A UNIVERSAL SCHEMA FOR RELATIONAL FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Traditional relational deep learning models use a database's relational structure to predict properties of its entities for a given task specification, but are tied to the schema and task on which they are trained. Recent work on relational foundation models aims to remove this constraint by pretraining a single model across databases and transferring it to unseen ones. Existing approaches address schema variation through the model architecture. We propose a complementary, representation-level approach: an information-preserving mapping that represents any database and task using a fixed universal schema, moving schema agnosticism from the model architecture to the input data. Any existing relational deep learning model can then be trained across databases and transferred to new ones. On RelBench classification and regression tasks, we show that key relational foundation-model capabilities can emerge from this representation alone. A standard GraphSAGE trained across databases achieves zero-shot transfer to an unseen database under the Relational Transformer protocol, while transferred models require substantially fewer examples to reach a given accuracy when fine-tuned on a new database. Overall, our results suggest that relational foundation-model capabilities need not be encoded entirely in specialized architectures: they can emerge from the representation itself, enabling a modular interface between relational data and learning architectures.[1]

1 INTRODUCTION

A machine-learning model for relational databases is typically designed for a specific relational schema. In particular, such a model is aware of the database's *signature*: its relations, their attributes and data types, and the foreign-key constraints that connect them. This dependence is particularly explicit in relational deep learning, where a database is commonly represented as a *heterogeneous graph* and where heterogeneous graph neural networks have become standard architectures for learning over relational databases (Wang et al., 2019; Hu et al., 2020; Zhao et al., 2021; Fey et al., 2024; Robinson et al., 2024). In this representation, tuples are nodes, with node types corresponding to relations, while foreign-key references are edges, with edge types corresponding to foreign-key constraints. The model's awareness of the schema is then reflected in its parameterization: a heterogeneous graph neural network can associate different parameters with different node and edge types, and thereby process different relations and foreign-key constraints differently.

In contrast, *relational foundation models* aim to pretrain a single model across databases of varying schemas, and then transfer to new databases and tasks. Such pretraining can support capabilities including zero-shot prediction, label-efficient fine-tuning, few-shot learning, and in-context learning. The prevailing approach is to construct a sufficiently flexible architecture to accommodate databases with widely varying signatures, including different relations, attributes, data types, and foreign-key structures. This flexibility is achieved through purpose-built, often complex neural architectures, typically based on transformers. Examples include the Relational Transformer (Ranjan et al., 2026), Griffin (Wang et al., 2025), KumoRFM (Fey et al., 2025; Hudovernik et al., 2026), and OpenRFM (Chen et al., 2026). A different approach is to flatten the relational database into a single feature table, using joins and predefined aggregations in the spirit of deep feature synthesis (Kanter & Veeramachaneni, 2015), and then apply a tabular foundation model such as TabPFN (Holl-

[1] Code to reproduce all experiments: `https://anonymous.4open.science/r/UniSchema`.

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.

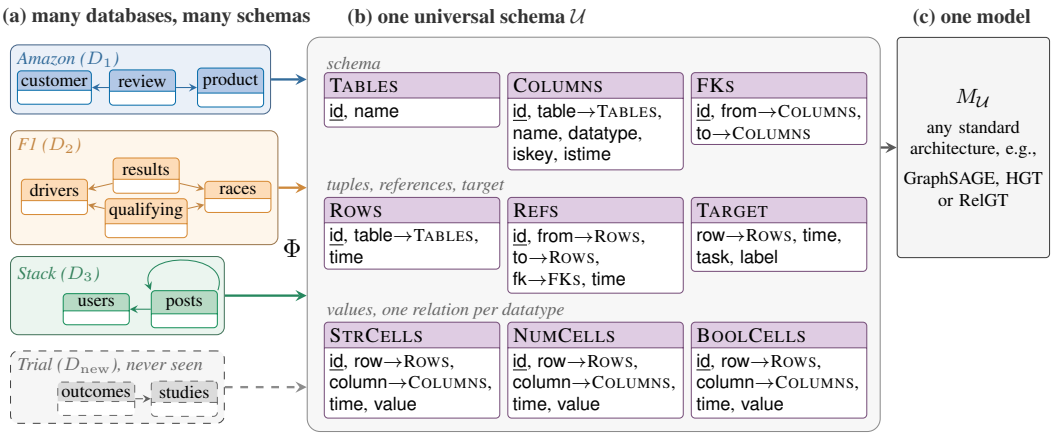


Figure 1: One model for many databases. (a) Four databases, each with its own schema (b) The mapping Φ (Section 3.2) takes each of them to a database over the same universal schema $\mathcal{U}$; a task populates TARGET (Section 3.3). (c) One schema-specific model for $\mathcal{U}$ is trained on all of them and, with no change, predicts on a new one.

mann et al., 2023; 2025; Grinsztajn et al., 2026). This approach is taken, for example, by TabPFN-Rel (Hayler et al., 2026). Although relational flattening has performed strongly on recent benchmarks (Hayler et al., 2026; Grinsztajn et al., 2026), models that operate directly on the relational structure are arguably more flexible: they can learn how information should propagate across tables and relationships, rather than restricting cross-table signals to a predetermined collection of aggregate queries.

This work proposes a third approach that separates the handling of schema variation from the design of the learning architecture. The first component maps every input database to a fixed universal schema through a data-exchange process (Fagin et al., 2005; Arenas et al., 2014). The mapping is lossless: the original database, including its schema and contents, can be reconstructed from the resulting database. Crucially, the target universal schema is fixed and independent of the source schema. Hence, databases with different *source schemas* are all represented as databases over exactly the same *target schema*. The second component is then an ordinary schema-specific model for this universal schema. Since the model always receives databases of the same schema, the exact same architecture and parameters can be trained and applied across databases. Thus, turning any schema-specific architecture into a relational foundation model requires no additional architectural machinery: it amounts to applying that architecture to the universal schema and training it across mapped databases. This separation considerably simplifies the design, use, and maintenance of relational foundation models. Importantly, it also allows us to examine the architectural complexity introduced by contemporary relational foundation models such as the aforementioned Relational Transformers, Griffin, and KumoRFM.

Figure 1 illustrates the construction. Similar to the previously proposed entity-attribute-value representations (Nadkarni et al., 1999), it stores relational data using a fixed collection of generic relations; and similar to the information-preserving mappings from relational databases to RDF (Sequeda et al., 2012), it encodes both the source schema and its instance in a representation from which the original database can be reconstructed. Relations, attributes, and foreign-key constraints become tuples describing the original schema; each original tuple, cell, and foreign-key reference is stored in a designated tuple. Prediction instances are represented uniformly in a universal TARGET relation, which records the target row, prediction time, and label. Notably, the universal schema contains the same fixed set of relations regardless of the source schema: differences among source relations, attributes, and foreign keys are encoded *as data* rather than in the database's signature. In this way, schema variation is moved from the learning architecture into the input data.

We show, quite remarkably, that the universal-schema approach can match, and even outperform, complex architectures designed specifically as relational foundation models, while using ordinary, off-the-shelf models. We instantiate the approach with GraphSAGE (Hamilton et al., 2017), HGT (Hu et al., 2020), and RelGT (Dwivedi et al., 2026), without modifying their architectures,

and evaluate two central foundation-model capabilities on RelBench (Robinson et al., 2024). First, we study *zero-shot transfer*, where a model trained across several databases is applied, without any parameter updates, to a new database whose schema and tasks were not encountered during training. The task is communicated to the model through a textual description and the target entity's historical labels, following the protocol of the Relational Transformer (Ranjan et al., 2026). Second, we study *label-efficient fine-tuning*, where the pretrained model is adapted to the new database using labeled examples and compared with the same architecture trained from scratch.

On zero-shot transfer, all three architectures match or surpass the Relational Transformer, with a mean AUROC of 71.6, 71.4 and 70.8 against 70.6 over the twelve classification tasks and a mean normalized MAE of 35.3, 34.1 and 34.4 against 39.0 over the nine regression tasks, and they surpass Griffin by a wide margin (66.3 and 46.2, respectively). In fine-tuning, the pretrained model leads the same architecture trained from scratch by 4.0 AUROC points and 5.0 nMAE points after 128 steps, and the gap narrows at the full budget; hence, pretraining on the universal schema improves sample efficiency rather than raising the final score.

2 Related Work

We now shortly review related work.

Relational deep learning. Conventionally, a database is represented as a heterogeneous graph and a relational deep learning model is trained on it (Cvitkovic, 2020; Fey et al., 2024; Robinson et al., 2024; Wang et al., 2024). The resulting model is tied to that representation at two levels. First, the mechanism used to translate a database into a graph is itself database-dependent (e.g., AutoG (Chen et al., 2025b) searches over possible graph constructions, while RDB2G-Bench (Choi et al., 2025) benchmarks different constructions). Second, models such as HGT (Hu et al., 2020), RelGNN (Chen et al., 2025a), and RelGT (Dwivedi et al., 2026) use schema-dependent parameters, with weights associated with the node and edge types of the input graph, so changing the schema changes the parameterization of the model (Section 3.1). We instead keep these models unchanged and change their input: the universal schema fixes the relation names, attributes, and foreign keys that a model sees, allowing the same model to transfer across databases.

Relational foundation models. Relational foundation models aim to learn representations that transfer across databases and tasks despite schema differences. Existing approaches achieve schema agnosticism in two ways. The first builds it into the architecture: RT tokenizes cells and pretrains a schema-agnostic transformer prompted with a task table (Ranjan et al., 2026), while PluRel (Kothapalli et al., 2026), Griffin (Wang et al., 2025), KumoRFM and KumoRFM-2 (Fey et al., 2025; Hudovernik et al., 2026), OpenRFM (Chen et al., 2026), and RDB-PFN (Wang et al., 2026) use related mechanisms to absorb schema variation. The second leaves the model unchanged and translates the database into its input: as text or a GNN-encoded prompt for language models (Wydmuch et al., 2024; Wu et al., 2025), or as parameter-free aggregates for tabular in-context models (Xu et al., 2026; Hayler et al., 2026). These approaches share our premise that schema agnosticism can come from the data rather than the model, but their transformations are generally lossy and designed for a particular model family. Aggregation, for example, flattens the database into a fixed-depth feature table, losing references, schema structure, and temporal information. The universal schema instead preserves the database as a relational representation that any relational model can read unchanged.

Schema variation as data. Mao et al. (2024) argue that a fixed vocabulary shared across datasets is the missing piece of graph foundation models; the universal schema provides such a vocabulary for relational databases. Concurrently, Pradel (2026) applies the same principle to knowledge graphs: reifying each triple into a node linked through fixed meta-relations enables standard GNNs to transfer zero-shot across graphs. Databases require normalizing not only relation types but also attributes, typed values, references, time, and tasks.

Generic schemas in data management. Storing data of any schema in a fixed set of relations is an established idea. Entity-attribute-value representations store every cell as a triple (Nadkarni et al., 1999; Dinu & Nadkarni, 2007), and direct mappings of relational databases to RDF encode the schema and instance as triples from which the database can be reconstructed (Sequeda et al., 2012). Neither was designed as input to a learning model: EAV is a storage scheme, while RDF

targets querying. The universal schema combines the relational form of the former with the lossless encoding of the latter, and adds references, time, and tasks as relations of its own.

3 UNIVERSAL SCHEMA

A machine-learning model for relational databases is typically designed for one particular schema, and it is applicable only to databases over this schema; for example, the parameters of a neural network are typically tied to the relation names and the attributes of the schema. Relational foundation models lift this restriction through their architecture (Section 2), we lift it through the input instead. In this section, we present a *universal schema* to which every database with any relational schema can be mapped losslessly. Consequently, a machine-learning model for the universal schema applies, through the mapping, to every database, so one model can be trained and used across databases of different schemas. We begin with necessary definitions (Section 3.1), then present the universal schema and the mapping (Section 3.2), and finally discuss how a model for the universal schema is used as a relational foundation model (Section 3.3).

