LLMs AS FEATURE ENGINEERS: SIMPLIFYING AGENTIC RELATIONAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational forecasting tasks, such as predicting customer churn from a database of customers, transactions, and products, can be solved by transforming the database into a single table of entity features and training a standard tabular model on it. The main difficulty of this approach is feature engineering, and recent works delegate it to large language models (LLMs). In particular, RelAgent gives an LLM agent control over the entire pipeline: the agent explores the database, writes SQL feature programs, chooses the downstream model and its hyperparameters, and revises all these decisions using validation feedback over many iterations. In this work, we ask which of these responsibilities actually require an LLM. We study a simple pipeline in which the LLM acts only as a feature engineer. Given a static description of the database, it writes several SQL feature programs independently, using a single request per program and receiving no predictive feedback. Model tuning and aggregation of predictions are handled by standard methods. On 21 classification and regression tasks from RelBench, this pipeline outperforms the features engineered by an expert data scientist on average and stays close to RelAgent (77.52 vs. 78.38 average AUROC on classification). At the same time, one of our programs requires $146\times$ fewer LLM requests and about $1{,}200\times$ fewer processed input tokens than one search rollout of RelAgent. When a top-tier proprietary LLM is replaced with a much smaller open-source one, the average AUROC of our pipeline decreases by 2.59 points, compared with 8.25 points for RelAgent. Our analysis shows that a static database description is as informative as interactive exploration on most tasks and that conventional tuning further improves frozen LLM-generated features. It also shows that validation scores are a weak guide for choosing between programs: on classification, averaging independent programs outperforms selecting one of them by validation score. These results suggest that LLMs are most useful in relational learning as feature engineers, while model tuning and selection can be left to conventional methods.

1 INTRODUCTION

Many practical prediction problems are defined over relational databases: for example, whether a customer will stop purchasing, how many votes a new post will receive, or whether a clinical trial will achieve its primary outcome. In such relational forecasting tasks, each example is an entity at a given prediction time. The information relevant to it is distributed across multiple tables connected by primary–foreign key relations and restricted to the records created before this time. One line of work represents the database as a temporal heterogeneous graph and learns entity representations end-to-end with graph neural networks, graph transformers, or relational foundation models (Fey et al., 2023; Robinson et al., 2024; Chen et al., 2025; Dwivedi et al., 2025; Ranjan et al., 2026). A much older alternative, known as propositionalization (Kramer et al., 2001), is to transform the database into a single table. For each entity and prediction time, one computes features that summarize its relational history, such as counts, recency, and trends over selected join paths and time windows, and then trains a standard tabular model on these features. Recent evidence shows that this approach remains competitive. In the RelBench user study, features engineered by an experienced data scientist and processed by a tuned LightGBM model (Ke et al., 2017) were competitive with relational deep learning models, which matched or outperformed them on 11 of 15 tasks (Robinson et al., 2024). TabPFN-Rel, which applies a tabular foundation model to a single table automatically constructed

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.

from the database, currently ranks first among the models evaluated on the RelArena-α benchmark (Hayler et al., 2026). However, feature engineering remains the weak point of this approach. Generic methods such as Deep Feature Synthesis (Kanter & Veeramachaneni, 2015) enumerate aggregations along foreign-key paths without accounting for the semantics of the task, while manual engineering is expensive: the data scientist in the RelBench user study spent 12.3 hours and wrote about 880 lines of code per task on average (Robinson et al., 2024).

Large language models (LLMs) are natural candidates for this role: they can read the names of tables and columns, task descriptions, and sample rows, reason about which relations are likely to be predictive, and write executable queries. This idea was first explored for single-table data (Hollmann et al., 2023; Han et al., 2024; Nam et al., 2024; Abhyankar et al., 2026) and was recently extended to relational databases in the form of LLM agents (Kim et al., 2026; Huang et al., 2026). Among them, RelAgent (Huang et al., 2026) acts as an autonomous data scientist that controls the entire pipeline. In each of five independent rollouts, its agent spends up to 60 turns on exploring the database with SQL tools and writing SQL feature programs. The agent also chooses among seven families of tree-based models together with their hyperparameters, submits the resulting candidates for validation, and inspects the validation scores and predictions to revise them. The candidate with the best validation score across all rollouts is then deployed. In our comparison, RelAgent achieves the best average results on both classification and regression tasks from RelBench, but this comes at a considerable cost. According to our reproduction, a single rollout makes 146 LLM requests and processes 5.8 million input tokens, and the agent runs five rollouts per task. RelAgent's own ablation study appears to justify this cost: when the agent has to output its final program without validation feedback, its average AUROC on RelBench classification tasks drops from 78.38 to 66.86 (Huang et al., 2026).

At the same time, such an agent combines responsibilities that are very different in nature. Constructing relational features requires understanding the semantics of the database and the task, which is the strength of LLMs and what generic methods lack. In contrast, choosing the hyperparameters of a tabular model is a numerical black-box optimization problem. Dedicated methods such as Bayesian optimization solve it by systematically tracking all previous evaluations and without any LLM calls (Bergstra et al., 2011; Akiba et al., 2019). Finally, selecting the final solution by its validation score is a statistical decision, and its reliability depends on how well validation performance predicts test performance. In relational forecasting, this signal can be weak. Validation and test examples come from consecutive time periods, and on several RelBench tasks the validation set contains only a few hundred examples (e.g., 566 and 588 for `driver-dnf` and `driver-top3` in `rel-f1`). Consistent with this, Chen et al. (2026) found validation performance to be an unreliable guide for selecting architectures for relational tasks, and selection over many candidates is known to overfit to the validation set in general (Cawley & Talbot, 2010). When all these decisions are made by a single LLM within a long feedback loop, it is unclear which of them are responsible for the final performance. Moreover, each decision requires more LLM requests, and each request includes the entire interaction history, which grows with every step.

In this work, we ask whether competitive relational learning requires involving an LLM in anything beyond feature engineering. In the RelBench user study, the data scientist explored the database and wrote SQL features, while a standardized LightGBM script with hyperparameter tuning trained the model (Robinson et al., 2024). We study a simple pipeline that keeps this division of labor and assigns the role of the data scientist to an LLM (Section 3). The LLM receives a static, automatically constructed description of the task and database, which includes the schema, key relations, temporal metadata, column statistics, and sample rows. Based on this description, it writes $K = 5$ SQL feature programs independently, using a single request per program and receiving no predictive feedback. Each program is paired with a LightGBM model tuned with Optuna (Akiba et al., 2019), and the predictions of the resulting K models are averaged for classification and median-aggregated for regression. Table 1 compares how this pipeline and prior systems distribute responsibilities between the LLM and conventional components.

We evaluate our pipeline on 12 classification and 9 regression tasks from RelBench (Robinson et al., 2024), using GPT-5.2 (OpenAI, 2025) as the LLM, as in RelAgent. Our pipeline outperforms the data scientist features on average on both task types. It stays within 0.86 AUROC points of RelAgent on classification and within 3.3% of its normalized MAE on regression (Section 5.1). At the same time, one of our programs requires 146$\times$ fewer LLM requests and about 1,200$\times$ fewer processed input tokens than one search rollout of RelAgent (Section 5.2). When GPT-5.2 is replaced with the much smaller open-source Qwen3-32B (Yang et al., 2025), the average AUROC of our

Table 1: Distribution of responsibilities in pipelines that transform a relational database into a single table for a tabular model. *Data scientist* is the expert baseline from the RelBench user study (Robinson et al., 2024). LLM requests are medians over the classification tasks and include error-only repair requests; for RelAgent, they are measured in our reproduction. Both RelAgent and our pipeline make five attempts per task.

System	Database context	Feature construction	Model and hyperparameters	Use of validation	LLM requests per attempt
Data scientist	Manual exploration	Hand-written SQL features	LightGBM with a standard tuning script	Hyperparameter tuning	None (12.3 hours of work per task)
ReFuGe	Schema; LLM selects relevant tables	LLM proposes features; validation decides which to keep	Fixed LightGBM	Feature filtering; feedback to the LLM	Several per iteration
RelAgent	Interactive SQL exploration	LLM writes and revises programs over up to 60 turns	LLM chooses a model family and hyperparameters	Feedback to the LLM; best of 5 rollouts deployed	146 per rollout
Ours	Static description	$K = 5$ independent programs, one request each	LightGBM tuned with Optuna	One check of the tuned hyperparameters	1 per program

pipeline decreases by 2.59 points, compared with 8.25 points for RelAgent (Section 5.3). We then reintroduce the responsibilities of the agent into our pipeline one at a time (Section 6). A static database description performs on par with interactive exploration on average (Section 6.1). Even without tuning and aggregation, a single program of our pipeline reaches 75.17 average AUROC, while RelAgent reports 66.86 without validation feedback. Adding feedback improves regression results but degrades classification results on average, mainly because validation often fails to identify the best of the generated programs (Section 6.2). For the same reason, averaging the predictions of the K models outperforms deploying only the model with the best validation score by 2.75 AUROC points on classification (Section 6.3). Finally, tuning LightGBM on frozen programs improves the results on 19 of 21 tasks without any LLM calls (Section 6.4). Overall, our results indicate that RelAgent-style agents give the LLM more control than necessary: LLMs are most valuable as feature engineers, while model tuning and selection can be handled by conventional methods.

