RIDE: AN OPEN DATASET AND BENCHMARK FOR TRAIN DELAY PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Train delay prediction is an important problem for both passengers and railway operators, yet progress in the field remains difficult to assess due to the lack of standardized datasets, prediction targets, and evaluation protocols. To address this gap, we introduce RIDE, an open dataset and benchmark for train delay prediction built at nationwide scale over the Belgian railway network. RIDE covers 94.5M train events, 3.6M journeys, and 35.7M weather records from 2023 to 2025. It is organized as a layered data pipeline from raw railway and weather sources to two public releases: a reusable intermediate relational dataset and model-ready benchmark datasets. The benchmark standardizes the prediction task and the training and testing data. It also provides a unified evaluation protocol that supports direct comparison across models. Using this framework, we provide the first comprehensive evaluation of representative non-learning, tree-based, and deep learning model families under a shared setup. The resulting comparison reveals distinct behavior across model families: learning-based models clearly outperform non-learning approaches, graph neural networks achieve the best mean performance, yet the strongest sequential, attention-based, and graph-based models remain surprisingly close overall. Beyond aggregate mean absolute error (MAE) and root mean squared error (RMSE), the framework also provides breakdowns by prediction horizon and delay change, enabling more detailed analysis of model behavior across forecasting regimes.

1 INTRODUCTION

Railway networks are a central component of large-scale and sustainable mobility, supporting vast numbers of passenger journeys every day. Because rail operations are tightly scheduled and strongly interconnected, local disruptions can propagate across trains through the network, degrading reliability for both passengers and operators. Accurate delay prediction is an important task: it can help railway operators anticipate how local disruptions propagate between trains, thereby improving passenger information, dispatching, and traffic management. These challenges are further amplified by the scale and complexity of modern railway systems, where operational, infrastructural, and exogenous factors such as weather jointly shape delay dynamics Tiong et al. (2023); Spanninger et al. (2022; 2020).

Train delay prediction has been studied through a broad range of modeling approaches, from non-learning, rule-based methods to statistical approaches and more recently deep learning models Tiong et al. (2023); Spanninger et al. (2022; 2020); Tiong & Palmqvist (2023). Yet the field remains fragmented: delay prediction studies typically rely on different datasets, formulate different prediction targets, and evaluate under different protocols. Public datasets for train delay prediction have only recently begun to emerge Zhang et al. (2022); Wu et al. (2025); Döllmann (2025); Borin et al. (2026), and they differ substantially in scope, coverage, and intended prediction task, without establishing a unified benchmark for downstream comparison. This makes it difficult to draw conclusions across studies, and motivates the need for a dataset and evaluation framework for train delay prediction.

To address these issues, we introduce RIDE (T**R**a**I**n **DE**lay Prediction Dataset and Benchmark), a comprehensive open resource for standardized train delay prediction nationwide. Unlike prior work, RIDE combines a reusable data release with a benchmark framework that standardizes prediction targets, temporal splits, and evaluation metrics. It covers three years of passenger railway operations over the Belgian network, spans diverse service types, and enables models to be compared on shared

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.

prediction instances under a common evaluation protocol. This provides a controlled setting for studying how different modeling assumptions and inductive biases perform on large-scale structured spatiotemporal prediction. The main contributions are the following:

(i) We release a layered data pipeline that transforms raw railway and weather sources into an intermediate relational data layer and then into model-ready benchmark datasets tailored to different model families; the intermediate layer enables users to construct alternative model-specific datasets without reproducing the core cleaning and integration steps.

(ii) We define a unified benchmark protocol with a common prediction target, common splits, and shared evaluation metrics, allowing diverse model families to be compared on identical prediction instances and metrics.

(iii) We provide the first comprehensive benchmark on a diverse set of approaches: non-learning methods, with a Translation baseline and Graph-event model; tree-based models, with XGBoost; and deep learning models, with MLP, LSTM, Transformer, and GNN. These experiments illustrate a clear advantage for learning-based methods, with graph neural networks achieving the best mean performance, while the strongest learning-based architectures remain close in absolute performance.

(iv) We analyze performance across prediction horizons and delay-change regimes, revealing regime-specific model strengths, such as stronger short-horizon performance for sequential models and stronger performance under large delay accumulation for models that explicitly capture interactions between trains.

2 Related work

Datasets. Progress in train delay prediction is shaped not only by modeling choices, but also by the availability of shared datasets and standardized evaluation protocols. Public datasets for train delay prediction have only recently begun to emerge. Existing open resources include a dataset of high-speed rail operations on the Chinese railway network Zhang et al. (2022), a multi-train-type operations dataset for Italy Wu et al. (2025), and more recent open archives for Germany Döllmann (2025) and Finland Borin et al. (2026) that cover train operations over longer time spans, all of which include weather and scheduling information. While these resources are valuable, they remain limited in different ways, including temporal coverage, train diversity, or network scope, and do not by themselves establish a common benchmark framework combining event-level targets, fixed temporal splits, and a unified evaluation protocol for model comparison. As a result, papers in the literature rely on diverse public and proprietary datasets and use their own evaluation protocols, making it difficult to compare results across studies even when they address closely related problems.

Tasks. Prediction targets are another area where standardization remains limited. Prior work considers a variety of formulations, including next-station delay prediction, multiple-stations-ahead delay prediction, delay after a fixed time interval, station-level aggregate quantities, and event-level outputs Tiong et al. (2023); Spanninger et al. (2022; 2020). While these formulations can be well suited to specific operational settings, they further complicate comparison across studies. The common factor underlying these approaches is that train delays are measured at discrete scheduled events. In particular, scheduled arrivals, departures and passages of a train at an operational point are the finest-grained unit at which predictions can be validated against ground truth observations, and from which coarser quantities can be derived by aggregation. For this reason, we choose the delay at the next n scheduled events of each train as the common prediction target for this benchmark.

Metrics. Evaluation methods in the literature are also heterogeneous, although MAE and RMSE remain the dominant metrics Tiong et al. (2023); Spanninger et al. (2022); Yong et al. (2025). Percentage-based measures such as mean absolute percentage error (MAPE) are used, but are problematic in train delay prediction because delays are often close to zero Yong et al. (2025). Some works further stratify performance by prediction horizon or by delay ranges in order to study different operating regimes Spanninger et al. (2022); Nair et al. (2019). Our evaluation protocol follows this general direction, while extending it with a unified set of breakdowns by prediction horizon and delay change, designed to make regime-specific strengths and weaknesses easier to compare across models.

Network Scope. Another important point of variation in the literature is the scope of prediction. Many studies focus on isolated train lines, single-track systems, or otherwise restricted operational settings Liu et al. (2023); Meesit & Suwannasri (2025), which can be well suited to use cases where services can reasonably be treated in isolation. Other settings instead require modeling interactions at broader network scale Chowdhury et al. (2025); Elliker et al. (2026). In this work, we adopt a full-network benchmark so that potential interactions and delay propagation across services remain part of the prediction problem. At the same time, the dataset remains flexible enough to support narrower benchmarks derived from specific lines or sub-networks, allowing future work to follow the same general framework under different assumptions about operational scope.

Input Features. Across most modeling approaches, prior work relies on railway operational information, timetable and infrastructure context, and external variables such as weather Tiong et al. (2023); Spanninger et al. (2022; 2020), suggesting that effective delay prediction depends not only on recent train operations, but also on network structure and exogenous conditions. RIDE follows this pattern by combining operational event data with timetable and infrastructure information published by Infrabel, the manager of the Belgian railway network, whose data have supported delay propagation and prediction studies Elliker et al. (2026); Dekker et al. (2022), and weather-derived features from Open-Meteo, which has been incorporated into train delay datasets Wu et al. (2025).

Models. Prior work has also explored a broad range of modeling approaches, which can be grouped into three families: non-learning approaches, traditional machine learning methods, and deep learning models. Non-learning approaches include rule-based formulations like graph models and Markov chains Spanninger et al. (2022; 2020); Tiong & Palmqvist (2023). Traditional machine learning methods include linear regression, support vector machines, random forests, and gradient-boosted trees Tiong et al. (2023); Spanninger et al. (2022; 2020). Deep learning models include multilayer perceptrons, recurrent architectures such as LSTMs, transformer-based and graph-neural approaches Tiong et al. (2023); Spanninger et al. (2022); Arthaud et al. (2024); Li et al. (2024); Huang et al. (2024). Despite this diversity, individual studies typically compare only a narrow subset of methods under study-specific datasets, prediction tasks, metrics, scopes, and input assumptions. As a result, the relative strengths of different model families remain only partially understood, motivating a benchmark that evaluates them on common prediction instances under a shared protocol.

Taken together, these considerations motivate RIDE as both a public dataset release organized into processing layers and a standardized benchmark framework. The processed release provides a reusable foundation from which downstream users can construct alternative datasets for their own modeling choices, while the benchmark defines a common prediction target and unified evaluation protocol used across a wide range of model families in our experiments.

3 RIDE DATASET

As a first major contribution, we introduce the RIDE dataset, organized as a layered data pipeline that transforms raw railway and weather data into both a reusable intermediate release and model-ready benchmark datasets. It covers three years of passenger railway operations from 2023 to 2025 over the Belgian railway network at nationwide scale, and includes a broad range of rail traffic, from local and intercity to high-speed trains. As illustrated in Figure 1, RIDE is designed as a tiered release, organized into four stages: *raw*, *bronze*, *silver*, and *gold*. This tiered design is intended to support different levels of reuse: the raw and bronze stages support source acquisition and standardization, the silver stage provides a processed relational dataset that can be adapted to alternative modeling choices, and the gold stage provides concrete benchmark datasets tailored to different delay prediction approaches while sharing common splits and evaluation targets. We provide full details of the processing pipeline and resulting datasets in Appendix B.

