Télécom Paris

PRJ_4SD06_TP • Projet

Beyond Flat Tables: Does Structure Matter?

Anatomy of the Relational Advantage in Relational Deep Learning

Team

ZHENG Tong
NARANJO RODRIGUEZ Natalia
SOUSSOU Koffi Désiré
CIZERON Jules
RYSZFELD Sacha

Datasets: `rel-f1`, `rel-amazon`, `hm` (RelBench)

Instructor: **Nikola Simidjievski**

June 2026

Abstract

We study whether modelling relational database structure improves prediction over flat, single-table baselines, using RelBench to compare gradient-boosted trees with heterogeneous graph neural networks (Relational Deep Learning).

For `rel-f1`, ZHENG Tong analyzes the role of relational structure through controlled ablations over depth, tables, and content, showing that most of the predictive signal can be captured by a minimal one-hop subgraph.

For `rel-amazon`, NARANJO RODRIGUEZ Natalia investigates the interaction between feature engineering and relational modelling at scale, showing that relational structure provides benefits in low-feature settings, but that engineered tabular features can capture most of the same signal.

For `rel-amazon`, SOUSSOU Koffi Désiré investigates the `review-rating` autocompletion task: predicting missing 1 to 5 star ratings from the customer product review graph without access to review text, showing that heterogeneous message passing provides a measurable advantage over flat features, while dampened class weighting is essential to avoid collapse on the dominant 5 star class.

Contents

1 Introduction, Hypotheses, and Motivation **3**

2 Proposed Plan and Team Organization **3**

3 Implementation Process **4**

3.1 Common Experimental Framework 4

3.2 ZHENG Tong: `rel-f1` Classification Tasks 5

3.2.1 Tasks, Data and Baselines 5

3.2.2 Anatomy of the Advantage: a Controlled Ablation 5

3.3 NARANJO RODRIGUEZ Natalia: `rel-amazon` Classification 5

3.3.1 Tasks, Data and Baselines 5

3.3.2 Relational Model and Graph Construction 6

3.4 SOUSSOU Koffi Désiré: `rel-amazon` Review-Rating Autocompletion 7

3.4.1 Task, Data, and Leakage Constraints 7

3.4.2 Flat Baseline: LightGBM 7

3.4.3 Relational Model: Heterogeneous GNN 7

3.4.4 Results 8

3.5 CIZERON Jules: `rel-f1` Regression 8

3.5.1 Task Definition and Relation to the Classification Tasks 8

3.5.2 Flat Baseline 9

3.5.3 Relational Model 9

3.6 RYSZFELD Sacha: `hm` Regression 9

3.6.1 Task, Data and Baselines 9
3.6.2 Relational Model 10
3.6.3 Anatomy of the (Non-)Advantage: a Controlled Ablation 10

4 Challenges and Findings **11**
4.1 ZHENG Tong: Findings on `rel-f1` Classification 11
4.2 NARANJO RODRIGUEZ Natalia: Findings on `rel-amazon` Classification 12
4.3 Findings from the Other Tasks 14
4.4 SOUSSOU Koffi Désiré: Findings on `rel-amazon` Regression 14
4.5 CIZERON Jules: Findings on `rel-f1` Regression 15
4.6 RYSZFELD Sacha: Findings on `hm` Regression 15

5 Alignment with the Initial Plan **16**
5.1 Did We Manage To Do Everything We Set Out To Do? 16
5.2 What Would We Do Differently? Lessons Learned 19

6 Conclusion **19**

References **20**

Contribution Section **20**

Use of AI Tools **22**

1 Introduction, Hypotheses, and Motivation

Many real-world machine-learning problems are not naturally stored as one flat feature matrix. Instead, information is distributed across multiple linked tables: for example, users, items, events, transactions, races, drivers, and results. Flattening such data into a single table usually requires manual feature engineering, aggregation choices, and a fixed prediction horizon. The motivation of this project is to test whether models that preserve the relational structure of the data can exploit signals that are difficult to capture through entity-table-only features.

We study this question through RelBench, a benchmark designed for machine learning on relational databases. Our general comparison is between:

- **flat tabular baselines**, mainly LightGBM and XGBoost, trained on entity-level features;
- **relational deep learning models**, where the database is converted into a heterogeneous temporal graph and processed by a graph neural network.

Our hypotheses are:

1. **H1 – relational structure matters.** If useful predictive information is distributed across linked tables, a GNN operating on the primary–foreign key graph should outperform a flat entity-only baseline.
2. **H2 – the benefit is task-dependent.** Structure should help more when the target depends on recent relational history than when it can be approximated from static entity identity or simple calendar features.
3. **H3 – flat baselines remain strong.** Gradient-boosted trees can be competitive, especially when the entity table already contains strong identity or demographic signals.
4. **H4 – if structure helps, the advantage is localizable.** A graph neural network nominally exploits the whole database over several hops, but the useful signal may live in a small subschema. We hypothesise that the advantage can be attributed to a few tables reached in few hops, rather than to deep, whole-database message passing.

The `rel-f1` classification track (ZHENG Tong) is structured precisely to test H4: rather than stopping at the flat-vs-GNN comparison, it treats the relational structure as a controllable variable and removes it piece by piece.

The `rel-amazon` classification track (NARANJO RODRIGUEZ Natalia) complements this perspective by focusing on the interplay between feature engineering and relational structure at scale: rather than isolating graph components, it evaluates whether rich temporal and behavioral features can capture similar signals to those learned through relational message passing, and to what extent graph-based models still provide an advantage in such feature-rich settings.

2 Proposed Plan and Team Organization

The initial project plan covered a nine-week period, from the kickoff to the final submission. The main milestones were: literature review and research plan, data pipeline, flat baselines, GNN prototype, preliminary results, extra experiments, draft report, and final submission.

Table 1. Team workload split.

Team member	Dataset	Task type	Assigned work
ZHENG Tong	`rel-f1`	Classification	`driver-dnf`, `driver-top3`; flat baselines; GNN/RDL baselines; result comparison and interpretation.
NARANJO RODRIGUEZ Natalia	`rel-amazon`	Classification	`user_churn`, `item_churn`; flat baselines; GNN/RDL baselines; comparison across feature regimes; result analysis and interpretation.
SOUSSOU Koffi Désiré	`rel-amazon`	Regression	`review-rating` auto-completion: predicting missing customer product ratings without using review specific columns (review-text, summary) to avoid leakage; result comparison and interpretation.
CIZERON Jules	`rel-f1`	Regression	`driver-position`: entity-only XGBoost baseline (Optuna-tuned); HeteroGraphSAGE relational model over the full 9-table graph; flat-vs-GNN comparison and interpretation.
RYSZFELD Sacha	`hm`	Regression	`item-sales`: weekly article-level sales regression; LightGBM flat baseline with lag features; HeteroSAGE relational model; controlled ablation study (depth, features, aggregator, density, hop decay); result comparison and interpretation.

3 Implementation Process

3.1 Common Experimental Framework

The project uses RelBench for standardized relational datasets, task definitions, train/validation/test splits, and evaluation. The repository is organized around notebooks for experimentation, a shared Python environment, and generated reports/figures. For each task, we follow the same broad workflow:

1. load the RelBench dataset and task;
2. inspect the entity table, target definition, prediction window, and split statistics;
3. implement one or more flat baselines;
4. implement a relational model based on the heterogeneous temporal graph;
5. compare models with the official RelBench evaluator and task-specific metrics.

