**	67.57
rel-avito	user-clicks	59.14	**69.04**
rel-avito	user-visits	61.96	**65.49**
rel-event	user-ignore	**85.58**	82.52
rel-event	user-repeat	**76.80**	75.04
rel-f1	driver-dnf	**82.49**	70.87
rel-f1	driver-top3	**91.70**	79.69
rel-hm	user-churn	67.13	**68.05**
rel-stack	user-badge	75.25	**85.26**
rel-stack	user-engagement	83.94	**89.39**
rel-trial	study-outcome	66.46	**71.58**
Mean		74.85	**75.97**

H. LLM Use Disclosure

In line with the workshop policy, we disclose that LLMs were used as follows. (1) **Corpus construction.** The data pipeline of § 3 uses Claude Code agents to author per-domain data collectors and adapt standardization routines, and uses Claude Opus to propose forecasting tasks for each database. Both uses are gated by deterministic validators (mechanical loadability checks for collectors; the schema and template checker of § C.4 for tasks), so every released artefact is mechanically verifiable. (2) **Baselines.** The LLM Agent baseline (§ 5) is itself a method we evaluate, not a research aid. (3) **Writing.** An LLM was used to draft and polish text in this paper, under author review. The technical contributions, experiments, and results are entirely the authors'.