3.1 DATABASES AND MODELS

A *database schema* $\mathcal{S}$, or *schema* for short, is associated with a finite set of *relation names*. Each relation name R is associated with a sequence of distinct *attributes*, and each attribute A of R has a *domain*, denoted $\mathrm{dom}(A)$, which is one of the following *datatypes*: string, integer, real number, Boolean, and timestamp. In addition, R is associated with at most one *key*, denoted $\mathrm{key}(R)$, which is an attribute of R, and with at most one *time attribute*, which is a timestamp attribute of R. Note that we assume (without loss of generality) that keys are unary, as holds in standard benchmarks such as RelBench (Robinson et al., 2024).

The schema $\mathcal{S}$ also includes a set of *foreign-key constraints* (FKs). An FK is an expression of the form $R[C] \sqsubseteq \mathrm{key}(R')$, stating that the attribute C of R, which we call a *foreign key* of R, references the key of R'. Here, R and R' are relation names of $\mathcal{S}$ such that R' has a key, and C has the same domain as $\mathrm{key}(R')$. As in Rissaki et al. (2026), an attribute of R is a *key attribute* if it is the key of R, a *foreign-key attribute* if it is a foreign key of R, and a *data attribute* if it is neither of the two.

A *tuple* t over a relation name R assigns to every attribute A of R a value $t[A]$, which either belongs to $\mathrm{dom}(A)$ or is missing. A *relation* over a relation name R is a finite set of tuples over R. A *database* D over $\mathcal{S}$ assigns to every relation name R of $\mathcal{S}$ a relation $D(R)$ over R, so that the following two constraints hold. First, if R has a key, then $t[\mathrm{key}(R)]$ is not missing for every tuple t of $D(R)$, and $t_1[\mathrm{key}(R)] \neq t_2[\mathrm{key}(R)]$ for every two distinct tuples t_1 and t_2 of $D(R)$. Second, for every FK $R[C] \sqsubseteq \mathrm{key}(R')$ of $\mathcal{S}$ and every tuple t of $D(R)$ such that $t[C]$ is not missing, there is a tuple t' of $D(R')$ such that $t[C] = t'[\mathrm{key}(R')]$. This tuple t' is unique by the first constraint. We say that t *references* t', and that the pair (t, t') is a *foreign-key reference* for this FK. For a tuple t over R, we denote by $\mathrm{time}(t)$ the value of t in the time attribute of R. If R has no time attribute, or if this value is missing, then we define $\mathrm{time}(t) = -\infty$, which is earlier than every timestamp. In words, we regard t as present from $\mathrm{time}(t)$ on, similarly to Fey et al. (2024); in particular, we assume that t is present at all times if $\mathrm{time}(t) = -\infty$.

We next define prediction tasks over a single schema. We fix a *prediction domain* $\mathcal{Y}$, which is one of the datatypes, such as the Booleans for classification or the reals for regression.[2] Following Cucumides & Geerts (2025), a *prediction task* over a schema $\mathcal{S}$ is specified by a *target relation* T, which is a relation name of $\mathcal{S}$, together with an attribute L of T whose domain is $\mathcal{Y}$, which we call the *label attribute*. In this paper, a target relation T has three attributes: a foreign-key attribute, which states which tuple the prediction is about, the time attribute of T, which states when the prediction is made, and the label attribute L. Given a database D over $\mathcal{S}$ and a tuple t of $D(T)$ whose value $t[L]$ is missing, the task is to predict this value. Adapting Rissaki et al. (2026), a *model* for $\mathcal{S}$ and T is a function $M_{\mathcal{S},T}$ that maps a database D over $\mathcal{S}$ and a tuple t of $D(T)$ to a prediction $M_{\mathcal{S},T}(D, t)$ in $\mathcal{Y}$. This definition places no restriction on how $M_{\mathcal{S},T}$ is realized: it can be, for example, a neural network, a decision tree, or an SQL query. Crucially, such a model is schema-specific: it is defined only on databases over $\mathcal{S}$. In particular, it cannot be applied to a database over a different schema.

[2] We consider binary classification and regression. A multi-class task fits the definitions with integer labels.

3.2 THE UNIVERSAL SCHEMA

We refer to a schema $\mathcal{S}'$ as *universal* if there is a mapping that takes every database D over any schema $\mathcal{S}$ to a database D' over $\mathcal{S}'$, and that preserves the information of the database (that is, we can reconstruct D, given D', up to the values of its keys).

We propose a universal schema, which we denote by $\mathcal{U}$ and refer to as *the* universal schema in the rest of the paper. The idea is that a database over $\mathcal{U}$ stores another database as data, including its schema (metadata), which is why a fixed set of relation names suffices for every schema. The universal schema $\mathcal{U}$ consists of the following nine relation names. An underlined attribute is the key of its relation name, an entry $A \to R'$ stands for the attribute A, which is a foreign key that references the key of R', and time is the time attribute of every relation name that has it. The labels on the left state what the relations of each group store.

Schema	TABLES(id, name)
	COLUMNS(id, table → TABLES, name, datatype, iskey, istime)
	FKS(id, from → COLUMNS, to → COLUMNS)
Tuples	ROWS(id, table → TABLES, time)
References	REFS(id, from → ROWS, to → ROWS, fk → FKS, time)
Values	STRCELLS(id, row → ROWS, column → COLUMNS, time, value)
	Similar: NUMCELLS(...), BOOLCELLS(...)
Target	TARGET(row → ROWS, time, task, label)

The domain of value is the strings, real numbers, and Booleans in STRCELLS, NUMCELLS, and BOOLCELLS, respectively, while the domain of label is the prediction domain $\mathcal{Y}$. Our implementation also assigns a name to every foreign key and augments the schema tuples (TABLES, COLUMNS, and FKS). The relation name TARGET is *distinguished*: whenever we consider a model for $\mathcal{U}$, its target relation is TARGET and its label attribute is label. We refer to such a model as a *model for the universal schema*, and we denote it by $M_{\mathcal{U}}$. Note that $\mathcal{U}$ is an ordinary schema, thus $M_{\mathcal{U}}$ is an ordinary model as defined in Section 3.1.

Next, we describe the mapping. The mapping takes a schema $\mathcal{S}$ and a database D over $\mathcal{S}$, and it does not depend on any task. The mapping returns a database over $\mathcal{U}$, which we denote by $\Phi(D)$, leaving $\mathcal{S}$ implicit, and it creates the tuples of $\Phi(D)$ as follows. • *Schema.* Every relation name of $\mathcal{S}$ becomes a tuple of TABLES, every attribute becomes a tuple of COLUMNS, and every FK becomes a tuple of FKS. • *Tuples.* Every tuple t of D becomes a tuple of ROWS whose time is $\mathrm{time}(t)$. • *References.* Every foreign-key reference (t, t') among these tuples becomes a tuple of REFS, whose time is the later of $\mathrm{time}(t)$ and $\mathrm{time}(t')$. • *Values.* Every value $t[A]$ of a data attribute A that is not missing becomes a tuple of STRCELLS, NUMCELLS or BOOLCELLS, according to $\mathrm{dom}(A)$, whose time is $\mathrm{time}(t)$. • *Target.* No tuples: TARGET is filled according to a task, as Section 3.3 describes.

Every such tuple references, through the foreign keys of $\mathcal{U}$, the tuples of the objects that it belongs to. For example, the tuple of a value $t[A]$ references the ROWS tuple of t and the COLUMNS tuple of A. Two exceptions apply to the values: the value of the time attribute is held by the ROWS tuple and gets no tuple of its own, and every other timestamp is stored in NUMCELLS as a number. Appendix B gives further details, and Appendix C shows an example. Notice that the original database D can be reconstructed from these tuples. Explicit key values are omitted since they serve solely to identify tuples; the identifiers of ROWS take their place. Consequently, D is recovered exactly up to the arbitrary values of its keys, making the mapping described so far invertible and structurally lossless.

The mapping normalizes numbers. The attribute value of NUMCELLS pools attributes of $\mathcal{S}$ whose scales differ by orders of magnitude, such as ages and prices, so the mapping transforms the values of every attribute of $\mathcal{S}$ separately by quantile transformation to the standard normal distribution (Pedregosa et al., 2011). This step keeps the order of the values and not the values themselves, so it is not invertible. Storing the original number as well would make it invertible.

3.3 USE AS A RELATIONAL FOUNDATION MODEL

A *foundation model* is trained on broad data and adapted to a wide range of tasks (Bommasani et al., 2021). Accordingly, a *relational foundation model* is trained on many databases with many different

schemas, and applied to a new database over a new schema and a new task (Ranjan et al., 2026; Wang et al., 2025; Fey et al., 2025). In this paper, we focus on two ways in which such a model is used. In *fine-tuning*, a pretrained model is trained further on labeled data of the new database, aiming to reach a given score with fewer labels than a model trained from scratch. In *zero-shot learning*, the model receives no labeled examples of the task. Instead, it may receive a textual description of the task, as well as earlier labels of the target tuple from the history, called *self-labels* (Ranjan et al., 2026), which exist when the task is asked repeatedly over time. Other ways, such as *few-shot* and *in-context learning*, apply as well; our experimental study focuses on the former two.

Recall that an ordinary relational model is defined with respect to both a database schema and a designated target relation. We apply this definition to the universal schema by designating TARGET as its target relation. Hence, a model $M_{\mathcal{U}}$ for the universal schema is simply an ordinary schema-specific model for the fixed pair $(\mathcal{U}, \text{TARGET})$. Given a source database D over a schema $\mathcal{S}$ and a task with target relation T, the database mapping produces $\Phi(D)$, while the task mapping populates its TARGET relation. For every tuple t of $D(T)$, it creates a corresponding tuple of TARGET whose row references the relevant tuple of ROWS, whose time records the prediction time, whose task identifies the task, and whose label contains $t[L]$ when it is known and is missing when it is to be predicted. Applying $M_{\mathcal{U}}$ to the resulting database and the corresponding TARGET tuple therefore yields a prediction for t. Since both $\mathcal{U}$ and its target relation TARGET are fixed, the same model can be applied to every source schema and task.

This construction implies how the different foundation-model settings are realized. In fine-tuning, labeled tuples of TARGET are used to update a pretrained model, which is then applied to tuples whose labels are missing. In zero-shot prediction, the model parameters remain fixed, and the task is instead described within the mapped database. The task attribute may contain a textual description of the task, while earlier labeled TARGET tuples provide *self-labels*; these are connected to the prediction tuple through their common foreign-key reference to the corresponding ROWS tuple.

4 EXPERIMENTS

We test whether the universal schema provides standard models with the capabilities expected of a relational foundation model. We compare with RT (Ranjan et al., 2026) and Griffin (Wang et al., 2025). Following RT, we evaluate two forms of transfer to an unseen database: zero-shot prediction, where the model is applied without task-specific training (Section 4.2), and label-efficient fine-tuning (Section 4.3). Section 4.4 ablates what the model reads from the universal schema and measures the cost of mapping a database to the universal schema.

4.1 EXPERIMENTAL SETUP

Datasets and tasks. We use the official temporal splits of RelBench (Robinson et al., 2024) (package version 2.1.2). The seven RelBench V1 databases with entity-level tasks (Amazon, Avito, Event, F1, H&M, Stack, and Trial) provide twelve binary classification tasks and nine regression tasks. Following RelArena (Hayler et al., 2026), we score classification by AUROC and regression by MAE, which we report as normalized MAE (nMAE) by dividing it by the standard deviation of the task's training labels, as on the RelBench leaderboard (Stanford Tabular and Relational Project, 2026); see Appendix D.3 for the rationale for using nMAE rather than R^2.

Models and input. We realize $M_{\mathcal{U}}$ with three published architectures, unchanged: GraphSAGE (Hamilton et al., 2017), HGT (Hu et al., 2020), and RelGT (Dwivedi et al., 2026). A database, either in its native schema or in the universal schema representation, is mapped to a graph using the standard RelBench protocol (Fey et al., 2024; Robinson et al., 2024), and the resulting graph is used as input to the model. Model configurations are in Appendix D.1. For each prediction, we sample a subgraph around the target tuple using RelBench's time-aware sampling procedure (Robinson et al., 2024), which only includes tuples available before prediction time. We augment each sampled subgraph with two additional pieces of information: the task description and *self-labels* (Ranjan et al., 2026). The task is represented by the task attribute of TARGET, containing the RelBench task name and its one-sentence description (Appendix D.2), and encoded using the same text encoder as string-valued attributes. Self-labels are earlier TARGET tuples corresponding to the same tuple and task. Both are part of the mapped database input and require no changes to the model architecture.