3.2 ZHENG Tong: `rel-f1` Classification Tasks

3.2.1 Tasks, Data and Baselines

Both tasks are binary, with entity table `drivers` and a 30-day prediction window (Table 2). `driver-dnf` (a "did not finish" in the next 30 days) is large and majority-positive; `driver-top3` (a top-3 qualification) is small (∼17% positive), so overfitting is a concern. I report the threshold-free metrics ROC-AUC and average precision (AP); ROC-AUC is also the safe cross-paper metric, since the package and the paper use opposite `driver-dnf` label conventions.

Table 2. Split statistics for the two `rel-f1` classification tasks.

Task	Train	Tr. pos.	Val	Val pos.	Test	Test pos.
`driver-dnf`	11,411	0.880	566	0.779	702	0.705
`driver-top3`	1,353	0.171	588	0.202	726	0.176

Flat baselines. LightGBM and XGBoost on entity-only features (`driverRef`, `nationality`, age, and calendar year/month), Optuna-tuned and early-stopped on validation ROC-AUC.

Relational model. I build the heterogeneous temporal graph with `make_pkey_fkey_graph` (one node type per table, typed PK/FK edges) and train a HeteroGraphSAGE (per-table encoder, a temporal encoder, two SAGE layers, linear head). A temporal `NeighborLoader` (`disjoint=True`) samples only neighbors *before* each seed's prediction time, preventing future leakage; `driver-top3` needed a smaller model (64 channels) to curb overfitting.

3.2.2 Anatomy of the Advantage: a Controlled Ablation

The flat-vs-GNN comparison changes two things at once – model family *and* use of structure – so it cannot attribute the gain to structure. To test H4 I hold the model fixed (always the GNN) and remove structure piece by piece, through one function, `train_and_eval(num_neighbors, seed, use_temporal)`, fed either the full graph or a sub-graph. Three axes are varied: **depth** (0/1/2 hops, where 0 hops = no message passing, the no-structure control); **breadth** (restricting to the entity table plus one neighbor table at a time with `edge_type_subgraph` – a driver is one hop from exactly `results`, `qualifying` and `standings`); and **content** (dropping the temporal encoder, so the model sees *which* rows exist but not *when*). Together they define a *minimal model* (entity + the single best linked table, one hop), compared against the full nine-table, two-hop GNN. All code is consolidated in one notebook, `rel-f1_structure_anatomy.ipynb`.

3.3 NARANJO RODRIGUEZ Natalia: `rel-amazon` Classification

3.3.1 Tasks, Data and Baselines

Two binary classification tasks are defined on `rel-amazon`: `user_churn` and `item_churn`, predicting whether customers stop writing reviews or products stop receiving reviews. Both follow a temporal setup with a cutoff at 60% of the timeline and a 90-day prediction window. An entity is labeled as churned if it does not appear in this future window. Features and graph structure are built from past interactions only, while labels use future activity.

The dataset is composed of three relational tables: `review`, `product`, and `customer`. The `review` table contains interaction data (customer–product pairs, ratings, verification flags, timestamps), while `product` and `customer` provide metadata and identifiers. After temporal filtering, approx-

imately 6.59M historical reviews remain, producing around 1.15M users and 339k products (Table 3).

Table 3. Dataset and split statistics for `rel-amazon`.

Task	Entities	Features	Pos. rate	Train	Val/Test
`user_churn`	1,145,797	14	0.734	802,057	171,870 / 171,870
`item_churn`	338,950	15	0.530	237,265	50,842 / 50,843

Four configurations are defined by combining two feature regimes and two model families. In the raw snapshot (no-feature) setting, each entity is represented by its last interaction before the cutoff (rating, verified, price, category, timestamp), without aggregation. In the feature-based setting, temporal and behavioral aggregates are added, including interaction counts, recency, activity windows (30/90 days), temporal gaps, and short interaction statistics. All features are computed using past data only to prevent leakage.

Flat baselines (RandomForest, XGBoost, LightGBM, CatBoost) are trained under both regimes with hyperparameter selection based on validation AUROC.

3.3.2 Relational Model and Graph Construction

A heterogeneous graph is constructed with three node types (users, products, reviews) and two edge types (user–review, review–product), where review nodes represent interactions. The resulting graph is summarized in Table 4.

Table 4. Graph construction statistics for `rel-amazon`.

Graph component	Count
Raw reviews	20,862,040
Raw products	506,012
Raw customers	1,850,193
Historical review nodes	6,592,542
User nodes	1,145,797
Product nodes	338,950
User–review edges	6,592,542
Review–product edges	6,592,542

To ensure comparability, user and product node features reuse the same flat representations in both regimes. In the raw snapshot setting, GNNs aggregate information from multiple interactions instead of relying only on the last observation; in the feature-based setting, they use the same engineered features while additionally exploiting review-node context through message passing.

The graph is implemented with HeteroData and converted to an undirected form. Features are built from numeric and categorical attributes, excluding identifiers, labels, and datetime fields, and are normalized during training using statistics from the training split to avoid leakage.

Two architectures are used: GraphSAGE, which performs heterogeneous message passing with type-specific encoders, and R-GCN, which explicitly models relation types. Both are trained under identical feature regimes using mini-batch neighbor sampling (NeighborLoader), with the same 70/15/15 stratified splits as flat models and weighted binary cross-entropy for class imbalance.

Evaluation includes AUROC, AUPRC, and Recall@10%, with additional metrics computed using a fixed threshold of 0.5.

3.4 SOUSSOU Koffi Désiré: `rel-amazon` Review-Rating Autocompletion

3.4.1 Task, Data, and Leakage Constraints

The task is `review-rating`: given a (customer, product) pair, predict the missing star rating $r \in \{1, 2, 3, 4, 5\}$, framed as matrix completion over the interaction graph. Review-specific columns (text, summary, verification status) are excluded from all model inputs to prevent data leakage; only temporally aggregated node-level signals are permitted.

The dataset is severely class-imbalanced: ≈62% 5-star, 16% 4-star, 13% 3-star, 6% 2-star, 3% 1-star. A naive majority-class baseline (always predict 5) achieves MAE ≈ 0.585. All experiments use a 60% random subsample of the review table with referential integrity preserved.

Critical data-integrity issue. During implementation, a primary-key join bug was identified in the edge construction pipeline: the task table's positional primary key (0-indexed) was being joined against the raw `review` DataFrame's non-contiguous `.index` (ranging from 2 to ∼20M), silently dropping ≈30–33% of train/validation pairs and causing a factor-of-two discrepancy between internal and official MAE. Fix: `.reset_index(drop=True)` before using the index as join key. A secondary bug in `predict_all()` padded missing rows with `label=0` (not a valid rating), corrupting early-stopping metrics; fixed by masking `label==0` rows before computing validation MAE.

3.4.2 Flat Baseline: LightGBM

LightGBM is trained on a rich set of tabular features extracted from the joined review and product tables (review-text columns excluded):

- *Numeric*: log-price; temporal encodings (year, sin / cos month, sin / cos day-of-week).
- *Frequency encodings*: log-frequency of brand, main category, and subcategory.
- *Target encodings* (Bayesian smoothing): historical mean rating per customer, product, brand, and category – computed on training data only to prevent leakage.
- *Text metadata*: title and description length, word count, word density.

