The Henry and Marilyn Taub
Faculty of Computer Science

Module 6: Relational Foundation Models

StructML Course

Benny Kimelfeld
Technion – Israel Institute of Technology

Spring 2026

Module Goals

- Learn the concept of a *foundation model* on relational data
- Understand principal ideas in *using* a tabular/relational foundation model
- Understand principal ideas in *designing* a tabular/relational foundation model
- Learn details of *specific foundation models*, representing different approaches to the challenge

Outline

Outline

Foundation Models in General

- Pre-trained models that generalize to *different tasks* from *cross-domain data*
  - Trained once, adapted many times
- **Canonical examples:**
  - Natural language (LLMs): GPT, T5, PaLM, Llama, DeepSeek
  - Vision+text: DALL-E, CLIP, DINO, Moondream
  - Graphs+text: GraphGPT, GraphLLM
- We will focus on foundation models for tabular and relational data

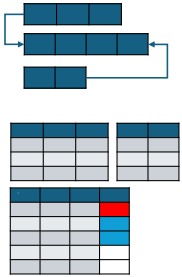
Tabular/Relational Foundation Models

- Importantly, pre-trained on datasets of many different schemas
  - Applicable to new datasets without knowing the schema in advance
- Involves encoding the data into a representation space of a fixed (predefined) dimension, regardless of the domain
- Need to capture the interplay between *metadata (schema) and data*
- To date, only transformer-like architectures are capable of doing so with reasonable performance
- We will see all of these in the examples

Relational Model Types: Transductive, Inductive, Foundation

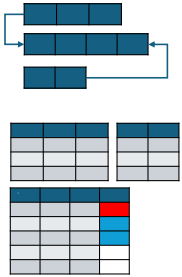
Transductive model:

Known schema, known task, known DB

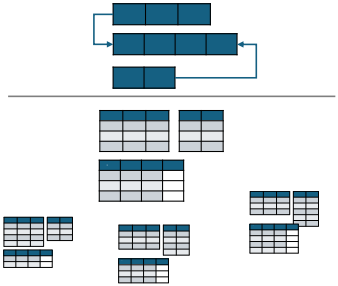


Relational Model Types: Transductive, Inductive, Foundation

Transductive model:
Known schema, known task, known DB

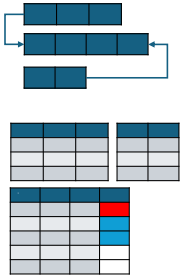


Inductive model:
Known schema, known task, unknown DB

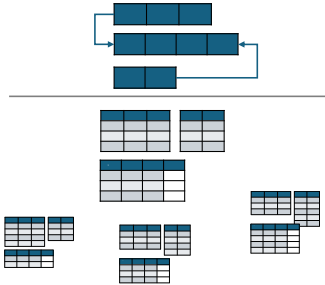


Relational Model Types: Transductive, Inductive, Foundation

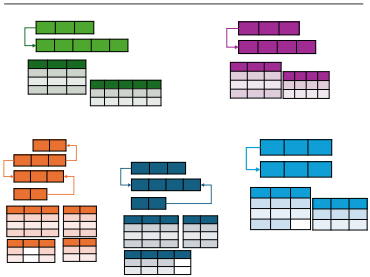
Transductive model:
Known schema, known task, known DB



Inductive model:
Known schema, known task, unknown DB



Foundation model:
Unknown schema, unknown task, unknown DB



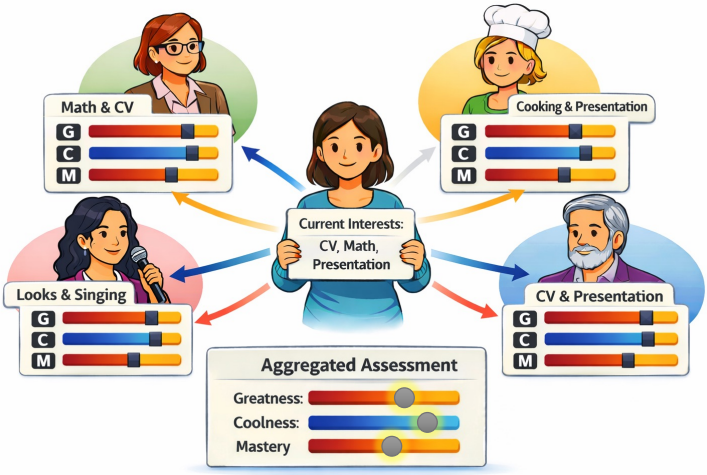
Using Foundation Models

Once pre-trained, an FM can be adapted to a new dataset in several modes:

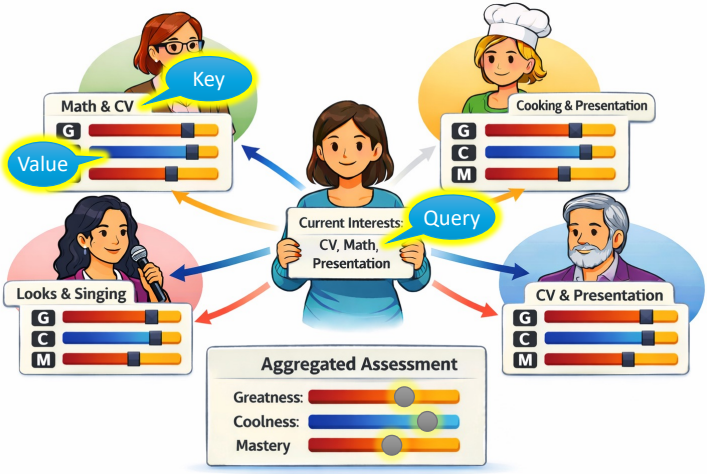
1. Fine-tuning: Starting with the pre-trained model, *update the model parameters* using labeled data from the target task
   - Or other types of learning, e.g., reinforcement
   - Works with *many-shot* (update all parameters) or *few-shot* (update a small number of the parameters)
2. In-context learning: Feed the model with *task examples as input*
   - Leave parameters intact
   - The model is *trained to learn*
3. Prompting (zero-shot): Specify the task via some (typically textual) *task description*, without any labeled data
   - The model is *trained to map a prompt into an answer*

Outline

Attention Analogy



Attention Analogy



Attention Unit

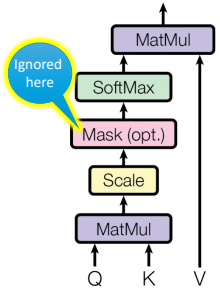
- Input: $\mathbf{q} \in \mathbb{R}^{d_k}$ $\mathbf{K} \in \mathbb{R}^{m \times d_k}$ $\mathbf{V} \in \mathbb{R}^{m \times d_v}$
- Apply the intuition with a conventional *scaling factor* $\sqrt{d_k}$

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) := \text{Softmax}\left(\frac{\mathbf{q} \cdot \mathbf{K}^\top}{\sqrt{d_k}}\right) \cdot \mathbf{V}$$

- Typically, a *collection* of instances (query posers) instead of one
- In this case, we get $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ instead of $\mathbf{q} \in \mathbb{R}^{d_k}$

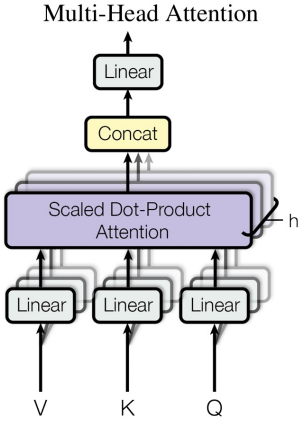
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{d_k}}\right) \cdot \mathbf{V}$$

Scaled Dot-Product Attention



[Vaswani et al., 2017]

Multi-Head Attention (by Figure)



[Vaswani et al., 2017]

Multi-Head Attention Detailed

- Concatenate multiple attentions, each scaled with separate parameters:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) :=$$

$$\Big(\text{Attention}(\mathbf{Q}_1, \mathbf{K}_1, \mathbf{V}_1) \oplus \ldots \oplus \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)\Big) \cdot \mathbf{W}^O$$

  where $\mathbf{Q}_i = \mathbf{Q} \cdot \mathbf{W}_i^Q$, and $\mathbf{K}_i = \mathbf{K} \cdot \mathbf{W}_i^K$, and $\mathbf{V}_i = \mathbf{V} \cdot \mathbf{W}_i^V$

- The matrix concatenation $\mathbf{A} \oplus \mathbf{B}$ means that we simply place $\mathbf{B}$ to the right of $\mathbf{A}$ (i.e., row-wise concatenation)