Baselines. We compare against two relational foundation models with published zero-shot results on RelBench: RT (Ranjan et al., 2026) and Griffin (Wang et al., 2025). We take their scores from the TabPFN-3 report (Grinsztajn et al., 2026), which reports both models on all 21 tasks, including the Event tasks omitted from RT's original paper. The report notes that RT's scores were likely obtained under a protocol that overestimates performance. We also report two constant baselines from RelArena (Hayler et al., 2026): *Entity Mean* for classification, which predicts each entity's training-set positive rate, and *Entity Median* for regression, which predicts each entity's median training label. For entities without training labels, both baselines fall back to a global constant. For fine-tuning, we use the same architectures trained from random initialization, both on the universal schema and on the original database schema (Section 4.3).

Training. We match RT's training budget for a direct comparison. In pretraining, every step samples 512 labeled TARGET tuples from the training tasks, in proportion to their training-set sizes. A batch can contain both classification and regression examples; classification examples are processed by a classification head using binary cross-entropy, and regression examples by a regression head using Huber loss. The two losses are treated equally, with no additional weighting. We use the AdamW optimizer (Loshchilov & Hutter, 2019). We train for 25,000 steps, corresponding to the same number of labeled examples as RT's 50,000 steps with batch size 256. For fine-tuning, we train on a single task for 32,768 steps with batch size 256, again matching RT's budget. Regression targets are normalized using the training-split quantile transformation described in Section 3.2 and transformed back before scoring. Training parameters are given in Appendix D.1.

4.2 Zero-Shot Transfer to Unseen Databases

In zero-shot evaluation, we follow RT's leave-one-database-out protocol (Ranjan et al., 2026, §4.1). For each of the seven databases, we train one model on all tasks from the other six databases and evaluate it on every task in the held-out database without updating the model weights. As in RT, we report the score on each target task's test split using the checkpoint with the best validation score.

Observations. On average, all three architectures match or surpass RT. Mean AUROC is 71.6 for GraphSAGE, 71.4 for HGT and 70.8 for RelGT, against 70.6 for RT and 66.3 for Griffin, and mean nMAE is 35.3, 34.1 and 34.4 against 39.0 for RT and 46.2 for Griffin (Table 1). Per task, each of our models beats RT on 6 to 8 of the 12 classification tasks, with the largest gain on Stack user-engagement (up to +11.4 AUROC) and RT ahead on both F1 tasks. On regression the gap is wider: each of our models beats the Entity Median baseline on at least 6 of the 9 tasks, whereas RT beats it on 2 and Griffin on 1. None of these architectures can be trained across databases without first mapping them to a universal schema. Nevertheless, once databases are mapped to the universal schema, all three architectures achieve zero-shot transfer across databases at performance comparable to RT. These results suggest that cross-database zero-shot transfer can come from the input representation rather than requiring an architecture specifically designed for it.

4.3 Label-Efficient Fine-Tuning

We repeat RT's fine-tuning experiment to test whether pretraining on the universal schema provides the same benefit: a pretrained model should reach a given score on a new database in fewer training steps than the same model trained from scratch. Each run fine-tunes on one task of a held-out database, starting either from the pretrained model of Section 4.2, which never saw that database, or from random initialization, and records the test score at every evaluation step. A third run trains the same architecture from random initialization on the original schema, the schema-specific model of Section 3.1, under the same protocol. Figure 2 averages the curves over tasks for RelGT, and Appendix D.4 shows every task for all three architectures, with RT's published curve overlaid where RT reports one.

Observations. Pretraining substantially accelerates fine-tuning. For RelGT, after 128 steps (32,768 labeled examples), the pretrained model surpasses the same architecture trained from scratch on the original schema by 4.0 AUROC points and 5.0 nMAE points, leading on 17 of the 21 tasks. By then, the model trained from scratch has still not reached the pretrained model's zero-shot AUROC (Appendix D.5). Under the full training budget the gap narrows, though the pretrained model stays ahead, at 76.9 against 75.0 AUROC and 30.3 against 33.4 nMAE (Section D.6). Under this

Table 1: Zero-shot AUROC (top) and nMAE (bottom) on the 21 RelBench V1 tasks, each scored on its held-out database and trained on the other six. Entity is RelArena's Entity Mean (classification) or Entity Median (regression) baseline (Hayler et al., 2026). Griffin and RT come from the TabPFN-3 report (Grinsztajn et al., 2026).

		Baselines			Universal schema		
DB	Task	Entity	Griffin	RT	SAGE	HGT	RelGT
Classification: AUROC (%) ↑							
Amazon	user-churn	63.4	62.3	64.0	63.2	64.1	62.6
	item-churn	72.9	69.0	70.9	74.7	71.1	71.6
Avito	user-visits	60.3	60.7	61.8	64.2	62.1	62.9
	user-clicks	50.4	45.9	59.5	63.6	59.1	61.1
Event	user-repeat	75.2	71.9	73.2	70.5	73.2	73.6
	user-ignore	84.0	83.3	77.5	76.1	68.3	79.9
F1	driver-dnf	69.9	57.7	81.2	76.8	78.2	76.2
	driver-top3	55.7	82.5	89.3	85.2	87.9	83.1
H&M	user-churn	64.8	60.2	62.8	66.3	68.9	68.4
Stack	user-engage	82.7	77.5	75.7	82.6	87.1	80.7
	user-badge	78.9	73.5	80.1	78.8	81.0	77.7
Trial	study-out	50.0	51.0	51.8	57.0	55.9	52.1
Mean		67.3	66.3	70.6	71.6	71.4	70.8
Wins vs. Entity			3/12	7/12	7/12	9/12	6/12
Regression: nMAE (%, Section 4.1) ↓							
Amazon	user-ltv	30.3	61.9	32.7	28.2	26.9	27.2
	item-ltv	11.1	9.0	9.8	10.4	9.9	10.0
Avito	ad-ctr	43.1	52.3	60.6	41.9	44.9	44.1
Event	user-attend	35.2	60.2	49.5	33.4	32.2	38.6
F1	driver-pos	58.4	63.5	41.3	59.5	48.7	50.8
H&M	item-sales	15.8	30.5	18.0	15.0	14.9	15.3
Stack	post-votes	13.6	18.0	21.6	13.7	15.4	12.7
Trial	site-succ	92.7	97.3	95.6	99.3	97.0	94.7
	study-adv	17.0	23.1	21.8	16.6	16.8	16.6
Mean		35.2	46.2	39.0	35.3	34.1	34.4
Wins vs. Entity			1/9	2/9	6/9	6/9	6/9

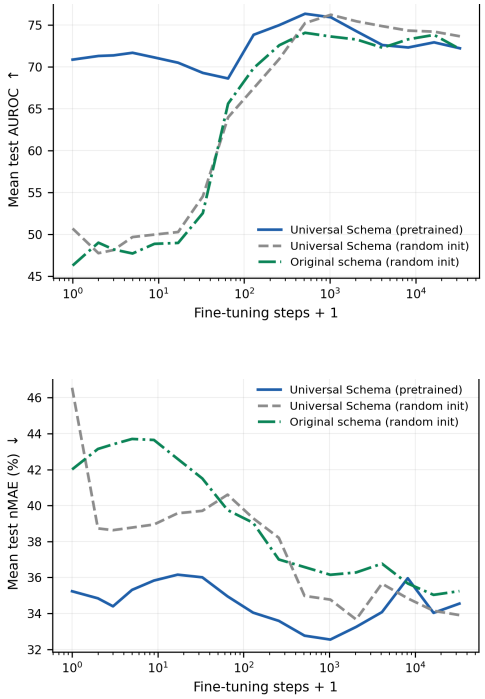


Figure 2: Mean test AUROC (top) and nMAE (bottom) of RelGT against fine-tuning steps, averaged over tasks. Three runs per task: from the pretrained model and from random initialization on the universal schema, and from random initialization on the original schema, the schema-specific model of Section 3.1.

protocol, the same architecture trained from random initialization on the universal schema also outperforms the original-schema run, by 2.6 AUROC points, showing no loss in final accuracy from the universal schema mapping, and it ends within 0.7 points of the pretrained run on both metrics. Pretraining on the universal schema thus improves sample efficiency rather than raising the final score. Appendix D.6 compares the final scores with the same architecture trained per task on its own schema.

4.4 Additional Details

Ablation. We ablate additional elements of the universal schema that are design choices. These include schema names, schema statistics, and task descriptions. Because these elements are represented as nodes in the sampled subgraph rather than architectural components, they can be ablated without retraining. Using the zero-shot models from Section 4.2, we mask each element in turn at inference time at the task's best-validation checkpoint.

Figures 3a and 3b report the change from the full RelGT model. *Task context* blanks the text of task. *Schema names* blanks the names in the Tables, Columns and FKs tuples; blanked text is replaced by the embedding of the empty string, identical for every tuple. *Schema stats* replaces each statistic with the average for its tuple type in that database, so the statistics no longer distinguish between schema tuples. *Schema tuples* drops the Tables, Columns and FKs tuples from the graph. Task context has the largest effect on performance. Without it, classification tasks lose an average of 4.3 AUROC points; and notably, Stack *user-engagement* drops from 80.7 to 58.2. Regression tasks show similar sensitivity, with Stack *post-votes* increasing by 43.2 nMAE points. Schema names have little effect: no task changes by more than 0.6 points. Schema statistics show mixed per-task effects, but little average change. In contrast, dropping schema tuples decreases performance by 1.3 AUROC points for classification and 5.2 nMAE points for regression. This is consistent with the model using

(a) Classification: AUROC (%)

DB	Task	Full	Masked input: Task context	Schema names	Schema stats	Schema tuples
Amazon	user-churn	62.6	+0.2	0.0	−0.1	−1.8
	item-churn	71.5	−0.1	0.0	−0.2	−3.3
Avito	user-visits	62.8	−1.0	0.0	+0.1	−0.8
	user-clicks	60.8	+0.9	−0.6	−0.2	−4.4
Event	user-repeat	74.9	−11.0	0.0	−4.3	−4.7
	user-ignore	79.7	−6.4	0.0	+2.5	−6.1
F1	driver-dnf	76.1	+0.3	+0.2	−4.1	−0.2
	driver-top3	83.1	−8.1	+0.2	+3.8	+1.6
H&M	user-churn	68.3	−2.8	0.0	−3.4	−1.3
Stack	user-engage	80.7	−22.5	0.0	−0.3	+2.8
	user-badge	77.8	−0.4	0.0	−0.4	+0.5
Trial	study-out	52.4	−0.2	0.0	+0.3	+2.0
Mean		70.9	−4.3	0.0	−0.5	−1.3

(b) Regression: nMAE (%, lower is better)

DB	Task	Full	Masked input: Task context	Schema names	Schema stats	Schema tuples
Amazon	user-ltv	27.2	+0.1	0.0	0.0	0.0
	item-ltv	10.0	+0.1	0.0	0.0	0.0
Avito	ad-ctr	44.0	+7.1	0.0	+0.4	−0.6
Event	user-attend	38.8	−2.0	0.0	−3.1	+4.0
F1	driver-pos	50.8	+0.5	0.0	+0.4	+1.7
H&M	item-sales	15.3	0.0	0.0	0.0	0.0
Stack	post-votes	12.7	+43.2	0.0	0.0	+30.1
Trial	site-succ	94.8	−1.5	0.0	+0.9	−8.5
	study-adv	16.6	−0.5	0.0	0.0	+19.9
Mean		34.5	+5.2	0.0	−0.2	+5.2

Figure 3: Input ablation of the zero-shot RelGT model. *Full* is the unmasked zero-shot score. Every other column is the change in points when we mask that input at inference. A negative change in AUROC, or a positive change in nMAE, means the model relied on the input. *Full* reproduces the RelGT scores of Table 1 to within 0.22 AUROC points on average.

schema tuples primarily for their structural connectivity, linking values to their attributes, relations, and other values, rather than for their semantic names.