2 Relational forecasting and agentic feature engineering

Problem statement A relational database consists of tables connected by primary–foreign key relations, and the rows of event tables carry timestamps. A relational forecasting task is given by labeled examples (v, t, y), where v is an entity, t is a prediction time, and y is a class label or a real value computed from the events within a fixed window after t. The examples are split into training, validation, and test sets by their prediction times, so that validation and test examples come from later periods than training examples. We call a set of SQL queries that maps each pair (v, t) to a feature vector a *feature program*. A valid feature program uses only the records created before t, and a tabular model is trained on the resulting feature vectors.

Agentic feature engineering ReFuGe (Kim et al., 2026) extends LLM-based feature generation to relational databases with three agents. The first selects relevant tables and columns from the schema and task description, the second proposes candidate features using several LLM instances, and the third filters them, keeping a feature only if it improves the validation score of a fixed LightGBM model. The effect of each feature is summarized in natural language and passed to the agents in the next iteration, and the loop stops when no new feature is retained. RelAgent (Huang et al., 2026) gives the LLM broader control. In each rollout, the agent explores the database with read-only SQL queries, submits candidates consisting of a feature program and a model configuration to a validation tool, and inspects an evaluation workspace that stores all previous trials together with their validation predictions. A model configuration specifies one of seven tree-based model families and its main

hyperparameters, and, for regression, also the training objective and an optional log transform of the target. The candidate with the best validation score across rollouts is refit on the training and validation data and deployed. The ablations of RelAgent attribute its performance to several of these components. Removing validation feedback reduces the average classification AUROC from 78.38 to 66.86, and removing the evaluation workspace reduces it from 78.38 to 75.78. Restricting the agent to a single model family, LightGBM or CatBoost, reduces the average AUROC from 78.38 to 75.94 and 75.95, respectively. The authors also regard validation as a reliable selection criterion, since validation and test scores are close on most tasks. We revisit these conclusions by reintroducing exploration, feedback, validation-based selection, and model adaptation into our pipeline and measuring their effect on quality and cost with the same LLM (Section 6). Further related work on relational deep learning, single-table representations of databases, and LLM-based feature engineering is discussed in Section A.

3 Our pipeline

Our pipeline assigns the LLM a single responsibility: writing feature programs. All other steps are performed by conventional software. The pipeline consists of three steps.

Static database description For each task, we automatically construct a description of the task and database. It contains the task definition and split cutoffs, a summary of the training targets, the schema with primary keys, foreign keys, and time columns, column statistics, and three sample rows per table (Section B.1 lists all fields). All statistics and rows are computed from the records created before the validation cutoff, and no validation or test labels are shown. The description contains no feature suggestions, no information about the downstream model, and no validation scores. The resulting prompts contain 3–8K tokens, which corresponds to roughly 2,000–6,000 words.

Independent feature programs Given the description, the LLM returns a feature program in a single request: a short plan and 3 to 8 named DuckDB queries, each of which returns one row per prediction key with one or more feature columns. A validator checks that the queries are read-only, do not access the labels, return at most one row per key, and contain no forward-looking time conditions (Section B.3 describes the exact checks). The outputs of the queries are then joined into one feature table. If a program cannot be parsed, fails these checks, or fails to execute, the LLM receives the error message and may fix the program at most twice. We call this procedure *error-only repair*: the repair request contains no labels, predictions, or scores, so it can make a program runnable but cannot steer it toward better performance. We sample $K = 5$ programs independently from the same prompt.

Downstream models and aggregation For each program, we train a LightGBM model on the resulting feature table and tune its hyperparameters with Optuna. The programs stay fixed during this procedure. Finally, the K models are combined without any validation-based weighting: we average the predicted probabilities for classification and take the pointwise median of the predictions for regression.

4 Experimental setup

Tasks and metrics We use all 21 entity classification and regression tasks of RelBench v1 (Robinson et al., 2024), which are defined on seven databases, with the official temporal splits of RelBench 2.1.2 (Section C lists the tasks and their split sizes). Following RelAgent, we report test AUROC ($\times 100$) for classification and MAE for regression. We aggregate classification results by the average AUROC and regression results by the normalized MAE μ_n, which divides the MAE on each task by the MAE of the single-table LightGBM baseline and averages these ratios over tasks. Some tasks have been corrected since the original RelBench release. In particular, the labels of `user-ignore` were regenerated in 2026 to fix a temporal leakage, and methods re-evaluated on the corrected task lose 2–6 AUROC points (Hayler et al., 2026). Published baseline results obtained before this fix are therefore not directly comparable on this task.

Baselines Our main points of comparison are the manual and agentic approaches to feature engineering. The manual approach is represented by the data scientist features from the RelBench user study processed by LightGBM (Robinson et al., 2024), AutoGluon, or TabPFN-2.5 (Hudovernik et al., 2026), which we denote by DS+LightGBM, DS+AutoGluon, and DS+TabPFN-2.5. The

Table 2: Test AUROC (×100) on the RelBench classification tasks; full task names are given in Table 7. Red and underlined values mark the best and second-best results in each column. Avg. is the average over tasks, and rank is the average rank over tasks among the methods with results on all tasks. ReFuGe reports one `rel-amazon` churn task, which we place under `user` because its LightGBM baseline matches `user-churn`. †Average over 11 tasks. ‡Evaluated with access to records after the test cutoff on `rel-f1` (Hayler et al., 2026).

	f1		avito		event		trial	amazon		stack		hm		
Method	dnf	top3	click	visit	repeat	ignore	outcome	user	item	engage	badge	churn	Avg. ↑	Rank ↓
Single-table model														
LightGBM	68.56	73.92	53.60	53.05	68.04	79.93	70.09	52.22	62.54	63.39	63.43	55.21	63.66	16.42
Supervised relational models														
GraphSAGE	72.62	75.54	65.90	66.20	76.89	81.62	68.60	70.42	82.81	90.59	88.86	69.88	75.83	6.00
HGT	70.77	70.75	63.76	64.32	64.96	82.47	58.37	66.43	77.97	88.47	86.08	66.95	71.78	12.00
HGT_{PE}	71.17	76.27	64.57	64.95	65.36	81.61	59.21	66.19	78.03	88.17	85.66	65.69	72.24	11.17
RelGNN	75.29	85.69	68.23	66.18	79.61	86.18	71.24	70.99	82.64	90.75	88.98	70.93	78.06	3.17
RelGT	75.87	83.52	68.30	66.78	76.09	81.57	68.61	70.39	82.55	90.53	86.32	69.27	76.65	5.67
Foundation, LLM-based, and hybrid models														
LLM_1	75.80	91.40	59.80	62.70	71.40	69.30	57.40	58.10	62.10	78.00	80.00	59.80	68.82	13.17
LLM_2	80.03	87.11	53.36	54.07	70.11	68.65	59.17	62.55	73.41	81.23	79.99	63.81	69.46	12.75
Rel-Zero	71.84	70.64	62.28	56.17	68.12	61.32	59.02	60.07	64.10	69.46	62.12	55.95	63.42	15.75
TabPFN-2.5	61.21	77.35	54.44	51.45	69.10	70.91	71.74	57.14	59.30	59.66	52.26	54.99	61.63	16.25
Griffin	57.70	82.50	45.90	60.70	71.88	83.27	51.00	62.30	69.00	77.50	73.50	60.20	66.29	14.58
$\text{RT}^{\ddagger}_{\text{zero}}$	81.20	89.30	59.50	61.80	73.22	77.47	51.80	64.00	70.90	75.70	80.10	62.80	70.65	12.17
GNN+TabPFN	64.26	74.67	60.68	65.35	73.58	84.13	56.68	64.30	77.89	87.48	85.34	64.34	71.56	12.00
RDBLearn	70.87	79.69	69.04	65.49	75.04	82.52	71.58	67.57	82.07	89.39	85.26	68.05	75.55	7.83
Tabular models on engineered features														
DS+LightGBM	69.80	82.40	64.27	64.46	75.42	84.23	72.00	67.60	81.80	90.30	86.20	69.00	75.62	7.42
DS+AutoGluon	70.01	82.53	65.66	65.87	74.85	82.31	70.51	68.12	80.45	89.94	85.38	68.71	75.36	8.25
DS+TabPFN-2.5	71.16	77.70	64.21	64.53	76.43	85.70	70.75	66.28	79.83	89.07	85.08	68.31	74.92	8.83
ReFuGe	76.26	83.04	63.20	66.43	77.14	83.40	72.22	67.00	–	87.66	83.80	68.10	75.30†	–
RelAgent	78.34	85.23	68.36	67.79	78.20	87.25	71.86	70.78	82.84	90.41	88.42	71.07	78.38	2.58
Ours	73.56	86.18	69.22	67.58	79.27	82.85	72.58	69.47	81.08	90.37	87.88	70.16	77.52	4.00