Cross-Attention vs. Self-Attention

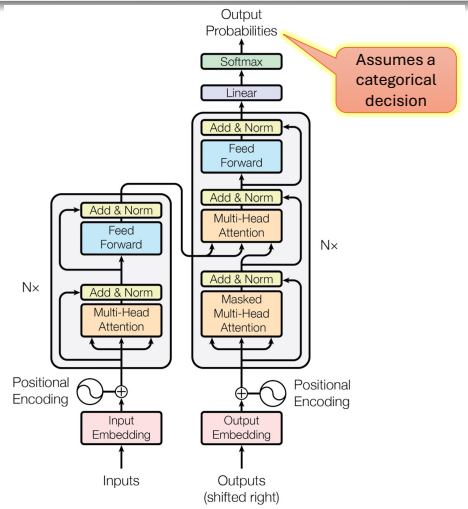
- *Where are* $\mathbf{Q}$, $\mathbf{K}$ *and* $\mathbf{V}$ *coming from?*
- Common scenario:
  - We have embeddings (representations) for the observers, represented by a matrix $\mathbf{X} \in \mathbb{R}^{n \times d_x}$, and embeddings for our experts, represented by $\mathbf{C} \in \mathbb{R}^{m \times d_c}$
    - These are predefined or determined by lower layers
  - Then, we apply weight matrices: $\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}^Q \quad \mathbf{K} = \mathbf{C} \cdot \mathbf{W}^K \quad \mathbf{V} = \mathbf{C} \cdot \mathbf{W}^V$
    - Importantly, the dimensions of the $\mathbf{W}$ does not depend on the number n of instances
- Two scenarios of how the attention mechanism is used in a bigger network:
  - Cross-attention: The observers and the context items are different creatures, as we assumed so far; in particular, $\mathbf{X}$ and $\mathbf{C}$ are not necessarily related
    - Example: textual question x about an image c
  - Self-attention: The observers and the context items are the same, and $\mathbf{X} = \mathbf{C}$
    - Example: language modeling, where each token observes the other tokens

Transformer Architectures

By a *transformer* we refer to an architecture with the following layers:

1. Input encoding
2. N layers of: multi-head attention followed by a feed-forward network, each includes a skip connection
3. A *decision layer* that typically consists of a simple NN layer followed by Softmax

The Original Text Transformer (A.I.A.Y.N)



[Vaswani et al., 2017]

Outline

Outline

TabPFN

- A foundation model for tabular data
- Designed for in-context learning
- Nature article [Hollmann et al., 2025]
  - Very few algorithm articles have been published in Nature over the years
  - Prominent example: AlphaFold (2024 Nobel Prize in Chemistry)
- Yet, not a foundation model in the usual sense
- It is **not** trained on a large collection of real data (like LLMs)
- Rather, it is trained on a large collection of *synthetic data* generated from random probabilistic models
  - Good news for fans of traditional closed models
- Based on the earlier PFN model [Müller et al., 2022]

Nature Article

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Open access | Published: 08 January 2025

Accurate predictions on small data with a tabular foundation model

Noah Hollmann ✉, Samuel Müller ✉, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister & Frank Hutter ✉

Nature **637**, 319–326 (2025) | Cite this article

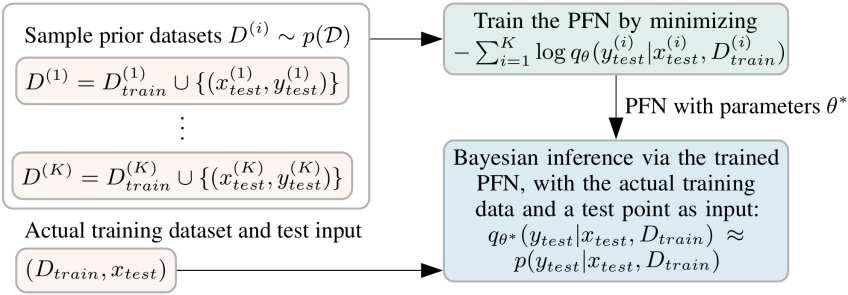
Prior-Data Fitted Network (PFN)

- Focuses on classification into a set C of classes
- The usual learning paradigm learns to predict $y \in C$ given an input vector $\mathbf{x}$
- PFNs *learn to learn*: they translate a **whole dateset** into a **distribution** to predict with
  - Input:
    - A bag $D = \{\!\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}\!\}$ of input-output examples, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in C$
    - An input $\mathbf{x}_{n+1}$
  - Output: y_{n+1}

PFN: Technical Idea

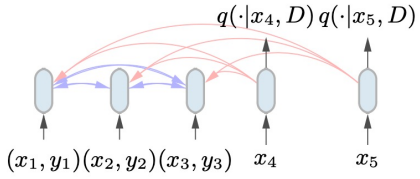
- Adapt the *transformer* model to represent the distribution
- Pretraining focuses on a class of $(\mathbf{x}, y)$ distributions
  - E.g., Bayesian networks
- In effect, pretraining aims to learn a function that maps the input $D = \{\!\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}\!\}$ into a distribution over xs and y

Visual of PFN



[Müller et al., 2022]

PFN's Transformer Model



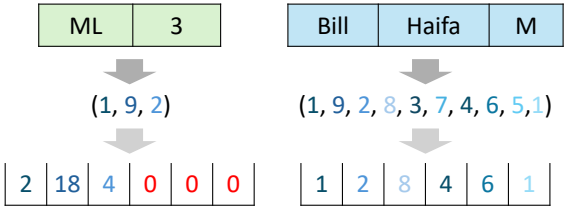
[Hollmann et al., 2023]

- Adaptation of the original transformer model [Vaswani et al., 2017] to a tabular setting
  - Without positional encoding!
- Given the input $(D, \mathbf{x})$, we do the following:
  1. Several layers of multi-head self-attention (plus skip connection) among the pairs $(\mathbf{x}_i, y_i)$ in D, followed by a feed-forward network
  2. Several layers of cross-attention (plus skip connection) from $\mathbf{x}$ to the pairs $(\mathbf{x}_i, y_i)$ in D, followed by a feed-forward network
  3. A decision layer that consists of a linear layer followed by Softmax, towards a categorical decision of y

TabPFN: PFN as a Foundational Tabular Model

- Recall: a *tabular signature* is a pair $(\mathbf{A}, \mathbf{dom})$
  - $\mathbf{A} = A_1, \ldots, A_m$ is a sequence of *attribute names*
  - $\mathbf{dom} = dom_1, \ldots, dom_m$ is a sequence of *domains* (of values)
- The TabPFN model is applicable to all tabular signatures
- *How to do so?*

Row Encoding in TabPFN (Illustration)



Row Encoding in TabPFN (Details)

- First, we need to translate every tuple $\mathbf{t} = (t_1, \ldots, t_m)$ into a feature space $\mathbb{R}^d$ of the same predefined dimension space d, regardless of $(\mathbf{A}, \mathbf{dom})$
- TabPFN first encodes every tuple $\mathbf{t}$ using off-the-shelf per-value encoders:

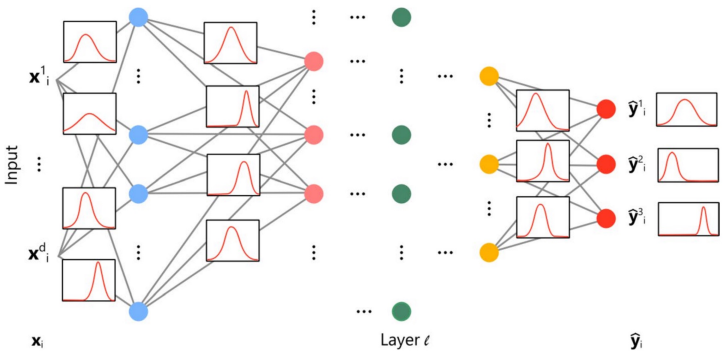
$$\text{Enc}(\mathbf{t}) := \text{Enc}_{\text{dom}_1}(t_1) \oplus \cdots \oplus \text{Enc}_{\text{dom}_m}(t_m) \ \in \mathbb{R}^k$$

- Then:
  - If $k < d$, pad $\text{Enc}(\mathbf{t})$ with 0s, and normalize—multiplication by d/k
  - If $k > d$, sample d positions randomly from the k positions in $\text{Enc}(\mathbf{t})$

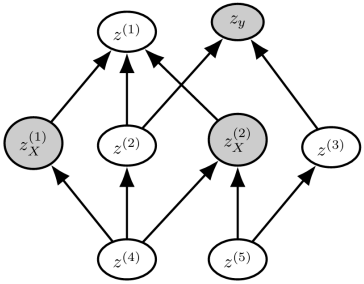
Pre-Training

- In principle, TabPFN could train on realistic datasets; this is **not** what they do
- They produce synthetic training data by invoking randomly-constructed *probabilistic graphical models*
- Specifically, they train on:
  - *Bayesian Neural Networks* (BNNs)
  - *Structural Causal Models* (SCMs)