3.1 FROM DATA SOURCES TO THE BRONZE STAGE

RIDE is built from two complementary public data sources. The first is the Infrabel Open Data portal Infrabel (2026), which provides the core railway data used throughout the pipeline, including train movement records with scheduled and observed timings, as well as infrastructure tables describing operational points and railway line sections. The second source is Open-Meteo Zippenfenig (2023),

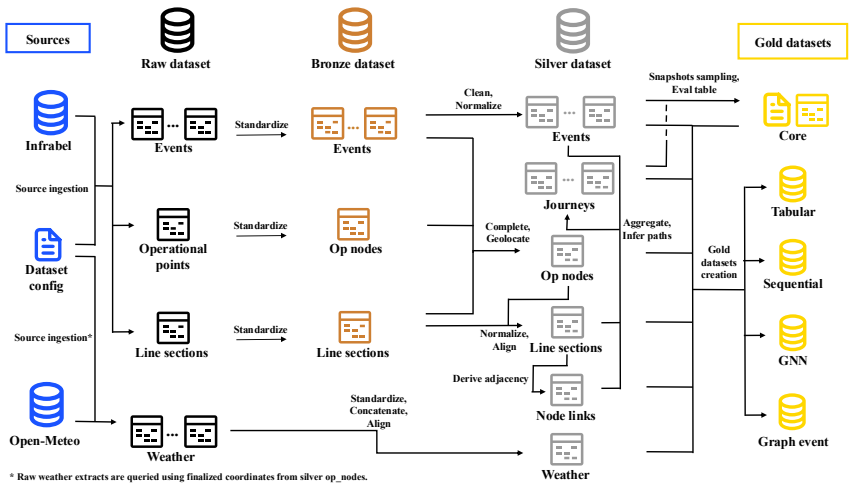

Figure 1: Overview of the RIDE data processing pipeline from sources to gold benchmark datasets.

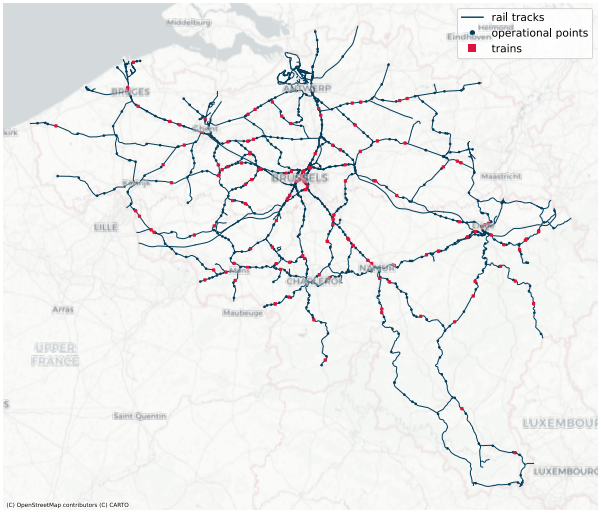

Figure 2: Visualization of the railway network for the snapshot at `2024-01-15 08:00:00`.

from which we obtain hourly historical weather observations aligned with operational points. The *bronze* stage provides a standardized ingestion layer for the acquired data. Conceptually, it performs schema harmonization, identifier and type normalization, coordinate extraction, and lightweight integrity checks while preserving the source-level structure of the data.

3.2 Silver Release: Relational Dataset

The *silver* release provides the reusable intermediate representation of RIDE: a relational dataset over events, journeys, infrastructure, and weather tables, as described in Table 1. It spares users from reproducing the core data preparation steps required to build a usable train delay prediction dataset, including cleaning, standardization, completion of missing or inconsistent information, and enrichment. In particular, the silver tier completes missing operational point information, reconstructs the railway network topology, sanitizes event timelines to enforce journey-level consistency, and infers the exact path of trains throughout the railway network. It provides a standardized foundation from which downstream users can construct alternative model-specific datasets.

The silver release contains 1,355 operational points, 1,212 line sections, and 1,797 node links. Across the 2023–2025 period, it includes 3.6 million journeys and 94.5 million train events, corresponding to

Table 1: Description of the Silver release tables.

Table	Description
events	train events, each corresponding to one train at one operational point with scheduled and observed timings, forming the core temporal table for delay prediction
journeys	journeys, each corresponding to one train itinerary on one day, with train-level metadata, journey statistics, and inferred exact paths through the railway network
op_nodes	operational points with names, types, and coordinates, serving as the nodes of the railway network
line_sections	fine-grained geospatial railway sections of the network whose geometry passes through multiple operational points
node_links	links between consecutive operational points with distances, serving as the graph edges induced by physical rail connections
weather	hourly weather observations aligned to operational points, providing exogenous context for downstream models

averages of 100,301 journeys and 2.6 million events per month. The weather table covers this period with hourly observations aligned to operational points, for a total of 35.7 million weather records.

3.3 Gold Release: Benchmark Datasets

The *gold* release is the model-ready tier of RIDE, materializing the silver data into two types of artifacts: (1) a shared benchmark *core*, which defines the common train/test data splits, targets, and evaluation metadata, and (2) model-specific datasets tailored to different prediction approaches. Its central concept is the *snapshot*, which serves as the fundamental unit of the benchmark. A snapshot represents the state of the railway network at a given time: it contains information on active trains (including schedule, past delays, ...), which is then used as model inputs for predicting future delays (Figure 2). The gold datasets are defined over a common list of training snapshots and testing snapshots provided by the benchmark *core*. The training and test sets are extracted from separate time periods, with training snapshots drawn from 2023–2024 and test snapshots drawn from 2025, so that evaluation reflects a realistic forward-in-time prediction setting. The benchmark *core* also provides a table containing all information required for evaluation. It is designed to support new gold datasets provided by downstream users, with alternative feature designs or model-specific representations, while preserving the same snapshot splits and test evaluation table under a common evaluation protocol, so that all models are compared on the same prediction instances and target values.

On top of this shared *core*, the gold release provides four model-specific datasets. The *tabular* dataset represents each train instance of each snapshot as a fixed-size feature vector and is used by the MLP, XGBoost, and Transformer models. The *sequential* dataset arranges the tabular features into event sequences and static features for the LSTM model. The *GNN* dataset represents each snapshot as a heterogeneous graph with node and edge features, used by the GNN model. The *graph-event* dataset computes aggregate statistics and retains relevant information from the silver dataset for the Graph-event model. The feature families used in the *tabular*, *sequential*, and *GNN* datasets are summarized in Table 2, while full schema and construction details are provided in Appendix B.5.

The gold release is provided in two tiers, *lite* and *standard*, covering the same 2023–2025 period but differing in scale: *lite* contains 15,000 training and 3,000 test snapshots, whereas *standard* contains 50,000 training and 10,000 test snapshots, with an average of 210 active trains per snapshot. The *lite* tier is a smaller benchmark configuration intended for faster experimentation and lower computational requirements, whereas the *standard* tier is the full benchmark setting used in Section 4.

4 RIDE Benchmark

As a second major contribution, we introduce a unified benchmark built on top of the RIDE dataset, with a common prediction task, fixed temporal splits, and a shared evaluation protocol across model families, and we provide benchmark experiments across a diverse set of models.

Table 2: Feature families in gold machine learning datasets.

Feature family	Description
train information	train operator and train category indicators derived from journey metadata
snapshot time features	cyclical and calendar features derived from the prediction snapshot timestamp
past event context	planned timings, delays, event types, and node embeddings for 15 previous events
future event context	planned timings, event types, and node embeddings for 15 future events
local network context	context for oriented rail segments and operational points, including distance, number of trains, and average delays
weather features	weather variables for each operational point at prediction time, including temperature, rain, snowfall, humidity, and wind speed

4.1 Delay Prediction Task

For a snapshot s at time t^s, the goal is to predict, for every active train, the delay at the next n scheduled events, where each event corresponds to a scheduled arrival, departure, or passage at an operational point (see Table 1). For a train i in snapshot s, we define the delay at future scheduled event $j \in \{1, \dots, n\}$ as $d_{s,i,j} = t^{\text{obs}}_{s,i,j} - t^{\text{sched}}_{s,i,j}$, where $t^{\text{sched}}_{s,i,j}$ and $t^{\text{obs}}_{s,i,j}$ denote the scheduled and observed times of that event, respectively. The model prediction is denoted by $\hat{d}_{s,i,j}$. We set $n = 15$, which corresponds to an average prediction horizon of approximately 40 minutes, a range relevant to operational use cases such as passenger information updates and short-term dispatching decisions.

For later use in this section, we define $d^{\text{last}}_{s,i}$ as the last known delay of train i at snapshot s. It is computed from the previous observed event delay $d_{s,i,-1}$ with an adjustment ensuring that the implied time of the next event does not fall before the snapshot: $d^{\text{last}}_{s,i} = d_{s,i,-1} + \max\big(0,\ t^s - t^{\text{sched}}_{s,i,1} - d_{s,i,-1}\big)$, where $t^{\text{sched}}_{s,i,1}$ is the scheduled time of the next event.