agentic approach is represented by ReFuGe and RelAgent. For context, we also report the other methods evaluated by Huang et al. (2026), from whose paper all baseline results are taken. These are LightGBM trained on the entity table alone (Ke et al., 2017; Robinson et al., 2024); supervised relational models GraphSAGE (Hamilton et al., 2017; Robinson et al., 2024), HGT and HGT with positional encodings (Hu et al., 2020; Dwivedi et al., 2025), RelGNN (Chen et al., 2025), and RelGT (Dwivedi et al., 2025); LLM-based predictors LLM_1 (as reported by Ranjan et al., 2026), LLM_2 (Wydmuch et al., 2024), and Rel-Zero (Wu et al., 2025); and foundation and hybrid models TabPFN-2.5 (Grinsztajn et al., 2025), Griffin (Wang et al., 2025), RT_{zero} (Ranjan et al., 2026), GNN+TabPFN (Meyer et al., 2026), and RDBLearn (Xu et al., 2026). For regression, several of these methods report no results, and we add the global and entity median predictors. We omit the closed-source KumoRFM models (Fey et al., 2025; Hudovernik et al., 2026). Their tuning regime is not documented, and KumoRFM-2, as well as models based on the Relational Transformer such as RT_{zero}, uses an evaluation regime that gives access to records after the test cutoff on `rel-f1` (Hayler et al., 2026). With this caveat, KumoRFM-2 reaches 79.60 average AUROC on classification and $\mu_n = 0.822$ on regression, as reported by Huang et al. (2026).

Implementation details We use GPT-5.2 with temperature 1.0 and medium reasoning effort in all experiments except for the study of sensitivity to the LLM, which uses Qwen3-32B; RelAgent uses the same two LLMs. For each program, Optuna runs 24 trials that train LightGBM on the earlier 75% of the training examples and evaluate it on the latest 25%. The best configuration is checked once on the official validation set and kept only if it outperforms the default configuration there. The final model is refit on the training and validation sets together (Section B.5 gives the search space and all other settings). Each model is trained on at most 200,000 rows. Unlike RelAgent, our feature programs cannot read the labels of the training set.

Measuring LLM usage For predictive comparisons with RelAgent, we use its published results. To measure its cost, we ran its released code with GPT-5.2 and the settings of the paper on the classification tasks, with three rollouts per task. The resulting test AUROC is within 2 points of the published results on 10 of 12 tasks (Section D gives the details of this reproduction). We count every LLM call as a request; since a turn of the agent can contain several calls, the number of requests exceeds the number of turns. We take processed input tokens from the API usage records. Each

Table 3: Test MAE on the RelBench regression tasks; full task names are given in Table 7. Values are rounded to two or three decimals depending on the scale of each task, while μ_n and ranks are computed from the published values with three decimals. Red and underlined values mark the best and second-best results in each column, and ties share the mark. μ_n is the MAE normalized by that of LightGBM and averaged over tasks, and rank is the average rank over tasks. ‡Evaluated with access to records after the test cutoff on `rel-f1` (Hayler et al., 2026).

	f1	avito	event	trial		amazon		stack	hm		
Method	position	ctr	attend	adverse	success	user	item	votes	sales	$\mu_n \downarrow$	Rank $\downarrow$
Constant predictors											
Global median	4.40	0.043	0.264	57.53	0.462	16.78	64.23	0.068	0.076	1.062	12.39
Entity median	8.52	0.046	0.269	57.93	0.441	17.42	66.44	0.069	0.078	1.190	14.17
Single-table model											
LightGBM	4.17	0.041	0.264	44.01	0.425	16.78	60.57	0.068	0.076	1.000	9.89
Supervised relational models											
GraphSAGE	4.02	0.041	0.258	44.47	0.400	14.31	50.05	0.065	0.056	0.918	6.67
HGT	4.23	0.046	0.264	45.17	0.443	15.41	55.87	0.068	0.064	0.987	10.89
HGT_{PE}	4.39	0.048	0.261	42.65	0.440	15.86	55.85	0.068	0.064	0.992	10.28
RelGNN	3.80	0.037	0.238	44.46	0.301	14.23	48.77	0.065	0.054	0.861	4.33
RelGT	3.92	0.035	0.250	43.99	0.326	14.27	48.92	0.065	0.054	0.870	4.78
Foundation and hybrid models											
TabPFN-2.5	4.45	0.044	0.527	55.19	0.469	17.51	61.00	0.104	0.096	1.260	14.89
Griffin	4.46	0.050	0.461	78.23	0.463	35.59	53.21	0.092	0.151	1.471	15.56
$\text{RT}^{\ddagger}_{\text{zero}}$	2.90	0.058	0.379	74.00	0.455	18.80	58.00	0.110	0.089	1.240	13.78
RDBLearn	3.83	0.034	0.237	43.91	0.424	14.54	48.56	0.068	0.064	0.906	5.83
Tabular models on engineered features											
DS+LightGBM	3.96	0.044	0.284	40.58	0.407	13.93	41.12	0.065	0.036	0.880	4.94
DS+AutoGluon	4.25	0.045	0.256	44.71	0.430	14.40	45.39	0.068	0.043	0.921	8.06
DS+TabPFN-2.5	4.37	0.044	0.318	47.17	0.432	15.63	47.91	0.080	0.059	1.010	10.56
RelAgent	4.02	0.033	0.241	37.19	0.386	13.95	41.77	0.064	0.035	0.817	2.39
Ours	4.14	0.032	0.254	40.76	0.405	14.03	43.46	0.064	0.037	0.844	3.61

request sends the entire interaction history to the LLM, so the history is counted again with every request. Prompt caching on the provider side reduces the price of the repeated part but does not shorten the context that the model receives. RelAgent reports its own cost by counting each message only once, which gives about 22K input tokens per run on average (Huang et al., 2026).

5 Comparison with manual and agentic feature engineering

We compare our pipeline with the other methods in terms of predictive performance (Section 5.1), LLM usage (Section 5.2), and sensitivity to the choice of the LLM (Section 5.3).

5.1 Predictive performance

Tables 2 and 3 report the results. Compared with the data scientist features processed by LightGBM, our pipeline is better on 10 of 12 classification tasks and by 1.9 AUROC points on average. On regression, it reduces the normalized MAE from 0.880 to 0.844, although it is better on only four of nine tasks. On the 11 classification tasks reported by ReFuGe, our average AUROC is 77.19, compared with 75.30 for ReFuGe. RelAgent remains better on average: by 0.86 AUROC points on classification and by 3.3% in normalized MAE on regression. On classification, this gap comes from two tasks, `driver-dnf` and `user-ignore`, where our pipeline is worse by 4.78 and 4.40 points. On the remaining ten tasks, the average difference is only 0.12 points, and our pipeline is better on four of them. The gap on `user-ignore` is not caused by the label fix mentioned in Section 4: in our reproduction on the corrected task, RelAgent stays within 2 points of its published result. On regression, RelAgent is better on seven of nine tasks, with the largest difference on `study-adverse` (37.19 vs. 40.76). Among the remaining methods, only the supervised relational model RelGNN reaches a higher average AUROC than our pipeline (78.06), and none reaches a lower normalized MAE.

Table 4: Median resource use per attempt on the classification tasks. An attempt is one program for our pipeline and its exploration variant (Section 6.1), one trajectory for the feedback variant (Section 6.2), and one rollout for RelAgent. Input tokens include the history resent with every request. Non-LLM time covers database setup, feature computation, and model fitting; for our pipeline and its variants, it includes one fit of LightGBM with default hyperparameters per program and excludes Optuna tuning.

	Ours	Ours with exploration	Ours with feedback	RelAgent
LLM requests	1	23	33	146
Processed input tokens	4.8K	71.3K	2.14M	5.8M
Completion tokens	5.4K	7.2K	154K	203K
LLM time (s)	35	119	1,950	3,742
Programs with a fitted model	1	1	24	27
Non-LLM time (s)	24	58	610	960

5.2 Number of LLM requests and processed tokens

Table 4 compares the resources used by one attempt: one program in our pipeline and one search rollout in RelAgent. Both systems make five attempts per task. A RelAgent rollout makes 146 LLM requests and processes 5.8M input tokens, because the history of exploration queries, programs, and validation results grows during the search and is sent again with every request. During the rollout, the agent also submits 27 programs for validation, and a model is fitted for each of them. One program of our pipeline requires a single request with a prompt of 4.8K tokens and one model fit. Thus, our pipeline uses 146× fewer requests and about 1,200× fewer processed input tokens per attempt, and its LLM time per attempt is 35 seconds instead of 3,742 seconds. The table does not include Optuna tuning, which adds 24 LightGBM fits and one refit per program. This part of the cost is conventional computation: it requires no LLM requests, and its budget can be changed without regenerating the programs. In RelAgent, in contrast, every additional validation trial requires further LLM requests with a longer context. Its authors report that the agent often recovers a large fraction of its final validation gain within the first 5–10 turns (Huang et al., 2026), so a shorter search would be cheaper, but every turn would still send the accumulated history to the LLM.