Illustration of a BNN and an SCM

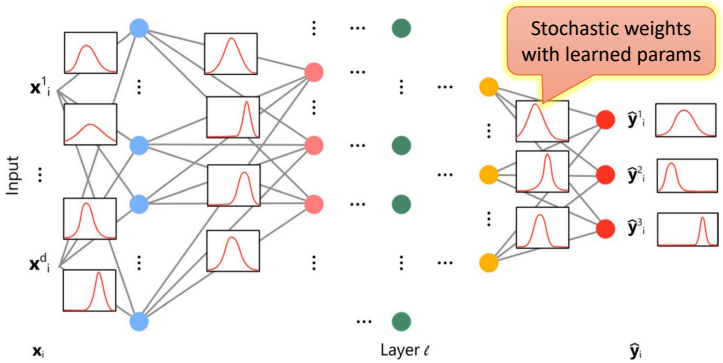


[Magris and Iosifidis, 2023]

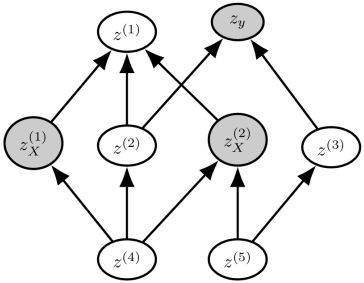


[Hollmann et al., 2023]

Illustration of a BNN and an SCM

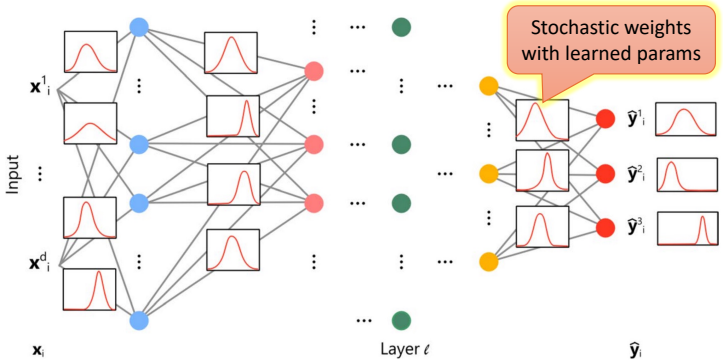


[Magris and Iosifidis, 2023]

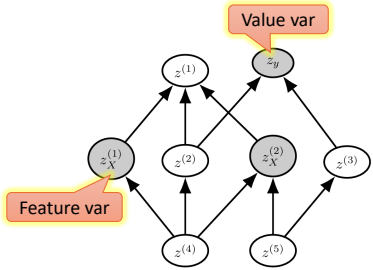


[Hollmann et al., 2023]

Illustration of a BNN and an SCM



[Magris and Iosifidis, 2023]



[Hollmann et al., 2023]

Sampling a Dataset for TabPFN Training

1. Construct a random probabilistic graphical model G
   - Techniques provided in the paper
2. Select m random input/intermediate nodes $v_1, \ldots, v_m$ of G, to be assigned to $A_1, \ldots, A_m$, respectively
3. Select a random output node v_y for y
4. Transform the numeric domain of v_y into a categorical domain:
   - Sample a number k of categories
   - Sample $k-1$ boundaries in the line domain of v_y
5. Apply a random (forward) invocation of G, getting a *random assignment* $\alpha(v)$ for each node v of G
6. Construct the instance $(\mathbf{t}, y)$ where $t_i = \alpha(v_i)$ and $y = \text{bucket}(\alpha(v_y))$
   - $\text{bucket}(z)$ is the index of the interval that contains z by our boundaries

Training

Algorithm 1: Prior-fitting of a PFN (Müller et al., 2022)

Input : A prior distribution over datasets $p(D)$, from which samples can be drawn and the number of samples K to draw

Output : A model q_θ that will approximate the PPD

Initialize the neural network q_θ;

for $j \leftarrow 1$ **to** K **do**

  Sample $D \cup \{(x_i, y_i)\}_{i=1}^{m} \sim p(D)$;

  Compute stochastic loss approximation $\bar{\ell}_\theta = \sum_{i=1}^{m}(-\log q_\theta(y_i|x_i, D))$;

  Update parameters θ with stochastic gradient descent on $\nabla_\theta \bar{\ell}_\theta$;

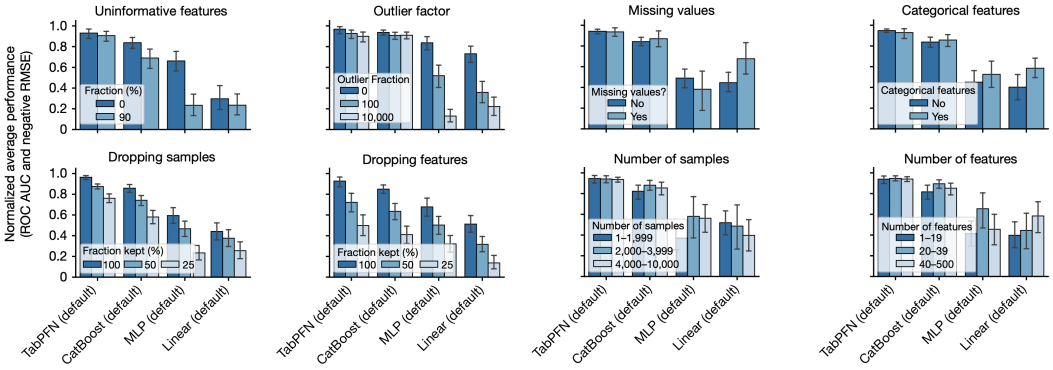
end

[Hollmann et al., 2023]

Implementation and Experimental Setup

- 12 layers
- Embedding size 512, hidden size 1024 in feed-forward layers
- 4-head attention
- Adam optimizer
- 25.82 M parameters
- Benchmarks datasets: AutoML Benchmark36, OpenML-CTR2337

Experiments from the Nature Article



[Hollmann et al., 2025]

AutoML Benchmark and OpenML-CTR23 ; TabPFN pre-trained on 8 GPUs, 2 weeks

Discussion

- As noted earlier, TabPFN is *not* a foundation model in the LLM sense, as it is trained exclusively on synthetic data
- Conceptually, TabPFN learns to approximate the posterior inference of a graphical probabilistic model given a dataset
- In principle, similar behavior could be obtained via classical maximum a posteriori inference in BNNS or SCMs
- However, such approaches are computationally expensive, whereas TabPFN performs inference in a single forward pass
- Final note: the TabPFN approach has been recently extended to relational databases [Kothapalli et al., 2026, Grinsztajn et al., 2026]

Outline

TabLLM

- As textual foundation models, LLMs typically encompass a huge volume of past knowledge
- For tabular data, an LLM can even do zero-shot prediction
  - That is, without any labeled data, processing only the given attribute names and values, and the name of the missing (predicted) attribute
- **TabLLM** [Hegselmann et al., 2023] focuses on how to do better than zero-shot with few-shot (in-context or fine-tuned) learning
- Approach also applied to *full relational databases* [Wu et al., 2025]
- Main focus of the TabLLM paper: how tabular examples (shots) are serialized textually into the LLM

Tabular Few-Shot via an LLM

- ML setting:
  - Input:
    - A small bag $D = \{\!\{(\mathbf{t}_1, y_1), \ldots, (\mathbf{t}_n, y_n)\}\!\}$ of input-output examples, each $y_i \in C$ for a finite set C of classes
    - An input $\mathbf{t}_{n+1}$
  - Output: y_{n+1}
- 3 ways of using an LLM for this task:
  1. In-context learning: prompt the context D and question $\mathbf{t}_{n+1}$
  2. Fine-tune the whole LLM for the task (*heavy...*)
  3. Fine-tune a tiny portion of parameters for the task
     - *Parameter-Efficient Fine-Tuning* (PEFT)
- TabLLM focuses on (3), using an open-source model

PEFT via $(IA)^3$ [Liu et al., 2022]

- Ordinary attention: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$
- Idea: add two new vectors that element-wise multiply the keys and values:

$$\text{ExtAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{Softmax}\left(\frac{\mathbf{Q} \cdot (\mathbf{w}_k \odot \mathbf{K}^\top)}{\sqrt{d_k}}\right)(\mathbf{w}_v \odot \mathbf{V})$$

- In addition, add a parameter to the FFN: $\left(\mathbf{w}_f \odot \sigma(\mathbf{W}_1 \cdot \mathbf{x})\right) \cdot \mathbf{W}_2$
- Then, we train $\mathbf{w}_k$, $\mathbf{w}_v$, and $\mathbf{w}_f$ on the task, leaving remaining parameters intact
- Done on each layer, so the total number of new parameters is $L \cdot (d_k + d_v + d_f)$

Tuple to Prompt

- The main focus of the paper is on different ways of *serializing* the examples $(\mathbf{t}, y)$ into a textual prompt
- Example techniques:
  - Direct list: "$(\mathbf{t}_1, y_1), \ldots, (\mathbf{t}_n, y_n)$"
  - Sentence per cell: "The [col-name] is [value]"
  - LLM (T0, GPT-3) prompt: "Write this info as a sentence: $(\mathbf{t}_1, y_1), \ldots, (\mathbf{t}_n, y_n)$"
  - LLM fine-tuned to *table-to-text* (HuggingFace)