Mapping overhead. The mapping increases graph size by adding tuples for relation names, attributes, foreign keys, and values, without changing the model architecture or number of parameters. Across the seven databases, graph size increases from 62 GiB to 143 GiB, with 7× as many nodes (728M) and 10× as many edges (3.0B); per-database memory increases by 1.1–5.5×. The increase is smallest for text-heavy databases, which already store text embeddings, and largest for those dominated by numeric and categorical values. Graph construction cost is largely unaffected, as it is dominated by text-value embedding, which is nearly identical under both schemas. The universal schema adds a fixed computational cost, so training can be faster or slower than on the original schema depending on its complexity; see Appendix D.7 for per-database training costs.

Limitations. Each zero-shot and fine-tuning result is a single run, and identical runs differ by about one AUROC point. The fine-tuning curves show the test score at every step, and Appendix D.6 reports each run at its best-validation checkpoint. Zero-shot evaluation selects each target task's checkpoint on its own validation split, as RT does. Finally, our models read a task description that RT does not, while RT reads other tuples' labels in its context, which ours do not.

5 Conclusions

The premise of our work is that the ability to operate across databases with different schemas need not be implemented entirely through specialized model architectures. Instead, each database and prediction task can be mapped into a fixed relational schema, allowing an ordinary schema-specific model to be trained across databases and transferred to new ones without architectural modification. We presented an experimental study that supports this premise. Standard and simple GNNs operating on the universal schema transfer effectively to databases and tasks unseen during training, matching or outperforming considerably more specialized relational foundation models. Moreover, the parameters learned across databases provide an effective initialization for adaptation, substantially improving label efficiency when fine-tuning on a new database.

These results suggest that the universal-schema approach can substantially lower the barrier to building relational foundation models. Rather than requiring expertise in designing and implementing complex architectures for various schemas, domain experts can map their databases to a common representation and reuse standard, well-supported learning architectures. This separation may make relational foundation models accessible to users with limited machine-learning engineering and programming expertise, while also simplifying their implementation, extension, and maintenance. In this context, an important direction for future work is to extend the universal-schema approach to other core capabilities of relational foundation models, particularly few-shot and in-context learning.

AI USE STATEMENT

In this work, we used generative AI tools (Claude, Anthropic) for the following tasks that require disclosure: implementing methods, in that the code for mapping databases to the universal schema, for training and evaluation, and for the analysis scripts was written with an AI coding assistant under the authors' direction, and providing feedback on the design of experimental protocols. We did not use AI tools to generate data, to develop the theoretical model or conceptual framework, to formulate or prove mathematical claims, for translation, or for qualitative analysis. We also used AI tools for tasks with recommended disclosure: drafting and editing parts of the text for readability, suggesting the structure of sections, creating and modifying figures, identifying and summarizing related literature, and formatting references. All AI-generated code was reviewed by the authors and is covered by an automated test suite; every reported number was produced by that code from logged runs and checked against the logs by at least one author; every sentence, claim and citation was reviewed and edited by the authors. The authors take full responsibility for the content of this paper, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite message passing for relational deep learning. In *Forty-second International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8296–8312, 2025a. URL https://openreview.net/forum?id=XXh3zmw2Uy.

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

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

Dongwon Choi, Sunwoo Kim, Juyeon Kim, Kyungho Kim, Geon Lee, Shinhwan Kang, Myunghwan Kim, and Kijung Shin. RDB2G-Bench: A comprehensive benchmark for automatic graph modeling of relational databases. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=ZbcUIxACQE. Datasets and Benchmarks Track.

Tamara Cucumides and Floris Geerts. From features to structure: Task-aware graph construction for relational and tabular learning with GNNs. In *VLDB 2025 Workshop: Tabular Data Analysis (TaDA)*, 2025. URL https://www.vldb.org/2025/Workshops/VLDB-Workshops-2025/TaDA/TaDA25_5.pdf.

Milan Cvitkovic. Supervised learning on relational databases with graph neural networks, 2020. URL https://arxiv.org/abs/2002.02046. arXiv:2002.02046.

Valentin Dinu and Prakash Nadkarni. Guidelines for the effective use of entity-attribute-value modeling for biomedical databases. *International Journal of Medical Informatics*, 76(11–12):769–779, 2007. doi: 10.1016/j.ijmedinf.2006.09.023.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational Graph Transformer. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2d3j6bt21A.

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

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 *Forty-first International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13592–13607, 2024. URL https://openreview.net/forum?id=BIMSHniyCP.

Matthias Fey, Vid Kocijan, Federico López, Jan Eric Lenssen, and Jure Leskovec. KumoRFM: A foundation model for in-context learning on relational data. Technical report, Kumo.AI, 2025. URL http://web.archive.org/web/20260702201348/https://kumo.ai/research/kumo_relational_foundation_model.pdf.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Mihir Manium, Shi Bin Hoo, Magnus Bühler, Anurag Garg, Dominik Safaric, Jake Robertson, Benjamin Jäger, Simone Alessi, Adrian Hayler, Vladyslav Moroshan, Lennart Purucker, Philipp Singer, Alan Arazi, Julien Siems, Jan Hendrik Metzen, Georg Grab, Nick Erickson, Siyuan Guo, Eliott Kalfon, Simon Bing, David Salinas, Clara Cornu, Lilly Charlotte Wehrhahn, Diana Kriuchkova, Kursat Kaya, Lydia Sidhoum, Marie Salmon, Jerry Chen, Madelon Hulsebos, Yann LeCun, Samuel Müller, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. TabPFN-3: Technical report, 2026. URL https://arxiv.org/abs/2605.13986. arXiv:2605.13986.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, pp. 1024–1034, 2017.

Adrian Hayler, Klemens Flöge, Alan Arazi, Rishabh Ranjan, Jure Leskovec, Felix Birkel, Brendan Roof, Anurag Garg, Kristina Collins, Lydia Sidhoum, Jonas Kübler, Siyuan Guo, Oscar Key, Jan Hendrik Metzen, Rylee Grace, David Salinas, Arthur Cahu, Simon Bing, Benjamin Jäger, Tuana Çelik, Mihir Manium, Vitor Monteiro, Jake Robertson, Jerry Chen, Eliott Kalfon, Tomás Pereda, Lilly Wehrhahn, Dominik Safaric, Tobias Schroeder, Georg Grab, Diana Kriuchkova, Clara Cornu, Philipp Singer, Nick Erickson, Vahid Balazadeh, Marie Salmon, Simone Alessi, Kürşat Kaya, Philipp Jund, Léo Grinsztajn, Yann LeCun, Bernhard Schölkopf, Madelon Hulsebos, Lennart Purucker, Sauraj Gambhir, Frank Hutter, and Noah Hollmann. Advancing open and reproducible relational learning: RelArena-α, TabPFN-Rel and RPI, 2026. URL https://arxiv.org/abs/2608.16319. arXiv:2608.16319.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cp5PvcI6w8_.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025. doi: 10.1038/s41586-024-08328-6.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous Graph Transformer. In *Proceedings of The Web Conference 2020 (WWW '20)*, pp. 2704–2710, 2020. doi: 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, 2026. URL https://arxiv.org/abs/2604.12596. arXiv:2604.12596.

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

Vignesh Kothapalli, Rishabh Ranjan, Valter Hudovernik, Vijay Prakash Dwivedi, Johannes Hoffart, Carlos Guestrin, and Jure Leskovec. PluRel: Synthetic data unlocks scaling laws for relational

foundation models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=rM9r9qyRSA.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34670–34692. PMLR, 2024.

Prakash M. Nadkarni, Luis Marenco, Roland Chen, Emmanouil Skoufos, Gordon Shepherd, and Perry Miller. Organization of heterogeneous scientific data using the EAV/CR representation. *Journal of the American Medical Informatics Association*, 6(6):478–493, 1999. doi: 10.1136/jamia.1999.0060478.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Camille Pradel. Reification as a transferable vocabulary: Zero-shot link prediction with vanilla GNNs, 2026. URL https://arxiv.org/abs/2609.11347. arXiv:2609.11347.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos I. Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational Transformer: Toward zero-shot foundation models for relational data. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=rpPtgMC5s9.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410.

Agapi Rissaki, Ilias Fountalis, Wolfgang Gatterbauer, and Benny Kimelfeld. Database views as explanations for relational deep learning. *Proceedings of the VLDB Endowment*, 19(7):1643–1658, 2026. doi: 10.14778/3801059.3801075.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. RelBench: A benchmark for deep learning on relational databases. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://openreview.net/forum?id=WEFxOm3Aez. Datasets and Benchmarks Track.

Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st International Conference on World Wide Web*, pp. 649–658. ACM, 2012. doi: 10.1145/2187836.2187924.

Stanford Tabular and Relational Project. RelBench leaderboard. https://star-project.stanford.edu/relbench/leaderboard/, 2026. Accessed 24 September 2026.

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

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788, 2020.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pp. 2022–2032. ACM, 2019. doi: 10.1145/3308558.3313562.

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*, volume 267 of *Proceedings of Machine Learning Research*, pp. 64604–64627, 2025. URL https://openreview.net/forum?id=TxeCxVb3cL.

Yanbo Wang, Jiaxuan You, Chuan Shi, and Muhan Zhang. Relational in-context learning via synthetic pre-training with structural prior. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=kiSm94YKor.

Fang Wu, Vijay Prakash Dwivedi, and Jure Leskovec. Large language models are good relational learners. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7835–7854, 2025. doi: 10.18653/v1/2025.acl-long.386. URL https://aclanthology.org/2025.acl-long.386/.

Marek Wydmuch, Łukasz Borchmann, and Filip Graliński. Tackling prediction tasks in relational databases with LLMs, 2024. URL https://arxiv.org/abs/2411.11829. arXiv:2411.11829.

Linjie Xu, Yanlin Zhang, Quan Gan, Minjie Wang, and David Wipf. No need to train your RDB foundation model. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=hrtEiSftwk.

Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. Heterogeneous graph structure learning for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4697–4705, 2021. doi: 10.1609/aaai.v35i5.16600.

A Attributes of the Universal Schema

Figure 1(b) shows the universal schema $\mathcal{U}$ of Section 3.2, and Table 2 lists its attributes with their domains, including the attributes that our implementation adds and those that the ablations of Section 4.4 mask.

B The Mapping in Full

This appendix gives the full mapping of Section 3.2, except for the normalization of numbers. Its input is a schema $\mathcal{S}$ and a database D over $\mathcal{S}$. Steps 1 to 3 map the schema, and Steps 4 to 6 map the tuples, the foreign-key references and the values of D. No step creates tuples of TARGET: they come from a task (Section 3.3). Below, R and R' are relation names of $\mathcal{S}$, and the Q tuple of an object x is the tuple of the relation name Q that x becomes. The values of id come from a fixed rule that never repeats an identifier within a relation, and a step that sets a time to $-\infty$ leaves time missing.

1. Every relation name R becomes a tuple of TABLES whose name is the string R.
2. Every attribute A of R becomes a tuple of COLUMNS that references the TABLES tuple of R. Its name is the string A, its datatype is the name of $\mathrm{dom}(A)$, and iskey and istime state whether A is the key of R and whether it is the time attribute of R.
3. Every FK $R[C] \sqsubseteq \mathrm{key}(R')$ becomes a tuple of FKS whose from references the COLUMNS tuple of C and whose to references the COLUMNS tuple of $\mathrm{key}(R')$.
4. Every tuple t of $D(R)$ becomes a tuple $\mathrm{row}(t)$ of ROWS that references the TABLES tuple of R and whose time is $\mathrm{time}(t)$.
5. Every foreign-key reference (t, t') for an FK $R[C] \sqsubseteq \mathrm{key}(R')$ becomes a tuple of REFS whose from, to and fk reference $\mathrm{row}(t)$, $\mathrm{row}(t')$ and the FKS tuple of the FK, respectively. Its time is the later of $\mathrm{time}(t)$ and $\mathrm{time}(t')$, since both tuples are present only from then on.