5.3 Replacing a top-tier LLM with a smaller open-source one

RelAgent reports its results with Qwen3-32B on all 12 classification tasks, and we run our pipeline with Qwen3-32B in the same way as with GPT-5.2. The average AUROC of our pipeline decreases from 77.52 to 74.93, while that of RelAgent decreases from 78.38 to 70.13. Thus, with Qwen3-32B, our pipeline outperforms RelAgent by 4.8 points. The absolute difference between the two LLMs is smaller for our pipeline on 8 of 12 tasks, with the largest reductions on `driver-top3`, `driver-dnf`, and `user-badge`. The main counterexample is `user-engagement`, where the difference is 10.67 points for our pipeline and 1.30 points for RelAgent (Section E gives per-task results). A likely reason is that a weaker LLM affects only the programs in our pipeline but all decisions in RelAgent.

6 Which responsibilities require an LLM?

RelAgent assigns the LLM several responsibilities that our pipeline leaves out. We reintroduce them into our pipeline one at a time, keeping the rest of the pipeline fixed where possible. Table 5 summarizes the effect of each modification, and Sections F to I provide per-task results.

6.1 Replacing the static description with interactive exploration

To measure how much interactive exploration improves the programs, we replace the static description with the read-only database tools of RelAgent. The LLM may make 1 to 24 tool calls (in practice, it made at least 8) and then writes the program (Section G describes the protocol). Our results show that the aggregate difference is small. Exploration is better on 11 of 21 tasks, improves the average

Table 5: Effect of modifying one component of our pipeline. ΔAUROC is the average difference in AUROC points, and ΔMAE is the average relative MAE reduction in percent, so positive values favor the modification; wins is the number of tasks on which the modification is better. Requests is the median number of LLM requests per program (per trajectory for feedback). *Compared with the initial program of the trajectory, fitted with default hyperparameters and not aggregated.

	Classification		Regression		
Modification	ΔAUROC	Wins	ΔMAE (%)	Wins	Requests
Interactive exploration instead of static description	$+0.29$	$6/12$	-0.15	$5/9$	23
Validation feedback over up to five rounds*	-0.73	$7/12$	$+1.91$	$6/9$	33
Validation-based selection instead of aggregation	-2.75	$1/12$	0.00	$5/9$	1
Default hyperparameters instead of Optuna tuning	-1.08	$1/12$	-0.74	$1/9$	1

AUROC by 0.29 points, and increases the MAE by 0.15% on average. Large and consistent gains occur only on `rel-event`, where exploration improves all three tasks: by 4.33 and 1.82 AUROC points on `user-ignore` and `user-repeat` and by 4.6% in MAE on `user-attendance`. In the traces for these tasks, the LLM inspects the relation between events and users and writes narrow temporal aggregates. The programs generated from the static description instead spread their queries over many relations. Thus, exploration seems to help the LLM focus on a few relevant relations rather than reveal schema information missing from the description. Consistent with this, when the LLM received both the description and the tools in a pilot of 15 attempts, it made only 4.4 tool calls on average (Section G). Exploration also requires 23 requests per program instead of one, $15\times$ more input tokens, and $3\times$ more time (Table 4).

6.2 Adding validation feedback to program generation

To measure how much validation feedback improves the programs, we run our generator in a feedback trajectory similar to a RelAgent rollout. The trajectory consists of at most five rounds. In each round, the LLM can submit several programs to a validation tool, which fits LightGBM with default hyperparameters on a program and returns its validation score. When the LLM stops submitting programs, the round ends, and we ask it to continue improving. At the end of each round, we save the current program, and we finally deploy the first saved program with the best validation score. The LLM receives only scalar scores, without an evaluation workspace or a choice of model. We use neither tuning nor aggregation in this experiment, so feedback is the only difference from the initial program.

The initial program already reaches 75.81 average AUROC, whereas RelAgent reports 66.86 for its agent without feedback. Note that our initial program can be fixed by error-only repair if it fails to execute, while the RelAgent variant cannot use its validation tool, which also reports execution errors. With feedback, the deployed program is better than the initial one on 13 of 21 tasks and reduces the MAE by 1.91% on regression. On classification, however, the average AUROC decreases by 0.73 points (Section H gives per-task results). This decrease comes from a few large drops. For example, on `driver-top3`, the AUROC decreases from 72.25 to 66.42, although the trajectory contains a program with 73.53. Overall, the deployed program reaches the best test score among the saved programs on only 6 of 12 classification and 4 of 9 regression tasks. Thus, the trajectories sometimes produce better programs, but validation often fails to select them. Feedback is also expensive: a trajectory requires 33 requests, 2.14M input tokens, and $43\times$ more time than a single program (Table 4). On a typical task, the best saved program already exists after 43.8% (classification) or 36.7% (regression) of the validation calls.

6.3 Aggregating programs instead of selecting one by validation score

Our pipeline produces five independent programs per task, each with its own LightGBM model. We compare two ways to use them. *Aggregation*, which our pipeline uses, averages predictions for classification and takes their pointwise median for regression. *Selection* deploys only the model whose program has the best score on the official validation set. Selection is only as good as the validation signal, and this signal is weak. Across the five programs of each task, the average rank

correlation between validation and test scores is only 0.19 on classification and 0.39 on regression. For comparison, a correlation of one would mean that validation orders the programs exactly as the test set does. A likely reason is the small size of some validation sets: for example, `user-repeat`, `driver-dnf`, and `driver-top3` have only 268, 566, and 588 validation examples (Section C lists all split sizes). This agrees with the observation of Chen et al. (2026) that more extensive tuning can lead to worse test performance on relational tasks.

As a result, selection decreases the AUROC relative to aggregation on 11 of 12 classification tasks and by 2.75 points on average. The largest drops, 15.13 and 11.61 points, occur on `driver-dnf` and `driver-top3`. This gap consists of two parts. First, aggregation outperforms an average single program by 1.5 points (77.52 vs. 76.01), which is the usual benefit of averaging several models. Second, selection falls another 1.2 points below an average single program (74.77). In other words, choosing a program by its validation score is, on average, worse than choosing one at random. This effect is driven mainly by the `rel-f1` tasks: the selected program is worse than an average program on 5 of 12 tasks (Section F gives per-task results). On regression, where validation and test scores agree more, selection and aggregation perform on par.

6.4 Reusing the programs with different downstream models

In our pipeline, the LLM does not know which model will consume its features. To test whether this information would help, we construct a variant whose prompt also specifies LightGBM and its hyperparameters, and keep the rest of the pipeline unchanged. We ran this comparison on six classification tasks. The programs generated without this information are better on five of them and by 1.58 AUROC points on average (Section F gives per-task results). Thus, the programs need not be tailored to the downstream model and can be reused with different models.

We test this in two ways. First, we compare LightGBM tuned with Optuna against LightGBM with default hyperparameters on the same frozen programs. Tuning improves the results on 19 of 21 tasks, by 1.08 AUROC points and 0.74% in MAE on average, with the largest gains on the `rel-f1` tasks. Second, we replace LightGBM with the tabular foundation model TabPFN-3 (Grinsztajn et al., 2026) without generating new programs. On classification, TabPFN reaches 77.29 average AUROC, close to the 77.52 of tuned LightGBM. On regression, its MAE is 14% higher on average, mostly because of large errors on `driver-position` and `item-sales` (Section I gives per-task results). Both experiments show that the choice and tuning of the downstream model can be handled by conventional methods on frozen programs, without any LLM calls.

7 Conclusion

In this work, we studied which responsibilities of an agentic relational learning system require an LLM. We found that a pipeline in which the LLM only writes feature programs from a static description performs close to RelAgent, an agent that controls the entire pipeline. At the same time, one of our programs requires 146× fewer LLM requests and about 1,200× fewer processed input tokens than one rollout of the agent. When we reintroduced the responsibilities of the agent, interactive exploration rarely added information beyond the static description. On classification, validation feedback and validation-based selection were misled by weak validation signals, and tuning a fixed model family on frozen programs improved the results without LLM calls. We conclude that LLMs are most cost-effective in relational learning as feature engineers, while model tuning and selection can be left to conventional methods.