Example

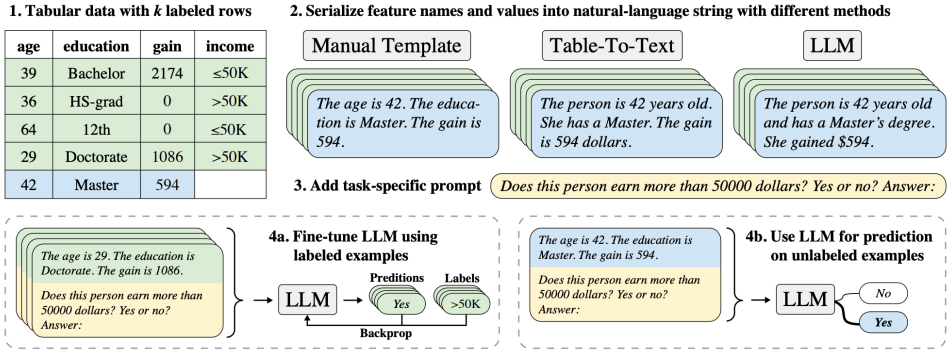
Bank Dataset (List Template):

```
- age: 69
- type of job: retired
- marital status: single
- education: tertiary
- has credit in default?: no
- average yearly balance, in euros:
2144
- has housing loan?: no
- has personal loan?: no
- contact communication type: cellular
- last contact day of the month: 29
- last contact month of year: jul
- last contact duration, in seconds:
417
- number of contacts performed during
this campaign and for this client:
- number of days that passed by after
the client was last contacted from a
previous campaign: 184
- number of contacts performed before
this campaign and for this client: 4
- outcome of the previous marketing
campaign: success
```

Bank Dataset (Text GPT-3):

```
The person is 69 years old, retired,
single, and has a tertiary education.
They have no credit in default, and
their average yearly balance is 2144
euros. They have no housing loan or
personal loan. The contact
communication type is cellular, and the
last contact was on the 29th day of the
month and lasted 417 seconds. They have
been contacted 4 times before this
campaign, and the outcome of the
previous marketing campaign was success.
```

TabLLM Architecture

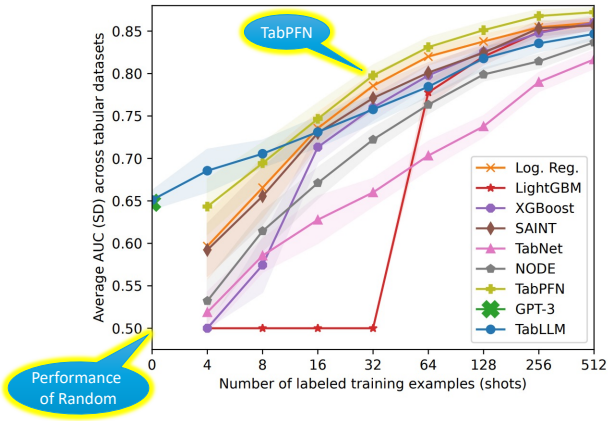


[Hegselmann et al., 2023]

TabLLM Experiments

- T0 LLM (HuggingFace)
- T-Few for fine-tuning
- Typically 31 epochs
- Example fine-tuning time: 64 examples (Income dataset) $\sim$ 3 minutes
- Inference time $\sim$ 10 ms per example

Chart from the TabLLM Paper



[Hegselmann et al., 2023]

9 public datasets:

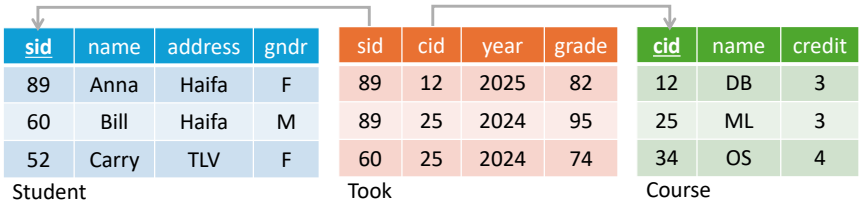
- **Bank** (45,211 rows, 16 columns)
- **Blood** (748, 4)
- **California** (20,640, 8)
- **Car** (1,728, 8)
- **Credit-g** (1,000, 20)
- **Income** (48,842, 14)
- **Jungle** (44,819, 6)
- **Diabetes** (768, 8)
- **Heart** (918, 11)

Outline

The Griffin System

- A relational foundation model from Amazon [Wang et al., 2025]
- Trained on both tabular datasets and relational databases
- Designed for *fine-tuning* (many-shot or few-shot)
- Consists of two main components: *fact representation* (tabular) followed by a *GNN* for structure processing (relational)
- As usual, we learn the details of the system, but the important takeaways are the challenges in being cross-schema, and the idea behind the solution design
  - Recall: relational FMs are nowadays still evolving and being replaced very quickly

Running Example



sid	name	address	gndr
89	Anna	Haifa	F
60	Bill	Haifa	M
52	Carry	TLV	F

Student

sid	cid	year	grade
89	12	2025	82
89	25	2024	95
60	25	2024	74

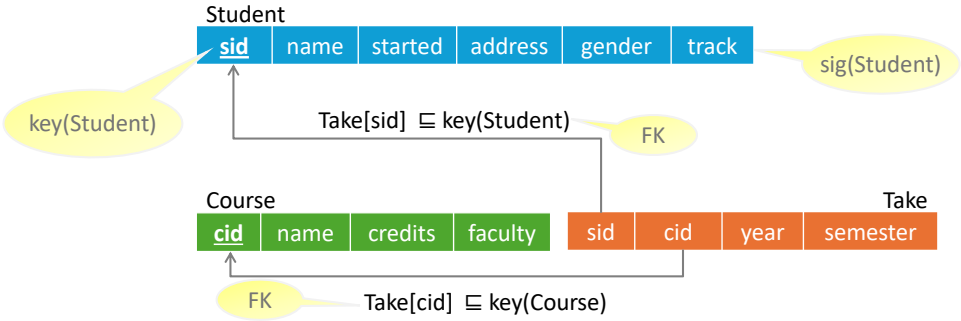
Took

cid	name	credit
12	DB	3
25	ML	3
34	OS	4

Course

Outline

Relational Schemas



Relational Schemas

A *relational schema* $\mathcal{S}$ is a tuple $(\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$ where:

- $\mathcal{R}$ is a finite set of *relation names*
- sig maps each relation name R to a tabular signature $\text{sig}(R)$
- key maps every relation name R into a *primary key*, which is a sequence of attributes from $\text{sig}(R)$
- **FK** is a set of *foreign-key constraints* (*FK*s for short)
  - An FK over the schema $\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$ has the form $R[\mathbf{C}] \sqsubseteq \text{key}(S)$:
    - R and S are relation names in $\mathcal{R}$
    - $\mathbf{C}$ is a sequence of attributes from $\text{sig}(R)$
    - $\mathbf{C}$ and $\text{key}(S)$ have the same length

Databases

Student

sid	name	started	address	gender	track
89	Anna	2022	Haifa	F	CS3
60	Bill	2024	Haifa	M	CS4
52	Carry	2023	Tel-Aviv	F	SE

Take

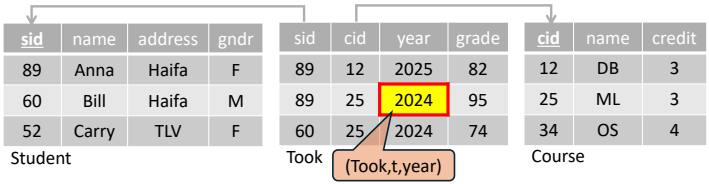
sid	cid	year	semester
89	12	2025	1
89	25	2024	2
60	25	2024	2

Course

cid	name	credits	faculty
12	DB	3	CS
25	ML	3	CS
34	OS	4	ECS

- Let $\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$ be a database schema
- A *database* D over $\mathcal{S}$ maps every $R \in \mathcal{R}$ to a table $D(R)$ over $\text{sig}(R)$ so that:
  - Primary keys are respected: for every relation name R, no two distinct tuples $\mathbf{t}_1, \mathbf{t}_2 \in D(R)$ satisfy $\mathbf{t}_1[\text{key}(R)] = \mathbf{t}_2[\text{key}(R)]$
  - FKs are respected: for every $R[\mathbf{C}] \sqsubseteq \text{key}(S)$ in $\mathbf{FK}$ and tuple $\mathbf{t} \in D(R)$ there is a tuple $\mathbf{s} \in D(S)$ such that $\mathbf{t}[\mathbf{C}] = \mathbf{s}[\text{key}(S)]$
- If D is a database over $\mathcal{S}$ and $\mathbf{t} \in D(R)$ for some $R \in \mathcal{R}$, then we call the expression $R(\mathbf{t})$ a fact (of D)