Hyperparameters are tuned with Optuna (20 trials, 5-fold time-series cross-validation) minimising multiclass log-loss with balanced class weights.

3.4.3 Relational Model: Heterogeneous GNN

The database is converted into a heterogeneous graph with three node types ($\mathcal{V}_{\text{customer}} \cup \mathcal{V}_{\text{product}} \cup \mathcal{V}_{\text{review}}$) and two typed edge relations: `customer` → `writes` → `review` and `review` → `concerns` → `product`.

Node features are constructed as follows. **Product**: Sentence-BERT (`all-MiniLM-L6-v2`, 384 dims) applied to title, description, brand, and category concatenated with log-price. **Review**: temporal and structural features only (year normalised, sin / cos month, day-of-week normalised). **Customer**: no static features; each customer is represented by a learnable embedding $\mathbf{e}_u \in \mathrm{R}^d$, gathered by local `n_id` to avoid the full-table broadcast bug.

The architecture, RelBenchHeteroGNN, chains per-type MLP encoders into a `to_hetero(GraphSAGE)`

stack (1 to 3 layers) and an MLP decoder applied to the concatenation of final customer and product embeddings, producing a 5 dimensional logit vector. Training uses AdamW with separate parameter groups (5× weight decay on the customer embedding), gradient clipping (norm = 1.0), and dampened inverse-frequency class weights $w_c = \left(\frac{N}{5\,N_c}\right)^{0.5}$. Hyperparameters are tuned via Optuna (12 trials) minimising validation MAE; the optimal configuration is a 1 layer SAGE with hidden size 64 and learning rate 1.15×10^{-3}.

3.4.4 Results

Table 5. Validation results for `review-rating` (`rel-amazon`).

Model	Val MAE (official)	Remark
LightGBM (Optuna-tuned)	0.77	flat baseline
GNN 1 layer SAGE	**0.592**	best weighted
GNN Unweighted loss	0.532	collapses to 5 rate

Table 6. Ablation study, `review-rating` (sorted by MAE).

Variant	MAE	5 star predicted
Unweighted loss	0.532	84.4%
SAGE Optimal – 1 layer	**0.592**	75.8%
SAGE Shallow (1 layer, seed B)	0.602	73.7%
No Text (price only)	0.602	69.4%
SAGE Deep – 3 layers	0.625	74.4%
No Graph (MLP, no message passing)	0.641	68.1%

3.5 CIZERON Jules: `rel-f1` Regression

3.5.1 Task Definition and Relation to the Classification Tasks

The task is `driver-position`, the entity-regression counterpart to ZHENG Tong's two `rel-f1` classification tasks: same entity table `drivers`, same nine-table relational database, but a continuous target – the driver's *average finishing position* over a 60-day prediction window – evaluated with MAE and RMSE (and R^2 reported alongside). Because both tracks share the entity table and the database, the two notebooks were built by directly adapting Tong's classification pipelines: the flat baseline mirrors `rel-f1_baseline_xgboost_driver_dnf_clf.ipynb` with the loss and evaluation metric swapped from classification to regression, and the GNN notebook mirrors `rel-f1_gnn_driver_dnf_clf.ipynb` with the `BCEWithLogitsLoss` classification head replaced by a single-scalar regression head trained with `L1Loss`.

Table 7. Split statistics for `driver-position` (`rel-f1`).

Split	Rows	Mean target	Window
Train	7,453	13.901	–
Val	499	11.083	60 days
Test	760	11.926	60 days

3.5.2 Flat Baseline

Features. Following the same entity-table-only protocol as Tong's flat baselines, the model uses only the `drivers` table: categorical `driverRef` and `nationality`, numerical `age_years` (derived from `dob` and the task timestamp), and calendar features `ts_year`/`ts_month`.

Model. XGBoost (`tree_method="hist"`, `objective="reg:squarederror"`), tuned with **Optuna** (search space: `max_depth`, `min_child_weight`, `subsample`, `colsample_bytree/bylevel`, L_1/L_2 regularization, learning rate), `num_boost_round=2000` with `early_stopping_rounds=50` on validation RMSE.

3.5.3 Relational Model

The heterogeneous temporal graph is built with `make_pkey_fkey_graph` over all nine `rel-f1` tables (`drivers`, `results`, `qualifying`, `standings`, `races`, `constructors`, `constructor_results`, `constructor_standings`, `circuits`), with per-column `stype` inferred via `get_stype_proposal` (text columns dropped, no text embedder configured). The model is the standard RelBench reference architecture: per-table `HeteroEncoder` (PyTorch-Frame ResNet) → 2-layer `HeteroGraphSAGE` → linear regression head, trained with `L1Loss` (Adam, `lr=1e-3`, `weight_decay=1e-4`, early stopping on val MAE, patience 3).

One deviation from Tong's classification setup: on this local Mac/Homebrew environment, only `torch-sparse` is available, not `pyg-lib` (`WITH_PYG_LIB=False`). Since `NeighborLoader`'s `disjoint=True` mode – and therefore the temporal-leakage-safe sampling used by Tong – requires `pyg-lib`, the loaders here run with `disjoint=False` and the `HeteroTemporalEncoder` is disabled accordingly (it expects per-seed time alignment that disjoint sampling provides). `NUM_NEIGHBORS=[128,128]` (2 hops), `batch_size=512`.

3.6 RYSZFELD Sacha: `hm` Regression

3.6.1 Task, Data and Baselines

The `rel-hm` dataset covers H&M's transaction history across three tables (`article`, `customer`, `transactions`), forming a heterogeneous bipartite graph of customer–article purchase events. The `item-sales` task predicts total weekly article sales over a 7-day horizon; the target is heavily right-skewed (≈81% zeros). Training spans 2019-09-09 to 2020-08-31; all evaluation uses the 2020-09-07 validation split (Table 8).

LightGBM flat baseline. Trained on five ordinally-encoded categorical article columns, four historical lag aggregates (`lag1`, `rec4`, `hmean`, `hmax`), and three per-row autoregressive features (`lag1`, `lag2`, `ma4`).

Table 8. Validation-set results for `rel-hm item-sales`. GNN figures are 3-seed means.

Model	MAE	RMSE	R^2
Persistence (lag1)	**0.0472**	0.4323	0.5739
LightGBM	0.0503	**0.4121**	**0.6129**
HeteroSAGE (0 hops, ours)	0.0572	0.4320	0.5745
HeteroSAGE (2 hops, ours)	0.0599	0.4397	0.5633

3.6.2 Relational Model

Graph construction. The raw 12.86 M (customer, article) co-purchase pairs are sparsified to K=30 customers per article (sampled by recency), reducing to 1.31 M edges while preserving full article coverage and fitting within the P100's 16 GB.

Architecture. A two-layer HeteroSAGE with per-type linear encoders, MEAN aggregation, a skip connection that injects raw article features directly into the MLP prediction head, and per-row autoregressive features (`lag1`/`lag2`/`ma4`) concatenated in the head. All code is consolidated in `hm_study.ipynb`.