4.2 Evaluation Metrics

We evaluate models using mean absolute error (MAE) and root mean squared error (RMSE), computed over the set $\mathcal{E}$ containing all prediction targets across all evaluation snapshots. Each element $(s, i, j) \in \mathcal{E}$ corresponds to a future event j of a train instance i in snapshot s. Then:

$$\text{MAE} = \frac{1}{|\mathcal{E}|} \sum_{(s,i,j)\in\mathcal{E}} |\hat{d}_{s,i,j} - d_{s,i,j}|, \quad \text{RMSE} = \sqrt{\frac{1}{|\mathcal{E}|} \sum_{(s,i,j)\in\mathcal{E}} (\hat{d}_{s,i,j} - d_{s,i,j})^2}.$$

Since delays are expressed in seconds, both metrics are also expressed in seconds and are therefore directly interpretable. While these aggregate metrics provide a global measure of predictive accuracy, they can hide important variations across prediction settings. We therefore complement MAE and RMSE with breakdowns along two dimensions:

- prediction horizon, defined by how far in the future the event occurs, i.e., $t^{\text{obs}}_{s,i,j} - t^s$.
- delay-delta bins, corresponding to delay change, defined from the difference between the target delay and the last known delay at prediction time, $d_{s,i,j} - d^{\text{last}}_{s,i}$.

These complementary breakdowns make it possible to analyze the strengths and weaknesses of models across different forecasting regimes.

4.3 Evaluation Protocol

The benchmark is built from fixed training and test snapshot splits defined once in the shared gold benchmark *core*. A single standardized `test_eval_table` is constructed from the test snapshots and used for final evaluation across all model families, ensuring that every method is evaluated on the same prediction instances and target values. For learning-based models, hyperparameters are selected on the training split using Optuna Akiba et al. (2019), optimizing validation MAE on the last 10% of training snapshots in temporal order. Final benchmark results are then obtained by retraining each

model with the selected hyperparameters on 10 seeds and evaluating on the common test split. In the main paper, we report results on the *standard* tier only; full details for training, hyperparameter search, and computation budgets are provided in Appendices C.3, C.4 and C.5.

4.4 MODELS

This subsection presents the models considered in our experiments. We compare three families of approaches under the RIDE benchmark setting: non-learning models, namely the Translation and Graph-event models, which provide deterministic reference points; tree-based models, represented by XGBoost, a strong tabular predictor for structured data; and deep learning models, namely MLP, LSTM, Transformer, and GNN, which exploit richer sequential, relational, and interaction structure. Due to the diversity of prediction targets, operational scopes, and data representations used in the literature, several prior models cannot be directly adopted in a comparable form. We therefore implement benchmark-specific versions inspired by the literature but adapted to the RIDE prediction setting and evaluation protocol. For all learning-based models, we predict delay relative to $d_{s,i}^{\text{last}}$ at prediction time and recover absolute delay by adding $d_{s,i}^{\text{last}}$ back at inference time. This reparameterization leaves the benchmark target unchanged but was found to improve predictive performance. These models are built from the same underlying feature families, which are structured differently depending on the architecture. Architectural and training details are provided in Appendix C.2.

Translation. A commonly used baseline in railway settings is the Translation Arthaud et al. (2024); Wang & Work (2015). For every target event, it predicts the last known delay $d_{s,i}^{\text{last}}$. Despite its simplicity, it is a strong reference point for evaluating the added value of more complex models.

Graph-event. The graph-event model is a non-learning rule-based approach for train delay prediction, inspired by prior graph- and event-based delay-propagation methods Goverde (2010); Wei et al. (2015). Starting from each prediction snapshot, it simulates the future evolution of active trains over the railway network using the train positions and schedules, graph connectivity, and precedence constraints between trains sharing infrastructure. Expected travel times between consecutive events are obtained from empirical travel-time statistics estimated from the training data, and delays are then propagated forward event by event to produce the future delay predictions. This model provides a deterministic graph-based reference point that contrasts with the learning-based benchmark models.

XGBoost. We also include an XGBoost model serving, a tree-based model, operating on the same fixed-size tabular representation used for the MLP Du et al. (2025); Gao et al. (2024). In this setting, we train one independent XGBoost regressor for each future event slot, so that each model predicts the delay at a fixed position among the next n scheduled events from the same tabular features.

MLP. We include a standard multilayer perceptron model applied to the fixed-size tabular representation of each train at each snapshot Peters et al. (2005); Oneto et al. (2018). The model maps the tabular input features directly to predicted delays at future events.

LSTM. Sequential models are a natural fit for train delay prediction. Recurrent neural architectures such as LSTMs have been explored for this task Huang et al. (2020); Yu et al. (2023). To match our multi-step prediction setting, we use an encoder-decoder LSTM that operates on one train at a time: the encoder processes the past event sequence, non-sequential features are embedded with an MLP, and the decoder combines this context with future event features to predict delays at future events.

Transformer. Transformer-based models have recently been used for train delay prediction Elliker et al. (2026); Arthaud et al. (2024). The model adopts an encoder-only architecture in which each train in the snapshot is represented as a token, allowing trains to attend to one another through self-attention mechanisms before predicting delays at future events.

GNN. Graph-based models are also a natural fit for train delay prediction, and graph neural network (GNN) approaches have been explored in the literature Li et al. (2024); Huang et al. (2024). To match our prediction setting, we implement a heterogeneous GNN following this line of work: each snapshot is represented as a graph with train nodes and station nodes, edges between adjacent stations, and train-to-station edges for past and future stations encoding train itineraries. Message passing and

Table 3: Standard-tier test performance. MAE/RMSE in seconds; ±: std. over 10 seeds.

Model	MAE	RMSE
Translation	96.65	233.42
Graph-event	88.41	232.48
MLP	77.20 ± 0.04	203.21 ± 0.40
XGBoost	76.58 ± 0.01	203.46 ± 0.02
LSTM	74.62 ± 0.27	202.63 ± 0.77
Transformer	74.54 ± 0.25	195.39 ± 0.59
GNN	$\mathbf{73.62 \pm 0.19}$	$\mathbf{194.56 \pm 0.88}$

Table 4: Test-set MAE in seconds by prediction horizon on the standard tier. Bins are in minutes.

Model	0:5	5:10	10:15	15:20	20:25	25:30	30:35	35:40	40:45	45+
Translation	37.5	57.2	72.3	84.9	95.7	107.4	121.7	141.7	166.9	312.2
Graph-event	30.5	48.5	63.1	75.2	86.3	99.0	115.2	135.2	159.1	305.9
MLP	25.9	42.2	55.2	66.4	76.3	87.0	99.9	116.9	138.6	266.8
XGBoost	**24.0**	41.1	54.7	66.2	76.0	86.7	99.6	116.9	138.4	267.4
LSTM	24.1	**39.7**	**52.6**	63.6	73.3	84.1	97.1	114.3	136.0	264.3
Transformer	25.2	41.2	53.9	64.5	73.9	84.3	96.7	112.8	133.3	252.6
GNN	25.8	41.4	53.2	**63.3**	**72.3**	**82.3**	**94.5**	**110.4**	**130.6**	**252.2**

edge updates are performed over this graph with heterogeneous GINE-style node aggregations Hu et al. (2019), and delays at future events are predicted on the future train-to-station edges.

4.5 Experimental Results and Analysis

Aggregate Results. Table 3 summarizes the main results on the standard benchmark tier. The GNN achieves the best mean performance, with 73.62 MAE and 194.56 RMSE, while the Transformer and LSTM follow closely at 74.54 and 74.62 MAE, respectively. Among tabular learning-based models, XGBoost outperforms the MLP on MAE (76.58 vs. 77.20), while the MLP remains marginally better on RMSE (203.21 vs. 203.46). The graph-event model clearly improves over the simple translation baseline (88.41 vs. 96.65 MAE), yet remains behind the learning-based approaches. Despite its simplicity, the translation baseline is surprisingly strong, providing a reference point that is difficult to beat by a large margin. Interestingly, the LSTM yields strong performance, close to the Transformer, despite operating on one train at a time and lacking an explicit mechanism to propagate information through the network; this suggests that effective single-train sequence modeling, combined with local network context features, already captures much of the signal needed for this benchmark. This may also reflect, in part, the relative scarcity of situations where knowing the precise position and itinerary of each train is required for accurate delay prediction. More broadly, the strongest learning-based models remain close to each other in absolute terms, so the benchmark does not point to a single overwhelmingly dominant architecture. Taken together, these results indicate that architectural changes alone do not lead to dramatic gains on this benchmark, and that further progress may also depend on better feature design and problem-specific modeling choices. Confidence intervals and statistical significance tests for the Standard-tier results are reported in Appendices C.6.1 and C.6.2, respectively, confirming the robustness of the GNN results across random seeds.

Results by Prediction Horizon. Table 4 breaks down performance by prediction horizon. A clear pattern emerges: XGBoost performs best in the very shortest 0:5 minute bin, although the LSTM remains extremely close, and the LSTM is then strongest from 5 to 15 minutes. From 15 minutes onward, the GNN becomes the best-performing model across all remaining horizon bins, while the Transformer remains competitive throughout without clearly dominating either the short- or long-horizon regime. The strong 0:5 minute performance of XGBoost is also consistent with the delay-change analysis below, where it performs particularly well in the dominant regime of small delay changes, which is itself especially prevalent at very short horizons. One important caveat, however, is that the latest horizon bins are progressively biased toward trains that accumulate delay,

Table 5: Test-set MAE in seconds by delay-delta bin on the standard tier. Bins are in minutes; negative values correspond to delay recovery and positive values to delay accumulation.