Cells



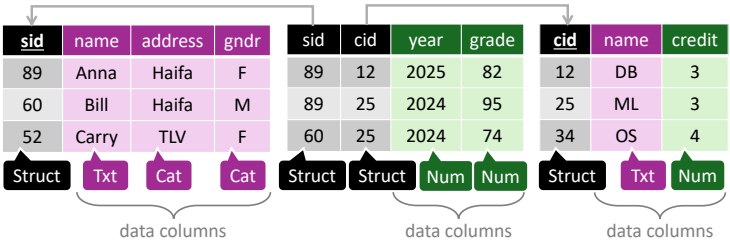
- Let D be a database over a schema $\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \textbf{FK})$
- A *cell* of D is defined as a triple $(R, \mathbf{t}, A)$ where:
  - R is a relation name in $\mathcal{R}$
  - $\mathbf{t}$ is a tuple in $D(R)$
  - A is an attribute in $\text{sig}(R)$
- We denote by $\text{Cells}(D)$ the set of all cells of D
- If $c \in \text{Cells}(D)$, then we denote by $\text{val}(c)$ the value $\mathbf{t}[A]$

Database Notation: Data and Structure Attributes

- Let D be a database over a schema $\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$
- If $R \in \mathcal{R}$ and $A \in \text{sig}(R)$, then we refer to $R.A$ as a *column*
- By a *structure column* we refer to every column $R.A$ that occurs in a key or a foreign key
- Every other column is a *data column*
- A *data cell* is a cell $(R, \mathbf{t}, A)$ such that $R.A$ is a data column
- By $\text{DCells}(D)$ we denote the set of all data cells of D

Outline

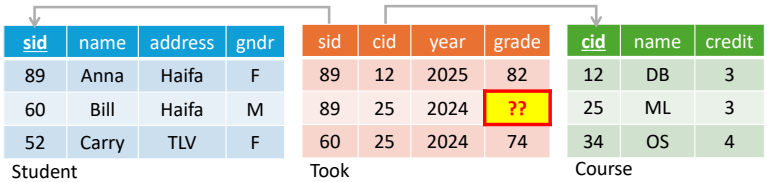
Griffin's View of Columns



- Griffin's data columns have few types: • *numerical (scalar) values* • *categorical values* • *textual values*
- Assumes tuples have *timestamps*

Training Tasks

- Griffin's training and evaluation is based on *masked* cells



sid	name	address	gndr
89	Anna	Haifa	F
60	Bill	Haifa	M
52	Carry	TLV	F

Student

sid	cid	year	grade
89	12	2025	82
89	25	2024	??
60	25	2024	74

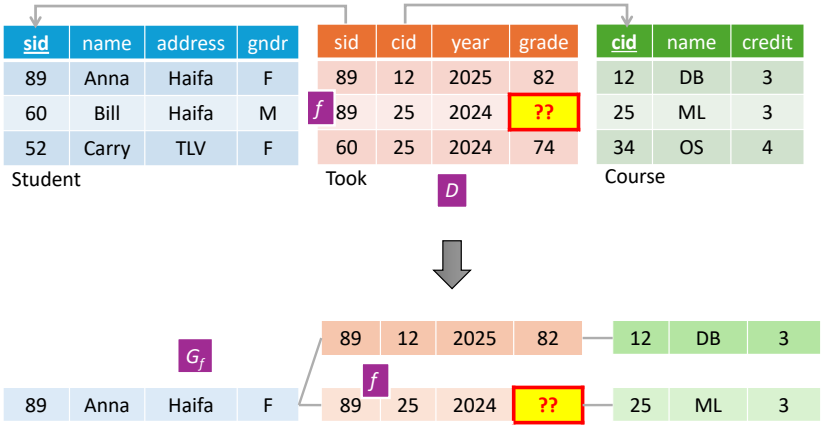
Took

cid	name	credit
12	DB	3
25	ML	3
34	OS	4

Course

- Task: given a database D and $c \in \text{DCells}(D)$, guess $\text{val}(c)$ when $\text{val}(c)$ is hidden
- Each masked data cell induces either a *classification task* (categorical) or a *regression task* (numerical)
  - No text tasks

Creating Small Graph Problems – Illustration First



Creating Small Graph Problems

- Each fact $g = R'(\mathbf{t}')$ of D is encoded as a feature vector $\mathbf{x}_g$ of dimension d
  - Same d for all schemas, databases, and tasks
  - We discuss this construction later
- We convert D to a graph G
  - Specifically, if there is an FK τ from the fact g to the fact f, then we get the labeled edge (g, τ, f)
- For every task $(R, A, \mathbf{t})$ over a fact $f = R(\mathbf{t})$, Griffin samples a random subgraph G_f, starting from f, similarly to *node-wise mini-batching*
  - Importantly, the sampling is *temporally restricted*: assuming we have time-stamps, we sample only facts that are older than f
- The problem is then a full-graph classification/regression: given G_f, guess $\mathbf{t}[A]$

Pretrained Primitive Encoders

- Griffin uses two types of primitive-type encoding, each produces a vector in $\mathbb{R}^d$:
  1. Text encoding: Based on BERT; applied for:
     - Representation of text and categorical cell values
     - Attribute names
     - Task description $\mathbf{s}_A$ — simply a text embedding of the task relation name R and attribute name A
  2. Numerical encoding: Applied to represent numerical cell values
     - The encoder of a trained encoder-decoder model using *MLP* + *normalization*
- Importantly, these encoders are pretrained once, and never changed throughout the phase of training over databases

Illustration of Griffin's Primitive Encoders

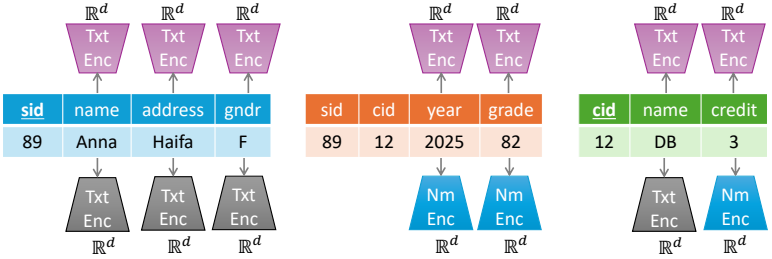
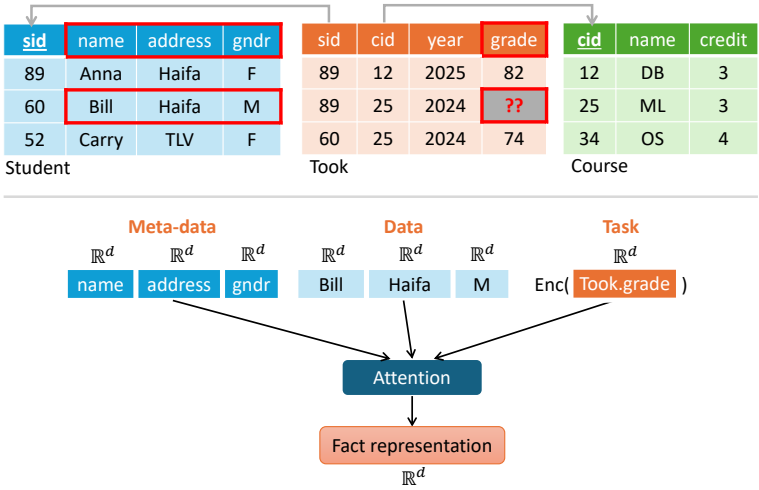