3.6.3 Anatomy of the (Non-)Advantage: a Controlled Ablation

I hold the architecture fixed and vary the relational structure along four axes — depth, feature content, node-feature design, and message-passing hyperparameters — running each configuration with 3 seeds and reporting mean ± std on the validation set.

Depth (Table 9). Varying only the number of message-passing layers, *zero hops strictly dominates*: each additional layer degrades R^2 by 0.011–0.012. This is the mirror image of the `rel-f1` finding, where one hop provided the entire structural advantage.

Table 9. Depth ablation on `rel-hm item-sales` (3-seed mean ± std, full features).

Hops	RMSE	R^2	ΔR^2
0 (no message passing)	0.4320 ± 0.0015	$\mathbf{0.5745 \pm 0.0030}$	—
1	0.4379 ± 0.0006	0.5628 ± 0.0014	-0.0117
2	0.4377 ± 0.0024	0.5633 ± 0.0047	-0.0112

Feature content. Removing all engineered features (temporal lags and historical statistics) while keeping the graph collapses R^2 to 0.04–0.05 regardless of depth or aggregator, matching a near-random predictor. Conversely, the no-hop model with all features achieves the best GNN result, confirming that the signal lives in the features, not the graph topology.

Node-feature ablation (Table 10). Removing the five categorical article columns (product type, colour, etc.) while keeping the four lag features *improves* R^2 by 0.0073: those columns describe what an article *is* rather than how it sells, and propagating them through message passing introduces GNN-specific noise. LightGBM ignores them naturally through feature selection; the GNN cannot. Removing all customer features is neutral (-0.003): a learned per-customer embedding carries as much information as the demographic columns.

Table 10. Node-feature ablation on `rel-hm` (2 hops, 3 seeds).

Variant	Art. dims	R^2	ΔR^2
Full (5 cat. + 4 lags)	9	0.5633 ± 0.0047	—
A: lags only	4	$\mathbf{0.5706 \pm 0.0048}$	$+0.0073$
B: customers empty	9	0.5604 ± 0.0082	-0.0029
C: A + B	4	0.5705 ± 0.0063	$+0.0072$

Aggregator, density, hop decay, and richer features. Varying the SAGEConv aggregator (mean/sum/max), the sparsification threshold (K=30 vs. K=100, adding 1.86 M edges), a per-hop decay scalar on the second layer ($\gamma \in \{0.8, 0.5, 0.2\}$), and enriched behavioural node features (purchase recency, frequency, repeat rate, sales trend, etc.) all produce changes within 0.010 R^2 of the baseline. None of these modifications recovers the 0.011 loss incurred by the first hop.

4 Challenges and Findings

4.1 ZHENG Tong: Findings on `rel-f1` Classification

Table 11. Flat vs. relational, test set (ROC-AUC / AP). GNN = our HeteroGraphSAGE (3-seed mean); paper rows report ROC-AUC only.

	`driver-top3`		`driver-dnf`	
Model	ROC-AUC	AP	ROC-AUC	AP
XGBoost (flat)	0.609	0.206	0.607	0.772
LightGBM (flat)	0.689	0.259	0.662	0.818
HeteroGraphSAGE (ours)	**0.797**	**0.360**	**0.726**	**0.851**
LightGBM (paper)	0.739	–	0.686	–
RDL/GNN (paper)	0.755	–	0.726	–

The GNN beats both flat baselines on both tasks and matches the paper's RDL (Table 11), with the largest gain on the small, imbalanced `driver-top3`. But this comparison is confounded (model *and* structure change together); the ablation below shows *where* the gain comes from.

Depth (Fig. 1). Varying only the number of hops, 0 hops falls to near-chance (0.564 ± 0.106 top3; 0.582 ± 0.064 dnf, with one seed below 0.5), so the win is *structure*, not the network. One hop jumps above the flat baselines (0.823; 0.743); the second adds nothing and on `driver-dnf` even hurts and quadruples the variance (0.743 ± 0.010 vs. 0.726 ± 0.040). The signal is *shallow*.

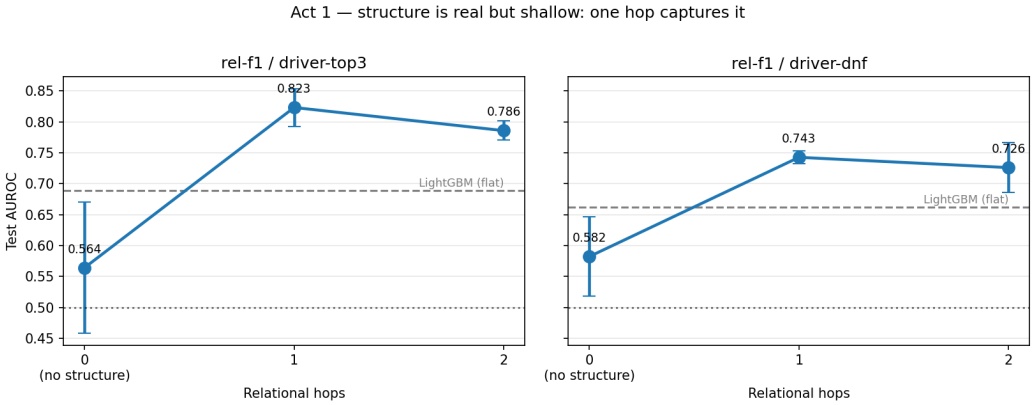


Figure 1. Depth ablation: test ROC-AUC vs. hops, same architecture throughout. One hop captures the advantage; the second does not. Dashed line = LightGBM.

Breadth and the minimal model (Fig. 2). A driver is one hop from three tables; *any one* recovers almost the whole gain (top3: `+results` 0.842 ± 0.001, `+standings` 0.844), and all three together do no better. So a minimal two-table, one-hop model matches or beats the full nine-table, two-hop GNN, far more stably (Table 12): the advantage is *sparse*.

Table 12. Minimal vs. maximal relational model (test ROC-AUC). One linked table at one hop is enough. "Full GNN" is the 3-seed mean of the 2-hop model.

Model	**Tables**	**Hops**	`top3`	`dnf`
LightGBM (flat)	1	0	0.689	0.662
Minimal RDL (`+results`)	2	1	**0.842**	**0.731**
Full GNN (default)	9	2	0.786	0.726

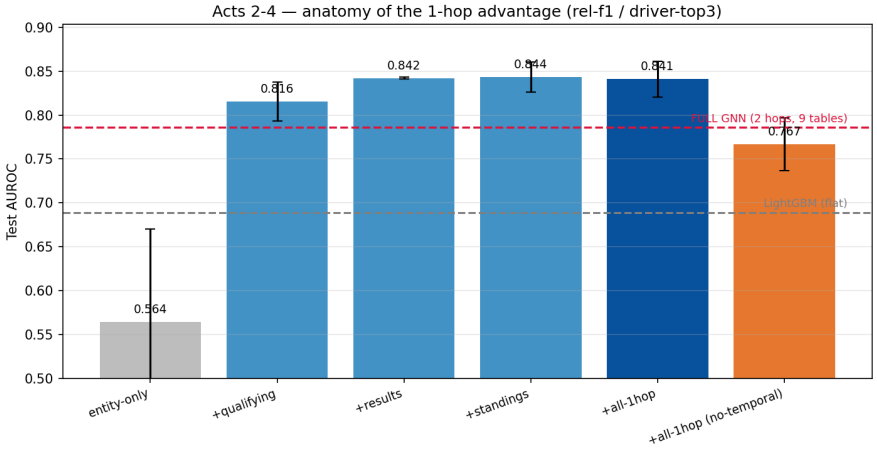


Figure 2. Breadth/content ablation (`driver-top3`). Single-table bars reach the full-GNN line (red); dropping the temporal encoder (orange) still beats LightGBM. `driver-dnf` shows the same pattern.

