PRETRAINING IN RELATIONAL DATABASES ACROSS TASKS AND ENTITY TYPES

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational databases support many predictive tasks over shared histories, yet relational deep learning models are typically trained separately for each target and entity type. We study whether a conventional relational encoder can instead be pretrained once on many query-defined forecasting tasks and reused. Query-Conditioned Relational Pretraining (QCRP) generates future-prediction objectives from structured query specifications and trains them jointly through a shared query-conditioned predictor. Evaluated on RelBench, frozen QCRP encoders improve over existing relational pretraining objectives on standard tasks. Because standard targets largely overlap with pretraining objectives, we introduce 24 strictly held-out tasks, where QCRP also demonstrates strong transfer. A single encoder pretrained across entity types remains competitive with type-specific specialists. We further find that query-defined tasks share low-dimensional structure, which the query encoder captures to support zero-shot prediction for unseen queries, although its advantage over nearest-query substitution is limited. These results suggest that task-diverse pretraining is a practical route to reusable database-level encoders, and that task specification is a promising interface for future work.

1 INTRODUCTION

Relational databases provide a shared history that supports diverse predictive tasks. By leveraging past records up to a specific cutoff, we can forecast future outcomes for a given entity, such as whether a customer will churn or how a product will perform. Because this historical context is distributed across multiple tables, a natural approach translates the database into a graph by representing rows as nodes and primary–foreign key relations as edges. Over this structure, relational deep learning (RDL) typically applies graph neural networks (GNNs) (Fey et al., 2024; Robinson et al., 2024; Chen et al., 2025; Dwivedi et al., 2025) to aggregate information and produce rich entity representations. This enables models to learn directly from interconnected tables, significantly reducing the need for manual feature engineering.

Despite sharing the same database history, supervised RDL pipelines typically train a separate model for each predictive task. This repeated training motivates learning representations that can be reused across tasks and root entity types (Peleška, 2026). While self-supervised pretraining offers a route to such reuse, methods like masked reconstruction (Hou et al., 2022) and graph contrastive learning (You et al., 2020; Peleška & Šír, 2025) capture useful information but do not directly supervise future outcomes. Consequently, these generic objectives often transfer inconsistently to relational data (Liu et al., 2022).

An alternative is to pretrain on generated tasks. STUNT (Nam et al., 2023) constructs tasks from columns of unlabeled tables, while TVE (Truong et al., 2025) and RelPrism (Yang et al., 2026) predict future statistics in relational databases. However, these relational methods pretrain for one entity type at a time, and overlap between pretraining objectives and benchmark targets limits the evidence for transfer to unseen tasks (Azevedo et al., 2026; Gu et al., 2026).

More broadly, relational prediction tasks often combine recurring components such as entity type, relational path, target column, and aggregation (Kanter & Veeramachaneni, 2015). Predictive Query Language (PQL) and Relational Task Generation Language (RTGL) express such tasks through

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

declarative specifications (Kocijan et al., 2026; Kolesnichenko et al., 2026). While task-conditioned and prompted multitask models can generalize from such task descriptions—most closely Every-Query for clinical outcomes (Mahabadi et al., 2021; Sanh et al., 2021; Liu et al., 2024; Chandak et al., 2026)—relational models are typically pretrained per root (entity) type. Yet, heterogeneous GNNs inherently compute representations for multiple node types via relational message passing (Hu et al., 2020), admit self-supervised pretraining (Wang et al., 2021; Jiang et al., 2021), and support joint learning across node types through selective structural sharing (Feng et al., 2025; Yoon et al., 2022; Caruana, 1997).

Unlike relational foundation models, which transfer across databases via large-scale pretraining and in-context learning (Wang et al., 2025; Fey et al., 2025; Ranjan et al., 2026a; Hudovernik et al., 2026), our goal is to maximize reuse within a single database without altering conventional encoders. We introduce Query-Conditioned Relational Pretraining (QCRP), a lightweight framework for database-level pretraining that constructs scalar prediction objectives from structured future queries and optimizes them across shared relational minibatches. A query encoder represents each task specification, while a shared predictor combines this query embedding with the GNN's entity embedding to predict the target. By pretraining jointly across root tables, QCRP consolidates distinct task families into a single relational encoder, while its zero-shot interface enables prediction conditioned entirely on the task specification.

Our contributions are:

1. **Task-aware relational representation learning.** We introduce QCRP and evaluate its representations using frozen-encoder MLP probes and full fine-tuning. Over standard RelBench targets and held-out query-defined tasks, frozen QCRP representations improve over existing relational pretraining baselines and remain competitive with task-specific supervised training (Section 3).
2. **Database-level pretraining across roots.** We study joint pretraining across root entity types using a single conventional GNN relational encoder. Comparisons with specialists under matched per-root exposure show that one checkpoint can remain competitive across roots, with positive transfer in several settings. We analyze training dynamics, including root scheduling and switch-induced interference, to understand when this shared pretraining is effective (Section 4).
3. **Explicit query-conditioned predictive interface.** We show that structured query conditioning enables prediction without fitting a separate head for each requested task. We evaluate this interface on pretraining and held-out query-defined tasks, and analyze how query semantics relate to the learned task representations and predictions (Section 5).

2 Query Conditioned Relational Pretraining

Following the temporal prediction setting of RDL and RelBench (Fey et al., 2024; Robinson et al., 2024), let v denote an entity from root table r, t a prediction cutoff, and $\mathcal{H}_{\leq t}$ the relational history available before t. We define a relational encoder E_θ as a table encoder that maps row attributes to initial entity embeddings, followed by a message-passing network that aggregates information across primary–foreign key links (Fey et al., 2024). QCRP pretrains this encoder by predicting query-defined future outcomes. The relational encoder computes $z_{v,t} = E_\theta(v, \mathcal{H}_{<t})$. An additional query encoder maps the task specification q to $e_q = G_\phi(q)$, and a shared predictor produces $\hat{y}_{v,t,q} = D_\psi(z_{v,t}, e_q)$. After pretraining, we can retain only the relational encoder for query-independent entity representations, or retain the entire system for zero-shot prediction.

2.1 Task Specification and Query Encoding

Relational prediction tasks often share components such as a root entity type, relational path, target column, aggregation, and prediction horizon. Declarative query languages make these components explicit (Kocijan et al., 2026; Kolesnichenko et al., 2026). We represent each task specification q through these recurring fields. For example, the task *"predict the total sales of a product, where reviews are verified, over the next month"* is specified by primitive fields such as: product root, product–review path, verified equals true filter, count aggregation, multiply by price context, one-month horizon, and regression output type.

Pretraining task construction. We construct a finite pretraining pool $\mathcal{Q}_r$ for each root type by combining these fields under schema and datatype constraints. Following future-statistic supervision

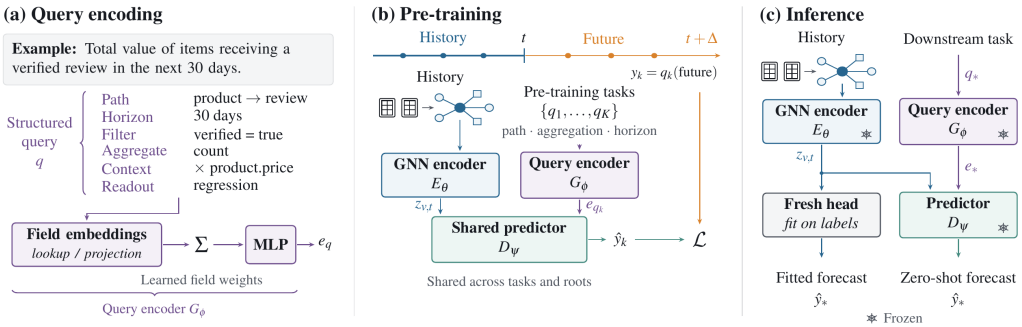

Figure 1: **QCRP overview.** (a) Recurring fields in relational prediction tasks are encoded into query embeddings. (b) A shared predictor combines historical entity representations with query embeddings to predict future targets. (c) The pretrained relational encoder supports task-specific fitted heads. Retaining the query encoder and shared predictor additionally enables zero-shot prediction.

(Truong et al., 2025), targets aggregate records reached through future root-linked events in $(t, t+\Delta]$. Supported aggregations include count, distinct count, mean, minimum, maximum, sum, mode, and standard deviation. Optional filters restrict the contributing records. We also combine aggregates with root attributes available at t, reflecting the complementary intrinsic and relational information emphasized by Yang et al. (2026). Examples include multiplying future review count by product price or ranking an aggregate within a root-attribute category. The readout returns the resulting scalar or applies a comparison to obtain a binary target. Thresholds and task-selection statistics are computed from training data. We exclude degenerate targets and insufficiently frequent predicates. The selected pools provide auxiliary supervision over a range of relational quantities.

Query encoder. Each field q_j has a type-specific encoder $E_j(q_j) \in \mathbb{R}^{d_q}$. Categorical fields use learned embeddings, numerical fields use normalized values followed by a small MLP, and schema paths use mean-pooled table embeddings. We combine them as $e_q = f_q(\mathrm{LN}[\sum_{j \in \mathcal{F}(q)} \alpha_j \, E_j(q_j)])$, where $\mathcal{F}(q)$ contains the fields present in q, α_j is a learned scalar for field type j, and f_q is a shared projection MLP. Absent optional categorical fields contribute a learned `None` embedding, and optional numerical fields are masked to zero. Shared field encoders reuse representations of common components across tasks and allow new combinations of supported values to be encoded.

2.2 Multi-Root Multi-Task Pretraining

Shared relational computation. For root type r, we sample a minibatch $\mathcal{B}_r = \{(v_i, t_i)\}_{i=1}^{B}$ from its entity–cutoff pool and a set $\mathcal{S}_r$ of $\min(K, |\mathcal{Q}_r|)$ queries uniformly without replacement. In each update, the relational encoder computes each historical representation z_i once, and the query encoder processes K sampled queries. We use $K = 16$ queries per root minibatch to balance training speed and maintain a good estimate of the entire task pool while serving as implicit regularization. The shared predictor applies an MLP to $[z_i \| G_\phi(q)]$ for every sampled query, producing a regression value or classification logit. This late conditioning shares temporal neighborhood sampling and message passing across objectives while preserving query-independent entity representations. Future observations supply targets only and never enter the encoder.

Training objective. Let $\mathcal{B}_{r,q}$ contain minibatch examples with defined targets for query q, and let $n_q = |\mathcal{B}_{r,q}|$. Writing $\mathcal{S}_{r,+}$ for sampled queries with $n_q > 0$, we compute

$$\mathcal{L}_r = \frac{\sum_{q \in \mathcal{S}_{r,+}} w_q \, \frac{1}{n_q} \sum_{i \in \mathcal{B}_{r,q}} \ell_q(\hat{y}_{i,q}, \tilde{y}_{i,q})}{\sum_{q \in \mathcal{S}_{r,+}} w_q}, \qquad w_q = \min\left(1, \sqrt{n_q/256}\right). \tag{1}$$

Here, $\tilde{y}_{i,q}$ denotes the training-transformed target. Binary objectives use binary cross-entropy with logits. Regression objectives use Huber loss with transition parameter one after training-set standardization. Count, distinct-count, and sum targets additionally receive a $\log(1+y)$ transform before standardization. The reliability weight reduces the influence of objectives supported by few valid examples. Undefined targets are masked, while valid zero counts remain supervision. Minibatches with no valid targets are skipped.

Training across root types. Multi-root pretraining shares parameters $\Theta = (\theta, \phi, \psi)$ across root-specific anchor and query pools, replacing separate checkpoints with one database-level model. Because shared optimization across multiple node types can introduce task competition (Standley et al., 2020; Feng et al., 2025), we study two training regimes: joint training and block-wise interleaving.

Joint training behaves like standard multitask learning: it samples a minibatch from each of the N roots, averages their losses, and takes a single optimizer step so all roots contribute gradients simultaneously. Block-wise interleaving (Mayo et al., 2023) instead trains exclusively on one root for a block of T contiguous updates before rotating to the next root, meaning successive blocks update the shared model using different root-specific objectives. The detailed training algorithms are in Appendix A. Section 4 compares these schedules, examines switching dynamics, and evaluates shared encoders against root-specific specialists under matched per-root exposure.

Table 1: **Frozen representation reuse across backbones.** Mean ± sample SD over three seeds. LTV/sales: MAE; churn: AUC $\times 100$. Pretrained encoders are frozen and receive fresh MLP probes; supervised training is end-to-end from scratch. Bold marks the best pretrained mean within each backbone.