Limitations and future work Our experiments cover the 21 entity tasks of RelBench and two LLMs, and extending them to other benchmarks, such as RelBench v2 and 4DBInfer, is a natural next step. Following common practice in this field, we compare with the published results of RelAgent, and we use our own reproduction only to measure its cost; a jointly rerun comparison would make the protocols fully matched. Our results do not imply that interactive exploration and validation feedback are never useful: exploration helps on `rel-event` and feedback helps on regression on average, so combining a static description with targeted interaction is a promising direction. A direct comparison of conventional and LLM-based hyperparameter tuning under equal budgets is outside the scope of this work. Finally, estimating the variance over repeated generations of the programs would further strengthen the conclusions.

AI USE STATEMENT

LLMs are the object of this study: GPT-5.2 and Qwen3-32B generate the feature programs in our pipeline and act as the agent in our reproduction of RelAgent, as described in Sections 3 and 4. In addition, we used generative AI tools to assist with drafting and editing the text of the paper and with writing code for the experiments. All AI-assisted text and code were reviewed, tested, and revised by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our experiments use the public RelBench datasets under their original licenses; we collected no new data and conducted no studies with human subjects. The databases contain anonymized records of user activity, and our pipeline only computes aggregate features from them. During feature generation, schema statistics and sample rows of these public databases are sent to the API of the LLM provider.

REPRODUCIBILITY STATEMENT

The code and artifacts of this work are available in an anonymous repository. The repository contains the code of our pipeline and of the additional experiments, including the prompts, the 105 frozen feature programs used in the main experiments, the task configurations, compact per-task and per-program results, the aggregated cost measurements, scripts that rebuild the result tables without LLM calls, and offline tests. Raw LLM conversations, API logs, downloaded datasets, and model checkpoints are not included. Section B describes the database description, the prompt, the checks applied to feature programs, error-only repair, and the training of downstream models. Section C lists the tasks and aggregation metrics, Section D describes the cost accounting and our reproduction of RelAgent, and Sections E to I provide further protocols and per-task results.

REFERENCES

Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. LLM-FE: Automated feature engineering for tabular data with LLMs as evolutionary optimizers. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=qvI35hkpOO. 2, 14

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2623–2631, 2019. 2

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011. 2

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010. 2, 14

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025. 14

Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite message passing for relational deep learning. In *International Conference on Machine Learning*, pp. 8296–8312, 2025. 1, 5, 14

Zhikai Chen, Han Xie, Jian Zhang, Jiliang Tang, Xiang Song, and Huzefa Rangwala. Relatron: Automating relational machine learning over relational databases. In *International Conference on Learning Representations*, 2026. 2, 9, 14

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. *arXiv preprint arXiv:2505.10960*, 2025. 1, 5, 14

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-Tabular: Robust and accurate AutoML for structured data. *arXiv preprint arXiv:2003.06505*, 2020. 14

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023. 1, 14

Matthias Fey, Vid Kocijan, Federico Lopez, Jan Eric Lenssen, and Jure Leskovec. KumoRFM: A foundation model for in-context learning on relational data, 2025. 5, 14

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, Mihir Manium, Rosen Yu, Felix Jablonski, Shi Bin Hoo, Anurag Garg, Jake Robertson, Magnus Bühler, Vladyslav Moroshan, Lennart Purucker, Clara Cornu, Lilly Charlotte Wehrhahn, Alessandro Bonetto, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. TabPFN-2.5: Advancing the state of the art in tabular foundation models. *arXiv preprint arXiv:2511.08667*, 2025. 5, 14

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. *arXiv preprint arXiv:2605.13986*, 2026. 9, 22

Justin Gu, Rishabh Ranjan, Charilaos Kanatsoulis, Haiming Tang, Martin Jurkovic, Valter Hudovernik, Mark Znidar, Pranshu Chaturvedi, Parth Shroff, Fengyu Li, et al. RelBench v2: A large-scale benchmark and repository for relational data. *arXiv preprint arXiv:2602.12606*, 2026. 14

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *International Conference on Machine Learning*, 2024. 14

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017. 5, 14

Sungwon Han, Jinsung Yoon, Sercan O. Arik, and Tomas Pfister. Large language models can automatically engineer features for few-shot tabular learning. In *International Conference on Machine Learning*, 2024. 2, 14

Adrian Hayler, Klemens Flöge, Alan Arazi, Rishabh Ranjan, Jure Leskovec, Felix Birkel, Brendan Roof, Anurag Garg, Kristina Collins, Lydia Sidhoum, et al. Advancing open and reproducible relational learning: RelArena-α, TabPFN-Rel and RPI. *arXiv preprint arXiv:2608.16319*, 2026. 2, 4, 5, 6, 14

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing CAAFE for context-aware automated feature engineering. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44753–44775. Curran Associates, Inc., 2023. 2, 14

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pp. 2704–2710, 2020. 5, 14

Xingyue Huang, Louis Tichelman, Jinwoo Kim, Krzysztof Olejniczak, and Ismail Ilkan Ceylan. RelAgent: Llm agents as data scientists for relational learning. *arXiv preprint arXiv:2605.07840*, 2026. 2, 3, 5, 6, 7, 18

Valter Hudovernik, Federico López, Vid Kocijan, Akihiro Nitta, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. KumoRFM-2: Scaling foundation models for relational learning. *arXiv preprint arXiv:2604.12596*, 2026. 4, 5, 14

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025. 14

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pp. 1–10. IEEE, 2015. 2, 14

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30, 2017. 1, 5

Kyungho Kim, Geon Lee, Juyeon Kim, Dongwon Choi, Shinhwan Kang, and Kijung Shin. ReFuGe: Feature generation for prediction tasks on relational databases with llm agents. In *Proceedings of the ACM Web Conference 2026*, pp. 8501–8504, 2026. 2, 3

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 *International Conference on Machine Learning*, 2026. 14

Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In Sašo Džeroski and Nada Lavrač (eds.), *Relational Data Mining*, pp. 262–291. Springer, 2001. 1, 14

Joe Meyer, Tom Palczewski, Afreen Shaikh, Mahmoud Mohammadi, Dinesh Katupputhur Ramprasath, Karan Paresh, Roshan Upendra Reddy, and Mark Li. Relational in-context learning on structured data via neighborhood aggregation and structural information. In *Proceedings of the AAAI Symposium Series*, volume 9, pp. 219–220, 2026. 5, 14

Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. Optimized feature generation for tabular data via LLMs with decision tree reasoning. In *Advances in Neural Information Processing Systems*, 2024. 2, 14

OpenAI. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, 2025. 2

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. In *International Conference on Learning Representations*, volume 2026, pp. 69149–69174, 2026. 1, 5, 14

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, et al. RelBench: A benchmark for deep learning on relational databases. *Advances in Neural Information Processing Systems*, 37:21330–21341, 2024. 1, 2, 3, 4, 5, 14

Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Ning Li, Jianheng Tang, Yanlin Zhang, Zizhao Zhang, Zunyao Mao, Yakun Song, et al. 4DBInfer: A 4D benchmarking toolbox for graph-centric predictive modeling on relational DBs. *Advances in Neural Information Processing Systems*, 37, 2024. 14

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 *International Conference on Machine Learning*, pp. 64604–64627, 2025. 5, 14

Yanbo Wang, Jiaxuan You, Chuan Shi, and Muhan Zhang. Relational in-context learning via synthetic pre-training with structural prior. *arXiv preprint arXiv:2603.03805*, 2026. 14

Fang Wu, Vijay Prakash Dwivedi, and Jure Leskovec. Large language models are good relational learners. In *Annual Meeting of the Association for Computational Linguistics*, 2025. 5, 14

Marek Wydmuch, Łukasz Borchmann, and Filip Graliński. Tackling prediction tasks in relational databases with LLMs. *arXiv preprint arXiv:2411.11829*, 2024. 5, 14

Linjie Xu, Yanlin Zhang, Quan Gan, Minjie Wang, and David Wipf. No need to train your RDB foundation model. *arXiv preprint arXiv:2602.13697*, 2026. 5, 14

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 2

A Additional related work

Relational deep learning and foundation models Relational deep learning (RDL) represents a database as a temporal heterogeneous graph whose nodes are table rows and whose edges are primary–foreign key links, and trains graph neural networks on this graph end-to-end (Fey et al., 2023). RelBench (Robinson et al., 2024; Gu et al., 2026) and 4DBInfer (Wang et al., 2024) established standard benchmarks for this setting, with heterogeneous GraphSAGE (Hamilton et al., 2017) as the reference RDL model. Subsequent works improve the architecture, for example, with composite message passing that crosses intermediate tables within a single layer in RelGNN (Chen et al., 2025) or with graph transformers such as HGT (Hu et al., 2020) and RelGT (Dwivedi et al., 2025). Relational foundation models are pretrained on many real or synthetic databases to make predictions on new databases and tasks with little or no task-specific training (Wang et al., 2025; Ranjan et al., 2026; Fey et al., 2025; Hudovernik et al., 2026; Kothapalli et al., 2026; Wang et al., 2026). Another group of methods uses LLMs directly as predictors by serializing the relational context of an entity into the prompt (Wydmuch et al., 2024; Wu et al., 2025). All these methods access the relational structure through a predefined mechanism, such as a sampled neighborhood of limited depth, and learn how to aggregate information within it. We report their results in Section 5 to show how a tabular model trained on LLM-generated features performs relative to models that learn directly from the relational structure.