Model	<-5	-5:-2	-2:-1	-1:-0.5	-0.5:0	0:0.5	0.5:1	1:2	2:5	5:10	10+
Translation	428.1	174.8	85.2	44.3	**14.7**	**13.5**	43.2	85.1	184.8	408.5	1260.2
Graph-event	**169.6**	**81.4**	53.2	39.3	31.2	32.2	45.7	75.9	169.5	398.2	1259.3
MLP	235.5	92.7	53.4	37.2	28.9	28.1	37.2	61.0	136.8	322.9	1053.6
XGBoost	265.3	100.7	56.5	37.2	24.9	23.9	**35.4**	59.8	135.7	324.0	1062.3
LSTM	200.3	81.5	**49.1**	**35.5**	29.0	28.7	37.2	**59.2**	131.3	314.3	1046.4
Transformer	197.1	81.6	50.6	36.9	27.8	27.8	39.5	63.7	136.6	312.3	**968.5**
GNN	197.8	82.0	50.2	36.9	29.2	28.6	38.4	61.2	**131.0**	**299.5**	970.5

with this effect being particularly pronounced in the final 45+ bin, since these bins become thinner by construction under the fixed future-event prediction window. This phenomenon is illustrated in Appendix C.1. As a result, performance differences across horizon bins should not be interpreted as a pure horizon effect alone, but rather as reflecting a mixture of horizon length and delay regime.

Results by Delay Change. Table 5 reports performance by delay-delta bin. The translation baseline is strongest around zero delay change, where propagating the current delay provides a strong approximation. Excluding this baseline, XGBoost is the strongest model in the central delay-delta regime around zero, roughly from -0.5 to 1 minute, the most common regime in the data. This suggests that XGBoost tends to make comparatively conservative predictions. The LSTM performs particularly well on moderate negative delay-delta bins, indicating that a sequential prior is well suited for modeling how trains recover from delays. For moderate delay accumulation, between 0.5 and 2 minutes, tabular and sequential models tend to outperform the GNN and Transformer, suggesting that, in this regime, the fixed train-level feature representation already contains enough information to support accurate predictions, without requiring explicit modeling of network interactions. At the opposite end, the graph-event model performs best on large negative delay deltas, i.e. strong delay recovery, which is consistent with its tendency to predict travel times that are on average faster than the scheduled ones. In contrast, the GNN performs best across most large positive delay-delta bins, while in the most extreme accumulation regime it remains essentially tied with the Transformer. This suggests that models with learned interaction patterns between trains are better able to capture substantial delay increases, likely because these are tied to propagation effects in the network.

Together, these two evaluation breakdowns provide a more informative view of model behavior than aggregate MAE and RMSE alone. They make it possible to identify which models perform best in specific forecasting regimes, such as short- versus long-horizon prediction or delay recovery versus delay accumulation, and thereby support broader insights into the dynamics of train delay prediction.

5 Conclusion

In this work, we introduced RIDE, an open dataset and benchmark for train delay prediction built at nationwide scale over the Belgian railway network. RIDE is the first resource to provide both a reusable intermediate silver release, intended as a standardized foundation for downstream dataset construction, and model-ready gold benchmark datasets, together with a unified evaluation protocol that we use to compare the non-learning Translation baseline and Graph-event model, the tree-based model XGBoost, and the deep learning models MLP, LSTM, Transformer, and GNN. Our benchmark results confirm a clear gap between learning-based and non-learning models, while differences among the strongest learning-based models remain comparatively small, with graph neural networks achieving the best mean performance. The competitiveness of recurrent, attention-based, and tabular models, despite their different inductive biases, suggests that further progress may depend as much on feature design as on architecture. More broadly, the proposed evaluation protocol goes beyond aggregate accuracy by enabling regime-specific analysis of model behavior, which helps reveal where different approaches succeed or fail and supports more nuanced insights into train delay prediction. While some findings may depend on the specific characteristics of the Belgian passenger railway network, we hope that RIDE will provide a useful foundation for future work and facilitate progress on delay prediction, delay propagation, and learning over large-scale transportation systems.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experimental design, to assist with method implementation, and to support dataset cleaning and reformatting. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, summarize or analyze existing literature, improve the readability of the manuscript, and identify relevant literature.

We did not use generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, assist with translation, support qualitative or thematic data analysis, or interpret results. The use of generative AI to formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in writing proofs, or propose or refine hypotheses was not applicable to this work.

We reviewed all AI-assisted work, including generated or modified code, research-related assistance, and manuscript edits, and verified the resulting content for correctness and appropriateness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We use open railway operational data and weather data released under licenses that permit reuse. The dataset contains only non-personal operational information, such as train times, stations, delays, infrastructure metadata, and weather observations, and no data about individual passengers or staff. Our use of the data therefore complies with the providers' licenses and does not raise additional privacy concerns.

REPRODUCIBILITY STATEMENT

All datasets, code, and instructions required to reproduce the data-processing and benchmark pipeline are publicly available. Links to the RIDE dataset releases and the accompanying code repository are provided in Appendix A.1. The repository includes instructions for reproducing the full pipeline end to end, while additional implementation, training, and hyperparameter details are documented in Appendices C.2, C.3, and C.4.

REFERENCES

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 international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Farid Arthaud, Guillaume Lecoeur, and Alban Pierre. Transformers à grande vitesse: Massively parallel real-time predictions of train delay propagation. *Journal of Rail Transport Planning & Management*, 29:100418, 2024.

Vinicius Pozzobon Borin, Jean Michel de Souza Sant'Ana, Usama Raheel, and Nurul Huda Mahmood. Integrating meteorological and operational data: A novel approach to understanding railway delays in finland. *arXiv preprint arXiv:2601.16592*, 2026.

Koyena Chowdhury, Paramita Koley, Abhijnan Chakraborty, and Saptarshi Ghosh. Rstgcn: Railway-centric spatio-temporal graph convolutional network for train delay prediction. *arXiv preprint arXiv:2510.01262*, 2025.

Mark M Dekker, Alexey N Medvedev, Jan Rombouts, Grzegorz Siudem, and Liubov Tupikina. Modelling railway delay propagation as diffusion-like spreading. *EPJ Data Science*, 11(1):44, 2022.

Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy*, pp. 287–290. 2022.

Theo Döllmann. Bahn-vorhersage dataset: an open archive of most train delays in germany since september 2021. 2025.

Jingjie Du, Jian Wang, Hui Zhang, Tao Wang, Runmei Li, and Yanran Ji. Xgboost-based train delay prediction. In *2025 IEEE International Conference on Intelligent Rail Transportation (ICIRT)*, pp. 168–173. IEEE, 2025.

Clément Elliker, Jesse Read, Sonia Vanier, and Albert Bifet. Simulation-driven railway delay prediction: An imitation learning approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 20977–20984, 2026.

Tianze Gao, Junhua Chen, and Huizhang Xu. Data-driven train delay prediction incorporating dispatching commands: An xgboost-metaheuristic framework. *IET Intelligent Transport Systems*, 18(10):1777–1796, 2024.

Rob MP Goverde. A delay propagation algorithm for large-scale railway traffic networks. *Transportation Research Part C: Emerging Technologies*, 18(3):269–287, 2010.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Ping Huang, Chao Wen, Liping Fu, Javad Lessan, Chaozhe Jiang, Qiyuan Peng, and Xinyue Xu. Modeling train operation as sequences: A study of delay prediction with operation and weather data. *Transportation research part E: logistics and transportation review*, 141:102022, 2020.

Ping Huang, Jingwei Guo, Shu Liu, and Francesco Corman. Explainable train delay propagation: A graph attention network approach. *Transportation Research Part E: Logistics and Transportation Review*, 184:103457, 2024.

Infrabel. Infrabel open data. `https://infrabel.opendatasoft.com/pages/home/`, 2026.

Zhongcan Li, Ping Huang, Chao Wen, Wei Dong, Yindong Ji, and Filipe Rodrigues. Railway network delay evolution: A heterogeneous graph neural network approach. *Applied Soft Computing*, 159: 111640, 2024.

Qianyi Liu, Shengjie Wang, Zhongcan Li, Li Li, Jun Zhang, and Chao Wen. Prediction of high-speed train delay propagation based on causal text information. *Railway Engineering Science*, 31(1): 89–106, 2023.

Ratthaphong Meesit and Rathathammanoon Suwannasri. Real-time delay prediction for single-track intercity railways. *Multimodal Transportation*, pp. 100288, 2025.

Rahul Nair, Thanh Lam Hoang, Marco Laumanns, Bei Chen, Randall Cogill, Jácint Szabó, and Thomas Walter. An ensemble prediction model for train delays. *Transportation Research Part C: Emerging Technologies*, 104:196–209, 2019.

Luca Oneto, Emanuele Fumeo, Giorgio Clerico, Renzo Canepa, Federico Papa, Carlo Dambra, Nadia Mazzino, and Davide Anguita. Train delay prediction systems: a big data analytics perspective. *Big data research*, 11:54–64, 2018.

Jan Peters, Bastian Emig, Marten Jung, and Stefan Schmidt. Prediction of delays in public transportation using neural networks. In *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, volume 2, pp. 92–97. IEEE, 2005.

Thomas Spanninger, Alessio Trivella, and Francesco Corman. Approaches for real-time train delay prediction. In *20th Swiss Transport Research Conference (STRC 2020)(virtual)*. STRC, 2020.

Thomas Spanninger, Alessio Trivella, Beda Büchel, and Francesco Corman. A review of train delay prediction approaches. *Journal of Rail Transport Planning & Management*, 22:100312, 2022.

Kah Yong Tiong and Carl-William Palmqvist. Quantitative methods for train delay propagation research. *Transportation Research Procedia*, 72:80–86, 2023.