Content. Dropping the temporal encoder costs 0.074 on `driver-top3` (0.841 → 0.767) but only 0.007 on `driver-dnf`, while both stay above the flat baselines. The gain is therefore mostly *aggregation of linked recent rows*, not deep reasoning; timing matters for qualifying form (`top3`) but not reliability (`dnf`), refining H2.

In sum, the relational advantage is *sparse, shallow, and mostly an aggregate of one table*, so the full heterogeneous-graph machinery is here overkill (supporting H1, H3, H4). The main challenges were preventing temporal leakage in neighbor sampling, the `pyg-lib` requirement for temporal sampling, controlling overfitting and thresholds on the tiny `driver-top3`, and the version-dependent `driver-dnf` label inversion (handled by relying on ROC-AUC).

4.2 NARANJO RODRIGUEZ Natalia: Findings on `rel-amazon` Classification

Results (Table 13) show a consistent pattern: relational structure provides clear gains in the raw snapshot setting but becomes negligible once strong feature engineering is introduced.

Table 13. Best models per task, feature regime, and model group (test set).

Task	Features	Group	Model	AUROC	AUPRC	R@10
item_churn	no features	Flat	LightGBM	0.7424	0.7374	0.1540
item_churn	no features	GNN	GraphSAGE	0.7774	0.7589	0.1553
item_churn	with features	Flat	XGBoost	0.7873	0.7679	0.1570
item_churn	with features	GNN	GraphSAGE	0.7887	0.7671	0.1562
user_churn	no features	Flat	XGBoost	0.6887	0.8455	0.1232
user_churn	no features	GNN	GraphSAGE	0.6958	0.8509	0.1249
user_churn	with features	Flat	XGBoost	0.7025	0.8560	0.1266
user_churn	with features	GNN	R-GCN	0.7035	0.8561	0.1261

The effect of feature engineering and relational modeling is summarized in Table 14.

Table 14. Effect of feature engineering and relational modeling (test).

Task	Comparison	Δ AUROC	Δ AUPRC
item_churn	features (Flat)	+0.0449	+0.0305
item_churn	features (GNN)	+0.0113	+0.0082
item_churn	GNN vs Flat (no features)	+0.0350	+0.0215
item_churn	GNN vs Flat (with features)	+0.0014	-0.0007
user_churn	features (Flat)	+0.0138	+0.0105
user_churn	features (GNN)	+0.0077	+0.0052
user_churn	GNN vs Flat (no features)	+0.0071	+0.0054
user_churn	GNN vs Flat (with features)	+0.0010	+0.0002

The deltas in Table 14 reflect performance differences under fully aligned settings (same cutoff, labels, splits, and feature regimes), so GNN vs flat comparisons correspond to the effect of relational modeling with review-node context.

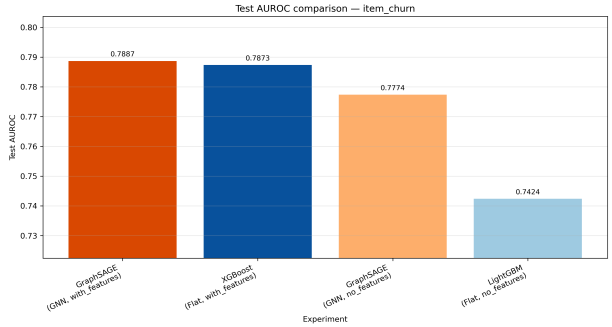


Figure 3. Test best AUROC for `item_churn`.

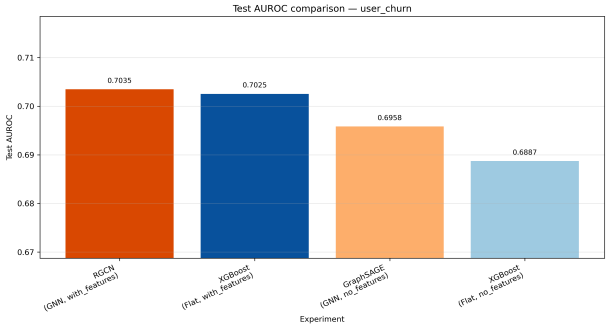


Figure 4. Test best AUROC for `user_churn`.

In the no-feature setting, GNNs outperform flat baselines, particularly for `item_churn` (+0.035 AUROC), while gains for `user_churn` are smaller (+0.007), showing that the graph captures interaction patterns beyond the last snapshot by aggregating multiple interactions.

In contrast, in the feature-based setting the gap nearly disappears. Flat models improve substantially (e.g., +0.0449 AUROC for item_churn), while GNN gains are limited, leading to comparable performance. Improvements appear mainly in AUROC and AUPRC, while Recall@10% remains stable, indicating gains in global ranking rather than top-risk selection.

This behavior is explained by the feature design: aggregates such as counts, recency, activity windows, temporal gaps, and short histories effectively approximate relational information. Variables like `num_reviews`, `reviews_last_30d/90d`, and `recency_days` encode signals that GNNs recover through neighborhood aggregation.

Task differences reinforce this interpretation. Larger gains for `item_churn` indicate stronger dependence on collective interaction patterns, while `user_churn` is driven more by individual activity and is easier to capture with features.

Overall, relational structure and feature engineering encode largely overlapping signals: GNNs help when only raw snapshots are available but offer limited gains with strong features. From a computational perspective, the difference is substantial: flat models required several hours of training in practice (approximately ∼3 hours for item_churn and up to ∼6–8 hours for user_churn due to grid search), whereas GNN models completed training in minutes (around 10–15 minutes after graph construction).

This reflects a shift in complexity: flat models concentrate cost in preprocessing and hyperparameter tuning, while GNNs introduce overhead through graph construction, memory usage, and neighbor sampling. These results suggest that neither approach is strictly superior. Instead, hybrid strategies where combining engineered tabular features with relational modeling or approximating relational signals when graph methods are impractical are often the most effective and practical solution.

4.3 Findings from the Other Tasks

4.4 SOUSSOU Koffi Désiré: Findings on `rel-amazon` Regression

Tables 5–6 summarise the results. The central finding is that **graph topology provides a measurable advantage**: removing message passing (No Graph) raises MAE from 0.592 to 0.641 (+8%), while the flat LightGBM baseline sits at 0.77 MAE, substantially worse than both GNN variants. The two-hop customer–review–product path is the key structural unit: a customer's latent preference, aggregated across their review history, flows through review nodes into the product representation and informs the pair-level prediction in a way that tabular target encodings only partially approximate.

Architecture depth. The 3-layer model is worse than the 1-layer model (0.625 vs. 0.592), consistent with over-smoothing on a shallow two-hop graph. This mirrors the `rel-f1` depth ablation (Section 3.2.2), where the second hop added noise rather than signal.