	Rel-Amazon				Rel-HM	
Method	user-LTV↓	item-LTV↓	user-churn↑	item-churn↑	item-sales↓	user-churn↑
GraphSAGE						
Supervised	14.501 ±0.034	47.403 ±0.083	69.338 ±0.234	81.642 ±0.061	0.039 ±0.000	68.984 ±0.176
Random	16.444 ±0.029	60.431 ±0.257	65.772 ±0.062	74.604 ±0.474	0.051 ±0.000	66.616 ±0.487
MaskAE	15.698 ±0.105	60.135 ±0.436	66.565 ±0.341	64.899 ±1.715	0.072 ±0.002	68.161 ±0.151
CTR	14.992 ±0.037	57.527 ±0.323	66.202 ±0.241	76.071 ±0.422	0.057 ±0.001	67.114 ±0.089
TVE-2	14.652 ±0.019	54.544 ±0.100	69.463 ±0.027	81.975 ±0.033	0.046 ±0.002	68.476 ±0.102
TVE-2+CTR	14.426 ±0.027	55.096 ±0.299	69.730 ±0.086	81.921 ±0.069	0.041 ±0.001	68.724 ±0.071
$\text{QCRP}_{\text{root}}$	14.343 ±0.008	**46.203** ±0.117	**69.976** ±0.077	82.268 ±0.061	**0.039** ±0.000	68.613 ±0.172
$\text{QCRP}_{\text{multi}}$	**14.328** ±0.040	47.252 ±0.375	69.872 ±0.031	**82.301** ±0.055	0.040 ±0.000	**68.985** ±0.052
RelGNN						
Supervised	14.123 ±0.012	47.897 ±0.183	70.782 ±0.040	82.627 ±0.038	0.049 ±0.002	70.634 ±0.259
Random	16.414 ±0.018	60.325 ±0.351	64.721 ±0.228	73.412 ±0.757	0.050 ±0.001	66.527 ±0.381
TVE-2	14.040 ±0.016	52.498 ±0.388	70.861 ±0.042	**83.030** ±0.036	0.040 ±0.001	69.441 ±0.186
TVE-2+CTR	13.950 ±0.028	52.442 ±0.352	70.957 ±0.008	82.952 ±0.014	0.038 ±0.000	69.839 ±0.094
$\text{QCRP}_{\text{root}}$	**13.808** ±0.011	**44.692** ±0.637	**70.958** ±0.064	83.012 ±0.040	0.036 ±0.001	69.749 ±0.098
$\text{QCRP}_{\text{multi}}$	13.834 ±0.024	46.103 ±1.690	70.874 ±0.027	82.902 ±0.035	**0.036** ±0.001	**69.955** ±0.156

$\text{QCRP}_{\text{root}}$: separate root-specific checkpoints; $\text{QCRP}_{\text{multi}}$: one checkpoint per database, with matched per-root exposure. MaskAE: masked reconstruction; CTR: contrastive pretraining alone. TVE uses two-hop statistics; TVE+CTR combines TVE with the contrastive learning objective ($\mathcal{L} = 0.9\mathcal{L}_{\text{TVE}} + 0.1\mathcal{L}_{\text{CTR}}$). SDs are rounded; bold does not imply statistical significance.

3 Evaluating Pretrained Representations

3.1 Frozen Adaptation on RelBench

Setup. We evaluate four tasks on Rel-Amazon and two on Rel-HM using the official temporal splits (Robinson et al., 2024). Each pretrained relational encoder is frozen, including its table encoder and message-passing parameters, and receives a fresh task-specific MLP. We report mean absolute error (MAE) for regression and ROC-AUC for classification, averaged over three seeds. Task-specific supervised training from scratch provides an end-to-end reference.

Table 1 compares QCRP with masked reconstruction (MaskAE; Hou et al., 2022), GraphCL-based contrastive learning (CTR; You et al., 2020), and TVE with two-hop objective, with and without CTR (Truong et al., 2025). MaskAE and CTR follow the relational adaptations used by Truong et al. (2025). We evaluate two relational encoders: GraphSAGE (Hamilton et al., 2017) and RelGNN (Chen et al., 2025). $\text{QCRP}_{\text{root}}$ trains separate root-specific checkpoints like all other SSL methods, whereas $\text{QCRP}_{\text{multi}}$ trains one checkpoint per database under matched per-root exposure.

Future-target supervision outperforms generic SSL objectives on these six tasks. QCRP improves frozen adaptation across several tasks and both backbones. With GraphSAGE, TVE and both QCRP variants outperform MaskAE and CTR on all six tasks. This pattern supports the value of future-target supervision for the forecasting tasks evaluated here. $\text{QCRP}_{\text{root}}$ improves over TVE

on all six tasks and on TVE+CTR on five. $\text{QCRP}_{\text{multi}}$ improves over both baselines on all six. The RelGNN results also favor QCRP on the regression tasks, while classification results are more mixed. For example, $\text{QCRP}_{\text{root}}$ reduces item-LTV MAE from 52.442 with TVE+CTR to 44.692. On Amazon user-churn, the two methods have nearly identical means, while TVE retains the highest item-churn ROC-AUC.

Reuse across entity types. With GraphSAGE, $\text{QCRP}_{\text{multi}}$ improves over $\text{QCRP}_{\text{root}}$ on three of six tasks and remains competitive overall. RelGNN exhibits a similar trade-off: multi-root training improves HM tasks but underperforms the root-specific model on Amazon. Multi-root QCRP remains competitive with root-specific models while using one encoder per database; Section 4 examines this trade-off and its training dynamics in detail.

3.2 Task Coverage and Downstream Alignment

The six RelBench tasks considered here have compact semantic definitions based on future spending, sales, or inactivity. Five appear directly in TVE's pretraining targets, and all six are included in QCRP's supervision. Such overlap arises naturally when constructing auxiliary tasks from common relational operations. It does not invalidate frozen adaptation, but limits what these benchmarks establish about transfer to unseen tasks (Azevedo et al., 2026).

Table 2: **Downstream alignment of TVE targets.** Diagnostic prediction from ground-truth TVE two-hop target statistics. Scores are linear-regression R^2 for continuous targets and logistic-regression ROC-AUC for churn.

Database	Task	Target coord. present	Full vector	After removal
Amazon	user-LTV	Yes	1.000	0.982
	item-LTV	No	0.191	0.191
	user-churn	Yes	1.000	1.000
	item-churn	Yes	1.000	1.000
HM	item-sales	Yes	1.000	0.966
	user-churn	Yes	1.000	1.000

Removal excludes sum-price coordinates for user-LTV and item-sales; for churn, it excludes the inactivity indicator and all count and distinct-count coordinates. Other statistics are retained at every hop. Item-LTV has no direct target coordinate, so its vector is unchanged.

Table 2 examines this overlap by predicting downstream labels from ground-truth TVE statistics. Even after removing directly aligned coordinates, linear regression recovers user-LTV and item-sales with $R^2 = 0.982$ and 0.966, respectively, while logistic regression retains 1.000 ROC-AUC for churn. These results reflect redundancy among simple relational targets. For example, an aggregate that is positive whenever activity occurs can identify inactivity. Likewise, a sum can be reconstructed from count and mean over the same non-null population, allowing a nonlinear predictor to recover information absent as an explicit coordinate. Removing a target coordinate therefore need not produce a meaningfully distinct evaluation task.

Item-LTV exposes a different aspect of target coverage. Its definition combines product price with future review count, a contextualized quantity absent from the evaluated TVE vector. Linear prediction from that vector achieves only $R^2 = 0.191$. QCRP explicitly includes this combination through contextualization, and item-LTV exhibits the largest improvement over TVE-based frozen adaptations. This pattern is consistent with better target coverage contributing to the gain, although the diagnostic does not isolate its effect from other differences between the methods.

Table 3: **Frozen adaptation with GraphSAGE on 24 held-out tasks.** Score in each cell is averaged across three tasks and three seeds. Underlining marks the best mean or lowest rank.

Method	Amazon-Customer		Amazon-Product		HM-Customer		HM-Article		Mean rank ↓		
	R^2	AUC	R^2	AUC	R^2	AUC	R^2	AUC	Reg.	Cls.	**All**
TVE-2	.162±.002	75.78±.13	.223±.003	86.41±.16	.070±.002	66.19±.10	.653±.011	91.32±.04	3.75	3.58	**3.67**
TVE-2+CTR	.172±.001	74.44±.23	.221±.004	85.37±.13	.068±.003	66.13±.09	.676±.003	91.32±.23	3.67	4.08	**3.88**
$\text{QCRP}_{\text{root}}$	.180±.002	76.62±.06	.273±.003	90.36±.01	.069±.001	65.24±.13	.718±.002	92.58±.06	1.75	2.50	**2.12**
$\text{QCRP}_{\text{multi}}$	.180±.000	76.68±.09	.272±.001	90.35±.11	.067±.001	64.93±.25	.708±.003	92.30±.01	2.33	2.75	**2.54**
Supervised	.149±.008	76.75±.21	.222±.010	90.00±.59	.065±.003	66.09±.15	.710±.012	92.41±.10	3.50	2.08	**2.79**

For each seed, scores are averaged across the three tasks in each group; we report the mean and sample SD of these averages across three seeds. SD therefore reflects variation across seeds, not across tasks. AUC is reported as ROC-AUC × 100.

3.3 Frozen Adaptation to Held-Out Tasks

We evaluate 24 held-out forecasting tasks across Rel-Amazon and Rel-HM, with three regression and three classification tasks per root. Targets include spending by sales channel, rating diversity, and demand from active members. All task definitions are excluded from every compared pretraining pool. Selection uses training labels to require positive regression-target variance and classification prevalence between 0.05 and 0.95. Appendix B gives the definitions and selection procedure.

Table 3 shows that QCRP's advantages extend beyond the standard benchmark targets. $\text{QCRP}_{\text{root}}$ achieves the best overall mean rank of 2.12, followed by $\text{QCRP}_{\text{multi}}$ at 2.54 and supervised training at 2.79. Both QCRP variants improve over TVE and TVE+CTR in mean regression and classification scores for Amazon customers, Amazon products, and HM articles, yet the gains are not uniform: TVE retains the strongest group means for HM customers.

We evaluate all pretraining methods by freezing the relational encoder; only a 16.9K-parameter prediction head is trained per task. The supervised baseline instead fits the entire relational encoder of 1.36M and 2.25M parameters for Rel-Amazon and Rel-HM (parameters differ by table encoder; Rel-HM contains more columns). The comparison with supervised training differs by task type. $\text{QCRP}_{\text{root}}$ achieves the best regression mean rank (1.75), whereas supervised training ranks best on classification (2.08). Thus, frozen reuse is competitive across much of this suite, without consistently matching task-specific training in every group. Multi-root pretraining largely preserves these results in one checkpoint per database. Its group means remain close to those of root-specific QCRP, although its overall mean rank is slightly worse. Together, these results support reuse across unseen task definitions and multiple root types. They do not establish transfer to statistically unrelated targets, since the held-out tasks may share predictive structure with pretraining supervision.

4 Database Level Multi-Root Pretraining

We examine whether one encoder can retain root-specific predictive quality when pretrained across entity types. We compare training schedules (Section 4.1), measure performance relative to specialists (Section 4.2), and analyze shared and root-private computation (Section 4.3).

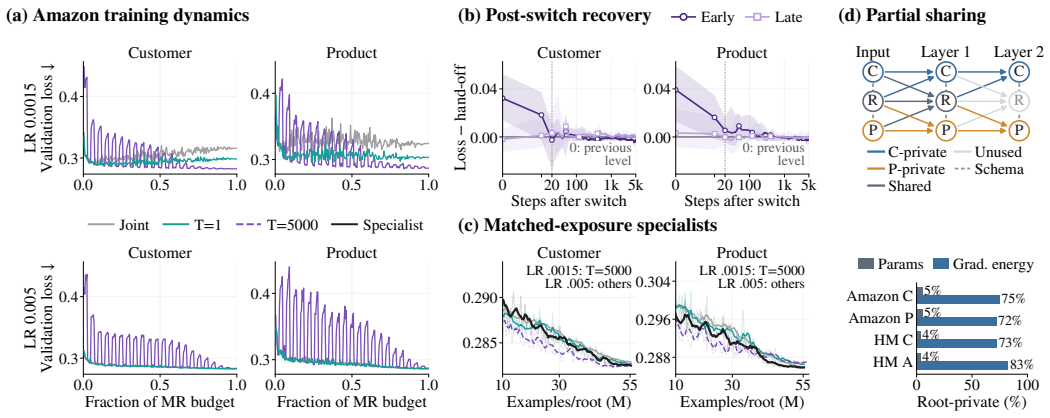

Figure 2: **Database-level consolidation.** (a) Amazon per-root validation dynamics at two learning rates. (b) Post-switch training-loss recovery for T=5000, LR 0.0015; bands show variation across early/late blocks. (c) Smoothed validation curves at matched root exposure: LR 0.0015 for T=5000, 0.005 otherwise. Dashed curves retain active-root observations. (d) Schema-aligned partial sharing and root-private parameter versus gradient-energy shares, using matched finite parameter scopes. Protocol details and HM dynamics appear in Appendix C.

4.1 Which Training Conditions Matter?

We evaluate the joint and interleaved regimes from Section 2.2 with heterogeneous GraphSAGE (Hamilton et al., 2017) on Rel-Amazon and Rel-HM. Joint updates use 256 examples from each root, while interleaved updates use 512 examples from one root at a time. We match the number of

examples seen per root: single-root models train for 108K updates and multi-root models for 216K. We compare joint training with interleaving at short (T=1) and long (T=5000) switching intervals. Figure 2a shows the Amazon results; Rel-HM exhibits the same qualitative trends and is reported in Appendix C.

Learning rate and switching interval interact. At learning rate 0.0015, long blocks (T=5000) outperform frequent switching (T=1) on both databases. At learning rate 0.005, frequent switching and joint training become competitive on Rel-Amazon. A plausible explanation is that frequent switching requires sufficiently fast adaptation: with a small step size, each short visit to a root makes little progress before the active objective changes. At matched root exposure, Amazon multiroot curves remain close to specialists with LR 0.005 or blocked interleaving at LR 0.0015, whereas low-LR joint and T=1 training show clear degradation. This supports joint selection of the learning rate and training schedule instead of a universal advantage for one regime.