Table 2: Attributes of the universal schema $\mathcal{U}$ (Section 3.2) and their domains. Keys are underlined, and $A \to R'$ denotes a foreign key that references the key of R'. Attributes in typewriter font are added by our implementation, under the names it uses. The implementation names the relations `tables`, `cols`, `fks`, `rows`, `fk_entities`, `str_cells` and `bool_cells`, splits NUMCELLS into `int_cells` and `float_cells`, and does not store TARGET tuples (Appendix D.1). The schema-names and schema-statistics ablations of Section 4.4 mask the attributes marked ‡ and †, respectively. The domain of value is the strings in STRCELLS, the real numbers in NUMCELLS, and the Booleans in BOOLCELLS; the domain of label is the prediction domain $\mathcal{Y}$.

Relation	Attribute	Domain	Holds
TABLES	id	integers	an identifier
	name‡	strings	the relation name R
	`n_cols`†	integers	the number of attributes of R
	`n_numeric_cols`†	integers	the number of numeric attributes of R
	`n_text_cols`†	integers	the number of string attributes of R
	`n_bool_cols`†	integers	the number of Boolean attributes of R
	`n_time_cols`†	integers	the number of timestamp attributes of R
	`n_fk_out`†	integers	the number of FKs of R
	`n_fk_in`†	integers	the number of FKs that reference the key of R
	`has_time_col`†	Booleans	whether R has a time attribute
COLUMNS	id	integers	an identifier
	table $\to$ TABLES	integers	the relation name R of the attribute A
	name‡	strings	the attribute A
	`full_column_name`‡	strings	the string "$R.A$"
	datatype†	strings	the name of $\mathrm{dom}(A)$
	iskey†	Booleans	whether A is the key of R
	istime†	Booleans	whether A is the time attribute of R
	`is_fk`†	Booleans	whether A is a foreign-key attribute of R
FKS	id	integers	an identifier
	from $\to$ COLUMNS	integers	the attribute C of the FK $R[C] \sqsubseteq \mathrm{key}(R')$
	to $\to$ COLUMNS	integers	the key attribute of R'
	`full_fk_relation`‡	strings	the FK as a string of the form `R.C->R'.K`
	`fk_is_self_reference`†	Booleans	whether R and R' are the same relation name
	`fk_child_out_degree`†	integers	the number of FKs of R
	`fk_parent_in_degree`†	integers	the number of FKs referencing the key of R'
	`fk_parent_out_degree`†	integers	the number of FKs of R'
ROWS	id	integers	an identifier
	table $\to$ TABLES	integers	the relation name of the tuple t
	time	timestamps	$\mathrm{time}(t)$
REFS	id	integers	an identifier
	from $\to$ ROWS	integers	the tuple t of the reference (t, t')
	to $\to$ ROWS	integers	the tuple t'
	fk $\to$ FKS	integers	the FK of the reference
	time	timestamps	the later of $\mathrm{time}(t)$ and $\mathrm{time}(t')$
STRCELLS, NUMCELLS, BOOLCELLS	id	integers	an identifier
	row $\to$ ROWS	integers	the tuple t of the value $t[A]$
	column $\to$ COLUMNS	integers	the attribute A
	time	timestamps	$\mathrm{time}(t)$
	value	see caption	the value $t[A]$
TARGET	row $\to$ ROWS	integers	the tuple that the prediction is about
	time	timestamps	the time of the prediction
	task	strings	the name of T; in our experiments, the task text of Appendix D.2
	label	$\mathcal{Y}$	the label $t[L]$ of the tuple, when it is known

6. Every value $t[A]$ that is not missing, where A is a data attribute of R, becomes a tuple of STRCELLS if A is a string attribute, of BOOLCELLS if it is a Boolean attribute, and of NUMCELLS otherwise. This tuple references $\mathrm{row}(t)$ and the COLUMNS tuple of A, its value is $t[A]$, and its time is $\mathrm{time}(t)$. Timestamps are the exception: the time attribute of R gets no such tuples, because $\mathrm{row}(t)$ already holds its value, and the value of every other timestamp is its number of seconds from a fixed origin.

In Steps 1 to 3, our implementation also fills the attributes in typewriter font in Table 2, all of which it computes from $\mathcal{S}$ alone. The result $\Phi(D)$ is a database over $\mathcal{U}$: identifiers are distinct, and every tuple that a step references comes from an earlier step. The mapping stores no key or foreign-key values, since they only identify tuples and REFS records every foreign-key reference. Given $\mathcal{S}$, one can therefore rebuild D from $\Phi(D)$, with the identifiers of ROWS in place of its key values.

C THE MAPPING ON A SMALL DATABASE

Table 3 applies the mapping of Section 3.2 to a small database with a churn prediction task, and Figure 4 draws the result as a graph.

Table 3: A small database with a churn prediction task (left) and the database over $\mathcal{U}$ that the mapping of Section 3.2 creates from it (right). On the left, an underlined attribute is a key, Orders and the target relation Churn reference Customers through cid, and date is the time attribute of both. The label attribute of Churn is label, and its value in the second tuple is missing: it is the value to predict. On the right, tuples follow the attribute order of Section 3.2, and TARGET has one tuple per Churn tuple, as in Section 3.3. For brevity, we omit the tuples that the mapping also creates for Churn itself. The identifiers of ROWS take the place of key values, each reference from an order becomes a REFS tuple, and the time attribute becomes the time of the ROWS tuple and its cells. NULL marks a missing value, such as the time of the customer, since Customers has no time attribute, and NUMCELLS shows the numbers before normalization.

Source database D	Mapped database $\Phi(D)$
Customers($\underline{\mathrm{cid}}$, name) (c1, Noa Levi) Orders($\underline{\mathrm{oid}}$, cid, amount, date) (o1, c1, 259.90, 2024-03-01) (o2, c1, 49.50, 2024-05-14) Churn(cid, date, label) (the target relation) (c1, 2024-04-01, false) (c1, 2024-07-01, NULL)	TABLES: (1, Customers), (2, Orders) COLUMNS: (1, 1, cid, string, true, false), (2, 1, name, string, false, false), (3, 2, oid, string, true, false), (4, 2, cid, string, false, false), (5, 2, amount, real number, false, false), (6, 2, date, timestamp, false, true) FKS: (1, 4, 1) ROWS: (1, 1, NULL), (2, 2, 2024-03-01), (3, 2, 2024-05-14) REFS: (1, 2, 1, 1, 2024-03-01), (2, 3, 1, 1, 2024-05-14) STRCELLS: (1, 1, 2, NULL, Noa Levi) NUMCELLS: (1, 2, 5, 2024-03-01, 259.90), (2, 3, 5, 2024-05-14, 49.50) BOOLCELLS: no tuples TARGET: (1, 2024-04-01, Churn, false), (1, 2024-07-01, Churn, NULL)

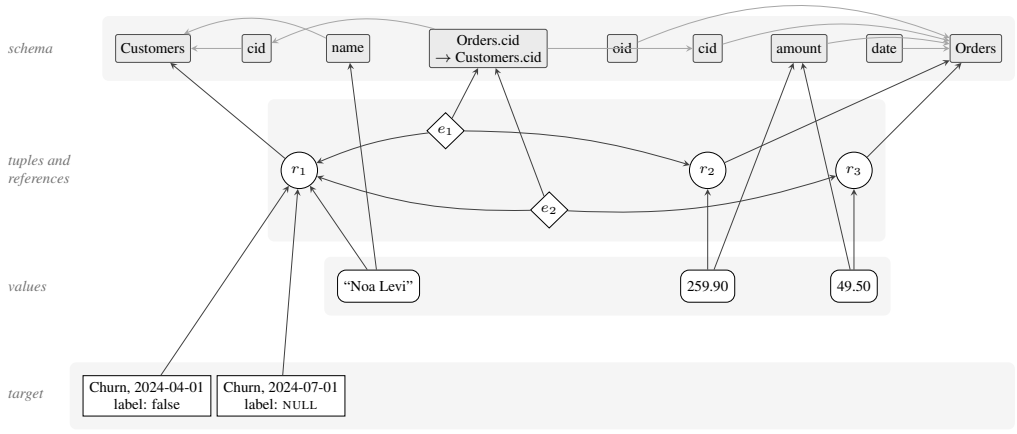


Figure 4: The mapped database of Table 3 as a graph: each tuple is a node, and each foreign-key reference is an edge, drawn in one direction. Gray boxes are the tuples of TABLES, COLUMNS and FKS, circles the ROWS tuples, diamonds the REFS tuples, rounded boxes the cells, and the boxes at the bottom the TARGET tuples. No key value appears in the graph: the value c1 that both orders hold in cid becomes the nodes e_1 and e_2, and the names of the source schema appear only as values of TABLES and COLUMNS. Our implementation does not store the TARGET tuples (Appendix D.1): the model predicts the label of the second, and receives the first as a self-label attached to r_1.

D ADDITIONAL EXPERIMENTAL RESULTS

D.1 HYPER-PARAMETERS AND INPUT

Table 4 lists the configuration that the three architectures share in the zero-shot and the fine-tuning experiments. We tune nothing per database: we choose only the learning rate, once per architecture, by the sweep below. The sweep held out Amazon, Stack, and H&M. We left out Trial: its zero-shot transfer is at chance, so its validation score cannot rank learning rates. We swept $\{10^{-4}, 3\cdot 10^{-4}, 10^{-3}, 3\cdot 10^{-3}\}$ for HGT and RelGT and $\{3\cdot 10^{-4}, 10^{-3}, 3\cdot 10^{-3}, 10^{-2}\}$ for GraphSAGE, and every zero-shot cell uses the winner: $3 \cdot 10^{-3}$ for GraphSAGE and $3 \cdot 10^{-4}$ for HGT and RelGT. Each architecture uses its rate for every target database, so the rate has seen the validation labels of Amazon, Stack, and H&M.

Implementation of the input. The mapped database holds no TARGET tuples: our implementation passes their content to the model directly. It appends the text of task to the features of every ROWS tuple of the sampled subgraph, and it attaches the self-labels to the ROWS tuple of the entity, as nodes that carry their time and label.

D.2 TASK DESCRIPTIONS

Table 5 lists the value of task for every task: the RelBench task name and a one-sentence description. The model embeds it with the encoder of string values, all-MiniLM-L6-v2 (Wang et al., 2020; Reimers & Gurevych, 2019).

D.3 BASELINES, METRICS AND DATA VERSIONS

All runs use version 2.1.2 of the `relbench` package, which includes pull request 357 (`https://github.com/snap-stanford/relbench/pull/357`), merged on 1 February 2026. That change fixes the Event user-ignore labels affected by the temporal leakage that RT describes (Ranjan et al., 2026, §4.1), and it removes the `AcceptedAnswerId` column from the posts table of Stack. RT's and Griffin's published Stack results predate it. Their scores come from Tables 14 and 15 of the TabPFN-3 report (Grinsztajn et al., 2026), which reuses previously reported scores for both models and marks RT as likely following a different evaluation protocol that overestimates performance.

Table 4: Configuration of the runs. One configuration per architecture serves every database; only the learning rate was chosen per architecture, by the sweep described in Appendix D.1.

	Zero-shot pretraining (Section 4.2)	Fine-tuning (Section 4.3)
Hidden dimension	128	as pretrained
Attention heads (HGT, RelGT)	4	as pretrained
Sampled neighbors per hop	20, 10, 5, 5 (four hops)	same
Sampling from schema tuples	never expanded: a TABLES, COLUMNS or FKS tuple is reached but draws no data tuples	same
RelGT tokens per prediction / local layers	512 / 2	same
Text encoder	all-MiniLM-L6-v2, for string values and the task text	same
TARGET tuples per step	512, drawn from all training tasks in proportion to their training-split sizes	256, from one task
Optimizer steps	25,000	32,768 per run
Optimizer	AdamW, weight decay 0.1, gradient norm clipped at 1.0	AdamW, weight decay 0, clipping at 1.0
Peak learning rate	GraphSAGE $3 \cdot 10^{-3}$; HGT and RelGT $3 \cdot 10^{-4}$	the architecture's pretraining rate
Schedule	one-cycle: linear warm-up over the first 20% of the steps, then linear decay	one-cycle per run, same shape
Precision	bfloat16 mixed precision	same
Evaluation	every 1,000 steps and at steps 1, 2, 4, …; each task scored at its own best-validation checkpoint	at steps 0, 1, 2, 4, …, 2^{15}, on 40 batches of validation and of test rows; test score at the best-validation step
Regression labels	quantile transformation to the standard normal, fitted on the training split; predictions mapped back before scoring	same
Random seed	0	0

Table 5: The task text of every task: the RelBench task name and the one-sentence description that the model receives as the task attribute of TARGET (Section 4.1), verbatim.