Illustration of Fact Encoding



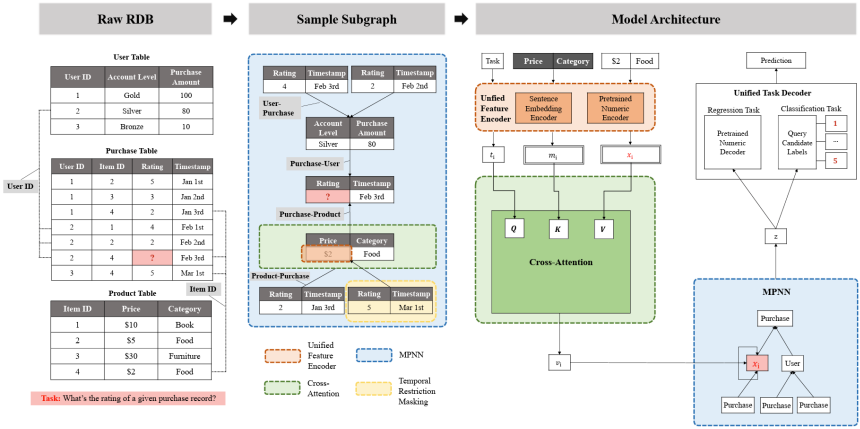
Fact Encoding

- To embed a fact f, we use 3 components:
  1. The embeddings of the values of f: $\mathbf{v}_1, \ldots, \mathbf{v}_\ell$ (matrix $\mathbf{V}_f \in \mathbb{R}^{\ell,d}$)
  2. The embeddings of the attribute names of f: $\mathbf{k}_1, \ldots, \mathbf{k}_\ell$ (matrix $\mathbf{K}_f \in \mathbb{R}^{\ell,d}$)
  3. The embedding of the task description: $\mathbf{s}_A$ (in $\mathbb{R}^d$)
- Each vector is of dimension d (512 in the experiments)
- Challenge: should work for varying ℓ: ℓ may be different for different relations R
- f's embedding obtained by *cross-attending* to the values $\mathbf{v}_j$ through the keys $\mathbf{k}_j$ by a learned query $\mathbf{q}$; multiple layers

$$\mathbf{u}_f^{(i)} := \mathbf{u}_f^{(i-1)} + \text{Attention}(\mathbf{q}_f^{(i)}, \mathbf{K}_f, \mathbf{V}_f)$$

where i is the layer number, and $\mathbf{q}_f^{(i)} := \text{MLP}(\mathbf{u}_f^{(i-1)}, \mathbf{s}_A; \boldsymbol{\theta}^{(i)}) \in \mathbb{R}^k$ is a learned MLP layer that produces a query for the attention layer

Griffin Architecture



[Wang et al., 2025]

So far we analyzed each record independently, so we designed a tabular model ; next, we account for the cross-tabular structure (FKs):

MPNN Layer

- Once we have the attention output $\mathbf{u}_f$, we apply a message-passing neural network (MPNN) over the graph
- *Heterogeneous* — handles each relationship type (FK) separately
  - Yet, importantly, it **does not** maintain any FK-specific parameters, since we do not know these FKs in advance
- Starting with the attention output $\mathbf{u}_f^{(0)} := \mathbf{u}_f$:

$$\mathbf{h}_{f,\tau}^{(i)} := \text{Avg}\{\!\{\text{MLP}(\mathbf{u}_g^{(i-1)}) \mid (f,\tau,g) \in E\}\!\} \quad \text{(avg over } \tau\text{-neighbors)}$$
$$\mathbf{h}_f^{(i)} := \text{Max}\{\!\{\mathbf{h}_{f,\tau}^{(i)} \odot \mathbf{e}_\tau \mid \tau \in L_e\}\!\} \quad \text{(max-agg over elem-wise product)}$$
$$\mathbf{u}_f^{(i)} := \mathbf{u}_f^{(i-1)} + \mathbf{h}_f^{(i)} \quad \text{(skip connection)}$$

where $\mathbf{e}_\tau \in \mathbb{R}^d$ is an encoding of the relationship type (FK) τ

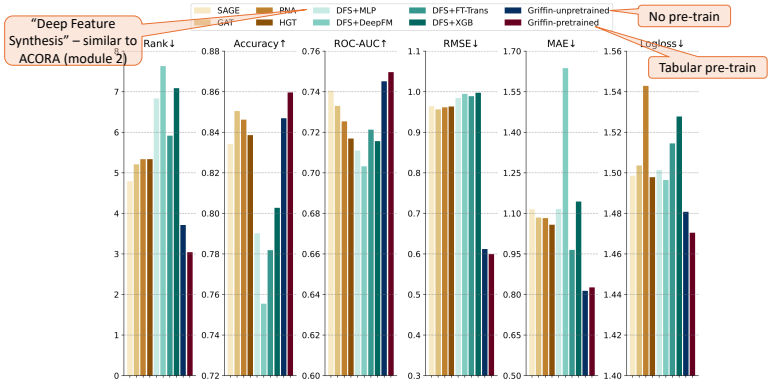
Prediction Head

- Finally, Griffin deploys two options for parameter-free *prediction heads* (i.e., the layer making the final decision), based on the type of the task:
  - Classification task: $\text{Softmax}(\mathbf{o} \cdot \mathbf{z}_1, \ldots, \mathbf{o} \cdot \mathbf{z}_m) \in \mathbb{R}^d$
    - Let $c_1, \ldots, c_m$ be the different categories in the task column
    - $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathbb{R}^d$ are the text encodings of $c_1, \ldots, c_m$
    - $\mathbf{o} \in \mathbb{R}^d$ is the output of the topmost layer of the MPNN
  - Regression task: Done with a *decoding function* $\text{Dec}(\mathbf{o}) \in \mathbb{R}$, where $\text{Dec} : \mathbb{R}^d \to \mathbb{R}$ is trained in advance along with the numerical encoding
- Importantly, the model selects the prediction according to the task's type, while the rest of the model (specifically, the set of parameters) is **shared**

Training

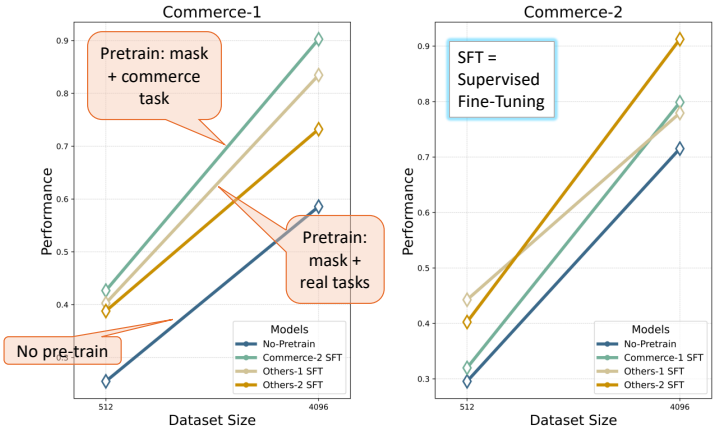
- The system is trained on two types of data:
  - **Single tables**
  - **Relational databases**
- On each type, it pre-trains on 2 types of tasks:
  - Mask: Completion of auto-masked values
  - Real: Realistic tasks from specific benchmarks (not completion)
- Numbers in the experiments:
  - 200 single tables curated from TPBerta and CARTE (accessible from Hugging Face)
  - 24 tasks from 4DBInfer [Wang et al., 2024] and RelBench [Robinson et al., 2024] repositories (both accessible from GitHub)

Experiments from the Griffin Paper: Effect Real-Task Pretraining



[Wang et al., 2025]

Experiments from the Griffin Paper: Effect of Relational Pretraining



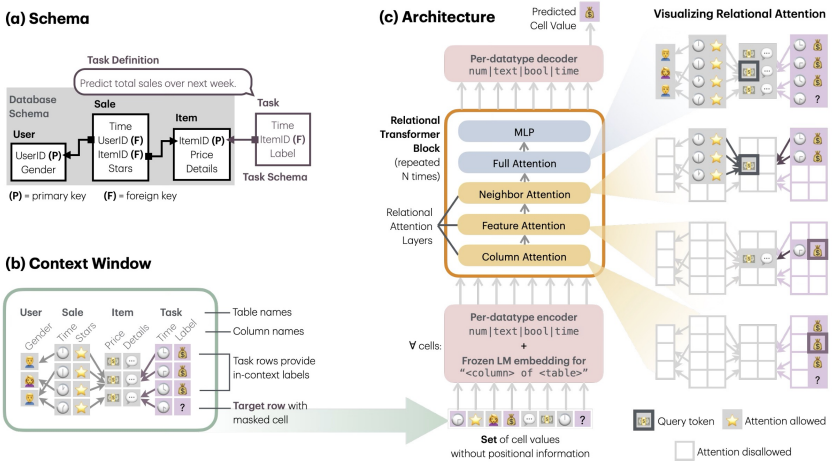
[Wang et al., 2025]

Outline

Relational Transformers (RT)

- A relational foundation model from Stanford [Ranjan et al., 2025]
- Operates directly over the database
  - Does not involve any explicit graph representation
- Designed for in-context learning
  - But can also be used in zero-shot and fine-tuning
- Many ideas similar to Griffin, with one major difference: the **attention**
  - Griffin uses cross-attention: *facts* attend to *cells*
  - RT uses self-attention: *cells* attend to *cells*

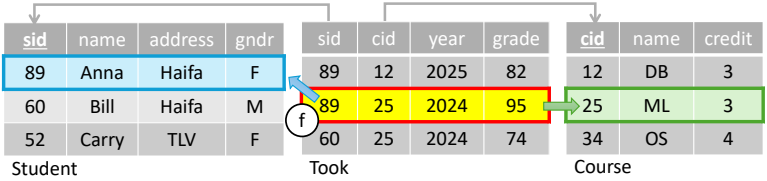
RT Architecture (Explained Next)



[Ranjan et al., 2025]