Blocked training temporarily increases loss on the inactive root, with larger excursions at LR 0.005 than at 0.0015. This suggests that a smaller learning rate can limit cross-root disruption in long blocks, complementing the benefit of a larger learning rate under frequent switching. Most of the degradation is recovered shortly after the root becomes active again. In early Rel-Amazon blocks, product's excess loss relative to its previous active block falls by 86% within 30 updates, and customer's falls below its previous level within the same window. By late training the switching cost nearly vanishes, consistent with learning-rate decay and possibly with greater agreement between the two roots' objectives. This pattern is consistent with the rapid recovery reported for interleaved multitask learning (Mayo et al., 2023), rather than irreversible forgetting.

4.2 How Much Specialist Performance Is Retained?

Tables 1 and 3 show that multi-root QCRP remains close to root-specific pretraining on both standard RelBench tasks and held-out objectives. It matches or exceeds the specialist on 3/6 downstream tasks with GraphSAGE and 2/6 with RelGNN, while held-out transfer shows larger but still modest gaps. To examine consolidation more granularly, we compare validation loss over the full 221-objective pretraining pool across Rel-Amazon and Rel-HM.

Consolidation maintains overall quality but introduces asymmetric trade-offs. Mean loss improves by 3.51% on Amazon product and increases by only 0.20–0.78% on the other roots. Task-wise changes reveal stronger asymmetry: Amazon product improves on 37/38 objectives, while Amazon customer worsens on all 40. On Rel-HM, customer improves on 53/65 objectives despite a higher mean loss, indicating that a few large regressions dominate the average, whereas article improves on only 13/78. One checkpoint thus covers every root of the database with near-specialist quality, and with positive transfer for Amazon products. The cost is a small, root-dependent regression elsewhere, most consistently for Amazon customers.

4.3 A Schema-Aligned Basis for Partial Sharing

Single-root training leaves root-specific capacity unused. A heterogeneous GNN assigns separate transformations to each edge type and destination node type (Schlichtkrull et al., 2018; Hu et al., 2020). With L layers, a root's representation depends only on the modules inside its L-hop computation tree; modules whose destinations lie outside that tree receive no gradient. This property is well known. Yoon et al. (2022) trace HGNN gradient paths and show that embeddings of different node types come from largely separate parameter sets, so a model trained with labels on one type leaves the other type's feature extractor untrained. Efficient mini-batch implementations exploit the identical structure by computing final-layer representations only for seed nodes (PyG Team, 2023). In our two-layer customer–review–product encoder, a customer specialist never trains the product-destination modules, and vice versa (Figure 2d). Dependency tracing and output-preserving pruning show that these modules are inert in each specialist.

Multi-root training activates this capacity. Multi-root training uses the same architecture and simply activates these modules, while the review transformations and table encoders remain shared (Figure 2d). The result is a form of hard parameter sharing (Caruana, 1997; Ruder, 2017) in which the schema, rather than a design choice, determines which computation is shared and which is root-private. Although each private branch holds only 5.4% (Amazon) and 3.9% (HM) of a specialist's trained parameters, it receives 72–83% of that root's squared-gradient energy. This shows where

learning concentrates, not that private modules are necessary or that sharing causes transfer. We view the heterogeneous GNN as a convenient structural basis for consolidation, not an explanation of why it succeeds.

5 Task Structure and Query Conditioned Prediction

5.1 Tasks Share Structure Beyond Entity Activity

For QCRP to predict an unseen task without labeled adaptation, the query must select useful predictive information learned during pretraining. We examine two aspects of this transfer: shared variation across task labels, and whether similarities between query embeddings reflect relationships between those labels.

For each root, we form the entity $\times$ task label matrix over a common anchor sample using its K non-degenerate pretraining tasks, and compute its effective rank, roughly the number of independent directions the labels span. It is far below K on all four roots (Figure 3a): 14.9 of 36 (Amazon product), 16.7 of 35 (Amazon customer), 2.2 of 39 (HM customer), and 15.8 of 63 (HM article), whereas shuffling each task's labels across entities keeps it near K. Because an active entity may score high on every future-window task, we also residualize each task on a per-cutoff fixed effect and the entity's mean across tasks. Three spectra barely change; only HM customer rises, from 2.2 to 4.6, so activity explains part of that root's concentration. The concentration therefore persists after controlling for this activity proxy, consistent with the target redundancy observed in Section 3.2. Appendix D.4 details the sampling, filtering, normalization, and spectral analysis.

Pretraining aligns query geometry with label structure. We measure the Spearman correlation between label correlation r_{ij} and query-embedding cosine c_{ij} for within-root task pairs. The baseline is an untrained encoder rather than zero: the field encoder is additive, so tasks sharing fields start with similar embeddings. Pretraining raises alignment from 0.12 to 0.44 on Rel-Amazon and from 0.59 to 0.69 on Rel-HM, with disjoint ranges across five initializations and three seeds (Figure 3b). The V-shaped relationship shows that the embedding encodes the strength of association more than its sign: anti-correlated tasks sit closer than unrelated ones.

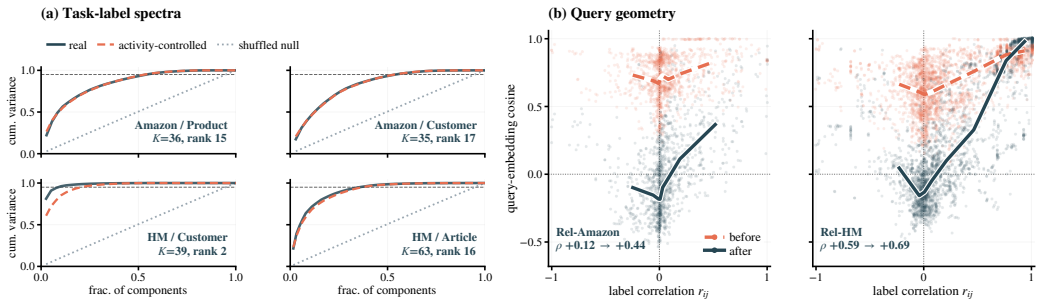

Figure 3: **Task structure and query geometry.** (a) Cumulative explained variance of each root's task-label matrix: real (solid), activity-controlled (dashed), within-task shuffled null (dotted). (b) Query-embedding cosine versus label correlation for within-root task pairs, before (five initializations) and after pretraining (three seeds); lines are binned medians, labels give mean Spearman alignment.

5.2 Zero-Shot Query Transfer

We call prediction *zero-shot* when the task is specified by its query q, and the pretrained model scores each entity as $\hat{y}_{v,t,q} = D_\psi\big(E_\theta(v,t),\, G_\phi(q)\big)$ without additional parameter fitting, target-task labels for adaptation or selection, or in-context examples. This differs from relational foundation models that condition on labeled examples (Fey et al., 2025; Hudovernik et al., 2026; Chen et al., 2026) and from the frozen probes of Section 3, which fit task-specific heads. Because zero-shot scores are uncalibrated, we report Spearman ρ for regression and ROC-AUC for classification. Results

are averaged across tasks, three seeds, and single-root and multi-root checkpoints; Appendix D.5 reports per-task results.

In-domain queries. The six RelBench tasks are field-matched to pretraining queries. Zero-shot classification approaches fitted-probe performance (AUC 72.9 versus 73.7; Table 4). These results show that the pretrained predictor remains useful without a new head, but do not test transfer to unseen query specifications.

Held-out queries. Across the 24 held-out tasks, QCRP reaches $\rho = 0.42$ and AUC 68.4 without adaptation. Replacing query fields with randomly sampled pretraining values reduces performance on most tasks (Table 4), showing that predictions depend on the supplied specification. Corrupted queries nevertheless perform above chance on average, consistent with predictive signals shared across tasks (Section 5.1). A substantial gap remains to fitted probes, particularly for classification, where they reach AUC 81.1. Transfer is weakest on HM customer tasks, whose labels have the most concentrated spectrum.

Table 4: **Zero-shot query transfer.** Task means over seeds and both checkpoint types. Only MLP probes use target-task labels for downstream fitting. AUC is multiplied by 100.

Query supplied	$\rho \uparrow$	AUC↑
In-domain (RelBench)		
Correct	0.48	72.9
Fitted MLP probe (ref.)	0.55	73.7
Held-out (24 tasks)		
Correct	0.42	68.4
Nearest, embedding	0.41	67.1
Nearest, field overlap	0.36	63.2
Corrupted fields	0.19	58.9
Fitted MLP probe (ref.)	0.49	81.1
Chance	0.00	50.0

Predictive similarity between queries. Replacing a held-out query with its nearest pretraining query in embedding space preserves much of the performance (Table 4). Matching by field overlap is less effective: on some tasks, changing a single field, such as the aggregation, reverses the association between targets, and embedding-based selection avoids several such mismatches. Query embeddings therefore capture predictive relationships that field overlap misses, although the limited advantage of the correct query leaves generalization beyond nearby pretraining tasks unresolved.

6 Discussion and Conclusion

QCRP learns better frozen representations than existing SSL methods. We introduced QCRP, a lightweight framework for database-level pretraining of conventional relational encoders. Structured future-prediction tasks provide supervision across entity types, while query conditioning exposes a shared predictive interface. Among standard RelBench targets and held-out task definitions, QCRP learns frozen representations that improve aggregate performance over TVE-based pretraining and remain competitive with task-specific supervised training. Multi-root pretraining largely preserves root-specific performance while using one encoder per database. These findings show conventional relational encoders as a practical starting point for reusable database-level forecasting representations.

Beyond standard forecasting targets. Standard targets can overlap directly with auxiliary objectives or remain recoverable from closely related statistics. Evaluation on held-out definitions therefore complements standard benchmarks, although exclusion alone does not establish statistical independence from pretraining supervision. We encourage benchmarks with richer schema, more prediction roots, and tasks spanning multiple relational paths, filters, contextual attributes, and horizons (Gu et al., 2026).

Toward stronger zero-shot prediction. Our results present a usable specification-only interface, although its limited advantage over nearest-query substitution leaves open whether it generalizes beyond relationships already represented by nearby pretraining tasks. A future direction is to co-design the relational encoder and query-conditioning module so that task semantics guide which relational information is retrieved and combined. Larger and more diverse task pools, such as recent cross-database task corpora (Ranjan et al., 2026b), would also allow systematic study of how task coverage and field interactions affect zero-shot generalization.

AI USE STATEMENT

In this work, we used generative AI tools to support selected parts of the research and writing process, including brainstorming and refining research ideas, provide feedback on experimental design, support implementation and data processing, and help examine and discuss experimental results. We also used generative AI tools to assist with literature discovery, writing and editing, creating or modifying scientific figures, software development, and constructing synthetic pretraining tasks. We used these tools as supporting aids within research processes led by the authors, and the authors independently reviewed, revised, and validated all outputs.

We did not use generative AI tools to generate synthetic datasets, develop theoretical results, formulate mathematical claims, or produce mathematical proofs.

We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To facilitate the reproducibility of the results presented in this work, we have taken several steps detailed throughout the paper and supplementary materials. Complete architectural details, hyperparameter configurations, and training setups are provided in Appendix A. Finally, an implementation of our codebase to replicate our main experiments will be made publicly accessible upon publication.

REFERENCES

Francisco Galuppo Azevedo, Clarissa Lima Loures, and Denis Oliveira Correa. Task scarcity and label leakage in relational transfer learning. In *3rd Workshop on Navigating and Addressing Data Problems for Foundation Models at ICLR*, 2026. doi: 10.48550/arXiv.2603.29914. URL https://openreview.net/forum?id=nI2nsMMHXp.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Payal Chandak, Gregory Kondas, Liat Antwarg Friedman, Isaac Kohane, and Matthew McDermott. Everyquery: Zero-shot clinical prediction via task-conditioned pretraining over electronic health records. *arXiv preprint arXiv:2603.07900*, 2026.

Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite message passing for relational deep learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=XXh3zmw2Uy.

Zhikai Chen, Junyu Yin, Jialiang Gu, Siheng Xiong, Xiaoze Liu, Ruowang Zhang, Keren Zhou, and Kai Guo. Openrfm: Dissecting relational in-context learning. *CoRR*, abs/2606.04320, 2026. doi: 10.48550/ARXIV.2606.04320. URL https://doi.org/10.48550/arXiv.2606.04320.

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

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. In *International Conference on Learning Representations*, volume 2026, pp. 154348–154369, 2026.

Xinyue Feng, Shuxin Zhong, Jinquan Hang, Wenjun Lyu, Yuequn Zhang, Guang Yang, Haotian Wang, Desheng Zhang, and Guang Wang. Hierarchical structure sharing empowers multi-task heterogeneous gnns for customer expansion. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 4424–4434, 2025.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Position: Relational deep learning - graph representation learning on relational databases. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13592–13607. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/fey24a.html.

Matthias Fey, Vid Kocijan, Federico Lopez, J Lenssen, and Jure Leskovec. Kumorfm: A foundation model for in-context learning on relational data. *Technical whitepaper, Kumo AI*, 2025.