Database	Task	Description
Amazon	user-churn	For each user, predict 1 if the customer does not review any product in the next 3 months, and 0 otherwise.
	item-churn	For each product, predict 1 if the product does not receive any reviews in the next 3 months.
	user-ltv	For each user, predict their life-time value in the next 3 months: the sum of the prices of the products they review (a dollar amount).
	item-ltv	For each product, predict its life-time value in the next 3 months: the number of times it is purchased multiplied by its price (a dollar amount).
Avito	user-visits	Predict whether each customer will visit more than one Ad in the next 4 days.
	user-clicks	Predict whether each customer will click on more than one Ads in the next 4 days.
	ad-ctr	Assuming the ad will be clicked in the next 4 days, predict its click-through rate, a fraction between 0 and 1.
Event	user-repeat	Predict whether a user will attend an event (by responding yes or maybe) in the next 7 days if they have already attended an event in the last 14 days.
	user-ignore	Predict whether a user will ignore more than 2 event invitations in the next 7 days.
	user-attendance	For each user, predict the number of events they will attend in the next 7 days (a count).
F1	driver-dnf	For each driver predict if they will DNF (did not finish) a race in the next 1 month.
	driver-top3	For each driver predict if they will qualify in the top-3 for a race in the next 1 month.
	driver-position	For each driver, predict their average finishing position over all races in the next 2 months (a number, lower is better).
H&M	user-churn	Predict the churn for a customer (no transactions) in the next week.
	item-sales	For each article, predict its total sales in the next week: the sum of the prices of its transactions (an amount).
Stack	user-engagement	For each user predict if a user will make any votes, posts, or comments in the next 3 months.
	user-badge	For each user predict if a user will receive a new badge in the next 3 months.
	post-votes	For each question post, predict the number of upvotes it will receive in the next 2 years (a count).
Trial	study-outcome	Predict if the trials will achieve its primary outcome (defined as p-value $<$ 0.05).
	site-success	For each trial site, predict its success rate in the next 1 year, a fraction between 0 and 1.
	study-adverse	For each trial, predict the number of patients affected by severe adverse events or death in the next 1 year (a count).

Table 1 normalizes RT's MAE from Table 15 as we normalize ours. Entity Mean and Entity Median come from RelArena (Hayler et al., 2026).

Why nMAE. RT reports regression as R^2, which a few extreme targets dominate on these tasks. The labels of tasks such as post-votes and the two lifetime-value tasks are heavy-tailed, so a handful of test rows can move R^2 by tens of points for a fixed predictor (dropping the ten largest-error test rows of Amazon item-ltv moved $R^2 \times 100$ by 17 points in our pilot), and R^2 turns negative for predictors that are otherwise reasonable. MAE weighs every row alike, and RelArena (Hayler et al., 2026) reports it. Dividing each task's MAE by the standard deviation of its training labels puts all tasks on one scale, so means over tasks are meaningful, as on the RelBench leaderboard (Stanford Tabular and Relational Project, 2026).

D.4 Fine-Tuning Curves

Each panel shows one task's test score against fine-tuning steps for three runs of the same architecture: the universal schema from the pretrained model, the universal schema from random initialization, and the original schema from random initialization. Every run follows the protocol of Section D.6 and is evaluated at steps $0, 1, 2, 4, \ldots, 2^{15}$ on a fixed subsample of 10,240 test rows. RT's published curve (Ranjan et al., 2026) is overlaid where RT reports one; those curves start, like ours, from a model pretrained without the target database. RT publishes no MAE curves, so the regression panels carry no overlay. Figures 5 and 6 show RelGT, Figures 7 and 8 HGT, and Figures 9 and 10 GraphSAGE.

D.4.1 RelGT

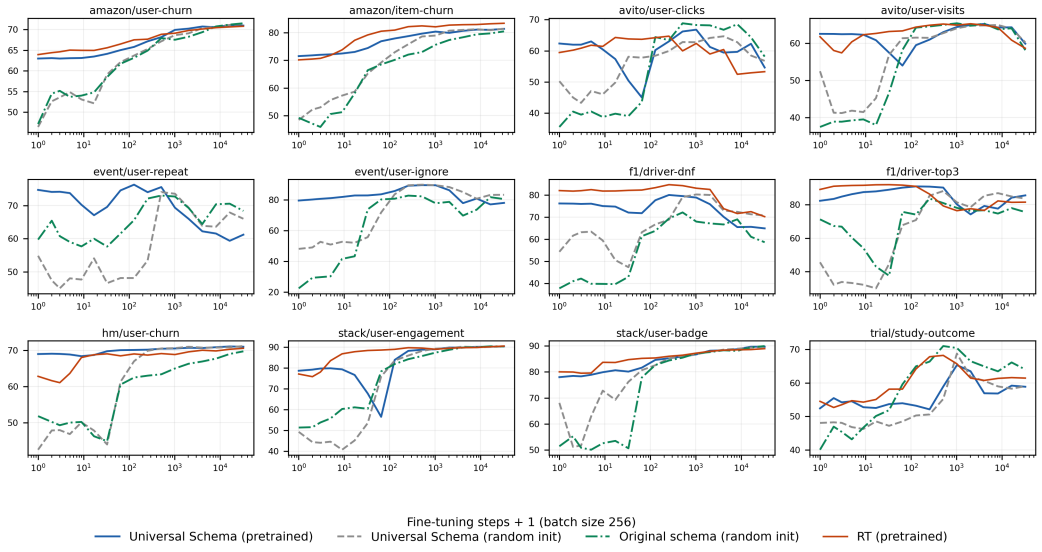


Figure 5: RelGT, the twelve classification tasks (test AUROC, higher is better).

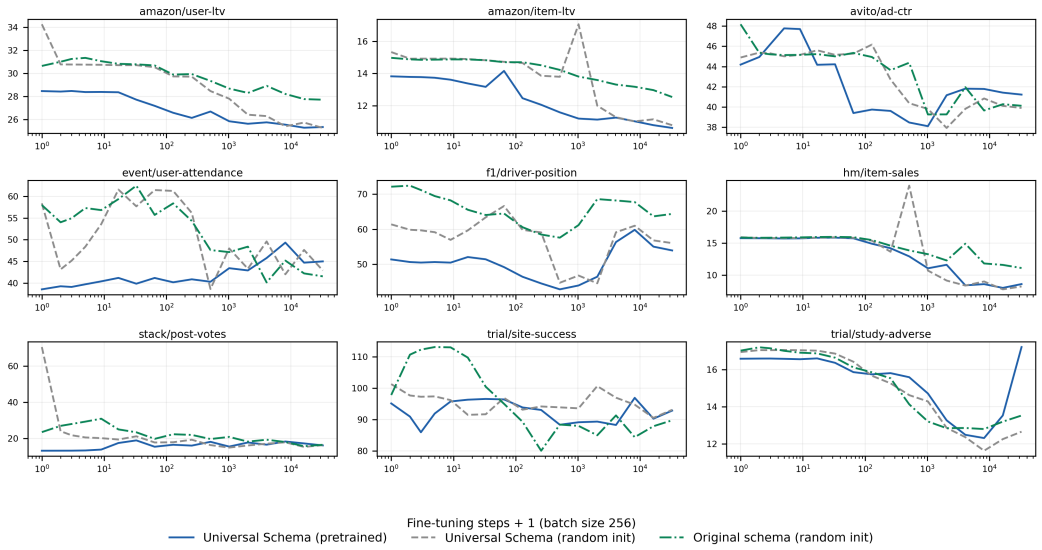


Figure 6: RelGT, the nine regression tasks (test nMAE, lower is better).

D.4.2 HGT

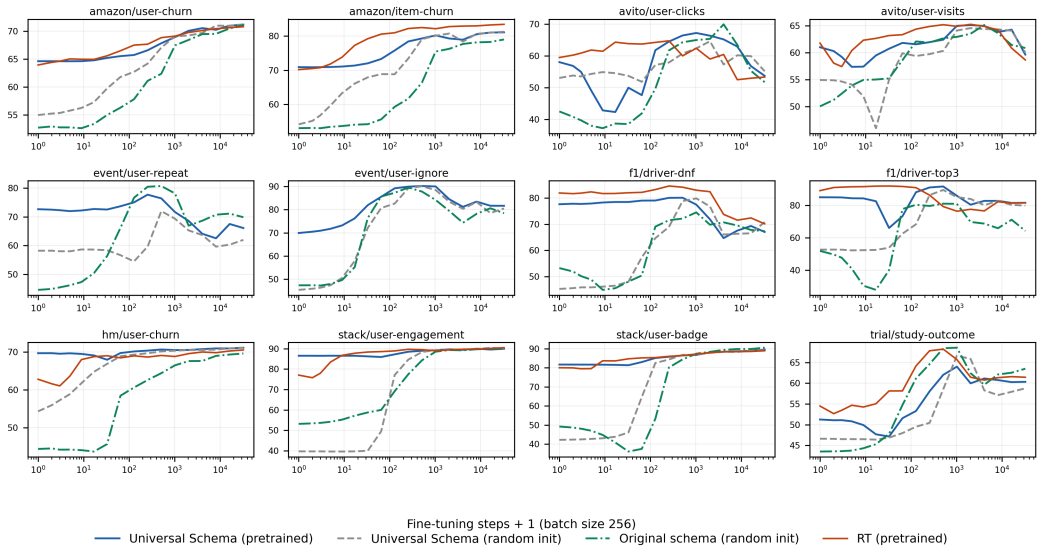


Figure 7: HGT, the twelve classification tasks (test AUROC, higher is better).

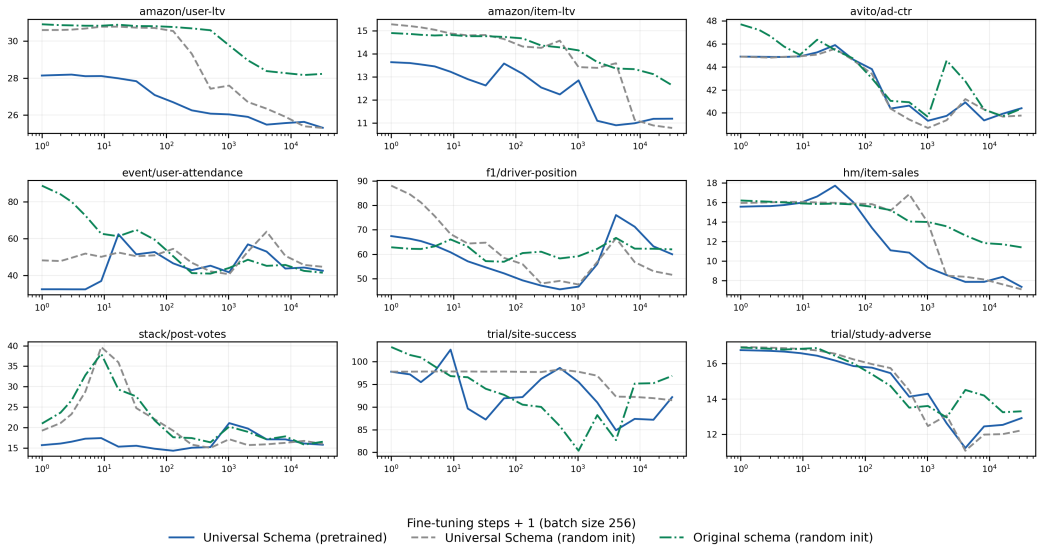


Figure 8: HGT, the nine regression tasks (test nMAE, lower is better).