Transforming relational databases into single tables Transforming a relational database into a single table of entity features is a long-standing approach in relational data mining, known as propositionalization (Kramer et al., 2001). Deep Feature Synthesis (DFS) (Kanter & Veeramachaneni, 2015) automates it by recursively applying predefined aggregation and transformation primitives along foreign-key paths, which produces the same generic features regardless of the task. RelBench complemented its RDL baselines with an expert data scientist who explored each database, wrote task-specific SQL features, and trained LightGBM on them (Robinson et al., 2024). The same features were later processed with AutoGluon (Erickson et al., 2020) and TabPFN-2.5 (Grinsztajn et al., 2025) as downstream models (Hudovernik et al., 2026). More recent methods apply tabular foundation models to automatically constructed tables: RDBLearn (Xu et al., 2026) and TabPFN-Rel (Hayler et al., 2026) apply TabPFN to generic relational aggregates, and GNN+TabPFN (Meyer et al., 2026) provides TabPFN with aggregated graph neighborhoods. Chen et al. (2026) compared DFS-based and RDL pipelines in a shared design space and found that RDL does not consistently outperform DFS. Thus, a single table processed by a strong tabular model is a competitive solution, and the open question is how to obtain task-specific features without the cost of manual engineering.

LLMs as feature engineers For single-table data, CAAFE (Hollmann et al., 2023) prompts an LLM with a dataset description to generate Python code for new features and keeps a feature only if it improves cross-validated performance. FeatLLM (Han et al., 2024) asks an LLM to write rules that are turned into binary features for few-shot learning with a simple downstream model. OCTree (Nam et al., 2024) iteratively refines generated rules using decision tree reasoning as feedback, and LLM-FE (Abhyankar et al., 2026) treats feature construction as evolutionary program search with the validation score as fitness. More broadly, LLM agents for machine learning engineering, such as AIDE (Jiang et al., 2025) and DS-Agent (Guo et al., 2024), automate entire pipelines by iteratively writing, executing, and revising code, and MLE-bench (Chan et al., 2025) evaluates such agents on Kaggle competitions. Most of these systems rely on iterative generation guided by validation feedback, which is the design choice we examine for relational data.

Model selection with a noisy validation signal Selecting among many candidates by their validation scores overfits to the validation set, and the resulting selection bias grows with the number of compared candidates and the variance of the validation estimate (Cawley & Talbot, 2010). In relational learning, Chen et al. (2026) observed that more extensive tuning can lead to worse test performance and proposed loss-landscape metrics to avoid brittle checkpoints. Averaging several models instead of selecting one of them is a standard way to reduce the variance of this decision, and our pipeline applies it to independently generated feature programs.

B IMPLEMENTATION DETAILS

B.1 STATIC DATABASE DESCRIPTION

Before any statistic is computed, the database is truncated at the validation cutoff, so neither the sample rows nor the statistics contain information from the validation or test periods. Table 6 lists the contents of the description. All tasks use the same template, and only the task-specific fields differ.

Table 6: Contents of the static database description, together with the output contract included in the same prompt.

Block	Information provided to the LLM
Task	Database and task names, official task description, task type, entity, time, and target columns, evaluation metric, split sizes, and validation and test cutoffs.
Target summary	Number of training examples and distinct entities, time range, and class balance for classification or target quantiles and the fraction of zeros for regression. Individual labeled examples are not shown.
Schema	All tables and columns with their types, primary keys, time columns, and foreign-key relations.
Column statistics	Number of rows, null fraction, number of distinct values, minimum and maximum of numerical and date columns, mean of numerical columns, average length of strings, and the eight most frequent values of low-cardinality columns.
Sample rows	Three rows per table, with long strings truncated.
Output contract	The table of prediction keys, the required output format, the point-in-time rule, and the SQL dialect.

B.2 PROMPT AND OUTPUT FORMAT

The system message instructs the LLM to read the description and return one feature program, and states that there is no exploration, no validation feedback, and no further turn except for error-only repair. The user message is filled automatically from the RelBench metadata and the truncated database:

> `TASK`: [dataset], [task], [task type], [official description].
> `Prediction key`: ([entity columns], [timestamp]).
> `Target`: [target column]; `metric`: [metric].
> `Splits and cutoffs`: [train/validation/test metadata].
>
> `DATABASE`: for each table, its keys, time column, foreign keys, columns, statistics, and three sample rows.
>
> `EXECUTION`: `eval_table` contains only the rows to score and never a label. Return named DuckDB `SELECT` queries. Every query must be anchored on `eval_table`, return the prediction key exactly once, and add at least one feature column. Historical rows must satisfy $t_{\text{event}} < t_{\text{prediction}}$; bounded windows must subtract a past `INTERVAL` from the prediction time.
>
> `OUTPUT`: one JSON object with fields `plan` and `feature_queries`, where each feature query has fields `name` and `sql`. Return 3–8 feature queries and no text outside the JSON object.

The field `plan` contains a short description of the joins and expected signals, and each `sql` field contains one DuckDB `SELECT` statement. The LLM is not asked to choose a model, hyperparameters, categorical columns, or a target transformation.

B.3 EXECUTION OF FEATURE PROGRAMS

For each split, the unlabeled prediction keys are registered as `eval_table`. Each query of a program returns the key columns and some feature columns. The feature columns are prefixed with the query name, and the outputs of all queries are left-joined to the prediction keys, which produces a feature table with the same rows as the split. Before a model is fitted, a validator checks that

- all queries are read-only `SELECT` statements and do not access the labeled training table;
- every query is anchored on `eval_table`, returns all key columns, and returns at most one row per key;
- queries with forward-looking time conditions or intervals are rejected, so that events must be strictly earlier than the prediction time;
- intermediate results are aggregated before being joined to the prediction keys; join explosions, missing feature columns, unknown columns, parser errors, and timeouts are treated as failures.

Numerical feature columns are imputed with the training median, and non-numerical columns are passed to LightGBM as categorical features with vocabularies built on the training set. The temporal check is syntactic: it catches common forward-looking predicates but does not prove that an arbitrary query is free of leakage.

B.4 Error-only repair

Each program is first executed without fitting a model. If parsing, validation, or execution fails, the LLM receives the task, the output contract, its previous program, and the error message, and is asked to fix the technical problem while keeping the intended features. At most two repairs are allowed. The repair request contains no labels, predictions, validation scores, or test information. With GPT-5.2, 91 of 105 programs were runnable immediately and the remaining 14 after one repair; the most common initial failure was a SQL timeout (6 of 14).

B.5 Downstream models, tuning, and aggregation

The default model is LightGBM with 200 trees, learning rate 0.05, maximum depth 6, minimum child size 20, and row and column subsampling rates of 0.8. For regression, we use the L1 objective. Optuna runs 24 trials with the TPE sampler, seeded with the index of the program. The search space consists of the number of trees (50–500), the learning rate (0.01–0.3, log scale), the maximum depth (2–10), the minimum child size (1–100), the row and column subsampling rates (0.5–1), and the L1 and L2 regularization coefficients (0–10). Each trial fits LightGBM on the earlier 75% of the training examples and evaluates it on the latest 25%; if the task has no usable time column, 25% of the entities are held out instead. The best configuration is then fitted on the full training set and evaluated once on the official validation set. It is kept only if its validation score is higher than that of the default configuration. The final model is refit on the training and validation sets together. The number of training rows is capped at 200,000: for classification, we draw a stratified subsample, and for regression, a uniform subsample without replacement, both with a fixed seed. For classification, the aggregated prediction is $\widehat{p}(v,t) = \frac{1}{K}\sum_{k=1}^{K} p_k(v,t)$, where p_k is the probability predicted by the k-th model; for regression, it is the median of the K predictions. A program that fails after the repair budget is recorded as a failure and is not resampled.

B.6 LLM settings

GPT-5.2 is queried with temperature 1.0, medium reasoning effort, and at most 16,000 output tokens per request. Qwen3-32B is served with vLLM 0.28.0 with tensor parallelism over two GPUs and a context length of 40,960 tokens.

C Tasks and aggregation

Table 7 lists the 21 tasks together with the sizes of their splits. For regression, the normalized MAE of a method a is $\mu_n(a) = \frac{1}{J}\sum_{j=1}^{J} \mathrm{MAE}_{a,j}/\mathrm{MAE}_{\mathrm{LightGBM},j}$, where J is the number of tasks. Ranks are computed on each task among the methods with results on all tasks, with tied methods receiving the average of their ranks, and are then averaged over tasks. In Section 6, the difference between two regression configurations on task j is the relative MAE reduction $100\,(\mathrm{MAE}_{\mathrm{orig},j} - \mathrm{MAE}_{\mathrm{mod},j})/\mathrm{MAE}_{\mathrm{orig},j}$.