Justin Gu, Rishabh Ranjan, Charilaos Kanatsoulis, Haiming Tang, Martin Jurkovic, Valter Hudovernik, Mark Znidar, Pranshu Chaturvedi, Parth Shroff, Fengyu Li, and Jure Leskovec. RelBench v2: A large-scale benchmark and repository for relational data, 2026. URL https://arxiv.org/abs/2602.12606.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

Adrian Hayler, Klemens Flöge, Alan Arazi, Rishabh Ranjan, Jure Leskovec, Felix Birkel, Brendan Roof, Anurag Garg, Kristina Collins, Lydia Sidhoum, et al. Advancing open and reproducible relational learning: Relarena-α, tabpfn-rel and rpi. *arXiv preprint arXiv:2608.16319*, 2026.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 594–604, 2022.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *The Web Conference*, pp. 2704–2710. ACM, 2020. doi: 10.1145/3366423.3380027. URL https://doi.org/10.1145/3366423.3380027.

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

Xunqiang Jiang, Tianrui Jia, Yuan Fang, Chuan Shi, Zhe Lin, and Hui Wang. Pre-training on large-scale heterogeneous graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 756–766, 2021. doi: 10.1145/3447548.3467396.

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

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

Vid Kocijan, Jinu Sunil, Jan Eric Lenssen, Viman Deb, Xinwei Xe, Federico Reyes Gomez, Matthias Fey, and Jure Leskovec. Predictive query language: A domain-specific language for predictive modeling on relational databases. *arXiv preprint arXiv:2602.09572*, 2026.

Oleksii Kolesnichenko, Jakub Peleška, et al. Relational task generation language: A declarative specification framework for relational deep learning. *arXiv preprint arXiv:2609.01292*, 2026.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4IT2pgc9v6.

Shengchao Liu, David Vazquez, Jian Tang, and Pierre-André Noël. Flaky performances when pretraining on relational databases. *arXiv preprint arXiv:2211.05213*, 2022.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 565–576, 2021.

David Mayo, Tyler R Scott, Mengye Ren, Gamaledin Elsayed, Katherine Hermann, Matt Jones, and Michael Mozer. Multitask learning via interleaving: A neural network investigation. In *Proceedings of the annual meeting of the cognitive science society*, volume 45, 2023.

Jaehyun Nam, Jihoon Tack, Kyungmin Lee, Hankook Lee, and Jinwoo Shin. STUNT: Few-shot tabular learning with self-generated tasks from unlabeled tables. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_xlsjehDvlY.

Jakub Peleška. Universal encoders for modular relational deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 73–90. Springer, 2026.

Jakub Peleška and Gustav Šír. Task-agnostic contrastive pretraining for relational deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 623–638. Springer, 2025.

PyG Team. Hierarchical neighborhood sampling. PyTorch Geometric documentation, 2023. https://pytorch-geometric.readthedocs.io/en/latest/advanced/hgam.html.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 69149–69174, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/7107d4d2e837bde2171c6b71b5bde954-Paper-Conference.pdf.

Rishabh Ranjan, Vignesh Kothapalli, Harshvardhan Agarwal, Charilaos I. Kanatsoulis, Roshan Reddy Upendra, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Large-scale pretraining unlocks few-shot prediction for relational data. In *2nd ICML Workshop on Foundation Models for Structured Data*, 2026b. URL https://openreview.net/forum?id=oQINTd9din.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. Relbench: A benchmark for deep learning on relational databases. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 21330–21341. Curran Associates, Inc., 2024. doi: 10.52202/079017-0672. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/25cd345233c65fac1fec0ce61d0f7836-Paper-Datasets_and_Benchmarks_Track.pdf.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International conference on machine learning*, pp. 9120–9132. PMLR, 2020.

Quang Truong, Zhikai Chen, Mingxuan Ju, Tong Zhao, Neil Shah, and Jiliang Tang. A pre-training framework for relational data with information-theoretic principles. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 8122–8153. Curran Associates, Inc., 2025. doi: 10.52202/085713-0277. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/0c0936a505992ce0d45169cd595b9755-Paper-Conference.pdf.

Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1726–1736, 2021. doi: 10.1145/3447548.3467415.

Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a graph-centric relational database foundation model. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=TxeCxVb3cL.

Jinyu Yang, Cheng Yang, Junze Chen, Zedi Liu, Muhan Zhang, Hanyang Peng, and Chuan Shi. Relprism: A multi-faceted pre-training framework with self-generated tasks for relational databases. *arXiv preprint arXiv:2605.23241*, 2026.

Minji Yoon, John Palowitch, Dustin Zelle, Ziniu Hu, Ruslan Salakhutdinov, and Bryan Perozzi. Zero-shot transfer learning within a heterogeneous graph via knowledge transfer networks. *Advances in Neural Information Processing Systems*, 35:27347–27359, 2022.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33: 5812–5823, 2020.

Yanlin Zhang, Linjie Xu, Quan Gan, David Wipf, and Minjie Wang. Rdblearn: Simple in-context prediction over relational databases, 2026. URL https://arxiv.org/abs/2602.18495.

A IMPLEMENTATION DETAILS

A.1 RELATIONAL ENCODERS

Graph construction and row features. Each database row is a node, with its table defining its node type. Foreign keys induce directed relations and their reverse relations. We sample temporal neighborhoods with fanouts $(128, 64)$, using the most recent eligible neighbors and a bidirectional sampled subgraph. Features and neighborhoods are restricted by the anchor's cutoff time; future windows are used only to construct targets. Both backbones operate on this sampled graph, including the four-layer RelGNN. We do not add learned entity-ID embeddings.

Each table has an independent row encoder. Categorical and multi-categorical columns use embedding encoders, numerical columns use standardized scalar projections, precomputed GloVe text features use learned projections, and timestamps use timestamp encoders. Each column produces a 128-dimensional feature. These features are concatenated and processed by four residual fully connected blocks of width 128. Each block contains two linear–LayerNorm–ReLU–dropout stages, with dropout 0.2; a learned shortcut matches the input width when necessary. A final LayerNorm–ReLU–linear projection produces the backbone input width. The GraphSAGE row encoder additionally applies dropout 0.2 after this predictor. For timestamped tables, a sinusoidal encoding of the difference between the anchor cutoff and the row timestamp, measured in days, is passed through a table-specific linear map and added to the row embedding.

Heterogeneous GraphSAGE. Our GraphSAGE backbone (Hamilton et al., 2017) assigns separate weights to each directed relation and layer. For relation $a \to b$ and destination node v, the relation-specific output at layer ℓ is

$$m^{(\ell)}_{a\to b,v} = W^{(\ell)}_{a\to b,\mathrm{nbr}} \,\mathrm{mean}_{u\in\mathcal{N}_{a\to b}(v)}\, h^{(\ell-1)}_u + W^{(\ell)}_{a\to b,\mathrm{self}} h^{(\ell-1)}_v + b^{(\ell)}_{a\to b}. \tag{2}$$

Relation outputs are summed at each destination, followed by a node-type-specific LayerNorm and ReLU. The self term is a learned destination transformation, not an additional identity residual. We use two layers of width 128, without jumping-knowledge connections or message-passing dropout. The root embedding is the final-layer feature of the seed node. Standard specialist training retains the instantiated relation modules, even when some have no path to the selected root loss; pruning is a separate analysis.

RelGNN. We use RelGNN's composite message passing (Chen et al., 2025), with four layers of width 256. Atomic routes are constructed from the foreign-key schema. Direct routes connect dimension tables; composite dimension–fact–dimension routes first aggregate source-dimension features into a fact row with a GraphSAGE operator, then use attention to propagate the enriched fact representation to the destination dimension. The intermediate fact representation also contributes to the fact-table update. Outputs of routes reaching the same node type are summed and passed through node-type-specific LayerNorm and ReLU. Route and layer parameters are distinct.

The composite aggregation uses sum aggregation. Each attention operator has 16 heads with 256 channels *per head*, concatenating to 4,096 channels before a learned projection back to 256. This internal width is substantially larger than the final node embedding. We use the full composite-message-passing implementation, without its simplified-message-passing option or attention dropout. For pretraining, we take the root embedding before the backbone's supervised prediction head and attach the query conditioned predictor described below.

A.2 PRETRAINING AND FROZEN-PROBE SETTINGS

Tables 6 and 7 report the configurations associated with the QCRP checkpoints and six RelBench frozen-probe results in the main comparison. The separate training-dynamics study uses the learning rates and schedules stated in its figures.

A.3 STRUCTURED QUERY ENCODER AND PREDICTION HEADS

Query fields. The clause encoder represents a query through the fields in Table 8; it does not use a learned embedding indexed by task ID. Each categorical field has its own 64-dimensional

Table 5: Backbone architecture. RelGNN's attention width is specified *per head*; the concatenated attention output is projected back to the node width.

Component	GraphSAGE	RelGNN
Row encoder hidden width / residual blocks	128 / 4	128 / 4
Node representation width d	128	256
Message-passing layers	2	4
Neighbor aggregation	Mean	Composite routes / attention
Attention heads $\times$ width per head	—	16×256
Relation/route combination	Sum	Sum
Layer normalization / activation	LayerNorm / ReLU	LayerNorm / ReLU
Query embedding width	128	128
Fusion hidden width / output width	256 / 128	256 / 256

Table 6: QCRP pretraining configurations. Budgets are configured training budgets; root-specific and multi-root settings are distinguished where they differ.

	GraphSAGE		RelGNN
Setting	Amazon	HM	Both databases
Root-specific learning rate	10^{-3}	5×10^{-3}	2×10^{-4}
Multi-root learning rate	1.5×10^{-3}	5×10^{-3}	2×10^{-4}
Multi-root switching interval	5,000 updates		
Optimizer	AdamW		
(β_1, β_2) / weight decay	$(0.9, 0.999)$ / 0.01		
Learning-rate schedule	5% linear warmup, then cosine decay		
Anchors per update	512 (one active root)		
Queries per root batch	16		
Root-specific budget	108 epochs $\times$ 1,000 updates		
Multi-root budget	216 epochs $\times$ 1,000 updates		
Neighbor fanouts	$(128, 64)$		
Temporal sampling / subgraph	Most recent / bidirectional		
Precision	bfloat16		
Regression loss	Huber, $\delta = 1$		
Classification loss	Binary cross-entropy with logits		
Effective-sample reference	256 valid targets		
Seeds	43, 44, 45		

embedding table. The schema path is the masked mean of table-token embeddings, truncated or padded to four tokens. This pooling does not encode token order. Numerical fields use separate $1 \to 64 \to 64$ MLPs with GELU between the linear layers. We apply $\log(1 + x)$ to the horizon value before its MLP. Filter and readout values have explicit presence masks, so a missing value is distinct from a numerical zero. Categorical vocabularies distinguish padding, unknown tokens, and absent fields; an absent categorical field contributes its learned `NONE` embedding.

There are 15 encoded fields, including the pooled path and three numerical fields. Each has a learned scalar weight α_f, initialized to one. With $e_f(q)$ denoting its encoding, the query representation is

$$e_q = G_\phi(q) = \mathrm{MLP}_{64 \to 128 \to 128}\left(\mathrm{LN}_{64}\left[\sum_f \alpha_f e_f(q)\right]\right). \tag{3}$$

Normalization is applied *after* summation. The output MLP contains GELU and dropout 0.1 after its first linear layer, and LayerNorm after its second. The interface can represent new combinations of known query primitives; unseen categorical tokens map to the unknown-token embedding.

Late fusion. For a root embedding $z \in \mathbb{R}^d$, we concatenate z with e_q and apply

$$h_q = \mathrm{Dropout}_{0.1}(\mathrm{GELU}(\mathrm{LN}(\mathrm{Linear}_{d+128 \to 256}([z; e_q])))),$$
$$\widetilde{z}_q = \mathrm{GELU}(\mathrm{Linear}_{256 \to d}(h_q)).$$

Table 7: Frozen entity-representation MLP-probe optimization for the six RelBench tasks, shared by the reported QCRP GraphSAGE and RelGNN runs. Only the fresh probe is updated; the pretrained encoder remains frozen in evaluation mode.

Setting	Value
Optimizer	AdamW
(β_1, β_2) / weight decay	$(0.9, 0.999)$ / 0.01
Regression learning rate (LTV, sales)	2×10^{-3}
Classification learning rate (churn)	5×10^{-3}
Learning-rate schedule / warmup	Constant / none
Epochs / maximum updates per epoch	20 / 1,000
Batch size	512
Neighbor fanouts	$(128, 64)$
Temporal sampling / subgraph	Most recent / bidirectional
Precision	float32
Probe dropout	0.1
Regression loss / target normalization	MAE / none
Classification loss	Binary cross-entropy with logits
Checkpoint metric	Validation MAE / ROC-AUC
Seeds	43, 44, 45

Table 8: Inputs to the clause query encoder. Each row lists fields with the same encoding.

Clause	Fields	Encoding
Scope	`root_table`, `horizon_unit`	Embedding
Path	`schema_path`	Masked mean
Aggregate	`aggregation`, `aggregation_column`	Embedding
Filter	`filter_column`, `filter_operator`, `filter_value_categorical`	Embedding
Context	`context_type`, `context_column`	Embedding
Readout	`output_type`, `operator`	Embedding
Horizon	`horizon_value`	$\log(1+x)$, MLP
Filter value	`filter_value_numeric`	Masked MLP
Readout value	`readout_value`	Masked MLP

Two linear scalar readouts serve regression and binary classification, respectively; they are shared across queries of the corresponding output type. Late fusion reuses the same relational embedding for all queries sampled for a root batch. Joint training requires one relational forward pass for *each* root batch.