D.4.3 GRAPHSAGE

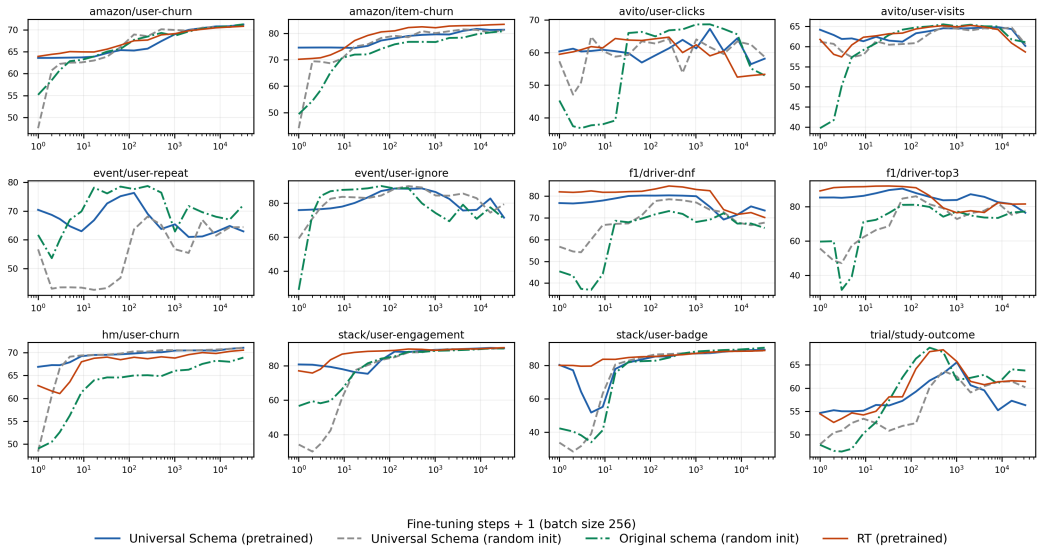


Figure 9: GraphSAGE, the twelve classification tasks (test AUROC, higher is better).

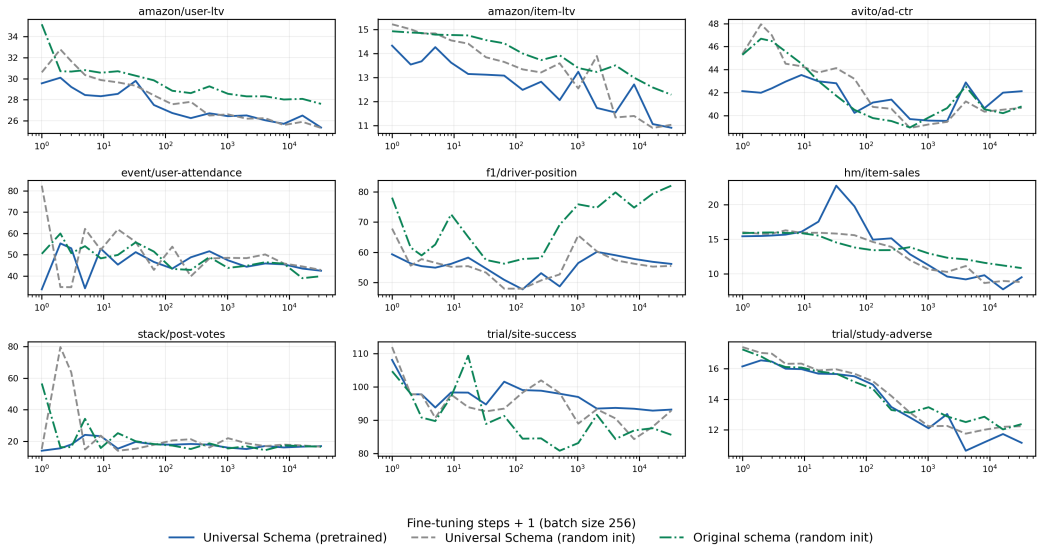


Figure 10: GraphSAGE, the nine regression tasks (test nMAE, lower is better).

D.5 Fine-Tuning at a Fixed Label Budget

The tables of Section D.6 report scores at the end of the full fine-tuning budget, where the two initializations converge. The budget at which they differ is much smaller. Here we hold the label budget fixed and compare the same two initializations before any gradient update and after 128 and 256 optimizer steps, which at batch size 256 is 32,768 and 65,536 labels.

The six tables below share one set of columns, one group per budget. *Pretrained* is our pretrained model on the universal schema and *Scratch* the same architecture from random initialization on the original RelBench schema, both under the protocol of Section D.6; nMAE is as defined in Section D.3. The scores are read from the curves of Section D.4, so they are on the fixed 10,240-row test subsample rather than the full test split used in Section D.6, and the two sets of numbers are not directly comparable. We report both budgets for every architecture: the architectures are not caught at the same point, and where each is caught is itself a result.

D.5.1 RelGT

Table 6: RelGT: test AUROC at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-churn	47.1	**63.0**	63.2	**65.8**	64.9	**67.2**
	item-churn	49.2	**71.5**	70.2	**77.9**	72.1	**78.8**
Avito	user-clicks	35.6	**62.4**	**64.4**	60.2	**63.4**	63.0
	user-visits	37.5	**62.5**	**64.2**	59.5	**64.5**	61.0
Event	user-repeat	59.7	**74.8**	65.6	**76.4**	72.2	**74.1**
	user-ignore	22.5	**79.6**	80.8	**85.8**	82.8	**89.2**
F1	driver-dnf	37.7	**76.1**	63.7	**77.6**	69.3	**80.0**
	driver-top3	71.2	**82.3**	74.1	**90.9**	83.6	**90.8**
H&M	user-churn	51.8	**68.9**	62.5	**70.1**	63.0	**70.2**
Stack	user-engagement	51.3	**78.6**	81.8	**83.9**	84.2	**88.2**
	user-badge	51.5	**78.0**	82.7	**84.6**	84.4	**85.2**
Trial	study-outcome	40.0	**52.4**	**64.9**	53.2	**66.3**	52.1
Mean (12 tasks)		46.3	**70.8**	69.8	**73.8**	72.6	**75.0**

Table 7: RelGT: test nMAE (%, lower is better) at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-ltv	30.6	**28.5**	29.9	**26.6**	29.9	**26.1**
	item-ltv	15.0	**13.8**	14.7	**12.5**	14.5	**12.1**
Avito	ad-ctr	48.2	**44.2**	44.9	**39.7**	43.6	**39.6**
Event	user-attendance	58.0	**38.6**	58.4	**40.2**	54.3	**40.9**
F1	driver-position	72.1	**51.4**	60.6	**46.4**	58.5	**44.5**
H&M	item-sales	**15.8**	**15.8**	15.5	**14.9**	14.6	**14.2**
Stack	post-votes	23.4	**13.1**	22.3	**16.5**	21.9	**16.0**
Trial	site-success	97.9	**95.2**	**89.3**	93.9	**80.1**	93.1
	study-adverse	17.0	**16.6**	15.9	**15.7**	**15.5**	15.8
Mean (9 tasks)		42.0	**35.2**	39.0	**34.0**	37.0	**33.6**

D.5.2 HGT

Table 8: HGT: test AUROC at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-churn	52.7	**64.6**	57.8	**65.7**	61.1	**66.6**
	item-churn	53.0	**70.9**	59.3	**75.8**	61.7	**78.4**
Avito	user-clicks	42.5	**58.0**	49.6	**61.8**	61.7	**64.5**
	user-visits	50.0	**61.0**	**62.1**	61.5	61.9	**62.0**
Event	user-repeat	44.6	**72.7**	**76.7**	75.1	**80.5**	77.7
	user-ignore	47.4	**69.9**	87.3	**89.2**	89.6	**90.0**
F1	driver-dnf	53.2	**77.7**	69.1	**79.1**	71.5	**80.1**
	driver-top3	51.8	**85.0**	80.4	**88.1**	79.6	**91.1**
H&M	user-churn	44.4	**69.7**	60.6	**70.1**	62.6	**70.4**
Stack	user-engagement	53.2	**86.6**	69.2	**87.2**	77.5	**88.5**
	user-badge	49.2	**81.7**	53.2	**85.1**	80.2	**85.8**
Trial	study-outcome	43.5	**51.2**	**61.1**	53.3	**64.6**	58.0
Mean (12 tasks)		48.8	**70.7**	65.5	**74.3**	71.0	**76.1**

Table 9: HGT: test nMAE (%, lower is better) at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-ltv	30.9	**28.1**	30.8	**26.7**	30.7	**26.3**
	item-ltv	14.9	**13.6**	14.7	**13.1**	14.4	**12.5**
Avito	ad-ctr	47.7	**44.9**	**43.0**	43.8	41.0	**40.4**
Event	user-attendance	88.8	**32.4**	50.4	**46.5**	**41.2**	42.8
F1	driver-position	**62.8**	67.4	60.4	**49.3**	61.0	**47.1**
H&M	item-sales	16.2	**15.6**	15.6	**13.4**	15.2	**11.1**
Stack	post-votes	20.9	**15.7**	17.6	**14.3**	17.4	**15.0**
Trial	site-success	103.2	**97.8**	**90.5**	92.2	**90.0**	96.2
	study-adverse	16.9	**16.8**	**15.4**	15.8	**14.7**	15.5
Mean (9 tasks)		44.7	**36.9**	37.6	**35.0**	36.2	**34.1**

D.5.3 GRAPHSAGE

Table 10: SAGE: test AUROC at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-churn	55.2	**63.6**	**67.7**	65.3	**68.5**	65.7
	item-churn	49.4	**74.6**	75.8	**78.1**	76.8	**79.0**
Avito	user-clicks	45.3	**60.3**	**65.0**	59.1	**66.8**	61.3
	user-visits	39.7	**64.2**	**64.8**	63.3	**65.0**	63.8
Event	user-repeat	61.7	**70.4**	**77.6**	76.3	**78.7**	69.0
	user-ignore	29.0	**75.9**	**88.6**	**88.6**	**88.9**	88.5
F1	driver-dnf	45.4	**76.8**	71.6	**80.2**	73.1	**80.3**
	driver-top3	59.6	**85.2**	81.1	**87.8**	79.8	**85.6**
H&M	user-churn	49.0	**66.9**	65.0	**69.8**	65.1	**70.0**
Stack	user-engagement	56.6	**80.8**	85.4	**88.2**	**88.1**	88.0
	user-badge	42.2	**80.3**	82.8	**85.4**	84.6	**85.3**
Trial	study-outcome	47.9	**54.7**	**66.3**	59.3	**68.7**	61.6
Mean (12 tasks)		48.4	**71.1**	74.3	**75.1**	**75.3**	74.9

Table 11: SAGE: test nMAE (%, lower is better) at increasing fine-tuning budgets, for the two initializations. Bold: better initialization at that budget.

		0 steps		128 steps		256 steps	
Database	Task	Scratch	Pretrained	Scratch	Pretrained	Scratch	Pretrained
Amazon	user-ltv	35.2	**29.6**	28.9	**26.7**	28.6	**26.3**
	item-ltv	14.9	**14.3**	14.0	**12.5**	13.7	**12.8**
Avito	ad-ctr	45.3	**42.1**	**39.8**	41.1	**39.5**	41.4
Event	user-attendance	50.4	**33.7**	**43.3**	43.5	**42.8**	48.8
F1	driver-position	78.1	**59.4**	57.8	**47.8**	58.1	**53.1**
H&M	item-sales	15.9	**15.4**	**13.4**	14.9	**13.4**	15.1
Stack	post-votes	56.6	**14.0**	**17.3**	17.8	**15.1**	18.3
Trial	site-success	**104.7**	108.1	**84.4**	99.0	**84.5**	98.8
	study-adverse	17.3	**16.2**	**14.7**	15.0	**13.3**	13.5
Mean (9 tasks)		46.5	**37.0**	**34.8**	35.4	**34.4**	36.5

At zero steps the gap is the zero-shot transfer already reported: the pretrained model averages 70.8, 70.7 and 71.1 AUROC for RelGT, HGT and GraphSAGE, ahead on all twelve classification tasks, while the randomly initialized model is near chance at 46.3, 48.8 and 48.4. Fine-tuning narrows that gap quickly, and by different amounts. At 128 steps the pretrained model still leads on classification for all three, by 4.0, 8.8 and 0.8 AUROC points. At 256 steps RelGT and HGT still lead, by 2.4 and 5.1 points, while GraphSAGE has been caught and passed, by 0.5. Regression follows the same order: RelGT and HGT lead at both budgets, by 5.0 and 2.6 nMAE points at 128 and 3.4 and 2.1 at 256, while GraphSAGE is already behind at 128, by 0.5, and further behind at 256, by 2.1.