Table 7: RelBench tasks used in our experiments and the numbers of examples in their splits in RelBench 2.1.2. Some of these numbers differ from those in the original RelBench paper because several tasks were updated after its publication. Short names are used in the column headers of Tables 2 and 3 and are unique within each task type.

Database	Task	Short name	Train	Validation	Test
Classification					
rel-f1	driver-dnf	dnf	11,411	566	702
rel-f1	driver-top3	top3	1,353	588	726
rel-avito	user-clicks	click	59,454	21,183	47,996
rel-avito	user-visits	visit	86,619	29,979	36,129
rel-event	user-repeat	repeat	3,842	268	246
rel-event	user-ignore	ignore	19,239	2,013	1,958
rel-trial	study-outcome	outcome	11,994	960	825
rel-amazon	user-churn	user	4,708,383	409,792	351,885
rel-amazon	item-churn	item	2,536,014	177,689	166,842
rel-stack	user-engagement	engage	1,360,850	85,838	88,137
rel-stack	user-badge	badge	3,386,276	247,398	255,360
rel-hm	user-churn	churn	3,832,692	76,556	74,575
Regression					
rel-f1	driver-position	position	7,453	499	760
rel-avito	ad-ctr	ctr	5,100	1,766	1,816
rel-event	user-attendance	attend	19,239	2,013	1,958
rel-trial	study-adverse	adverse	43,335	3,596	3,098
rel-trial	site-success	success	151,407	19,740	22,617
rel-amazon	user-ltv	user	4,708,383	409,792	351,885
rel-amazon	item-ltv	item	2,707,679	166,978	178,334
rel-stack	post-votes	votes	2,453,921	156,216	160,903
rel-hm	item-sales	sales	5,488,184	105,542	105,542

D COST ACCOUNTING

We count every completed LLM request, including repair and tool-use requests, and take input and completion tokens from the API usage records. Input tokens are summed over all requests, so the history resent with every request, including cached history, is counted every time. LLM time is the elapsed time of the requests, and non-LLM time covers database setup, feature computation, and model fitting. For each configuration, one attempt consists of the following requests and model fits:

- *Ours*: the generation request, possible repair requests, and one fit of the resulting program;
- *Ours with exploration*: the initial request, all tool-use requests, the final answer, possible repairs, and one fit of the final program;
- *Ours with feedback* and *RelAgent*: all search, validation, continuation, and repair requests, and a fit for every program submitted to the validation tool.

For each task, we average resource use over the attempts of this task and report the median over the 12 classification tasks in Table 4.

To measure the cost of RelAgent, we ran its released code with GPT-5.2 and the settings described in its paper on all 12 classification tasks, with three rollouts per task. The test AUROC of this reproduction is within 2 points of the published results on 10 of 12 tasks, and the two exceptions are the `rel-f1` tasks. We use this reproduction only to measure the cost of a rollout, and we use the published results for all predictive comparisons. Optuna tuning (24 trials and one refit per program) makes no LLM requests and is excluded from Table 4.

E PER-TASK RESULTS WITH QWEN3-32B

In the study of sensitivity to the LLM, the prompt, repair budget, Optuna search, training-row cap, number of programs, and aggregation rule are the same for both LLMs. With Qwen3-32B, 6 of 60 programs failed after the repair budget (one parse failure and five rejections by the validator) and were not resampled, so some ensembles contain fewer than five models. Table 8 reports the per-task results. The differences for RelAgent are computed from the results with GPT-5.2 and Qwen3-32B reported by Huang et al. (2026).

Table 8: Test AUROC ($\times 100$) of our pipeline with GPT-5.2 and Qwen3-32B, and the absolute difference between the two LLMs for our pipeline and for RelAgent. Red marks the smaller difference in each row. Differences are computed from unrounded scores for our pipeline and from published scores for RelAgent. The Average row of the difference columns averages absolute differences, and the last row gives the difference between average AUROCs.

		Ours		Absolute difference	
Database	Task	GPT-5.2	Qwen3-32B	Ours	RelAgent
rel-f1	driver-dnf	73.56	72.00	1.56	20.67
rel-f1	driver-top3	86.18	80.94	5.24	27.19
rel-avito	user-clicks	69.22	68.41	0.81	3.26
rel-avito	user-visits	67.58	66.87	0.71	3.17
rel-event	user-repeat	79.27	77.97	1.30	3.47
rel-event	user-ignore	82.85	86.01	3.15	11.34
rel-trial	study-outcome	72.58	71.32	1.26	0.50
rel-amazon	user-churn	69.47	67.37	2.10	1.56
rel-amazon	item-churn	81.08	79.00	2.08	2.92
rel-stack	user-engagement	90.37	79.70	10.67	1.30
rel-stack	user-badge	87.88	81.81	6.07	23.12
rel-hm	user-churn	70.16	67.79	2.38	0.45
Average		77.52	74.93	3.11	8.25
Difference of averages				2.59	8.25

F PER-TASK RESULTS OF THE COMPONENT ANALYSIS

Table 9 reports the per-task results summarized in the first, third, and fourth rows of Table 5, together with the average result of a single program used in Section 6.3. Validation-based selection, single programs, and default hyperparameters reuse the five programs of our pipeline, while the exploration variant generates its own five programs; all 21 tasks have complete ensembles of five models in this variant. The rank correlations reported in Section 6.3 are Spearman correlations between the validation and test scores of the five programs of each task, averaged over tasks.

Table 9: Per-task effect of the modifications from Section 6. The first column shows the results of our pipeline: AUROC ($\times 100$) for classification and MAE for regression. The other columns show differences from our pipeline: AUROC points for classification and relative MAE reduction in percent for regression, so positive values favor the modification. *Single program* is the average result of the five programs of our pipeline, each evaluated without aggregation. MAE is shown with two decimals for `driver-position`, `study-adverse`, `user-ltv`, and `item-ltv` and with four decimals for the other regression tasks, whose values are small. Differences are computed from unrounded values.

Database	Task	Ours	Interactive exploration	Validation-based selection	Single program	Default hyperparameters
Classification						
rel-f1	driver-dnf	73.56	−0.09	−15.13	−5.57	−4.63
rel-f1	driver-top3	86.18	−1.35	−11.61	−5.63	−3.79
rel-avito	user-clicks	69.22	−0.22	−2.18	−1.53	−1.41
rel-avito	user-visits	67.58	+0.06	−0.27	−0.29	−0.23
rel-event	user-repeat	79.27	+1.82	−0.71	−1.01	−0.19
rel-event	user-ignore	82.85	+4.33	−1.11	−0.84	−1.48
rel-trial	study-outcome	72.58	−1.17	−1.93	−1.49	−0.96
rel-amazon	user-churn	69.47	+0.03	+0.41	−0.46	−0.07
rel-amazon	item-churn	81.08	+0.61	−0.07	−0.31	−0.03
rel-stack	user-engagement	90.37	−0.49	−0.14	−0.14	−0.04
rel-stack	user-badge	87.88	+0.04	−0.11	−0.28	−0.12
rel-hm	user-churn	70.16	−0.12	−0.13	−0.49	+0.02
Average		77.52	+0.29	−2.75	−1.50	−1.08
Regression						
rel-f1	driver-position	4.14	−4.22	−2.90	−3.07	−4.16
rel-avito	ad-ctr	0.0325	−0.51	−1.95	−0.97	−0.72
rel-event	user-attendance	0.2544	+4.65	+3.02	−2.18	−0.88
rel-trial	study-adverse	40.76	−1.21	−2.21	−1.54	−1.71
rel-trial	site-success	0.4046	−2.33	+1.25	−0.11	+2.32
rel-amazon	user-ltv	14.03	+0.67	+0.22	−0.93	−0.03
rel-amazon	item-ltv	43.46	+0.29	+1.95	−1.06	−1.04
rel-stack	post-votes	0.0638	+0.32	−0.01	−0.61	−0.07
rel-hm	item-sales	0.0365	+0.98	+0.66	−1.31	−0.34
Average		–	−0.15	0.00	−1.31	−0.74

Table 10 reports the comparison from Section 6.4 between our programs and programs generated with a prompt that also specifies the downstream LightGBM model and its hyperparameters. We report the six classification tasks for which both variants were run under the same protocol.

Table 10: Test AUROC ($\times 100$) of our pipeline and of its variant whose prompt specifies the downstream model. Δ is the difference between the two.