Frozen entity-representation probe. For each downstream task, we freeze the relational encoder in evaluation mode and fit a fresh head directly on z. The head is $\mathrm{Linear}_{d\to d}$–LayerNorm–GELU–$\mathrm{Dropout}_{0.1}$– $\mathrm{Linear}_{d\to 1}$. It does not use the query encoder, fusion network, or pretrained readouts. Regression probes minimize MAE on the downstream targets; binary probes minimize binary cross-entropy with logits. Checkpoints are selected by the corresponding downstream validation metric. In contrast, zero-shot query evaluation uses the pretrained query encoder and decoder without fitting a new head.

A.4 Training procedure

Targets and root-batch objective. For each root, the anchor pool contains entity–cutoff pairs, and future-window targets are precomputed for its query pool. Missing or invalid targets are masked. For regression, configured count, distinct-count, and sum targets receive a $\log(1+y)$ transformation; regression targets are then standardized using the query's training-set mean and standard deviation. A constant target uses scale one. The same fitted transformations are used for validation and test; binary targets are unchanged. Regression uses Huber loss with transition parameter one, and binary classification uses cross-entropy with logits.

Each root batch samples up to K queries uniformly without replacement. A query's loss is the mean over its n_q valid anchors. We average these losses with weights $w_q = \min(1, \sqrt{n_q/256})$, omitting queries with no valid targets. This reduces the contribution of sparsely observed queries without weighting every task in proportion to its number of labels. Algorithm 1 defines this computation and the single-root update loop. Sampling denotes drawing from the root's training loader; invalid batches do not advance the optimizer or update counter.

Single-root and multi-root updates. A specialist repeatedly updates on one root. Interleaving uses the same update procedure but changes roots every T updates (Algorithm 2). Model weights, optimizer moments, and the global learning-rate schedule persist across root switches. Joint training computes a separate loss for each root and takes one optimizer step on their mean (Algorithm 3). For the matched two-root comparison, an interleaved update uses 512 anchors from one root, whereas a joint update uses 256 per root. Both use 216,000 optimizer updates; specialists use 108,000. For blocked interleaving with $T = 5{,}000$, the final partial block yields 110,000 and 106,000 updates for the first and second roots, respectively, versus 108,000 per specialist. Per-root exposure is therefore approximately matched, with deviations of $\pm 1.85\%$. We sample up to 16 queries per root batch. Joint training consequently evaluates more queries per update, even though its total anchor count matches interleaving. Optimization uses AdamW with warmup and cosine decay; the learning rate and residence time vary in the training-condition study. Final checkpoint comparisons use offline evaluation over every task's full time-grid validation pool.

A.5 Pretraining runtime

Table 9 reports the observed time spent on optimizer updates for the QCRP runs. A multi-root checkpoint uses twice as many updates as either specialist and, across these settings, takes about 1.0–$1.4\times$ the combined update time of the two specialists. These are observed runs on NVIDIA H100 GPUs, with up to two jobs sharing a GPU, rather than a controlled throughput benchmark; validation and checkpointing time are excluded because their schedules differ across runs.

Table 9: **Measured pretraining update time.** Training time over three seeds, excluding validation and checkpointing. Multi-root runs use twice the specialist update budget; concurrent workloads may affect the measured wall time.

Run	Updates	n	Mean (min)	Range (min)
GraphSAGE				
Rel-Amazon product (specialist)	108,000	3	109	108–111
Rel-Amazon customer (specialist)	108,000	3	118	114–121
Rel-Amazon multi-root	216,000	3	318	278–363
Rel-HM article (specialist)	108,000	3	151	148–156
Rel-HM customer (specialist)	108,000	3	148	147–150
Rel-HM multi-root	216,000	3	328	315–336
RelGNN				
Rel-Amazon product (specialist)	108,000	3	185	165–204
Rel-Amazon customer (specialist)	108,000	3	191	167–213
Rel-Amazon multi-root	216,000	3	422	409–446
Rel-HM article (specialist)	108,000	3	270	262–275
Rel-HM customer (specialist)	108,000	3	243	199–273
Rel-HM multi-root	216,000	3	530	497–570

Algorithm 1 Multi-task pretraining for one root

Require: Anchor pools $\mathcal{A}_r$, query pools $\mathcal{Q}_r$, cached targets and masks
Encoder E_θ, query encoder G_ϕ, predictor D_ψ; $\Theta = (\theta, \phi, \psi)$
Fitted target transforms τ_q, losses ℓ_q, queries per batch K, schedule η_u

function RootLoss(r, b, Θ)
  Draw b entity–cutoff anchors $\mathcal{B}$ from $\mathcal{A}_r$
  Sample $\min(K, |\mathcal{Q}_r|)$ queries $\mathcal{S}$ uniformly without replacement
  $Z \leftarrow E_\theta(\{(\mathcal{H}_{\le t}, v) : (v, t) \in \mathcal{B}\})$ ▷ One encoder pass
  $a \leftarrow 0$; $w \leftarrow 0$
  for each $q \in \mathcal{S}$ **do**
    Gather cached targets Y_q and validity mask M_q for $\mathcal{B}$; $n_q \leftarrow \sum_i M_{q,i}$
    if $n_q > 0$ **then**
      $\widehat{Y}_q \leftarrow D_\psi(Z, G_\phi(q))$; $Y_q \leftarrow \tau_q(Y_q)$
      $\bar{\ell}_q \leftarrow n_q^{-1} \sum_{i:M_{q,i}=1} \ell_q(\widehat{Y}_{q,i}, Y_{q,i})$
      $w_q \leftarrow \min(1, \sqrt{n_q/256})$; $a \leftarrow a + w_q \bar{\ell}_q$; $w \leftarrow w + w_q$
    end if
  end for
  if $w = 0$ **then**
    return $\varnothing$
  end if
  return a/w
end function
procedure MultiTaskPretrain(r, L, Θ, s, u)
  $j \leftarrow 0$
  while $j < L$ **do**
    $\mathcal{L} \leftarrow$ RootLoss(r, B, Θ)
    if $\mathcal{L} \neq \varnothing$ **then**
      $(\Theta, s) \leftarrow$ OptimizerStep($\Theta, s, \nabla_\Theta \mathcal{L}, \eta_u$)
      $u \leftarrow u + 1$; $j \leftarrow j + 1$
    end if
  end while
  return (Θ, s, u)
end procedure

Algorithm 2 Multi-root pretraining by interleaving

Require: Ordered roots $(r_0, \ldots, r_{R-1})$, update budget U, residence time T, batch size B
Initialize model Θ, optimizer state s, and global step $u \leftarrow 0$
while $u < U$ **do**
  $r \leftarrow r_{\lfloor u/T \rfloor \bmod R}$; $L \leftarrow \min(T, U - u)$
  $(\Theta, s, u) \leftarrow$ MultiTaskPretrain(r, L, Θ, s, u)
end while
return Θ

Algorithm 3 Multi-root pretraining by joint updates (mean reduction)

Require: Roots $(r_0, \ldots, r_{R-1})$, update budget U, total batch size B divisible by R
Initialize model Θ, optimizer state s, and global step $u \leftarrow 0$
while $u < U$ **do**
  for $r \in (r_0, \ldots, r_{R-1})$ **do**
    $\mathcal{L}_r \leftarrow$ RootLoss($r, B/R, \Theta$)
  end for
  $\mathcal{V} \leftarrow \{r : \mathcal{L}_r \neq \varnothing\}$
  if $|\mathcal{V}| > 0$ **then**
    $\mathcal{L} \leftarrow |\mathcal{V}|^{-1} \sum_{r \in \mathcal{V}} \mathcal{L}_r$
    $(\Theta, s) \leftarrow$ OptimizerStep($\Theta, s, \nabla_\Theta \mathcal{L}, \eta_u$)
    $u \leftarrow u + 1$
  end if
end while
return Θ

B HELD-OUT EVALUATION QUERIES

The six RelBench tasks cover only a small subset of the prediction functions that a relational encoder may need to support. We therefore evaluate an additional suite of 24 held-out query-defined tasks on Rel-Amazon and Rel-HM. The purpose is to test whether pretrained entity representations remain useful for targets beyond the pretraining objectives, rather than only for the standard downstream benchmarks.

Task selection and review. An AI agent selected the candidate tasks before seeing any model results. The tasks were subsequently reviewed to exclude degenerate targets: binary classification tasks were required to have positive-class prevalence in $[0.05, 0.95]$, and regression tasks were required to have strictly positive target variance. We also checked that none of the selected tasks appeared in any pretraining task pool used in our experiments. This exclusion concerns the tasks themselves; related query primitives and targets may still occur during pretraining. Table 10 documents the final task specifications.

Suite construction. Each database contributes 12 tasks: six regression and six binary classification tasks. The suite is balanced across its two roots, with three regression and three classification tasks per root. Rel-Amazon uses customer and product roots; Rel-HM uses customer and article roots. Table 10 specifies each target and its relational path. These tasks combine distinct-value counts, dispersion statistics, extrema, filtered activity, and thresholded aggregates. They include direct event aggregates, attributes of related dimension rows, and cohort aggregates over events sharing a customer or an item. For example, a customer target may count distinct prices of reviewed products, whereas an article target may predict the age of its oldest active buyer. Thus, the suite varies both the information that must be retrieved and the function applied to it.

What the evaluation measures. For frozen-representation evaluation, a fresh task-specific MLP is trained on the held-out task's training labels using only the entity embedding z. The relational encoder remains frozen, and neither the query encoder nor the pretrained predictor is used. "Held out" therefore refers to exclusion from pretraining supervision, not to an absence of labeled adaptation examples. This protocol measures representation reuse for new targets. It does not, by itself, establish zero-shot execution of unseen queries or systematic compositional generalization: related pretraining targets can still provide useful predictive information. Zero-shot evaluation instead retains the query encoder and predictor and fits no new prediction head; we report it separately.

Table 10: **The 24 held-out evaluation tasks.** Amazon uses a 91-day horizon; HM uses a 7-day horizon. The task `customer_price_point_variety` is distinct in Rel-Amazon (A-R1) and Rel-HM (H-R6). Specifications give concise descriptions of the future prediction targets.

ID	Task	Root	Path	Specification
Rel-Amazon — Regression				
A-R1	customer_price_point_variety	customer	customer→review→product	Number of distinct product prices
A-R2	customer_rating_level_variety	customer	customer→review	Number of distinct rating levels
A-R3	customer_verified_rating_dispersion	customer	customer→review	Std. deviation of verified ratings
A-R4	product_price_weighted_verified_activity	product	product→review	Price × number of verified reviews
A-R5	product_rating_level_variety	product	product→review	Number of distinct rating levels
A-R6	product_verified_cohort_volume	product	product→review→review_by_customer	Number of verified cohort reviews
Rel-Amazon — Classification				
A-C1	customer_dimension_price_dispersion_over_ten	customer	customer→review→product	Product-price std. deviation > 10
A-C2	customer_same_customer_min_verified_above_three	customer	customer→review→review_by_customer	Minimum verified cohort rating > 3
A-C3	customer_same_product_rating_diversity_verified	customer	customer→review→review_by_product	More than one verified rating level on reviewed products
A-C4	product_same_customer_price_weighted_activity_above_median	product	product→review→review_by_customer	Price-weighted cohort activity > 194.8
A-C5	product_same_product_mean_verified_above_four_and_half	product	product→review→review_by_product	Mean verified rating on the product > 4.5
A-C6	product_same_product_rating_diversity_verified	product	product→review→review_by_product	More than one verified rating level on the product
Rel-HM — Regression				
H-R1	article_buyer_age_variety	article	article→transactions→customer	Number of distinct buyer ages
H-R2	article_cohort_avg_price	article	article→transactions→transactions_by_article	Mean transaction price for the article
H-R3	article_oldest_active_buyer	article	article→transactions→customer	Age of the oldest active buyer
H-R4	customer_instore_avg_price	customer	customer→transactions	Mean in-store purchase price
H-R5	customer_online_spend	customer	customer→transactions	Total online spending
H-R6	customer_price_point_variety	customer	customer→transactions	Number of distinct purchase prices
Rel-HM — Classification				
H-C1	article_active_member_demand_over	article	article→transactions→customer	More than 5 purchases by active members
H-C2	article_cobuyer_basket_value_over	article	article→transactions→transactions_by_customer	Co-buyers' total basket value > 2.5
H-C3	article_premium_sale_over	article	article→transactions	Maximum sale price > 0.0381186
H-C4	customer_buys_trending_articles	customer	customer→transactions→transactions_by_article	Transactions on purchased articles > 1470
H-C5	customer_no_instore_purchase	customer	customer→transactions	No in-store purchases
H-C6	customer_product_group_breadth_over	customer	customer→transactions→article	Purchases span more than one product group

C ADDITIONAL MULTIROOT PRETRAINING EXPERIMENTS

We provide four supplementary views of multiroot training: full-pool validation, comparison with specialists, blocked-interleaving dynamics, and gradient energy.

Evaluation protocol. Validation uses the complete time-grid pool for each task, with RelBench-aligned cutoff dates and training-fitted target transformations. We average task losses within each root: Amazon has 40 customer and 38 product tasks; HM has 65 customer and 78 article tasks. Detailed multiroot runs use 216 epochs of 1,000 updates, with validation every epoch. Joint updates average losses from 256 examples per root; interleaved updates use 512 examples from one root. All trajectories use $\beta_1 = 0.9$ and one seed per configuration: Amazon uses seed 43 except $T = 1$ (seed 44), while HM uses seed 44 except $T = 5000$ (seed 43).