The pretrained model also improves on its own zero-shot score at every budget and on every architecture, by 3.0, 3.6 and 4.0 AUROC points at 128 steps and by 4.1, 5.3 and 3.7 at 256, so the two initializations are not merely converging on a shared ceiling.

D.6 Fully Fine-Tuned Tables

We fine-tune on one task at a time under the protocol of RT (Ranjan et al., 2026): 32,768 optimizer steps at batch size 256, AdamW with no weight decay, gradients clipped at 1.0 and a one-cycle schedule whose peak is the architecture's zero-shot learning rate (Table 4), reached by 20% linear warm-up and then annealed over the rest of the budget, evaluating at steps $0, 1, 2, 4, \ldots, 2^{15}$ and reporting the test score at the best-validation checkpoint.

The six tables below share one set of columns. *Pretrained* fine-tunes our pretrained model on the universal schema; the two *scratch* columns train the same architecture from random initialization, on the universal schema and on the original RelBench schema, the latter being ordinary schema-specific supervised learning, which reads neither the task text nor the self-labels. The published columns are the entity baselines, and on classification, the pretrained-then-fine-tuned results of Griffin and RT as published at Table 10 of RT (Ranjan et al., 2026). Since it reports regression as R^2, neither model appears in the regression tables. RT and Griffin report ten of the twelve classification tasks, omitting both rel-event tasks, so each classification table carries two mean rows: one over all twelve tasks, where those two columns are dashed, and one over the ten they share.

D.6.1 RelGT

Table 12: RelGT: supervised fine-tuning AUROC (%, higher is better) on RelBench V1 entity classification. Bold: best per task. Dashes: not reported.

		Published baselines			This work, RelGT		
Database	Task	Entity Mean	Griffin	RT	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-churn	63.4	69.4	**70.8**	70.6	**70.8**	70.7
	item-churn	72.9	79.9	**83.4**	81.4	81.4	80.2
Avito	user-clicks	50.4	64.7	59.0	58.5	62.5	**67.6**
	user-visits	60.3	62.6	65.2	65.2	65.3	**65.9**
Event	user-repeat	**75.2**	–	–	74.7	74.4	70.8
	user-ignore	84.0	–	–	85.8	**89.9**	78.5
F1	driver-dnf	69.9	66.7	**84.2**	80.2	79.9	68.3
	driver-top3	55.7	78.7	**91.9**	90.8	87.8	79.1
H&M	user-churn	64.8	68.0	70.5	**70.8**	**70.8**	69.6
Stack	user-engagement	82.7	90.4	90.2	**90.9**	90.8	**90.9**
	user-badge	78.9	87.0	88.7	**88.9**	88.8	88.5
Trial	study-outcome	50.0	64.6	68.2	64.9	69.1	**70.3**
Mean AUROC (12 tasks)		67.3	–	–	76.9	**77.6**	75.0
Mean AUROC (10 tasks)		64.9	73.2	**77.2**	76.2	76.7	75.1
Wins vs. RT		0/10	2/10		3/10	**6/10**	4/10

Table 13: RelGT: supervised fine-tuning nMAE (%, lower is better) on RelBench V1 entity regression. Bold: best per task.

		Published baselines	This work, RelGT		
Database	Task	Entity Median	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-ltv	30.3	**24.2**	**24.2**	26.4
	item-ltv	11.1	**7.2**	7.4	8.8
Avito	ad-ctr	43.1	38.2	**38.1**	39.7
Event	user-attendance	**35.2**	38.6	38.6	41.5
F1	driver-position	58.4	**42.7**	44.4	61.1
H&M	item-sales	15.8	8.3	**7.6**	10.5
Stack	post-votes	13.6	**12.7**	14.5	15.4
Trial	site-success	92.7	88.5	92.6	**84.3**
	study-adverse	17.0	12.3	**11.7**	12.8
Mean nMAE (9 tasks)		35.2	**30.3**	31.0	33.4

D.6.2 HGT

Table 14: HGT: supervised fine-tuning AUROC (%, higher is better) on RelBench V1 entity classification. Bold: best per task. Dashes: not reported.

		Published baselines			This work, HGT		
Database	Task	Entity Mean	Griffin	RT	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-churn	63.4	69.4	**70.8**	70.6	70.7	70.5
	item-churn	72.9	79.9	**83.4**	81.2	81.3	79.2
Avito	user-clicks	50.4	64.7	59.0	62.2	60.6	**67.6**
	user-visits	60.3	62.6	65.2	65.2	64.8	**65.3**
Event	user-repeat	75.2	–	–	**77.7**	72.2	77.3
	user-ignore	84.0	–	–	76.2	**88.5**	80.1
F1	driver-dnf	69.9	66.7	**84.2**	80.1	79.9	68.0
	driver-top3	55.7	78.7	**91.9**	91.7	89.5	80.4
H&M	user-churn	64.8	68.0	70.5	**70.8**	**70.8**	69.8
Stack	user-engagement	82.7	90.4	90.2	**90.7**	90.3	90.5
	user-badge	78.9	87.0	88.7	88.3	88.0	**89.6**
Trial	study-outcome	50.0	64.6	68.2	63.9	66.6	**68.7**
Mean AUROC (12 tasks)		67.3	–	–	76.6	**76.9**	75.6
Mean AUROC (10 tasks)		64.9	73.2	**77.2**	76.5	76.3	75.0
Wins vs. RT		0/10	2/10		3/10	3/10	**5/10**

Table 15: HGT: supervised fine-tuning nMAE (%, lower is better) on RelBench V1 entity regression. Bold: best per task.

		Published baselines	This work, HGT		
Database	Task	Entity Median	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-ltv	30.3	**24.2**	**24.2**	26.7
	item-ltv	11.1	**7.1**	7.2	9.0
Avito	ad-ctr	43.1	39.3	**38.7**	39.7
Event	user-attendance	35.2	**32.4**	40.6	41.4
F1	driver-position	58.4	**45.5**	47.6	59.1
H&M	item-sales	15.8	**6.9**	**6.9**	10.8
Stack	post-votes	**13.6**	14.1	15.1	15.7
Trial	site-success	92.7	87.6	96.3	**87.3**
	study-adverse	17.0	11.3	**11.1**	13.2
Mean nMAE (9 tasks)		35.2	**29.8**	32.0	33.7

D.6.3 GRAPHSAGE

Table 16: SAGE: supervised fine-tuning AUROC (%, higher is better) on RelBench V1 entity classification. Bold: best per task. Dashes: not reported.

		Published baselines			This work, SAGE		
Database	Task	Entity Mean	Griffin	RT	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-churn	63.4	69.4	**70.8**	70.5	70.5	70.7
	item-churn	72.9	79.9	**83.4**	81.4	81.3	80.9
Avito	user-clicks	50.4	64.7	59.0	63.1	62.5	**67.9**
	user-visits	60.3	62.6	65.2	65.2	65.2	**65.8**
Event	user-repeat	75.2	–	–	75.1	64.5	**77.1**
	user-ignore	84.0	–	–	83.4	**89.3**	88.8
F1	driver-dnf	69.9	66.7	**84.2**	80.3	77.9	67.0
	driver-top3	55.7	78.7	**91.9**	87.8	86.0	77.0
H&M	user-churn	64.8	68.0	70.5	70.6	**70.7**	69.0
Stack	user-engagement	82.7	90.4	90.2	**90.8**	**90.8**	90.5
	user-badge	78.9	87.0	88.7	88.5	88.1	**89.5**
Trial	study-outcome	50.0	64.6	68.2	63.3	63.7	**68.5**
Mean AUROC (12 tasks)		67.3	–	–	**76.7**	75.9	76.1
Mean AUROC (10 tasks)		64.9	73.2	**77.2**	76.1	75.7	74.7
Wins vs. RT		0/10	2/10		3/10	3/10	**5/10**

Table 17: SAGE: supervised fine-tuning nMAE (%, lower is better) on RelBench V1 entity regression. Bold: best per task.

		Published baselines	This work, SAGE		
Database	Task	Entity Median	pretrained	scratch (uni.)	scratch (orig.)
Amazon	user-ltv	30.3	**24.3**	**24.3**	26.4
	item-ltv	11.1	7.3	**7.2**	8.7
Avito	ad-ctr	43.1	39.8	39.2	**39.0**
Event	user-attendance	35.2	**33.7**	34.8	38.8
F1	driver-position	58.4	**48.7**	52.7	57.9
H&M	item-sales	15.8	**7.3**	7.9	10.4
Stack	post-votes	**13.6**	13.7	13.7	14.2
Trial	site-success	92.7	97.2	89.1	**82.1**
	study-adverse	17.0	**10.6**	11.8	12.4
Mean nMAE (9 tasks)		35.2	31.4	**31.2**	32.2

At the full budget, every architecture achieves scores comparable to RT, the strongest published fine-tuned model, and clearly outperforms Griffin. Across the ten tasks shared by all methods, RT averages 77.2 AUROC and Griffin 73.2, compared to 76.2 for RelGT, 76.5 for HGT, and 76.1 for GraphSAGE initialized from our pretrained model. On regression tasks, where RT and Griffin report R^2 rather than MAE, our pretrained models average 30.3, 29.8, and 31.4 nMAE, against 35.2 for the per-entity baseline and 33.4, 33.7 and 32.2 for the same architecture trained from scratch on the original schema.

Initialization plays a much smaller role here than under the budgets evaluated in Section D.5. On the universal schema, the pretrained and randomly initialized runs converge to within 0.7, 0.3, and 0.8 AUROC points of each other. However, the choice of schema still affects final performance for two of the three architectures. When trained from random initialization under the same protocol, the original schema underperforms the universal-schema run by 2.6 and 1.3 AUROC points for RelGT and HGT, respectively, though it scores 0.2 points higher for GraphSAGE. On regression tasks, the original schema trails the universal schema by 2.4, 1.7, and 1.0 nMAE points across the three models. We note that these schema comparisons may be partially confounded by hyperparameter selection. Ultimately, the benefit of pretraining is concentrated in the early steps of fine-tuning rather than in the final converged score.

D.7 Per-step cost under the universal schema

To measure the training-time cost of the mapping, we train each backbone on the same task under both the original and the universal schema, using an otherwise identical protocol. For each database we take the largest entity task by number of training rows and run 1,000 optimizer steps from random initialization at batch size 256, in our zero-shot configuration. Every measurement runs on a single L40S GPU. A step is timed from requesting the batch to the end of the optimizer step, so waiting on the data loader is included; the first 20 steps are discarded to exclude loader start-up. Task context and self-labels are enabled on the universal schema, as in our zero-shot runs, and neither exists on the original one. Table 18 gives the result. These are necessarily single-task steps, since training on the original schema is possible only one database at a time; they are therefore not comparable to the step times of our multi-task runs, where a single step comprises one forward pass per task in the roster and costs more per sample.

Table 18: Mean wall-clock time per optimizer step, in seconds, on the mapped (Univ.) and original (Orig.) schema, with their ratio (O/U). *Tables* is the number of tables in the original schema, by which the rows are ordered. The final row gives the spread, $\max / \min$, within each column.

			RelGT			GraphSAGE			HGT		
Database	Task	Tables	Univ.	Orig.	O/U	Univ.	Orig.	O/U	Univ.	Orig.	O/U
rel-amazon	user-churn	3	0.127	0.101	0.80	0.207	0.069	0.33	0.464	0.143	0.31
rel-hm	item-sales	3	0.134	0.093	0.69	0.206	0.070	0.34	0.475	0.132	0.28
rel-event	user-ignore	5	0.128	0.108	0.84	0.209	0.133	0.64	0.468	0.253	0.54
rel-stack	user-badge	7	0.151	0.137	0.91	0.233	0.181	0.78	0.497	0.371	0.75
rel-avito	user-visits	8	0.122	0.129	1.06	0.214	0.172	0.80	0.461	0.378	0.82
rel-f1	driver-dnf	9	0.143	0.166	1.16	0.214	0.224	1.05	0.613	1.195	1.95
rel-trial	site-success	15	0.112	0.232	2.07	0.209	0.256	1.22	0.450	0.564	1.25
Spread (max/min)			1.35	2.49		1.13	3.71		1.36	9.05	