Kah Yong Tiong, Zhenliang Ma, and Carl-William Palmqvist. A review of data-driven approaches to predict train delays. *Transportation Research Part C: Emerging Technologies*, 148:104027, 2023.

Ren Wang and Daniel B Work. Data driven approaches for passenger train delay estimation. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 535–540. IEEE, 2015.

Dali Wei, Hongchao Liu, and Yong Qin. Modeling cascade dynamics of railway networks under inclement weather. *Transportation Research Part E: Logistics and Transportation Review*, 80: 95–122, 2015.

Jianqing Wu, Xukai Xiao, Yitao Zhou, Bo Du, Jun Shen, Yishan Chen, Bi Wang, and Qiang Wu. A railway network dataset incorporating multi-type train operation records and train scheduling. *Scientific Data*, 2025.

Tiong Kah Yong, Zhenliang Ma, and Carl-William Palmqvist. Ap-grip evaluation framework for data-driven train delay prediction models: systematic literature review. *European Transport Research Review*, 17(1):13, 2025.

Ke Yu, Chuiyun Kong, Limin Zhong, Junfeng Fu, and Jie Shao. Delay prediction with spatial–temporal bi-directional lstm in railway network. *ICT Express*, 9(5):921–926, 2023.

Dalin Zhang, Yunjuan Peng, Yi Xu, Chenyue Du, Yumei Zhang, Nan Wang, Yunhao Chong, Hongwei Wang, Daohua Wu, Jintao Liu, et al. A high-speed railway network dataset from train operation records and weather data. *Scientific data*, 9(1):244, 2022.

Patrick Zippenfenig. Open-meteo.com weather api, 2023. URL `https://open-meteo.com/`.

A Data Release, Ethics, Reproducibility and Limitations

A.1 Data and Code Availability

All RIDE data releases and accompanying code are publicly available at the following links.

Code – anonymous GitHub repository
Silver – anonymous Hugging Face dataset
Gold Standard – anonymous Hugging Face dataset
Gold Lite – anonymous Hugging Face dataset

The full data-processing and benchmark pipeline can be reproduced end to end using the instructions provided in the GitHub repository.

A.2 Maintenance Plan

RIDE is distributed and maintained by the authors through public Hugging Face dataset repositories, with the accompanying processing and benchmark code hosted on GitHub. Published dataset releases are kept unchanged to preserve reproducibility, and persistent Digital Object Identifiers (DOIs) will be assigned through Hugging Face.

Any future correction, extension, or additional temporal coverage will be published separately rather than modifying the existing RIDE repositories. This ensures that the current datasets and benchmark splits remain stable reference points for previously reported results. Changes introduced in subsequent releases will be documented explicitly.

Potential data issues can be reported through the public code or dataset repositories. Confirmed issues will be documented and, when necessary, addressed in a separate corrected release so that the original published data remain unchanged.

A.3 Licenses

The source data used to construct RIDE are released under open-data licenses. The Infrabel punctuality, operational-point, and railway-line-section datasets are published through Infrabel Open Data under the CC0 Universal open license. The weather variables are obtained from the Open-Meteo

Archive API, whose API data are provided under the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

The released RIDE datasets are distributed under CC-BY 4.0. This license is chosen to preserve the attribution requirements of the Open-Meteo-derived weather variables while remaining compatible with the CC0 status of the Infrabel source data. Users of RIDE should attribute RIDE, Infrabel, and Open-Meteo, and should not imply endorsement by any source-data provider. The accompanying source code is released under the MIT license.

A.4 Broader Impact

RIDE is intended to support research on train delay prediction and delay propagation, with potential downstream benefits for passenger information, dispatching, traffic management, and the reliability of public transport systems. We do not identify direct negative societal impacts from releasing this benchmark: it is built from open operational and weather data, contains no personal information about passengers or staff, and is intended for offline research and evaluation rather than direct automated operational control.

A.5 Limitations

Geographical scope. RIDE is constructed from the Belgian railway network, and railway systems may differ across countries in rolling stock, signalling systems, network topology, traffic density, and operational rules. The quantitative results and relative performance of model families observed here may therefore not transfer unchanged to other railway systems, and evaluating the benchmark framework across countries remains an important direction for future work. Nevertheless, RIDE covers a heterogeneous national network and diverse passenger services, from high-speed to intercity and local trains, providing varied operating conditions within a single railway system.

Traffic coverage. The raw punctuality data used to construct RIDE covers domestic and international passenger trains and therefore does not represent all railway traffic operating on the Belgian network. In particular, freight trains are not included in the stop-level raw punctuality records used by RIDE. Their interaction with passenger services may nevertheless affect network conditions and delay propagation. Consequently, the benchmark does not explicitly model the complete traffic state of the railway network.

Infrastructure and operational information. The public data available for RIDE captures network topology and selected static infrastructure characteristics, but not the fine-grained operational state of the railway system. In particular, it does not include information such as individual signal states, interlocking decisions, block occupancy, route setting, or detailed traffic-management actions. These factors can directly influence train movements and delay propagation but cannot be represented explicitly in the current benchmark. The models must therefore infer their effects indirectly from observable train movements, network structure, timetable information, and historical operating conditions.

B Dataset Construction and Analysis

This section provides additional detail on the raw, bronze, silver, and gold stages of RIDE. It describes the original data sources, the main processing steps used to transform them into the released datasets and benchmark-ready artifacts, and additional analyses and visualizations of the resulting datasets.

B.1 Sources

Infrabel Open Data portal. The railway data used in RIDE are obtained from the Infrabel Open Data portal Infrabel (2026). Infrabel is the Belgian railway infrastructure manager and publishes open datasets covering punctuality, infrastructure, safety, and other aspects of the Belgian rail network. RIDE relies on this portal as the source of both operational train movement data and infrastructure descriptions, which are later combined to construct event, journey, and railway-network

representations. The data are made available by Infrabel under the portal's open-data terms, which publish the datasets under a CC0 Universal open licence.

Open-Meteo. Weather data are obtained from the Open-Meteo Historical Weather API Zippenfenig (2023). Open-Meteo provides open access to historical meteorological time series through an archive endpoint queried by geographic coordinates, date range, timezone, and selected weather variables. RIDE uses this source to add exogenous weather context to the railway data, using operational-point coordinates as the spatial interface between railway events and weather observations. Open-Meteo API data are provided under the Creative Commons Attribution 4.0 International licence (CC BY 4.0).

B.2 Raw Data

The raw data layer contains four source datasets: three railway datasets from the Infrabel Open Data portal and one weather dataset derived from the Open-Meteo Historical Weather API. The Infrabel sources provide complementary views of railway operations and infrastructure: monthly punctuality files describe train movements, while the operational-points and line-sections catalogues provide the spatial and topological reference needed to interpret those movements on the Belgian railway network.

Raw punctuality files. We download all 36 monthly raw punctuality files covering January 2023 through December 2025. These files contain arrival and departure records for domestic and international passenger trains at their stopping points, indexed by service day and train number. They provide the core operational information used in RIDE, including train identifiers, relations, operators, operational point identifiers, scheduled and observed arrival and departure times, line identifiers, and delays.

Operational points. The operational-points catalogue lists the railway network's operational points together with their identifiers, geographic positions, official names, and Infrabel internal classifications. This table provides the spatial anchor used to associate train events with locations in the Belgian railway network.

Line sections. The railway line-sections catalogue describes the segmentation of the rail network into line sections, including section geometries, endpoint operational points, line identifiers, and technical infrastructure attributes. This source provides the geometric and topological basis from which RIDE reconstructs railway links between operational points.

Weather. For each operational point in the finalized silver-stage `op_nodes` table, we query hourly weather data from the Open-Meteo Historical Weather API. The queried period covers the full 2023–2025 railway observation window, extended by one day on both sides, from 2022-12-31 to 2026-01-01 to account for edge cases. Each query uses the operational point latitude and longitude, the `Europe/Brussels` timezone, and the following hourly variables: `temperature_2m`, `rain`, `snowfall`, `relative_humidity_2m`, `wind_speed_10m`, and `weather_code`. Because these coordinates are only finalized after the silver operational-node construction, weather acquisition is performed after the silver stage.

B.3 Bronze Data

The bronze layer provides a standardized version of the raw source files while preserving their original informational content. Its purpose is to make the heterogeneous raw inputs easier to process in later stages, rather than to perform dataset-level cleaning or modeling decisions. The bronze layer keeps the same table structure as the raw layer: no source table is added or removed, so we describe it globally rather than table by table. In this layer, source-specific column names are mapped to a consistent schema, unused fields are removed, basic data types are normalized, and the resulting CSV inputs are stored as Parquet tables. Temporal fields from the punctuality data are still kept close to their source representation, with full timestamp reconstruction, consistency checks, and journey-level filtering deferred to the silver layer. The exact column mappings and bronze-stage transformations are specified in the bronze manifests released with the code.

The bronze layer therefore acts as a stable interface between raw data acquisition and the relational construction performed in the silver stage. It makes the pipeline reproducible and easier to inspect, while ensuring that substantive transformations, such as operational-node correction, railway-link reconstruction, event cleaning, journey construction, path inference, and weather alignment, remain explicit in the later stages.

B.4 SILVER RELEASE

The silver release is the main reusable relational data layer of RIDE. It converts the standardized bronze railway inputs and raw weather extracts into a coherent set of railway, weather, and infrastructure tables that can be used independently of the benchmark representations defined in the gold layer. Its purpose is to provide a cleaned and structurally consistent view of Belgian passenger railway operations over the full 2023–2025 period, while preserving enough event-level detail for users to construct alternative feature sets, temporal windows, or model-specific datasets.