Database Notation: Referenced Facts

- Let D be a database over a schema $\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$
- Recall: A fact $f = R(\mathbf{t})$ can *reference* a fact $S(\mathbf{t}')$ through an FK $R[\mathbf{C}] \sqsubseteq \text{key}(S)$
- We denote by $\text{ref}_D(f)$ the set of all facts referenced by f



Task Table

- To represent tasks in a uniform way across schemas, RT adds to each schema a designated *task table*:

$$\text{Task}(\underline{\mathbf{K}}, \text{value}, \underline{\text{time}})$$

  with $\text{key}(\text{Task}) = \mathbf{K}, \text{time}$
- Very important — used for in-context examples
- Like Griffin — avoid using future information for prediction training

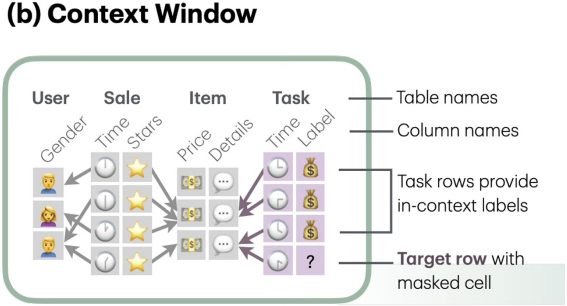
Task Cell and Sampled Context Window

- Similarly to Griffin, RT applies its model to a random subset D_f of D around the the Target fact f of the prediction instance ("seed row")
- The context window D_f is constructed by applying randomized BFS steps:

1. Start with $D_f := \{f\}$
2. Add to D_f a few Target facts with known values (for in-context learning)
3. Random BFS: when a fact g is visited:
   - All facts in $\text{ref}(g)$ enter the queue
   - A random subset of facts g' that reference g (i.e., $g \in \text{ref}(g')$) enters the queue

- Again, when timestamps are involved, facts newer than the task fact are excluded

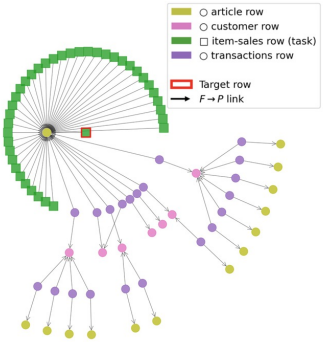
Context Example



[Ranjan et al., 2025]

Real Context on the H&M Dataset

(a) Graph visualization of the context window for an example from **rel-hm** (dataset) / **item-sales** (task)



(b) Tabular visualization of the same context window

item-sales (task) (39 rows x 3 cols)

timestamp	article_id	sales
2020-06-01	104264	[MASK]
2020-05-25	104264	0.337034
...	...	...
2019-10-07	104264	0.000000
2020-01-13	104264	0.000000

transactions (21 rows x 5 cols)

t_dat	customer_id	article_id	price	sales_channel_id
2020-02-20	29049	2251	0.033881	2
2020-06-01	29049	104264	0.041763	2
...	...	...	...	...
2020-06-01	1247634	104264	0.042356	2
2020-06-01	1247634	104264	0.042356	2

article (14 rows x 25 cols)

article_id	product_code	prod_name	...	garment_group_no	garment_group_name	detail_desc
2251	399223	Curvy Jeggings H...	...	1016	Trousers Denim	Jeggings in wash...
48500	692202	SPEED JAM SHIRT	...	1010	Blouses	Straight-cut blo...
...	...	...	...	...	...	...
101044	893796	Nejlika	...	1005	Jersey Fancy	Body in soft jer...
104264	920700	Dazzle top	...	1005	Jersey Fancy	Wide, slightly s...

customer (7 rows x 7 cols)

customer_id	FN	Active	club_member_status	fashion_news_frequency	age	postal_code
29049	NaN	NaN	ACTIVE	NONE	26.0	5cbf988955d931a7...
178380	NaN	NaN	ACTIVE	NONE	29.0	e777db329cc6dfe3...
...	...	...	...	...	...	...
856256	1.0	1.0	ACTIVE	Regularly	26.0	97cccf90aba93a67...
1247634	NaN	NaN	ACTIVE	NONE	24.0	0c0aaee59a3e86f5...

[Ranjan et al., 2025]

Cell Encoding

- Consider a cell $c = (R, \mathbf{t}, A)$ in DCells(D) where $dom(A)$ has the type τ
  - Recall: DCells(D) is the set of cells of the data columns
  - RT considers a few data types τ: *numeric*, *Boolean*, *timestamp*, and *text*
- The representation of c is given by $\mathbf{x}_c := \mathbf{W}_\tau \mathbf{z}_{t[A]} + \mathbf{W} \cdot e(A, R)$ where:
  - $\mathbf{W}_\tau$ and $\mathbf{W}$ are learned weight matrices
  - $\mathbf{z}_v$ is an encoding of v:
    - Numbers and timestamps are normalized over the column (expectation 0, variance 1)
    - Text is encoded via a predefined text embedding model Enc_{txt}
  - $e(A, R)$ is the encoding $\text{Enc}_{\text{txt}}(\text{“A of R”})$
- If c is the sought value cell of the seed row, then in $\mathbf{x}_c$ we replace $\mathbf{W}_\tau \mathbf{z}_{t[A]}$ with a learned parameter vector (mask vector) $\mathbf{m}_\tau$ of the required dimension
  - Same mask vector $\mathbf{m}_\tau$ for all predicted values of type τ

Illustration of Cell Encoding

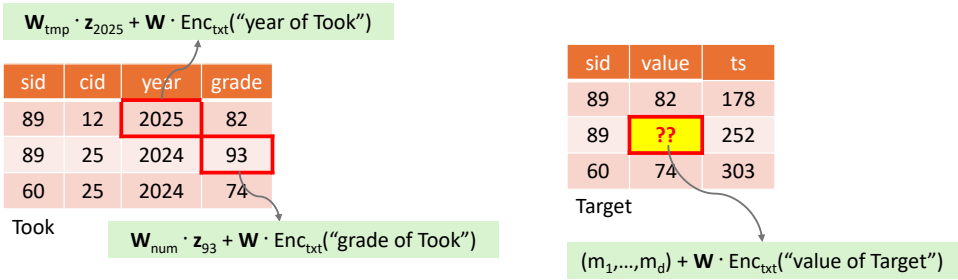
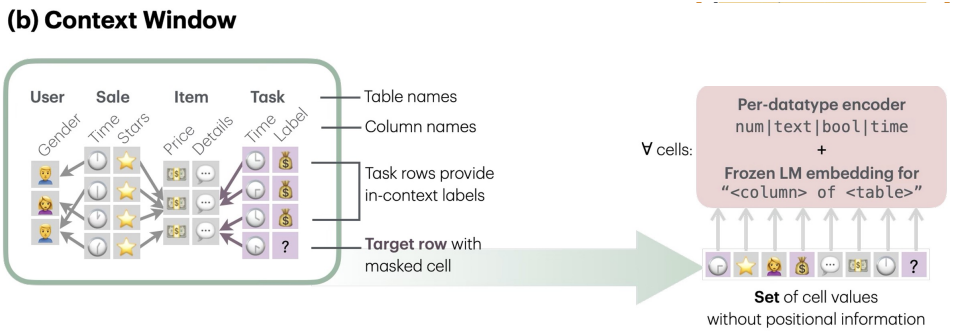


Illustration: From Context Window to Encoded Cells



[Ranjan et al., 2025]

Focused Cell Contexts

- Let $c = (R, \mathbf{t}, A) \in \text{DCells}(D)$ be a cell in a context window D_f
- We define 3 types of *specialized contexts* for c, each being a subset of $\text{DCells}(D_f)$
  1. The *column context* of c is the set of data cells in the column of c:
  $$\text{ColC}(c) := \{c' \mid c' = (R, \mathbf{t}', A) \in \text{DCells}(D_f)\}$$
  2. The *feature context* of c is the set of data cells in c's fact or its referenced facts
  $$\begin{aligned}\text{ColF}(c) :=& \{c' \mid c' = (R, \mathbf{t}, A') \in \text{DCells}(D_f)\} \cup \\ & \{c' \mid c' = (R', \mathbf{t}', A') \in \text{DCells}(D_f) \wedge R'(\mathbf{t}') \in \text{ref}_D(R(\mathbf{t}))\}\end{aligned}$$
  3. The *neighboring context* of c is the set of data cells of facts that reference c's fact
  $$\text{ColN}(c) := \{c' \mid c' = (R', \mathbf{t}', A') \in \text{DCells}(D_f) \wedge R(\mathbf{t}) \in \text{ref}(R'(\mathbf{t}'))\}$$