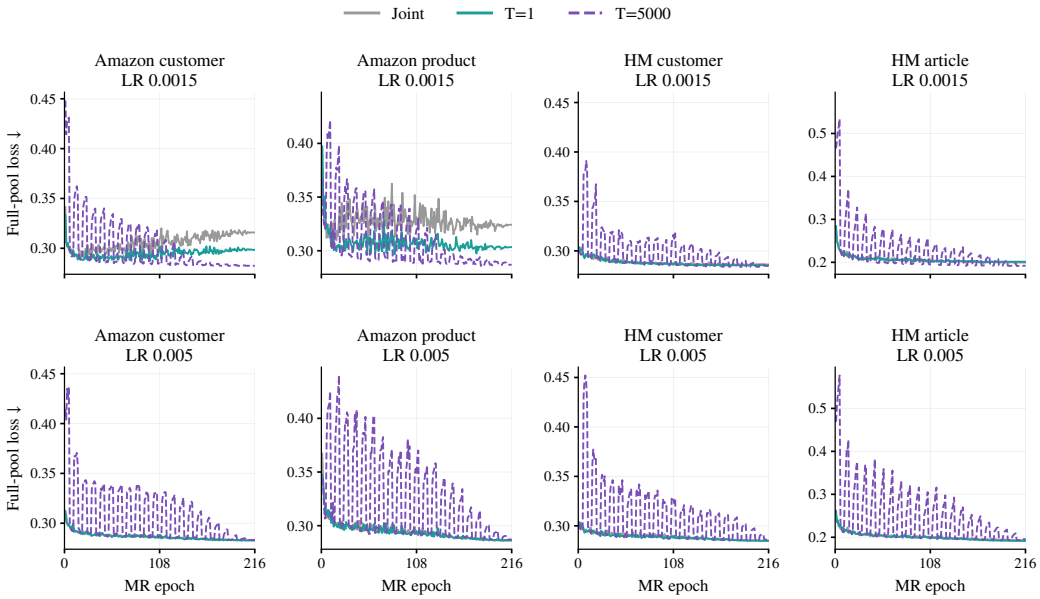

Figure 4: **Full-pool validation dynamics.** Columns separate database and root; rows separate learning rate. Unsmoothed curves retain inactive-root excursions, and vertical scales match across learning rates within each root. The horizontal axis counts all multiroot updates, including periods when the evaluated root is inactive.

Encoder storage. A specialist and a multiroot model instantiate the same schema-level encoder; standard training does not prune inactive modules. One multiroot encoder therefore replaces K equally sized specialist encoders, reducing encoder parameter storage by a factor of K ($K = 2$, or 50% less storage, here). This accounting excludes prediction heads, optimizer states, and file-format overhead.

Gradient measurement. The final-checkpoint audit uses eight cached validation batches of 64 examples per root, all root tasks, FP32, and no optimizer updates. Amazon uses LR 0.0015, $T = 5000$; HM uses LR 0.005, $T = 1$ (both seed 43). For a block B, per-parameter energy is $E_B = (8|B|)^{-1} \sum_{b=1}^{8} \|g_{B,b}\|_2^2$. Figure 7 compares private parameter and energy shares on the same scope and includes a within-layer comparison; these are local sensitivity measurements, not causal importance scores.

The within-layer comparison tests whether private-gradient concentration is explained solely by private modules being in the final layer, closer to the output. Layer 1 contains both private and shared modules; layer 2 has no shared modules on the paths to the evaluated root losses, so it admits no corresponding comparison. Private modules also receive greater energy per parameter within layer 1, ruling out layer index alone as the explanation. However, their computational roles and

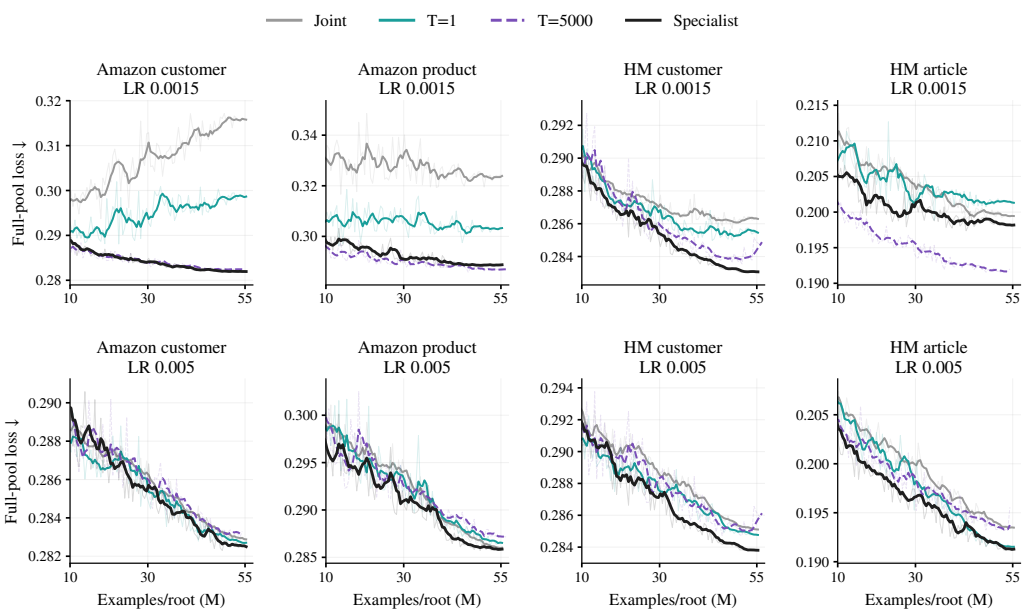

Figure 5: **Comparison with specialists at matched root exposure.** Each panel fixes the root and learning rate; specialists use seed 43 on Amazon and seed 44 on HM. One root-equivalent epoch comprises 512,000 examples: joint curves average adjacent epochs, $T = 1$ retains even epochs, and $T = 5000$ retains active-root epochs with exposure computed from the actual root order. Solid black denotes specialists and dashed purple denotes $T = 5000$. Prominent curves show trailing means over five root-equivalent epochs, with faint unsmoothed traces underneath. Curves end at their last available validation; the late-training view starts at ten million examples per root.

active paths still differ, and this observation does not establish that private branches cause successful consolidation.

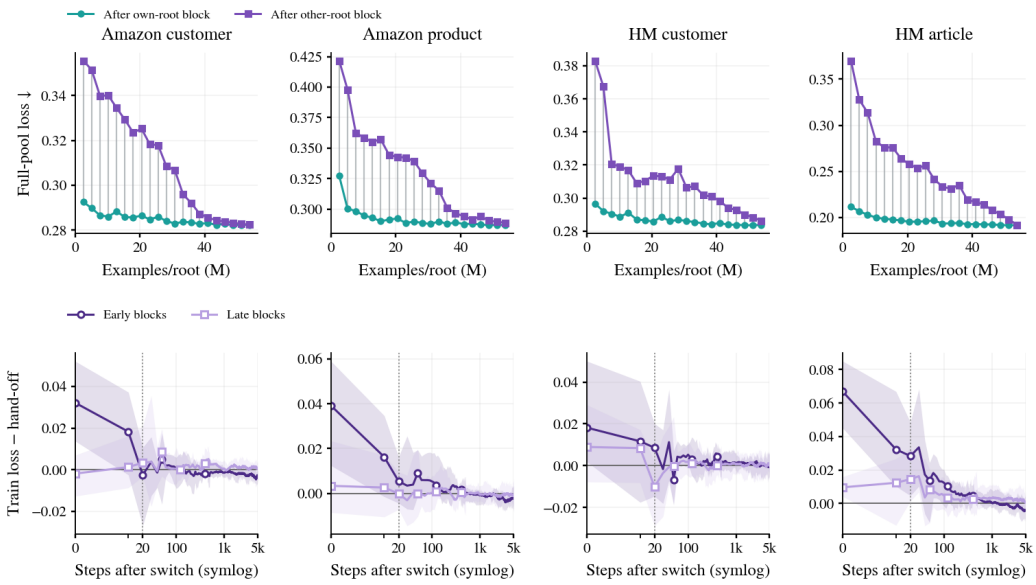

Figure 6: **Two views of blocked interleaving on Amazon and HM** ($T = 5000$, LR 0.0015, seed 43). *Top:* full-pool validation after an own-root block and the following other-root block, paired at identical own-root exposure. *Bottom:* excess sampled-task training loss after resuming a root, relative to the preceding own-root block's final ten logged windows; zero marks recovery of this training-loss reference. These finer logs come from separate runs with matching configurations. Early/late curves use the first/last thirds of complete resumed blocks; bands show across-block ranges, not seed uncertainty. The first point averages updates 0–9, and the point at 20 averages updates 20–29. The first six points are unsmoothed; subsequent points use centered nine-window means, widened to 51 windows after index 50.

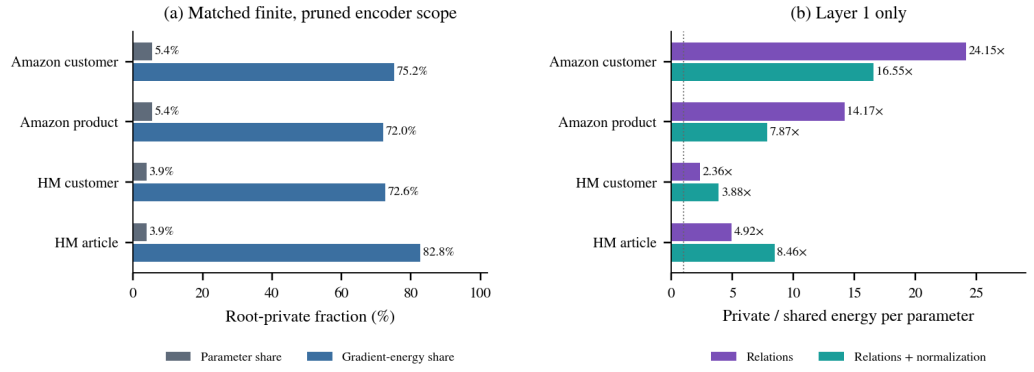

Figure 7: **Gradient energy relative to parameter share.** Left: private parameter and squared-gradient energy fractions within the same finite, structurally active encoder scope, including table encoders (excluding the HM customer table encoder). Right: private/shared energy per parameter within layer 1, for relation weights alone and with normalization; one denotes equal energy per parameter. The within-layer comparison controls for layer index, but not all differences in computation paths.

D ADDITIONAL RESULTS

D.1 FROZEN-PROBE COMPARISON WITH PUBLISHED BASELINES

Table 11 places the strongest multi-root configuration, QCRP with a RelGNN backbone, alongside published results on six RelBench tasks. It achieves the second-best average rank (2.33), behind RT-PluRel (2.00), and the lowest Amazon user-LTV MAE (13.83). This is competitive frozen-encoder transfer from one pretrained checkpoint per database. The comparison spans different adaptation protocols; in particular, several baselines train or fine-tune a task-specific model.

Table 11: **Published baselines vs. multi-root QCRP.** LTV/sales: MAE ↓; churn: AUC ↑. Top: pretrained foundation models; middle: conventional RDL models. Bold marks the best value in each column.

	Rel-Amazon				Rel-HM		
Method	u-LTV	i-LTV	u-churn	i-churn	i-sales	u-churn	Rank↓
Griffin (Wang et al., 2025)	35.59	53.21	62.30	69.00	.151	60.20	10.17
RDBLearn (Zhang et al., 2026)	14.54	48.56	67.57	82.07	.064	68.05	7.58
KumoRFM-2 (Hudovernik et al., 2026)	13.92	46.99	69.10	82.17	**.034**	69.27	4.58
RT-PluRel (Ranjan et al., 2026a)	13.94	**43.03**	**71.35**	**83.27**	.040	70.44	**2.00**
TabPFN-Rel (Hayler et al., 2026)	14.36	46.77	70.86	82.80	.061	70.52	4.50
LightGBM (Ke et al., 2017)	16.78	60.57	52.22	62.54	.076	55.21	10.67
HGT (Hu et al., 2020)	15.41	55.87	66.43	77.97	.064	66.95	9.08
GraphSAGE (Hamilton et al., 2017)	14.31	50.05	70.42	82.81	.056	69.88	5.50
RelGT (Dwivedi et al., 2026)	14.27	48.92	70.39	82.55	.054	69.27	5.83
RelGNN† (Chen et al., 2025)	14.23	48.77	70.99	82.64	.054	**70.93**	3.75
$\text{QCRP}_{\text{multi}}$ (RelGNN)	**13.83**	46.10	70.87	82.90	.036	69.96	2.33

Adaptation protocols differ: $\text{QCRP}_{\text{multi}}$ uses one pretrained RelGNN checkpoint per database with lightweight MLP probes; conventional RDL baselines are trained end-to-end per task; RT-PluRel performs task-specific fine-tuning and context selection; RDBLearn and TabPFN-Rel construct aggregate features before tabular in-context prediction. †Published RelGNN uses task-specific architectures and settings.

D.2 FULL FINETUNING

Table 12 reports end-to-end adaptation with GraphSAGE. For each pretrained model, we retain the relational encoder, discard the pretraining-specific modules, and jointly train the encoder and a fresh linear prediction head on the downstream task. The *Random* row denotes random initialization followed by supervised training from scratch under this adaptation protocol; it is not a frozen random-encoder probe. Learning rates are selected using validation performance, and results are reported as means and standard deviations over three seeds.