B.4.1 SILVER RELATIONAL DATA

This subsection documents the structure of the silver release as a relational dataset. In the following paragraphs, we describe the schemas of these tables, their relational structure, and provide a small illustrative example showing how they can be joined.

Table schemas. The silver release contains six Parquet tables. The `events` table (Table 6) contains event-level train records with timestamps, event types, delays, and arrival/departure line context. The `journeys` table (Table 7) contains one record per operated train service, with journey-level metadata, timing summaries, delay summaries, event counts, and inferred `deduced_paths`. The `op_nodes` table (Table 8) contains the operational-point catalogue with identifiers, names, types, and coordinates. The `line_sections` table (Table 9) describes railway line sections through line identifiers, endpoint operational points, geometries, track counts, and ordered matched operational nodes. The `node_links` table (Table 10) contains graph links between consecutive operational nodes along line-section geometries. Finally, the `weather` table (Table 11) contains hourly weather observations associated with operational points. These tables report the field name, data type, and role of each column for each table, while short notes below the tables clarify non-obvious keys and references.

Table 6: Schema of the silver `events` table.

Field	Type	Description
`train_id`	string	Train identifier for the operated service.
`service_date`	string	Scheduled service date associated with the train.
`op_id`	integer	Operational point at which the event occurs.
`event_type`	string	Event type: arrival, departure, or passage.
`arr_line_id`	string/null	Railway line identifier on the arrival side of the event.
`dep_line_id`	string/null	Railway line identifier on the departure side of the event.
`planned_ts`	timestamp	Scheduled timestamp of the event.
`observed_ts`	timestamp	Observed timestamp of the event.
`delay_sec`	integer	Event delay in seconds.

The pair `train_id`, `service_date` identifies the events belonging to the same operated train service. The fields `arr_line_id` and `dep_line_id` refer to railway line identifiers, i.e. `line_id`, not to `line_section_id`.

Table 7: Schema of the silver `journeys` table.

Field	Type	Description
`train_id`	string	Train identifier for the operated service.
`service_date`	string	Scheduled service date associated with the train.
`train_relation`	string	Train relation code.
`operator`	string	Railway operator.
`relation_direction`	string	Route description of the operated service.
`start_op_id`	integer	Operational point of the first event.
`end_op_id`	integer	Operational point of the last event.
`start_planned_ts`	timestamp	Scheduled timestamp of the first event.
`end_planned_ts`	timestamp	Scheduled timestamp of the last event.
`start_observed_ts`	timestamp	Observed timestamp of the first event.
`end_observed_ts`	timestamp	Observed timestamp of the last event.
`events_count`	integer	Number of events in the journey.
`max_delay_sec`	integer	Maximum observed delay along the journey.
`min_delay_sec`	integer	Minimum observed delay along the journey.
`deduced_paths`	array	List of inferred `link_id` sequences between consecutive events.

The field `deduced_paths` is a list of paths, with one path per transition between consecutive events; each path is itself a list of `link_id`s, and transitions that remain at the same operational point are represented by empty lists.

Table 8: Schema of the silver `op_nodes` table.

Field	Type	Description
`op_id`	integer	Operational point identifier.
`op_name`	string	Operational point name.
`op_type`	string	Operational point type or category.
`lat`	float	Latitude in WGS84 coordinates.
`lon`	float	Longitude in WGS84 coordinates.

Table 9: Schema of the silver `line_sections` table.

Field	Type	Description
`line_section_id`	string	Line-section identifier.
`line_id`	string	Railway line identifier.
`op_begin_id`	integer	Operational point at the beginning of the line section.
`op_end_id`	integer	Operational point at the end of the line section.
`nb_tracks`	integer	Number of tracks in the line section.
`geoshape`	string	LineString geometry of the line section.
`matched_op_node_ids`	array	List of `op_id`s aligned with the geometry vertices.

The field `matched_op_node_ids` has the same order as the coordinates in `geoshape`; entries are null for geometry vertices that are not matched to an operational node. A railway line identified by `line_id` may contain multiple line sections.

Table 10: Schema of the silver node_links table.

Field	Type	Description
link_id	integer	Node-link identifier.
u_node_id	integer	First endpoint operational node.
v_node_id	integer	Second endpoint operational node.
line_section_id	string	Line section from which the link is derived.
distance_m	float	Link distance in meters.

Node links are treated as undirected graph edges. The pair u_node_id, v_node_id is not unique: multiple rows may connect the same endpoints when they are derived from different line sections, so link_id is the unique link identifier.

Table 11: Schema of the silver weather table.

Field	Type	Description
op_id	integer	Operational point identifier.
time	timestamp	Hourly weather timestamp.
temperature_2m	float	Air temperature at 2 m above ground, in degrees Celsius.
rain	float	Rain amount, in millimeters.
snowfall	float	Snowfall amount.
relative_humidity_2m	float	Relative humidity at 2 m above ground, in percent.
wind_speed_10m	float	Wind speed at 10 m above ground.
weather_code	integer	WMO weather code.

The field time is stored at hourly resolution and represents the Europe/Brussels local time used when querying Open-Meteo.

Relational structure. Table 12 summarizes the relational structure of the silver release. At the train-operation level, events and journeys are linked by train_id and service_date: events provide ordered event-level observations, while journeys aggregate them into one record per operated train service. Railway infrastructure is represented through op_nodes, line_sections, and node_links. op_nodes define the operational points referenced by events, line_sections, node_links, and weather. line_sections store section geometries, endpoint identifiers, and the ordered operational nodes matched along each section; Figure 3 shows the corresponding full-network visualization of op_nodes and line_sections. node_links expand these ordered node sequences into graph edges. Journey paths are stored as ordered lists of link_ids, linking journey-level records back to the railway graph. Finally, weather provides exogenous context that can be aligned with train-operation records through operational points and timestamps. Figure 4 provides an illustrative example of how these tables connect around a single train operation and its associated infrastructure and weather context.

Table 12: Relational structure of the silver release.

Table	Key fields	Description
`events`	`train_id`, `service_date`, `op_id`	One train event at one operational point; stores event-level train movements, timestamps, event types, line context, and delays.
`journeys`	`train_id`, `service_date`, `deduced_paths`	One train journey on one service date; stores journey-level metadata, numeric summaries, event counts, and inferred paths describing the exact route taken by the train through the railway graph.
`op_nodes`	`op_id`	One operational point; stores node identifiers, names, coordinates, and classifications.
`line_sections`	`line_section_id`, `op_begin_id`, `op_end_id`, `matched_op_node_ids`	One railway line section; stores infrastructure geometry, line identifiers, endpoint operational points, and matched operational nodes along the geometry.
`node_links`	`link_id`, `u_node_id`, `v_node_id`, `line_section_id`	One graph edge between consecutive operational nodes; represents railway graph connectivity and link distances.
`weather`	`op_id`, `time`	One hourly operational-point record with temperature, rain, snowfall, humidity, wind speed, and weather code.

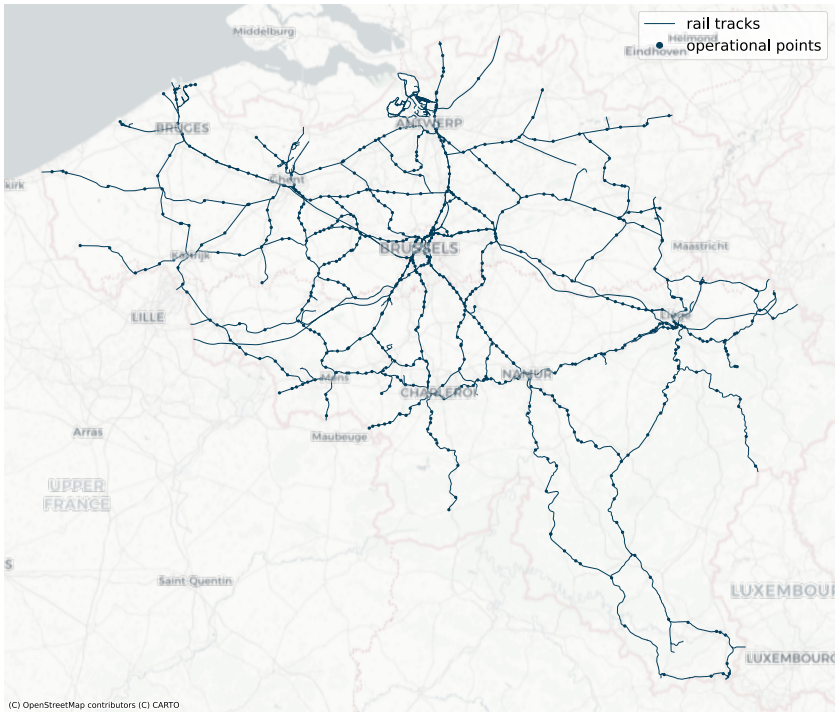

Figure 3: Visualization of the railway network. Some track segments appear to terminate away from a plotted operational point because the published line-section geometries can extend to infrastructure endpoints that are not represented in the published operational-point catalogue.