RT's Transformer Architecture

- Starting with $\mathbf{X}$ as the initial encoding, a transformer layer has the following form:

$$\begin{aligned}
\mathbf{x}_c &\leftarrow \mathbf{x}_c + \text{Update}(c, \text{ColC}(c)) && \#\text{column attention} \\
\mathbf{x}_c &\leftarrow \mathbf{x}_c + \text{Update}(c, \text{ColF}(c)) && \#\text{feature attention} \\
\mathbf{x}_c &\leftarrow \mathbf{x}_c + \text{Update}(c, \text{ColN}(c)) && \#\text{neighbor attention} \\
\mathbf{x}_c &\leftarrow \mathbf{x}_c + \text{Update}(c, \text{DCells}(D_f)) && \#\text{full context attention} \\
\mathbf{x}_c &\leftarrow \mathbf{x}_c + \text{MLP}(\mathbf{x}_c) && \#\text{MLP}
\end{aligned}$$

- The transformer architecture stacks multiple layers (12 in the experiments)

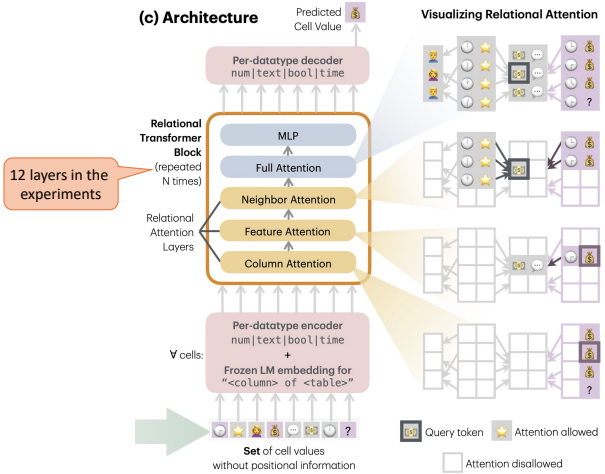
RT's Attention (Update)

On each set $S(c)$, RT's Update applies a sequence of multi-head self-attention layers for every cell c

$$\text{Head}_i(c, S(c)) \leftarrow \text{Attention}(\mathbf{x}_c \cdot \mathbf{W}_i^Q, \mathbf{X}_{S(c)} \cdot \mathbf{W}_i^K, \mathbf{X}_{S(c)} \cdot \mathbf{W}_i^V)$$
$$\text{Update}(c, S(c)) \leftarrow \text{Combine}(\text{Head}_1(c, S(c)), \ldots, \text{Head}_h(c, S(c)))$$

where $\mathbf{X}_S$ is the matrix with the representation $\mathbf{x}_{c'}$ for each cell $c' \in S(c)$

RT Architecture [Ranjan et al., 2025]



Batch Processing via Masking

- In practice, we update all of $\mathbf{X}$ in every step
- For each cell c to relate only to $S(c)$, RT uses pointwise-product by a *masking matrix* $\mathbf{M}^S \in \mathbb{R}^{s \times s}$ where s is the context size of the $|S(c)|$:

$$\mathbf{M}^S_{c,c'} = \begin{cases} 1 & \text{if } c' \in S(c) \\ 0 & \text{otherwise} \end{cases}$$

- Then:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top \odot \mathbf{M}^S}{\sqrt{d_k}}\right)\mathbf{V}$$

Training, Pre-Training, Zero-Shot

- *How do we apply RT to a new task?*
- 3 modes of operation:
  1. Ordinary (schema-specific, inductive) model: One schema, train using the training set, evaluate on a test set
     - Train: for each training task row, sample a context window, apply the model forward, update the model via back-propagation (e.g., Adam optimizer)
     - Test: for each testing task row, sample a context window, apply the model forward
  2. Fine-tuning: Pre-train the model on many datasets of many schemas, then continue with the ordinary (first) mode on your dataset of interest
  3. In-context: Simply apply RT without any training
     - *Where is the context?*
     - The other rows of the Task table!
     - In many cases, you get that for free (e.g., infer the future, past in the database)
     - Hence, can be seen as "zero-shot" in RelBench-type tasks

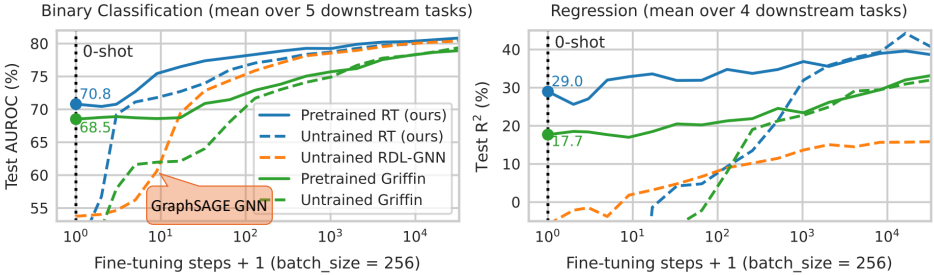
Implementation and Experimental Setup

- Tested on RelBench [Robinson et al., 2024]
  - *Leave-one-DB-out* pretraining
- 256-length hidden representation, 8 attention heads per layer
- MLP uses a *Sigmoid Linear Unit* (SiLU) activation:

$$\mathrm{SiLU}(x) = x\,\sigma(x) = \frac{x}{1 + e^{-x}}$$

- Context of 1024 cells
- Text encoding via Sentence-BERT [Reimers and Gurevych, 2019]

Experiments from the RT Paper



[Ranjan et al., 2025]

Key Takeaways from Module 6

- Foundation models for tabular and relational data generalize across schemas
- We saw several approaches to designing foundation models, all use transformers in one way or another
- TabPFN is trained on in-context learning from random synthetic datasets
- TabLLM is based on few-shot fine-tuning of a few extra parameters (PEFT)
- Griffin uses row-to-cell cross-attention, combined with a GNN
- Relational Transformers use cell-to-cell self-attention, stacking several layers of cell collections to attend
- Field is moving very fast ; specific systems will not last long ; their underlying ideas will live longer

The End

Many thanks for helping with the slides:

- **Shunit Agmon** (Technion)
- **Boaz Berger** (Technion)
- **Ilias Fountalis** (RelationalAI)

References I


Grinsztajn, L. et al. (2026).
Tabpfn-3: Technical report.
CoRR, abs/2605.13986.


Hegselmann, S., Buendia, A., Lang, H., Agrawal, M., Jiang, X., and Sontag, D. A. (2023).
Tabllm: Few-shot classification of tabular data with large language models.
In *AISTATS*, volume 206 of *Proceedings of Machine Learning Research*, pages 5549–5581. PMLR.


Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. (2023).
TabPFN: A transformer that solves small tabular classification problems in a second.
In *ICLR*. OpenReview.net.


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

References II

Kothapalli, V., Ranjan, R., Hudovernik, V., Dwivedi, V. P., Hoffart, J., Guestrin, C., and Leskovec, J. (2026).
Plurel: Synthetic data unlocks scaling laws for relational foundation models.
CoRR, abs/2602.04029.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. (2022).
Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.
In *NeurIPS*.

Magris, M. and Iosifidis, A. (2023).
Bayesian learning for neural networks: an algorithmic survey.
Artif. Intell. Rev., 56(10):11773–11823.

Müller, S., Hollmann, N., Pineda-Arango, S., Grabocka, J., and Hutter, F. (2022).
Transformers can do bayesian inference.
In *ICLR*. OpenReview.net.

References III

Ranjan, R., Hudovernik, V., Znidar, M., Kanatsoulis, C. I., Upendra, R., Mohammadi, M., Meyer, J., Palczewski, T., Guestrin, C., and Leskovec, J. (2025).

Relational transformer: Toward zero-shot foundation models for relational data.

CoRR, abs/2510.06377.

Reimers, N. and Gurevych, I. (2019).

Sentence-bert: Sentence embeddings using siamese bert-networks.

In *EMNLP/IJCNLP (1)*, pages 3980–3990. Association for Computational Linguistics.

Robinson, J., Ranjan, R., Hu, W., Huang, K., Han, J., Dobles, A., Fey, M., Lenssen, J. E., Yuan, Y., Zhang, Z., He, X., and Leskovec, J. (2024).

Relbench: A benchmark for deep learning on relational databases.

In *NeurIPS*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017).

Attention is all you need.

In *NIPS*, pages 5998–6008.

References IV

Wang, M., Gan, Q., Wipf, D., Zhang, Z., Faloutsos, C., Zhang, W., Zhang, M., Cai, Z., Li, J., Mao, Z., Song, Y., Tang, J., Zhang, Y., Yang, G., Lei, C., Qin, X., Li, N., Zhang, H., Wang, Y., and Zhang, Z. (2024).

4dbinfer: A 4d benchmarking toolbox for graph-centric predictive modeling on rdbs.

In *NeurIPS*.

Wang, Y., Wang, X., Gan, Q., Wang, M., Yang, Q., Wipf, D., and Zhang, M. (2025).

Griffin: Towards a graph-centric relational database foundation model.

CoRR, abs/2505.05568.

Wu, F., Dwivedi, V. P., and Leskovec, J. (2025).

Large language models are good relational learners.

In *ACL (1)*, pages 7835–7854. Association for Computational Linguistics.