Class weighting trade-off. The unweighted model achieves the lowest aggregate MAE (0.532) by concentrating 84% of predictions on the majority 5-star class (only 0.2% 1-star), collapsing on minority ratings. Dampened inverse-frequency weighting ($\gamma = 0.5$) recovers a balanced distribution at a modest MAE cost (+0.06) and is preferable for any application requiring reliable 1–3 star detection.

Text embeddings. Restricting product features to price alone (No Text) yields MAE 0.602 vs. 0.592 with full SBERT embeddings, confirming that semantic product information contributes, though graph topology remains the dominant structural signal.

Revisiting the hypotheses. H1 (structure matters) is *confirmed*: message passing provides a clear, reproducible gain over the flat MLP. H3 (flat baselines remain strong) is *partially refuted* on this task: the feature-engineered LightGBM baseline is significantly weaker than the GNN, unlike the other tracks.

4.5 CIZERON Jules: Findings on `rel-f1` Regression

Table 15. Flat vs. relational, test set, `driver-position` (`rel-f1`).

Model	MAE	RMSE	R^2
XGBoost (entity-only, Optuna)	4.1380	5.0120	0.0747
HeteroGraphSAGE (GNN, seed=42)	**4.1284**	5.0696	0.0533

On `driver-position`, the GNN edges out the flat XGBoost baseline on MAE (4.1284 vs. 4.1380) but is marginally worse on RMSE (5.0696 vs. 5.0120) and R^2 (0.0533 vs. 0.0747). Unlike Tong's classification tasks (Table 11), where the GNN's advantage over flat baselines is large, here the two model families are essentially *tied*: relational message passing over the full nine-table graph recovers, at best, the same signal that an entity-only XGBoost model already captures from `driverRef`, `nationality`, and age. Both R^2 values are low (< 0.08), indicating that average finishing position over a 60-day window is intrinsically hard to predict from either representation – consistent with motor racing being shaped by race-day factors (mechanical failures, collisions, weather) that are not encoded in the pre-race relational structure.

This near-tie is a useful complement to Tong's `rel-f1` classification ablation (Section 3.2.2): where classification showed a real but shallow one-hop relational advantage, the regression target here gets no measurable lift from structure at all, reinforcing H3 (flat baselines remain strong) and suggesting H2 in its strongest form – whether structure helps at all is itself task-dependent, not just *how much* it helps.

Challenges. The main obstacle was environment-specific: this notebook was run on a local Mac/Homebrew Python setup where `pyg-lib` could not be installed without breaking `torch_sparse` (the same `pyg-lib` dependency Tong flags for temporal `NeighborLoader` sampling). Falling back to `torch-sparse` forced `disjoint=False` sampling and disabling the `HeteroTemporalEncoder`, meaning the reported GNN, unlike Tong's, does *not* use relative-time positional encoding – a likely contributor to its weak margin over the flat baseline, and a clear next step (re-run with a `pyg-lib`-enabled environment, e.g. the shared team venv, once available). A second, more mundane issue was an early debugging run accidentally left at the top of the training loop (3 epochs only, val/test MAE far worse than the converged run); the numbers reported above are from the final, fully-trained model and were cross-checked against `task.evaluate()`.

4.6 RYSZFELD Sacha: Findings on `hm` Regression

The central finding on `rel-hm` *inverts* the `rel-f1` result: graph structure is not merely unnecessary — it is actively harmful.

The structure penalty. The depth ablation (Table 9) shows a monotone degradation with hops: 0 hops achieves R^2 0.5745; one hop drops to 0.5628; two hops give 0.5633. On `rel-f1`, one hop provided +0.10 ROC-AUC over the no-structure baseline; here, the neighbourhood aggregation corrupts the signal.

Why structure hurts: the autoregressive bottleneck. The `item-sales` target is dominated by temporal autocorrelation: next week's sales for an article correlates strongly with this week's sales (`lag1`). LightGBM receives `lag1` as a direct input column and exploits this trivially. The GNN must propagate article features through a bipartite customer–article graph, averaging over up to 30 heterogeneous purchasers and thereby *diluting* the article-specific lag signal. At zero

hops, the skip connection delivers `lag1` directly to the head without interference; adding message passing only corrupts this shortcut. The structural signal that does exist — which customers tend to co-purchase which articles — is not predictive of an article's absolute weekly sales, which are driven by factors (trend, seasonality, stock availability) not encoded in the topology.

Revisiting the hypotheses. H1 (structure matters) is *refuted* on this task: the optimal relational model uses zero hops, collapsing to a flat neural predictor. H2 (the benefit is task-dependent) is *confirmed*: unlike race outcomes, which are inherently relational (a driver's result depends on other drivers'), weekly article sales are largely self-correlated and independent of the relational neighbourhood. H3 (flat baselines remain strong) is *confirmed*: LightGBM's R^2 of 0.6129 is not matched by any GNN configuration. H4 takes an extreme form: the minimal predictive sub-schema is the entity table alone.

Engineering lessons. Three bugs produced the same surface symptom; diagnosing each required instrumenting individual tensors with `torch.isfinite` rather than reading the aggregate loss. The prediction collapse ($R^2 \approx 0 \rightarrow 0.56$) demonstrated that *training objective alignment* matters as much as architecture: a model trained on $\log(1+y)$ while evaluated on y_{raw} cannot reliably learn the dominant autoregressive pattern, even with an explicit skip connection.

Practical implications. The `rel-hm` result identifies a regime where relational deep learning should *not* be the default choice: tasks whose signal is dominated by entity-level autocorrelation. In such settings the relational neighbourhood is a noisy intermediary rather than a source of additional information, and the computational cost of graph construction and sparsification (here reducing 12.86 M to 1.31 M edges to fit GPU memory) yields no return. A simple diagnostic — run the 0-hop control first — identifies this regime before investing in the full graph machinery. Conversely, the contrast with `rel-f1` classification and `rel-amazon` regression (Koffi's track) shows that structure provides measurable gains whenever the target is genuinely relational, i.e. driven by interactions between entities rather than by each entity's own history.

5 Alignment with the Initial Plan

5.1 Did We Manage To Do Everything We Set Out To Do?

For ZHENG Tong's `rel-f1` classification part, the planned workflow was achieved: both flat baselines and relational GNN baselines were implemented, evaluated, and compared. The work then went well beyond the minimal prototype with a full controlled ablation (the depth/breadth/content study above), which was not in the original plan but emerged from noticing that the flat-vs-GNN comparison is confounded.

Table 16. Plan alignment for the completed `rel-f1` classification work.

Planned item	**Status**	**Comment**
RelBench/RDL literature review	Completed for this part	Used to define the flat-vs-relational comparison and interpret results.
Data pipeline	Completed	Loaded `rel-f1`, inspected tables, constructed task splits and heterogeneous graph.
Flat baselines	Completed	Implemented LightGBM and XGBoost for `driver-dnf` and `driver-top3`.
GNN prototype	Completed	Implemented HeteroEncoder + HeteroTemporalEncoder + HeteroGraphSAGE.
Extra experiments	Exceeded	Added task-specific tuning, multi-seed reporting, and a full depth/breadth/content ablation with a minimal-model result (3 seeds; 5 seeds would tighten the estimates further).
Final report	In progress	This document provides the report structure and completed `rel-f1` classification section.