Illustrative example. Figure 4 shows a small illustrative example of how the silver tables connect around one train journey with four consecutive events, together with the corresponding infrastructure and weather context.

journeys

train_id	service_date	train_relation	deduced_paths	...
8622	08JAN2023	P	[[354],[],[353,352]]	...

events

train_id	service_date	op_id	type	planned_ts	observed_ts	...
8622	08JAN2023	801	D	20:15:00	20:15:58	...
8622	08JAN2023	788	A	20:19:00	20:19:23	...
8622	08JAN2023	788	D	20:20:00	20:20:38	...
8622	08JAN2023	818	P	20:26:00	20:26:47	...

op_nodes

op_id	op_name	lat	lon	...
801	MARLOIE	50.203216	5.314306	...
788	MARCHE-EN-FAMENNE	50.222535	5.346178	...
1413	RACC.MARCHE-DEFENSE NATIONALE	50.244133	5.390791	...
818	MELREUX-HOTTON	50.283600	5.440353	...

line_sections

line_section_id	line_id	matched_op_node_ids	...
1297	43	[818, 1413, 788, 801]	...

node_links

link_id	u_node_id	v_node_id	line_section_id	distance_m
354	788	801	1297	3144.4
353	1413	788	1297	4053.0
352	818	1413	1297	5684.8

weather

op_id	time	temperature_2m	rain	...
801	20:00	6.50	0.60	...
788	20:00	6.50	0.60	...
818	20:00	6.75	0.20	...

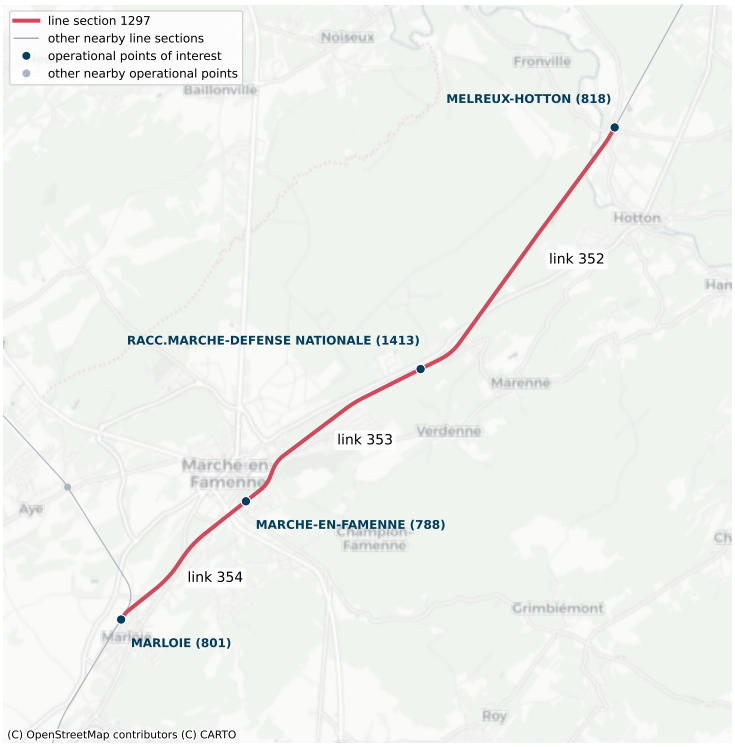

Figure 4: Illustrative example of silver tables for a single journey, together with a map view of the corresponding sub-network.

Figure 5 provides a complementary temporal view of a single silver journey, illustrating how scheduled and observed event times are represented along an ordered sequence of operational points.

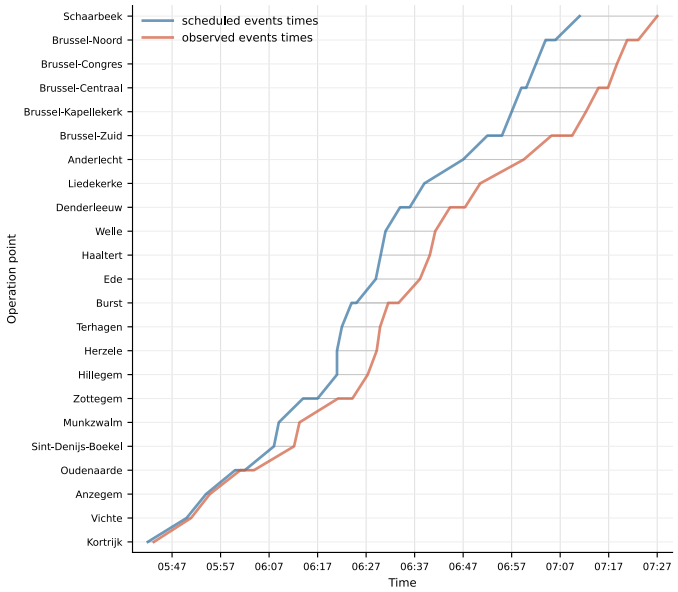

Figure 5: Visualization of the event sequence for one train on a given service date. The x-axis shows time and the y-axis lists operational points in journey order. Scheduled and observed event trajectories are plotted together. Horizontal trajectory segments at the same operational point correspond to dwell time between arrival and departure events.

B.4.2 Silver Processing and Descriptive Statistics

Processing overview. The silver release is constructed from the raw and bronze stages through a sequence of processing steps that transform heterogeneous source data into a coherent relational dataset. These steps combine cleaning, standardization, completion of missing or inconsistent information, and enrichment across train events, journeys, infrastructure, and weather data. In broad terms, the pipeline first consolidates operational points and infrastructure references, then reconstructs railway topology and graph connectivity, then sanitizes event timelines and aggregates them into journey-level records, and finally aligns hourly weather observations with operational points. The following paragraphs describe these components in turn.

Operational-node construction. The purpose of the `op_nodes` construction step is to obtain a node catalogue that covers all operational points referenced in the bronze stage, except a small number of clearly problematic cases that do not appear in event data, and assigns them coordinates consistent with railway line sections so that a graph matching the real network can be constructed. The `op_nodes` table is built by starting from the 1,337 operational points available in the bronze stage, among which 17 initially lacked coordinates, and then consolidating additional references required by infrastructure and event data. In particular, line-section information was used in two ways: to complete missing coordinates for 17 bronze operational points, and to add 18 additional operational points that appeared as line-section endpoints but were absent from the bronze operational-point catalogue. In addition, 4 event-only placeholder nodes were added from event files; their coordinates were then supplied manually for `op_id` 125, 864, 2291, and 2300 by cross-referencing external web sources with the surrounding event context and line-section geometries. At the same time, 4 problematic identifiers (`1375`, `1679`, `1296`, and `1297`) were removed because they were not attached to an existing line section or the main network and did not appear in event records. No duplicate `op_id` entries were present in the bronze input. The final silver `op_nodes` table contains 1,355 rows, covers all operational-point references required by downstream infrastructure and event processing, and has complete coordinate information. Figure 6 summarizes descriptive statistics of the final silver `op_nodes` table through two spatial diagnostics, namely average event activity per active day and average delay per operational point.

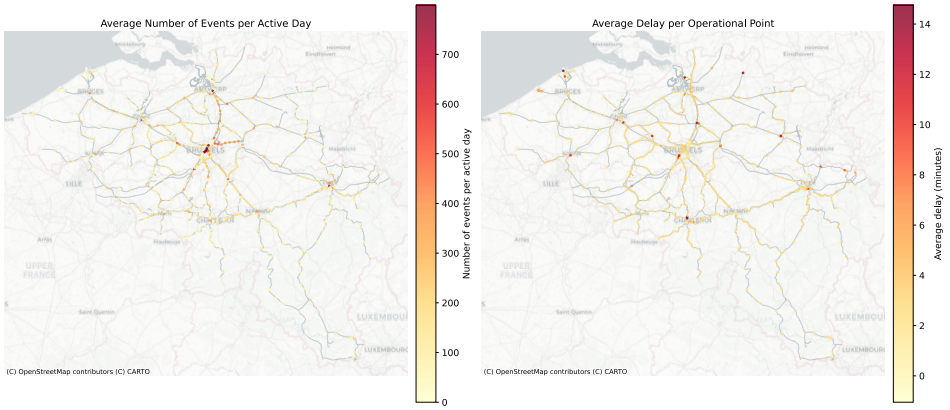

Figure 6: Descriptive statistics for the silver `op_nodes` table. The figure maps the average number of events per active day and the average delay per operational point over the full release period.

Line-section construction. The purpose of the `line_sections` construction step is to transform the bronze line-section data catalogue into an operational-point-aligned representation of railway geometry that can support exact graph construction. In the bronze stage, each line section is described by its begin and end operational-point identifiers together with a `geoshape` polyline, but intermediate geometry vertices are not yet linked to operational nodes. The silver construction step enriches these geometries by aligning them to the finalized `op_nodes` catalogue and recording the resulting ordered sequence of matched operational points in `matched_op_node_ids`. Concretely, line-section endpoints are anchored to `op_begin_id` and `op_end_id`, exact coordinate matches to geometry vertices are reused when available, and additional nearby operational points are projected onto the line geometry when they fall within a 1 m matching tolerance. For example, in the map view of Figure 4, the illustrated line section contains four matched operational points; in the bronze representation, only the two extreme endpoints are explicitly attached to the geometry, whereas this step recovers the two intermediate operational points and inserts them into the ordered sequence. One line section (`1165`) was removed because it is disconnected from the main network and has no linked event records. The final silver `line_sections` table contains 1,212 rows; all line sections have at least two matched operational points, 340 sections contain at least one interior matched operational point beyond their endpoints, for a total of 585 such interior assignments, every operational point appears in at least one line section, and all operational-point references in `line_sections` are valid. Figure 3 provides a visualization of the resulting network, while Figure 7 summarizes the distributions of cumulative section distance and matched operational-point counts.