Database	Task	Ours	Model specified in the prompt	Δ
rel-f1	driver-dnf	73.56	70.52	+3.04
rel-f1	driver-top3	86.18	82.53	+3.65
rel-avito	user-clicks	69.22	69.49	−0.27
rel-event	user-repeat	79.27	77.95	+1.32
rel-event	user-ignore	82.85	82.32	+0.53
rel-trial	study-outcome	72.58	71.40	+1.18
Average		77.28	75.70	+1.58

G Exploration protocol and traces

In the exploration variant, the static description is replaced with read-only access to the database, while the task metadata, point-in-time rule, and output contract stay the same. The LLM can call `list_tables`, `describe_table`, `sample_rows`, and `execute_query`, where the last tool runs one read-only DuckDB `SELECT` query. The LLM must make between 1 and 24 tool calls, one call per turn and at most 28 turns in total. A query has a timeout of 30 seconds and returns at most 20 rows and 8,000 characters, with long cell values truncated. After exploration, the LLM returns one program; repair requests have no tools and do not include the exploration transcript. Across the 105 programs, the LLM made between 8 and 24 tool calls, 20.87 on average (median 22), and 38 programs used the full budget of 24 calls.

Unlike the static description, the tools give access to individual rows of the labeled training table and to the unlabeled validation keys, but not to validation or test labels or to any scores. Since this variant changes both how the context is acquired and what the LLM can see, it measures the effect of the whole exploration setup rather than the effect of tools alone.

Besides the gains on `rel-event` discussed in Section 6.1, exploration is worse on `driver-top3` (-1.35 AUROC points), `study-outcome` (-1.17), `driver-position` (MAE $+4.22\%$), and `site-success` (MAE $+2.33\%$), and the remaining differences are small (Table 9). In the traces for `rel-event`, the LLM repeatedly inspects the relation between events and users, the time coverage, and a few behavioral columns before writing narrow temporal aggregates. In contrast, the programs generated from the static description more often spread their queries over many relations. We found no evidence that the tools reveal schema information missing from the description, which already lists all tables, keys, types, and column statistics; the tools additionally expose labeled training rows. This interpretation is based on the traces rather than on a controlled experiment.

In a pilot run of 15 attempts, we provided the LLM with both the static description and the tools. The LLM made 66 tool calls in total: 4.4 per attempt on average, with a median of 2 and a range from 1 to 15. In 7 of 15 attempts, it stopped after the single required call. Thus, when the description is available, the LLM rarely needs to explore the database.

H Validation feedback protocol and per-task results

The feedback variant follows one trajectory per task (Section 6.2). We denote the programs saved after rounds one to five by T1–T5; a trajectory can stop before the fifth round. The deployed program is the first saved program that attains the best validation score. The oracle, which is the saved program with the best test score, is reported only for analysis. Table 11 reports the per-task results.

The deployed program reaches the test score of the oracle on 6 of 12 classification tasks and 4 of 9 regression tasks; on `item-sales`, the deployed program differs from the oracle but has the same

test score. On `driver-dnf` and `driver-top3`, validation moves away from much better early programs, and these two tasks account for more than the entire average decrease on classification.

Table 11: Per-task results of validation feedback. *Initial* is the program before feedback, *Deployed* is the first saved program that attains the best validation score, and *Oracle* is the saved program with the best test score. Δ is the difference between the deployed and initial programs: AUROC points for classification and relative MAE reduction in percent for regression. *Round* is the round after which the deployed program was saved, and *Found at* is the position of the validation call that first produced the oracle, as a fraction of all validation calls in the trajectory. Classification reports AUROC ($\times 100$), and regression reports MAE. MAE is shown with two decimals for `driver-position`, `study-adverse`, `user-ltv`, and `item-ltv` and with four decimals for the other regression tasks, whose values are small.

Database	Task	Initial	Deployed	Oracle	Δ	Round	Found at (%)
Classification							
rel-f1	driver-dnf	70.72	66.73	68.72	−3.99	T2	3.6
rel-f1	driver-top3	72.25	66.42	73.53	−5.83	T2	36.8
rel-avito	user-clicks	66.35	65.92	66.17	−0.43	T2	42.1
rel-avito	user-visits	66.91	67.00	67.00	+0.09	T2	46.7
rel-event	user-repeat	80.35	78.94	78.94	−1.41	T1	10.3
rel-event	user-ignore	85.24	82.24	83.37	−3.00	T4	23.3
rel-trial	study-outcome	71.24	72.18	72.18	+0.94	T4	76.9
rel-amazon	user-churn	69.47	70.38	70.38	+0.91	T3	75.0
rel-amazon	item-churn	80.37	82.32	82.32	+1.95	T4	53.8
rel-stack	user-engagement	89.88	90.06	90.10	+0.18	T4	18.8
rel-stack	user-badge	87.58	88.11	88.11	+0.53	T5	100.0
rel-hm	user-churn	69.32	70.57	70.64	+1.25	T2	45.5
Average		75.81	75.07	75.96	−0.73		
Regression							
rel-f1	driver-position	4.25	3.97	3.94	+6.64	T3	36.7
rel-avito	ad-ctr	0.0328	0.0325	0.0322	+1.14	T4	84.6
rel-event	user-attendance	0.2480	0.2592	0.2492	−4.50	T2	11.1
rel-trial	study-adverse	41.68	40.65	40.65	+2.47	T1	50.0
rel-trial	site-success	0.4004	0.4064	0.4055	−1.50	T3	14.3
rel-amazon	user-ltv	14.19	13.88	13.87	+2.23	T4	81.0
rel-amazon	item-ltv	45.86	43.32	43.32	+5.54	T3	82.1
rel-stack	post-votes	0.0639	0.0641	0.0641	−0.36	T1	35.3
rel-hm	item-sales	0.0367	0.0347	0.0347	+5.49	T4	33.3
Average		–	–	–	+1.91		

I TABPFN ON FROZEN PROGRAMS

We replace LightGBM with TabPFN-3 (Grinsztajn et al., 2026) with 16 estimators and automatic handling of categorical features. TabPFN uses the training and validation sets as its context, capped at 200,000 rows. The programs, splits, and aggregation rule are the same as for LightGBM.

Table 12 reports the results. On classification, TabPFN is better than LightGBM with default hyperparameters on 10 of 12 tasks, by 0.85 AUROC points on average, but worse than tuned LightGBM by 0.23 points. On regression, TabPFN is better than tuned LightGBM only on two of nine tasks and has large errors on `driver-position` and `item-sales`. The tuned LightGBM therefore remains the better choice in our experiments. Since the validation set is part of the context of TabPFN, its validation scores are not held out, so we report only the aggregated models and not validation-based selection.

Table 12: Results of TabPFN-3 and of LightGBM with default and tuned hyperparameters on the same frozen programs. Classification reports AUROC ($\times 100$), and regression reports MAE. MAE is shown with two decimals for `driver-position`, `study-adverse`, `user-ltv`, and `item-ltv` and with four decimals for the other regression tasks, whose values are small. The last two columns show the differences of TabPFN from the two LightGBM configurations, computed from unrounded values: AUROC points for classification and relative MAE reduction in percent for regression, so positive values favor TabPFN.

		LightGBM			TabPFN vs. LightGBM	
Database	Task	Default	Tuned	TabPFN	Default	Tuned
Classification						
`rel-f1`	`driver-dnf`	68.93	73.56	73.39	+4.46	−0.18
`rel-f1`	`driver-top3`	82.39	86.18	85.46	+3.07	−0.72
`rel-avito`	`user-clicks`	67.81	69.22	68.33	+0.52	−0.89
`rel-avito`	`user-visits`	67.35	67.58	67.38	+0.04	−0.20
`rel-event`	`user-repeat`	79.08	79.27	79.89	+0.80	+0.62
`rel-event`	`user-ignore`	81.37	82.85	82.45	+1.09	−0.40
`rel-trial`	`study-outcome`	71.62	72.58	73.19	+1.57	+0.61
`rel-amazon`	`user-churn`	69.40	69.47	69.42	+0.02	−0.05
`rel-amazon`	`item-churn`	81.05	81.08	81.19	+0.14	+0.11
`rel-stack`	`user-engagement`	90.33	90.37	90.62	+0.29	+0.24
`rel-stack`	`user-badge`	87.76	87.88	86.18	−1.58	−1.70
`rel-hm`	`user-churn`	70.18	70.16	69.99	−0.18	−0.17
Average		76.44	77.52	77.29	+0.85	−0.23
Regression						
`rel-f1`	`driver-position`	4.31	4.14	6.02	−39.63	−45.44
`rel-avito`	`ad-ctr`	0.0327	0.0325	0.0315	+3.74	+3.05
`rel-event`	`user-attendance`	0.2566	0.2544	0.2409	+6.10	+5.27
`rel-trial`	`study-adverse`	41.46	40.76	43.70	−5.42	−7.22
`rel-trial`	`site-success`	0.3952	0.4046	0.4404	−11.42	−8.84
`rel-amazon`	`user-ltv`	14.03	14.03	14.16	−0.88	−0.91
`rel-amazon`	`item-ltv`	43.92	43.46	47.96	−9.22	−10.36
`rel-stack`	`post-votes`	0.0638	0.0638	0.0677	−6.05	−6.13
`rel-hm`	`item-sales`	0.0366	0.0365	0.0580	−58.23	−58.77
Average		–	–	–	−13.45	−14.37