Finetuning narrows the differences between pretraining methods. Allowing the encoder to adapt reduces the dependence on features that are immediately accessible to a frozen probe. For example, on Rel-Amazon item-LTV, finetuned MaskAE, CTR, and root-specific QCRP achieve MAEs of 46.833, 46.338, and 46.228, respectively. Generic self-supervised objectives also improve over supervised training from scratch on several tasks: both MaskAE and CTR improve item-LTV and Rel-HM user-churn, while CTR additionally improves Rel-Amazon user-LTV. These gains are consistent with pretraining providing a useful initialization even when its frozen representations are less effective. They are not universal: MaskAE degrades Rel-Amazon user-LTV, and neither generic objective improves Rel-HM item-sales over random initialization.

Task-aware pretraining remains competitive after adaptation. The best reported mean on each of the six tasks belongs to a task-aware method: root-specific QCRP leads the two Amazon LTV tasks, multi-root QCRP leads Amazon item-churn, and TVE+CTR leads Amazon user-churn and both HM tasks. Thus, finetuning reduces the gaps without removing the advantage of task-aware initialization as a group. It does not establish a uniform QCRP advantage: rankings vary across

tasks, and several differences are small relative to seed variation. In particular, the TVE+CTR and root-specific QCRP item-sales means both round to 0.038. Bold entries indicate mean rankings, not statistical significance.

Table 12: **Full finetuning with GraphSAGE.** Mean ± SD over three seeds. LTV/sales: MAE; churn: AUC ×100. Each method transfers only its relational encoder, discards pretraining-specific modules, and finetunes the encoder with a fresh linear head. Bold marks the best reported mean in each column.

	Rel-Amazon				Rel-HM	
Method	user-LTV↓	item-LTV↓	user-churn↑	item-churn↑	item-sales↓	user-churn↑
Random	14.501 ±0.034	47.403 ±0.083	69.338 ±0.234	81.642 ±0.061	0.039 ±0.000	68.984 ±0.176
MaskAE	15.537 ±1.110	46.833 ±0.107	68.820 ±0.232	81.621 ±0.142	0.040 ±0.001	69.764 ±0.090
CTR	14.418 ±0.051	46.338 ±0.686	69.347 ±0.051	81.494 ±0.026	0.041 ±0.001	69.486 ±0.109
TVE	14.419 ±0.064	46.382 ±0.515	69.471 ±0.170	81.799 ±0.064	0.042 ±0.001	69.531 ±0.127
TVE+CTR	14.353 ±0.017	47.098 ±0.425	**69.801** ±0.084	81.769 ±0.023	**0.038** ±0.001	**69.837** ±0.018
$\text{QCRP}_{\text{root}}$	**14.333** ±0.004	**46.228** ±0.544	69.741 ±0.127	81.636 ±0.261	0.038 ±0.001	69.425 ±0.015
$\text{QCRP}_{\text{multi}}$	14.511 ±0.092	46.339 ±0.355	69.648 ±0.154	**81.909** ±0.032	0.040 ±0.001	69.809 ±0.231

$\text{QCRP}_{\text{root}}$: separate root-specific checkpoints; $\text{QCRP}_{\text{multi}}$: one checkpoint per database, with matched per-root exposure. MaskAE: masked reconstruction; CTR: contrastive pretraining alone. TVE uses two-hop statistics; TVE+CTR combines TVE with the contrastive learning objective ($\mathcal{L} = 0.9\mathcal{L}_{\text{TVE}} + 0.1\mathcal{L}_{\text{CTR}}$). One learning rate per method and task is selected by mean validation performance. The learning-rate grid is $\{5\times10^{-4}, 10^{-3}, 2\times10^{-3}, 5\times10^{-3}, 10^{-2}\}$ for Rel-Amazon and the same grid without 10^{-2} for Rel-HM. SDs are rounded; bold does not imply statistical significance.

D.3 Frozen-probe transfer to held-out tasks

Table 13 reports frozen entity-representation MLP probes on 24 tasks excluded from every pretraining pool, compared with supervised training from scratch. Root-specific and multi-root QCRP exceed the supervised reference on 10/12 and 9/12 regression tasks, respectively. Classification is more mixed: both variants win on 4/12 tasks, with mean AUCs close to supervised training (81.20 and 81.06 versus 81.31). Both QCRP variants outperform both TVE baselines on all 12 Amazon tasks, while HM results vary by root and target. Overall, multiroot pretraining retains much of the specialist's held-out performance while allowing adaptation through a small frozen-encoder probe. Rankings and win counts describe reported means, rather than establishing significance across correlated tasks.

D.4 Task-Label Spectral Analysis

This section describes the task-label spectra in Figure 3a.

Sampling and matrix construction. For each root, we sample 200,000 entity–cutoff pairs uniformly without replacement from its full anchor pool, without stratifying by entity or cutoff. The pools contain 6,749,915 and 17,642,031 pairs for Rel-Amazon products and customers, and 35,309,177 and 2,647,097 pairs for Rel-HM customers and articles, respectively. We construct a matrix with one row per sampled pair (v, t) and one column per pretraining query. We first exclude queries with labels defined on fewer than 25% of sampled pairs, then retain only rows with observed labels for every remaining query. No missing labels are imputed. After removing columns with zero variance on these complete-case rows, we center and scale each column to unit variance, obtaining $Y \in \mathbb{R}^{n\times K}$. Thus, the spectrum reflects relationships among standardized targets rather than differences in their units or scales.

Retained queries and evaluation population. Coverage filtering removes 1, 4, 26, and 12 queries from Rel-Amazon product, Rel-Amazon customer, Rel-HM customer, and Rel-HM article pools, respectively. A further 1, 1, 0, and 3 queries have zero variance on the complete-case rows. The resulting matrices contain $K = 36$, 35, 39, and 63 queries, compared with 38, 40, 65, and 78 in the corresponding pretraining pools. Missingness often arises from aggregates that are undefined without sufficient future events. Complete-case selection therefore restricts the population toward active entity–cutoff pairs and can make activity indicators constant. These spectra characterize the retained queries and complete-case population, rather than the full pretraining pool or all entities.

Table 13: **Held-out evaluation across five methods.** IDs correspond to Table 10. Test-split R^2 (regression) and ROC-AUC×100 (classification), averaged over three seeds; higher is better throughout. All 24 tasks are absent from every method's pretraining pool. Pretrained methods use frozen-encoder MLP probes; Supervised trains the whole encoder from random initialization under a linear head. Bold marks the best method per task among those run.

Rel-Amazon	Regression (R^2)						Classification (AUC×100)					
	A-R1	A-R2	A-R3	A-R4	A-R5	A-R6	A-C1	A-C2	A-C3	A-C4	A-C5	A-C6
TVE-2	.242	.150	.092	.063	.421	.184	63.91	91.80	71.63	72.92	91.54	94.78
TVE-2+CTR	.274	.152	.089	.064	.417	.183	62.46	91.18	69.68	72.27	89.67	94.16
QCRP$_{\text{root}}$	**.283**	.162	**.097**	**.142**	**.441**	.236	64.62	93.47	**71.78**	**77.49**	96.35	97.25
QCRP$_{\text{multi}}$	.282	**.163**	.096	.140	.435	**.241**	64.73	93.64	71.66	77.48	96.29	**97.27**
Supervised	.219	.134	.093	.074	.377	.216	**65.06**	**93.91**	71.27	75.74	**97.00**	97.26
Rel-HM	Regression (R^2)						Classification (AUC×100)					
	H-R1	H-R2	H-R3	H-R4	H-R5	H-R6	H-C1	H-C2	H-C3	H-C4	H-C5	H-C6
TVE-2	.705	.908	.346	.010	**.147**	.052	93.45	82.70	97.80	**57.33**	81.61	**59.64**
TVE-2+CTR	.769	.901	.357	.004	**.147**	**.053**	93.71	83.05	97.19	57.31	81.82	59.27
QCRP$_{\text{root}}$	**.798**	.957	**.399**	.024	.146	.038	**94.23**	**84.01**	99.48	56.25	81.34	58.15
QCRP$_{\text{multi}}$	.780	.951	.393	.025	.143	.034	93.89	83.56	99.43	55.95	81.73	57.12
Supervised	.768	**.973**	.389	**.027**	.144	.025	94.20	83.38	**99.65**	57.29	**81.99**	59.00

Summary	Mean score		Mean rank		Wins vs. Supervised				Trainable
	R^2	AUC	Reg	Cls	Reg	p	Cls	p	params
TVE-2	.277	79.93	3.75	3.58	5/12	.774	3/12	.146	16.9K
TVE-2+CTR	.284	79.31	3.67	4.08	6/12	1.000	2/12	**.039**	16.9K
QCRP$_{\text{root}}$	**.310**	81.20	**1.75**	2.50	**10/12**	**.039**	**4/12**	.388	16.9K
QCRP$_{\text{multi}}$	.307	81.06	2.33	2.75	9/12	.146	**4/12**	.388	16.9K
Supervised	.287	**81.31**	3.50	**2.08**	–	–	–	–	1.36/2.25M

Ranks are computed within each task, with ties averaged. Reported p-values are two-sided exact sign tests against supervised training.

Spectrum and effective rank. Let $\sigma_1 \geq \cdots \geq \sigma_K \geq 0$ denote the singular values of Y. We define the variance shares and effective rank as

$$p_i = \frac{\sigma_i^2}{\sum_{j=1}^{K} \sigma_j^2}, \qquad r_{\text{eff}}(Y) = \exp\left(-\sum_{i=1}^{K} p_i \log p_i\right),$$

with $0 \log 0 = 0$. The cumulative-variance curve plots $\sum_{i=1}^{m} p_i$ against m/K. Our effective-rank definition uses squared singular values, consistent with these variance shares.

Control for overall task score and cutoff. Some task correlations may reflect differences in overall entity activity or shared temporal variation. To examine this possibility, we compute each entity–cutoff pair's mean standardized label across tasks. For each task, we then fit a linear regression using this mean score and cutoff indicators, and retain the residuals. The dashed curves show the spectra of these residuals after rescaling each column to unit variance. This adjustment removes variation linearly explained by the mean task score and cutoff-specific means. Because the mean score is constructed from task labels and is only an approximate proxy for activity, the remaining structure should not be interpreted as independent of activity.

Shuffled reference. We independently permute each column across rows and average the resulting spectral statistics over six random seeds. For the activity-controlled analysis, permutations are applied to the standardized residual matrix. Permutations are not stratified by cutoff, so the unadjusted reference disrupts both cross-task correspondence and shared calendar variation. Residualization removes cutoff-specific means before constructing the adjusted reference, but does not eliminate all possible temporal dependence.

D.5 DETAILED ZERO-SHOT QUERY EVALUATION

Table 14 reports every evaluated query for single-root and multi-root checkpoints. The pretrained query encoder and output head score each supplied query directly, without fitting a new predictor. Regression is evaluated with Spearman correlation, which measures ranking rather than calibration in the original target units; classification uses ROC-AUC multiplied by 100. The supervised column is a label-trained reference and is not a zero-shot method. Results average seeds 43–45 and report sample standard deviations.

Familiar queries and held-out specifications. The six standard RelBench queries are field-matched to pretraining objectives, so these rows evaluate reuse of the learned query interface rather than transfer to unseen task specifications. The other 24 queries are excluded from every pretraining pool; their definitions appear in Table 10. The *correct-query* condition supplies each task's own specification. The *shuffled-fields* control independently replaces encoded fields with values drawn from pretraining tasks, preserving familiar field values while disrupting their combination.

Correct queries outperform shuffled fields on 11 of 12 held-out regression tasks for both checkpoint types. This supports sensitivity to the supplied query, but the amount of useful transfer varies substantially. For example, H-R2 reaches correlations of 0.931 and 0.933 for single-root and multi-root checkpoints, whereas H-R6 reaches only 0.020 and 0.010 and does not beat the shuffled control. Classification is less consistent: A-C2 reaches about 89 AUC, but H-C4–H-C6 remain near or below chance in several settings. Query conditioning therefore provides a useful interface for some targets without reliably implementing every held-out specification.

Readout polarity is a limitation of the zero-shot interface. Some large classification standard deviations reflect inconsistent score orientation, not merely small changes in predictive accuracy. A fixed checkpoint can assign high scores to the opposite of a query's positive event; the direction can differ across queries and independently trained checkpoints. Such inversion is systematic for the affected readout, rather than stochastic variation from repeatedly evaluating the same checkpoint. For binary labels, negating a score changes its AUC from a to $1 - a$. Consequently, averaging runs with opposite polarity can obscure a strong but incorrectly oriented ranking. The particularly large standard deviations for A-C5 multi-root (64.30 ± 42.28) and H-C1 single-root (75.26 ± 31.93) should be read with this failure mode in mind, rather than as reliable evidence for a difference between single-root and multi-root training.

Table 14: **Zero-shot query transfer, every task.** Means over seeds 43–45 with sample SD. Held-out task identifiers are defined in Table 10.