For NARANJO RODRIGUEZ Natalia's `rel-amazon` classification tasks, the core objectives were achieved, although the process diverged from the initial plan. Flat and graph-based models were successfully implemented under both feature regimes using a consistent pipeline ensuring strict comparability (aligned splits, labels, and inputs). However, a substantial portion of the effort was dedicated to debugging and redesign, due to early issues such as feature leakage risks, inconsistencies between flat and graph representations, and preprocessing mismatches, which required multiple re-runs and reduced the time available for deeper experimentation.

Unlike the `rel-f1` track, this work focused on analyzing the interaction between feature engineering and relational modeling at scale rather than structured graph ablations. Computational constraints (dataset size, memory limits, and training cost) restricted extensive hyperparameter tuning and model exploration, particularly for GNNs. Overall, while the planned comparison was completed, the main outcomes highlight both the overlap between relational and feature-based signals and the practical challenge of building consistent and scalable pipelines for large relational datasets.

Table 17. Plan alignment for the completed `rel-amazon` classification work.

Planned item	**Status**	**Comment**
RelBench / RDL framework usage	Completed	RelBench-style temporal setup with aligned cutoff and prediction window.
Data pipeline	Completed (iterative)	Full pipelines for both regimes, requiring redesigns to ensure consistency.
Flat baselines	Completed	Random Forest, XGBoost, LightGBM, and CatBoost with GridSearchCV tuning.
GNN prototype	Completed	GraphSAGE and R-GCN with neighbor sampling, limited tuning due to scalability.
Consistency across experiments	Achieved after iterations	Early inconsistencies required re-runs for reliable comparisons.
Extra experiments	Partially achieved	Focus shifted to feature vs relational signal analysis instead of ablations.
Scalability and execution	Constrained	Large dataset required HPC; memory limited model exploration.
Final report	Completed	Comparison across four setups highlighting the feature vs relational trade-off.

For RYSZFELD Sacha's `rel-hm` regression part, the planned workflow was completed in full. The work also exceeded the initial plan with six controlled ablations informed by advisor feedback, none of which were in the original scope.

Table 18. Plan alignment for the `rel-hm` regression work.

Planned item	**Status**	**Comment**
RelBench/RDL literature review	Completed	Used to define the flat-vs-relational comparison and motivate the GNN architecture.
Data pipeline	Completed	Loaded `rel-hm`, inspected three tables, constructed task splits and heterogeneous graph.
Flat baseline	Completed	LightGBM with lag features and categorical article columns; persistence baseline also included.
GNN prototype	Completed	HeteroSAGE with skip connection and per-row autoregressive features; graph sparsification (K=30); three distinct bugs diagnosed and fixed.
Extra experiments	Exceeded	Six controlled ablations (depth, feature content, aggregator, graph density, hop decay, behavioural features), each with 3 seeds; results saved to `reports/ablations/`.
Final report	Completed	`rel-hm` sections of this document.

Table 19. Plan alignment for the `rel-amazon` regression work (SOUSSOU).

Planned item	Status	Comment
RelBench / RDL literature review	Completed	Used to define the matrix completion framing and leakage constraints.
Data pipeline	Completed (with debugging)	Fixed primary-key join bug (30% pair loss) and `predict_all()` label-padding bug; enforced review-text exclusion throughout.
Flat baseline	Completed	LightGBM with Optuna HPO, Bayesian-smoothed target encodings; evaluated with `task.evaluate()`.
GNN prototype	Completed	RelBenchHeteroGNN with SBERT product features, learnable customer embeddings, Optuna HPO (12 trials).
Ablation study	Completed	Six variants isolating depth, graph topology, text features, and class weighting; saved to `ABLATION_PATH`.
Final report	Completed	rel-amazon review-rating sections

[**TODO:** Add plan alignment for Jules once their sections are complete.]

5.2 What Would We Do Differently? Lessons Learned

If starting over, I would: (i) **pin the environment early**, since RelBench version differences silently flip `driver-dnf` label semantics; (ii) **tune to the target metric and use multiple seeds**, monitoring AP on the imbalanced `driver-top3` rather than default accuracy; and, most importantly, (iii) **prefer controlled ablations over single comparisons** — the informative result came not from "GNN beats trees" but from turning structure into a variable and removing it axis by axis, which answers "why, and how much can be removed?" rather than merely "is it better?".

For the `rel-amazon` tasks, an additional lesson is the importance of enforcing pipeline consistency early. Much of the effort was spent aligning feature and graph pipelines due to leakage risks and preprocessing inconsistencies, showing that reliable relational experiments depend as much on data handling and comparability as on model design, especially at large scale.

For the `rel-hm` regression task, if starting over I would: (i) **run the 0-hop control first** — if it already outperforms deeper models, the effort should go into feature engineering rather than graph topology; (ii) **align the training objective with the evaluation metric from day one** — training on $\log(1+y)$ while evaluating on y_{raw} silently biases the model and requires a second round of fixes to untangle; (iii) **instrument tensors before training, not after** — a single `assert torch.isfinite` on each feature matrix at construction time would have cut debugging time for the NaN propagation bug by hours.

6 Conclusion

The `rel-f1` classification experiments support the project's central idea – relational structure matters – but sharpen it: the GNN beats competitive flat baselines, yet the advantage is sparse, shallow, and largely a temporal/structural aggregate of a single linked table. Relational deep learning is therefore not automatically better, and not as deep as it looks; identifying the minimal predictive sub-schema is both cheaper and more informative. These lessons should guide the remaining tasks and the final team-level synthesis.

The `rel-amazon` experiments extend this result by showing that, at scale, strong temporal and behavioral features can capture much of the same signal as relational message passing. This highlights a practical trade-off: relational models offer clear gains in low-feature settings, but

their advantage diminishes when feature engineering is strong, suggesting that hybrid approaches balancing both representations are often the most effective in real-world scenarios.

The `rel-hm` experiments provide the sharpest counterexample: structure not only fails to help but actively degrades performance, with each additional message-passing hop costing roughly 0.011 R^2. The task is dominated by autoregressive signals already captured by the entity table, so the relational neighbourhood adds noise rather than information. Together, the three tracks span the full spectrum of H1: structure matters a lot (`rel-f1` classification), not at all (`rel-f1` regression), or harmfully (`rel-hm`), depending entirely on whether the predictive signal is inherently relational or self-contained.

References

References

[1] J. Robinson, R. E. Turner, C. Guo, F. Belletti, X. Yu, J. Kossaifi, R. Liao, G. Rabbat, M. Bronstein, and M. Zhang. *RelBench: A Benchmark for Deep Learning on Relational Databases*. arXiv:2407.20060, 2024.

[2] M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec. *Position: Relational Deep Learning – Graph Representation Learning on Relational Databases*. arXiv:2312.04615, 2023.

[3] V. P. Dwivedi, C. Kanatsoulis, S. Huang, and J. Leskovec. *Relational Deep Learning: Challenges, Foundations and Next-Generation Architectures*. arXiv:2507.05793, 2025.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. Advances in Neural Information Processing Systems, 2017.

[5] T. Chen and C. Guestrin. *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[6] W. Hamilton, Z. Ying, and J. Leskovec. *Inductive Representation Learning on Large Graphs*. Advances in Neural Information Processing Systems, 2017.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

Contribution Section

Individual Contributions

Team member	Specific contribution

ZHENG Tong	Worked on the `rel-f1` classification tasks `driver-dnf` and `driver-top3`. Implemented entity-table-only LightGBM and XGBoost baselines; built feature pipelines using `drivers` and calendar features; tuned hyperparameters with Optuna; implemented heterogeneous temporal GNN baselines using `make_pkey_fkey_graph`, temporal `NeighborLoader`, `HeteroEncoder`, `HeteroTemporalEncoder`, and `HeteroGraphSAGE`; handled threshold tuning and metric comparison; analyzed task imbalance and DNF label semantics. Designed and ran the controlled depth/breadth/content ablation (sub-graph construction with `edge_type_subgraph`, a temporal-encoder switch, and the minimal-model comparison), consolidated all of it into the single deliverable notebook `rel-f1_structure_anatomy.ipynb`, produced the figures, and wrote the `rel-f1` classification sections of this report.
NARANJO RODRIGUEZ Natalia	Worked on the `rel-amazon` classification tasks `user_churn` and `item_churn`. Implemented flat baselines (Random Forest, XGBoost, LightGBM, CatBoost) under both no-feature and feature-based regimes; designed preprocessing pipelines with temporal splits and strict no-leakage constraints; engineered temporal and behavioral features and constructed tabular datasets using Polars. Built heterogeneous graphs (users, products, reviews) in PyTorch Geometric and trained GraphSAGE and R-GCN models with neighbor sampling and weighted losses. Ensured comparability across four configurations (flat vs. GNN, with and without features), performed hyperparameter tuning and evaluation (AUROC, AUPRC, Recall@10%). Conducted result analysis highlighting the overlap between feature engineering and relational structure. Addressed challenges related to scalability, pipeline consistency, debugging, and experimental validation.
SOUSSOU Koffi Désiré	Worked on the `rel-amazon` regression task `review-rating`. Formulated the task as matrix completion on the customer–product interaction graph with a strict no-text-leakage constraint. Implemented the LightGBM flat baseline with Bayesian-smoothed target encodings, frequency features, and Optuna hyperparameter search (`scripts/train.py`). Built the RelBenchHeteroGNN pipeline: SBERT product embeddings, learnable customer embeddings gathered by local `n_id`, `to_hetero(GraphSAGE/GAT)`, class-weighted cross-entropy with dampening ($\gamma = 0.5$). Identified and fixed a critical primary-key join bug in `scripts/data.py`'s `build_edges()` (silent 30% pair loss) and a secondary bug in `predict_all()` (label-0 padding corrupting early-stopping MAE). Designed and ran the six-variant ablation study (`scripts/ablation.py`) isolating depth, topology, text, and class-weighting contributions; managed Optuna HPO. Produced the local exploration notebook (`amazon_review_gnn.ipynb`), the cluster analysis notebook (`comparison_analysis.ipynb`), and wrote the `rel-amazon` regression sections of this report.

CIZERON Jules	Worked on the `rel-f1` regression task `driver-position`. Adapted Tong's `driver-dnf` pipelines from classification to regression: built the entity-table-only XGBoost baseline (`driverRef`, `nationality`, `age_years`, calendar features) with Optuna-tuned hyperparameters and early stopping on validation RMSE; built the HeteroGraphSAGE relational model over the full nine-table `rel-f1` graph (`make_pkey_fkey_graph`, `HeteroEncoder`, 2-layer `HeteroGraphSAGE`, linear regression head, `L1Loss`). Diagnosed and worked around a local environment constraint (no `pyg-lib` on Mac/Homebrew, forcing `disjoint=False` sampling and a non-temporal GNN variant); debugged an early under-trained run left in the notebook and validated final metrics against `task.evaluate()`; produced the flat-vs-GNN comparison table and wrote the `rel-f1` regression sections of this report.
RYSZFELD Sacha	Worked on the `rel-hm` regression task (`item-sales`). Implemented the LightGBM flat baseline with historical lag and temporal features; built the heterogeneous customer–article graph and the HeteroSAGE model with skip connection, per-row autoregressive inputs, and graph sparsification (K=30); diagnosed and fixed three engineering bugs (uninitialised best state, CUDA OOM, NaN propagation via `OrdinalEncoder`); identified and corrected a prediction collapse caused by a misaligned training objective and non-linear feature transformation. Designed and ran six controlled ablations (depth, feature content, aggregator, graph density, hop decay, behavioural node features), each with 3 seeds, saving results to `reports/ablations/`. All code consolidated in `hm_study.ipynb`; ablation figures in `hm_synthesis.ipynb`. Wrote the `rel-hm` sections of this report.

Team Workflow and Organization

The team divided the workload by dataset and task type so that each member could independently implement and analyze one clear experimental track. We used GitHub to store code and notebooks, Notion to organize milestones, tasks, and team information, and regular team discussions to coordinate progress and resolve implementation issues. The planned milestones covered literature review, data pipeline, baseline models, GNN prototypes, preliminary results, extra experiments, and report writing.

[**TODO:** Add exact communication tools used by the team, e.g. WhatsApp/Discord/Teams, meeting frequency, code review habits, and how results were centralized.]

Use of AI Tools

For ZHENG Tong's `rel-f1` classification work, an AI assistant (Anthropic's Claude, via the Claude Code CLI) was used in a supporting role: mainly to debug code and resolve library/environment errors (e.g. the `pyg-lib` requirement for temporal sampling and a float64/MPS incompatibility), and to polish the writing and LaTeX formatting of this report. All experiments were designed and run by the author; every reported number comes from the executed notebooks and was checked against RelBench's official `task.evaluate`, and no metric or claim was produced by the AI tool itself.

For NARANJO RODRIGUEZ Natalia's `rel-amazon` classification work, AI tools (Microsoft Copilot) were used as a supporting aid to generate the initial code structure for preprocessing, feature engineering, and model pipelines, based on iterative instructions from the author and guidance from RelBench documentation. The author retained full control over the experimental design, including defining the feature versus no-feature setups, selecting models, features, evaluation metrics, and report structure, and progressively refined the pipelines with AI assistance based on observed results such as class imbalance and inconsistencies across configurations. AI tools were also used to assist with debugging, resolving errors, and adapting the implementation. In addition, Copilot helped polish the writing and LaTeX formatting of this report, while all content decisions remained defined by the author. All experiments were designed, run, and validated by the author, and all reported results were manually verified, with no conclusions or metrics produced without human oversight.

For RYSZFELD Sacha's `rel-hm` work, Claude Code (Anthropic's CLI) was used to debug CUDA and PyG errors (full-batch OOM, NaN propagation through `OrdinalEncoder`), to assist with LaTeX formatting, and to maintain a running project journal (`CLAUDE.md`). Experiment design, hyperparameter choices, ablation protocols, and all reported numbers come from the executed notebooks; every metric was verified against the `task.evaluate` call inside `hm_study.ipynb`.

For SOUSSOU Koffi Désiré's `rel-amazon` regression work, Claude, Gemini and Copilot was used to support debugging (primary-key join bug, early-stopping metric corruption, lazy PyG module initialisation), to assist with GNN pipeline structure and ablation script design, and to polish the LaTeX formatting of this report. All experiments were designed and run by me.

[**TODO:** Other members: briefly state which AI tools (if any) you used, for what, and how you verified the output.]