	Correct query		Shuffled fields		Supervised
Task	single-root	multi-root	single-root	multi-root	*(ref.)*
Regression (Spearman ρ)					
In-domain (RelBench)					
Rel-Amazon item-LTV	0.608 ± 0.002	0.602 ± 0.001	—	—	—
Rel-Amazon user-LTV	0.369 ± 0.002	0.366 ± 0.003	—	—	—
Rel-HM item-sales	0.468 ± 0.010	0.448 ± 0.029	—	—	—
Held-out					
A-R1	0.257 ± 0.009	0.258 ± 0.003	0.124 ± 0.023	0.137 ± 0.079	0.224 ± 0.061
A-R2	0.198 ± 0.005	0.198 ± 0.003	0.062 ± 0.071	0.061 ± 0.051	0.246 ± 0.003
A-R3	0.323 ± 0.010	0.321 ± 0.015	-0.069 ± 0.057	-0.018 ± 0.126	0.323 ± 0.000
A-R4	0.502 ± 0.003	0.495 ± 0.003	0.266 ± 0.067	0.330 ± 0.044	0.552 ± 0.007
A-R5	0.547 ± 0.005	0.552 ± 0.001	0.055 ± 0.252	0.050 ± 0.082	0.517 ± 0.017
A-R6	0.495 ± 0.004	0.508 ± 0.001	0.096 ± 0.028	0.379 ± 0.101	0.451 ± 0.013
H-R1	0.810 ± 0.007	0.813 ± 0.005	0.420 ± 0.065	0.577 ± 0.143	0.798 ± 0.014
H-R2	0.931 ± 0.003	0.933 ± 0.002	0.415 ± 0.134	0.387 ± 0.229	0.980 ± 0.001
H-R3	0.617 ± 0.001	0.612 ± 0.006	0.379 ± 0.116	0.477 ± 0.052	0.612 ± 0.003
H-R4	0.151 ± 0.002	0.123 ± 0.034	0.039 ± 0.015	0.053 ± 0.021	0.181 ± 0.002
H-R5	0.129 ± 0.006	0.192 ± 0.058	0.121 ± 0.030	0.132 ± 0.061	0.507 ± 0.014
H-R6	0.020 ± 0.001	0.010 ± 0.003	0.036 ± 0.010	0.033 ± 0.003	0.174 ± 0.011
Classification (AUC$\times$100)					
In-domain (RelBench)					
Rel-Amazon item-churn	81.74 ± 0.44	81.95 ± 0.19	—	—	81.64
Rel-Amazon user-churn	69.83 ± 0.23	69.69 ± 0.14	—	—	69.34
Rel-HM user-churn	66.24 ± 1.41	67.78 ± 0.16	—	—	68.98
Held-out					
A-C1	62.96 ± 0.78	62.51 ± 0.52	48.30 ± 2.07	47.14 ± 2.75	65.06 ± 0.03
A-C2	88.63 ± 0.76	88.72 ± 0.51	65.89 ± 4.56	53.55 ± 3.21	93.91 ± 0.05
A-C3	56.42 ± 13.52	54.53 ± 5.32	50.21 ± 1.73	50.53 ± 1.04	71.27 ± 0.58
A-C4	71.40 ± 0.54	69.87 ± 1.77	61.23 ± 7.87	64.57 ± 5.01	75.74 ± 1.66
A-C5	86.40 ± 3.72	64.30 ± 42.28	52.48 ± 2.57	56.65 ± 8.66	97.00 ± 0.09
A-C6	64.23 ± 4.19	62.53 ± 12.31	55.53 ± 3.43	61.09 ± 14.91	97.26 ± 0.04
H-C1	75.26 ± 31.93	92.46 ± 0.65	78.69 ± 2.72	76.43 ± 0.16	94.20 ± 0.14
H-C2	80.55 ± 6.77	81.75 ± 3.36	69.20 ± 8.27	72.41 ± 5.64	83.38 ± 0.40
H-C3	91.96 ± 7.96	91.45 ± 5.45	67.76 ± 1.64	68.20 ± 4.67	99.65 ± 0.01
H-C4	47.67 ± 0.25	48.40 ± 2.31	47.68 ± 0.20	47.58 ± 0.29	57.29 ± 0.07
H-C5	55.32 ± 2.58	42.01 ± 1.70	58.72 ± 2.40	55.03 ± 3.42	81.99 ± 0.14
H-C6	51.64 ± 0.74	51.61 ± 1.23	52.75 ± 0.33	51.05 ± 0.59	59.00 ± 0.24

E FULL PRETRAINING TASK POOL

The following inventory enumerates the selected objectives (pre-training task pools). Targets without a comparison are regression objectives, except a within-category rank, which is also continuous. Comparisons are binary objectives. Categorical column names refer to attributes, not ordinal measurements; their numeric codes are treated as categories. All Amazon objectives use 91 days and all HM objectives use seven days. "Rows" denotes event count. Scope abbreviations: D, direct events; C/P/A, events sharing the customer/product/article of a triggering event; PA/AA/CA, product/article/customer attributes reached through that event. Numbering restarts for each root.

Rel-Amazon / Product (38)

1. *D*: count(rows)
2. *D*: count(rows; verified=True)
3. *D*: price × count(rows)
4. *D*: max(rating)
5. *D*: mean(rating)
6. *D*: mean(rating; verified=True)
7. *D*: min(rating)
8. *D*: sd(rating)
9. *D*: sum(rating)
10. *D*: distinct(verified)
11. *C*: count(rows)
12. *C*: price × count(rows)
13. *C*: max(rating)
14. *C*: mean(rating)
15. *C*: min(rating)
16. *C*: sd(rating)
17. *C*: sum(rating)
18. *C*: distinct(verified)
19. *P*: count(rows)
20. *P*: price × count(rows)
21. *P*: max(rating)
22. *P*: mean(rating)
23. *P*: min(rating)
24. *P*: sd(rating)
25. *P*: sum(rating)
26. *P*: distinct(verified)
27. *D*: count(rows) = 0
28. *D*: count(rows) $>$ 1
29. *D*: mean(rating) $>$ 4.667
30. *D*: mode(verified) = 1
31. *C*: count(rows) $>$ 8
32. *C*: mean(rating) $>$ 4.428
33. *C*: mean(rating) $<$ 4.428
34. *C*: mode(verified) = 1
35. *P*: count(rows) $>$ 7
36. *P*: mean(rating) $>$ 4.432
37. *P*: mean(rating) $<$ 4.432
38. *P*: mode(verified) = 1

Rel-Amazon / Customer (40)

1. *D*: count(rows)
2. *D*: count(rows; verified=True)
3. *D*: max(rating)
4. *D*: mean(rating)
5. *D*: mean(rating; verified=True)
6. *D*: min(rating)
7. *D*: sd(rating)
8. *D*: sum(rating)
9. *D*: distinct(verified)
10. *PA*: max(price)
11. *PA*: mean(price)
12. *PA*: min(price)
13. *PA*: sd(price)
14. *PA*: sum(price)
15. *C*: count(rows)
16. *C*: max(rating)
17. *C*: mean(rating)
18. *C*: min(rating)
19. *C*: sd(rating)
20. *C*: sum(rating)
21. *C*: distinct(verified)
22. *P*: count(rows)
23. *P*: max(rating)
24. *P*: mean(rating)
25. *P*: min(rating)
26. *P*: sd(rating)

27. *P*: sum(rating)
28. *P*: distinct(verified)
29. *D*: count(rows) = 0
30. *D*: count(rows) $>$ 1
31. *D*: mode(verified) = 1
32. *PA*: mean(price) $>$ 12.99
33. *PA*: mean(price) $<$ 12.99
34. *C*: count(rows) $>$ 1
35. *C*: mean(rating) $>$ 4.5
36. *C*: mode(verified) = 1
37. *P*: count(rows) $>$ 7
38. *P*: mean(rating) $>$ 4.4
39. *P*: mean(rating) $<$ 4.4
40. *P*: mode(verified) = 1

Rel-HM / Customer (65)

1. *D*: mean(price)
2. *D*: min(price)
3. *D*: max(price)
4. *D*: sum(price)
5. *D*: count(price)
6. *D*: sd(price)
7. *D*: count(sales channel id)
8. *D*: distinct(sales channel id)
9. *AA*: count(product type no)
10. *AA*: distinct(product type no)
11. *AA*: count(product type name)
12. *AA*: distinct(product type name)
13. *AA*: count(product group name)
14. *AA*: distinct(product group name)
15. *AA*: count(graphical appearance no)
16. *AA*: distinct(graphical appearance no)
17. *AA*: count(colour group code)
18. *AA*: distinct(colour group code)
19. *AA*: count(perceived colour value id)
20. *AA*: distinct(perceived colour value id)
21. *AA*: count(perceived colour master id)
22. *AA*: distinct(perceived colour master id)
23. *AA*: count(department no)
24. *AA*: distinct(department no)
25. *AA*: count(department name)
26. *AA*: distinct(department name)
27. *AA*: count(index code)
28. *AA*: distinct(index code)
29. *AA*: count(index group no)
30. *AA*: distinct(index group no)
31. *AA*: count(section no)
32. *AA*: distinct(section no)
33. *AA*: count(garment group no)
34. *AA*: distinct(garment group no)
35. *A*: mean(price)
36. *A*: min(price)
37. *A*: max(price)
38. *A*: sum(price)
39. *A*: count(price)
40. *A*: sd(price)
41. *A*: count(sales channel id)
42. *A*: distinct(sales channel id)
43. *C*: mean(price)
44. *C*: min(price)
45. *C*: max(price)
46. *C*: sum(price)
47. *C*: count(price)
48. *C*: sd(price)
49. *C*: count(sales channel id)
50. *C*: distinct(sales channel id)
51. *D*: mode(sales channel id) = 2
52. *AA*: mode(product type no) = 272
53. *AA*: mode(product type name) = Trousers
54. *AA*: mode(product group name) = Garment Upper body
55. *AA*: mode(graphical appearance no) = 1010016
56. *AA*: mode(colour group code) = 9
57. *AA*: mode(perceived colour value id) = 4
58. *AA*: mode(perceived colour master id) = 5
59. *AA*: mode(index code) = A
60. *AA*: mode(index group no) = 1
61. *AA*: mode(section no) = 15
62. *AA*: mode(garment group no) = 1005
63. *A*: mode(sales channel id) = 2
64. *C*: mode(sales channel id) = 2
65. *D*: count(rows) = 0

Rel-HM / Article (78)

1. *D*: count(rows)
2. *D*: count(rows; sales channel id=1.0)
3. *D*: count(rows; sales channel id=2.0)
4. *D*: rank within colour group code of count(rows)
5. *D*: rank within department name of count(rows)
6. *D*: rank within department no of count(rows)
7. *D*: rank within garment group no of count(rows)
8. *D*: max(price)
9. *D*: max(price; sales channel id=1.0)
10. *D*: max(price; sales channel id=2.0)
11. *D*: mean(price)
12. *D*: mean(price; sales channel id=1.0)
13. *D*: mean(price; sales channel id=2.0)
14. *D*: min(price)
15. *D*: min(price; sales channel id=1.0)
16. *D*: min(price; sales channel id=2.0)
17. *D*: sd(price)
18. *D*: sd(price; sales channel id=1.0)
19. *D*: sd(price; sales channel id=2.0)
20. *D*: sum(price)
21. *D*: sum(price; sales channel id=1.0)
22. *D*: sum(price; sales channel id=2.0)
23. *D*: distinct(sales channel id)
24. *CA*: count(rows; club member status=PRE-CREATE)
25. *CA*: count(age; club member status=PRE-CREATE)
26. *CA*: max(age)
27. *CA*: max(age; club member status=PRE-CREATE)
28. *CA*: mean(age)
29. *CA*: mean(age; club member status=PRE-CREATE)
30. *CA*: min(age)
31. *CA*: min(age; club member status=PRE-CREATE)
32. *CA*: sd(age)
33. *CA*: sd(age; club member status=ACTIVE)
34. *CA*: sd(age; club member status=PRE-CREATE)
35. *CA*: sum(age)
36. *CA*: sum(age; club member status=PRE-CREATE)
37. *CA*: distinct(club member status)
38. *CA*: distinct(fashion news frequency)
39. *A*: count(rows)
40. *A*: sd(price)
41. *A*: sum(price)
42. *C*: count(rows)
43. *C*: rank within colour group code of count(rows)
44. *C*: rank within department name of count(rows)
45. *C*: rank within department no of count(rows)
46. *C*: rank within garment group no of count(rows)
47. *C*: max(price)
48. *C*: mean(price)
49. *C*: min(price)
50. *C*: sd(price)
51. *C*: sum(price)
52. *C*: distinct(sales channel id)
53. *D*: count(rows) = 0
54. *D*: count(rows) $>$ 2
55. *D*: count(rows) $<$ 2
56. *D*: count(rows; sales channel id=1.0) = 0
57. *D*: count(rows; sales channel id=2.0) = 0
58. *D*: count(rows; sales channel id=2.0) $>$ 1
59. *D*: mean(price) $>$ 0.02371
60. *D*: mean(price) $<$ 0.02371
61. *D*: mean(price; sales channel id=1.0) $>$ 0.01693
62. *D*: mean(price; sales channel id=1.0) $<$ 0.01693
63. *D*: mean(price; sales channel id=2.0) $<$ 0.02541
64. *D*: mode(sales channel id) = 2
65. *CA*: mean(age) $>$ 35
66. *CA*: mean(age) $<$ 35
67. *CA*: mean(age; club member status=ACTIVE) $>$ 35
68. *CA*: mean(age; club member status=ACTIVE) $<$ 35
69. *CA*: mean(age; club member status=PRE-CREATE) $<$ 38.67

70. *CA*: mode(fashion news frequency) = 0
71. *A*: count(rows) $>$ 3
72. *A*: mean(price) $>$ 0.02371
73. *A*: mean(price) $<$ 0.02371
74. *A*: mode(sales channel id) = 2
75. *C*: count(rows) $>$ 13
76. *C*: count(rows) $<$ 13
77. *C*: mean(price) $<$ 0.0253
78. *C*: mode(sales channel id) = 2