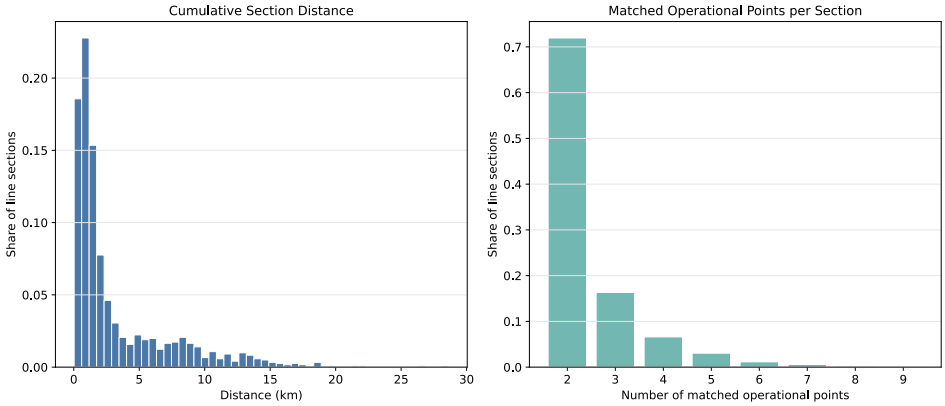

Figure 7: Descriptive statistics for the silver `line_sections` table. The figure shows the distribution of cumulative line-section distance and the distribution of the number of matched operational points per section.

Node-link construction. The purpose of the `node_links` construction step is to derive the exact railway graph induced by the finalized `line_sections`. For each line section, consecutive matched operational points in `matched_op_node_ids` are expanded into graph edges, and the corresponding link distance is computed by summing haversine distances along the geometry segment between the two matched nodes. This produces one undirected graph edge per consecutive pair of matched operational points along each line section. This construction is important because it recovers the physical railway topology directly from infrastructure geometry, rather than approximating it from successive train events: some operational points have no event records at all, and even when an event-based adjacency can be inferred, it does not provide the actual rail geometry and the true along-track distance between nodes. Figure 8 illustrates this distinction. The final silver `node_links` table contains 1,797 rows; no link connects an operational point to itself, and all link distances are strictly positive. Because multiple line sections may connect the same pair of operational points, 121 undirected endpoint pairs appear more than once, with a maximum multiplicity of 6; Figure 10 shows a local zoom on the most extreme such case. All 1,355 operational points appear in at least one node link, and the resulting railway graph forms a single connected component. Figure 9 summarizes descriptive statistics of the resulting table through the link-distance distribution and a map of average traversals per active day inferred from journey paths.

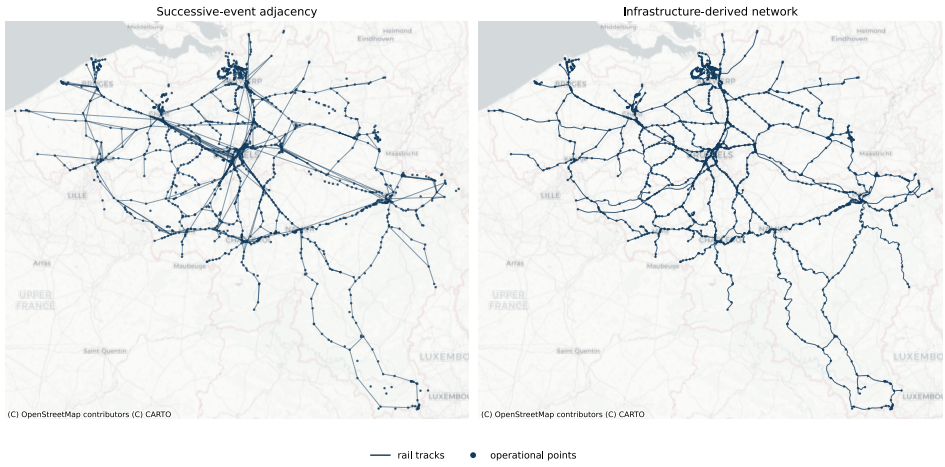

Figure 8: Comparison between an event-derived railway graph (left) and the infrastructure-derived graph used in RIDE (right).

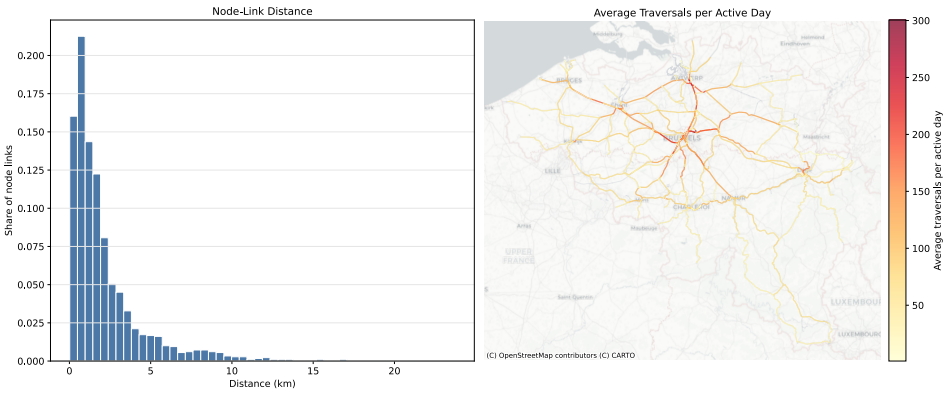

Figure 9: Descriptive statistics for the silver `node_links` table. The figure shows the distribution of link distance and a map of average traversals per active day inferred from `deduced_paths`. Because distinct node links can overlap visually on the same railway geometry, some map segments may appear superposed.

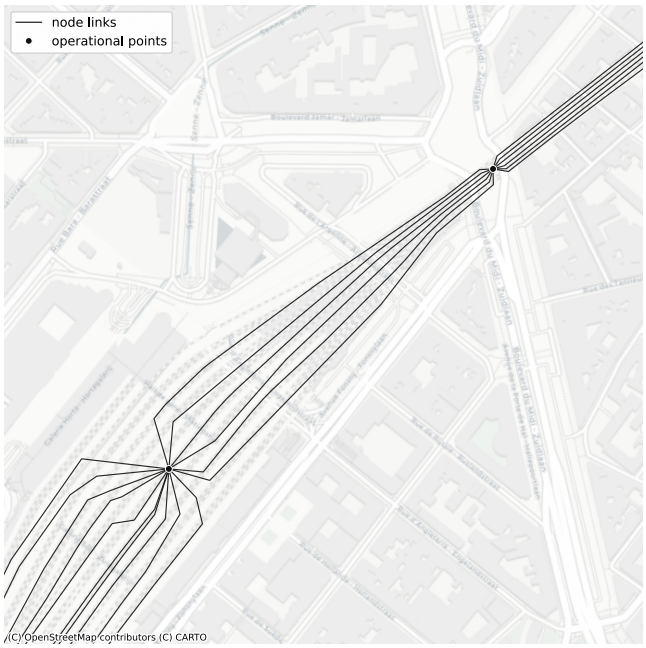

Figure 10: Local zoom on the operational-point pair with the highest node-link multiplicity. The figure illustrates how several distinct silver `node_links` can share the same two endpoint nodes while following overlapping local railway geometries.

Event cleaning. The purpose of the event-cleaning step is to transform the bronze train movement records into journey-level event sequences that remain faithful to the original event order while satisfying the temporal consistency expected from a clean journey representation. A central assumption of this step is that the original row order in the raw files already reflects the correct physical progression of the train along its journey; we therefore preserve this order throughout cleaning and apply all repairs in place rather than reordering events. Inspection of the raw data indicated that the original row order generally preserves the physical progression of trains along their journeys, including in cases where timestamps exhibit non-monotonicity. In the bronze data, each row corresponds to one train-at-operational-point record and may contain both arrival and departure information. Event-type separation converts such records into silver event rows: when scheduled arrival and departure times differ, the row is split into one arrival event and one departure event, whereas when they are equal, it is represented as a single passage event. Figure 11 illustrates these two cases. The resulting silver table therefore contains one row per arrival, departure, or passage event, with one scheduled timestamp, one observed timestamp, and event timelines that are non-decreasing in both scheduled and observed time within each journey. The cleaning pipeline first reconstructs timestamps from the raw date and time fields, then resolves local arrival/departure conflicts within the same operational point. Concretely, a conflict occurs when a scheduled departure precedes a scheduled arrival within the same bronze row. Using a tolerance of 120 s, such cases are repaired in place when the discrepancy remains small by setting the arrival timestamp equal to the departure timestamp, which subsequently yields a single passage event after event-type separation; otherwise the whole journey is discarded. Planned and observed event timelines are then enforced separately with the same 120 s tolerance: small backward steps are clamped in place, whereas journeys whose cumulative regressions exceed the threshold are removed. Figure 12 illustrates this repair-versus-drop logic on a simplified event sequence; the same illustration is also valid for local arrival/departure conflicts within a bronze stop record. Delays are then computed explicitly as the difference in seconds between observed and scheduled timestamps, and journeys are filtered if any event delay exceeds 4 h late or 1 h early, so as to exclude rare extreme outliers that are unlikely to be informative for the benchmark task. A sensitivity analysis on the complete January 2025 bronze data showed that moderate variations of the repair tolerances and delay bounds have only a limited effect on the resulting silver data. The largest tested change, obtained by reducing the repair tolerance from 120 s to 60 s, altered the retained event count by 0.127%, while variations of the delay bounds had a negligible effect. Finally, duplicate

events with the same `train_id`, `service_date`, `event_type`, and `op_id` are removed. Table 13 summarizes the main journey-level removals and row-level adjustments introduced by this event-cleaning step. Figure 13 summarizes descriptive statistics of the final silver `events` table, including delay distributions, temporal variation in average delay, and event-type frequencies.

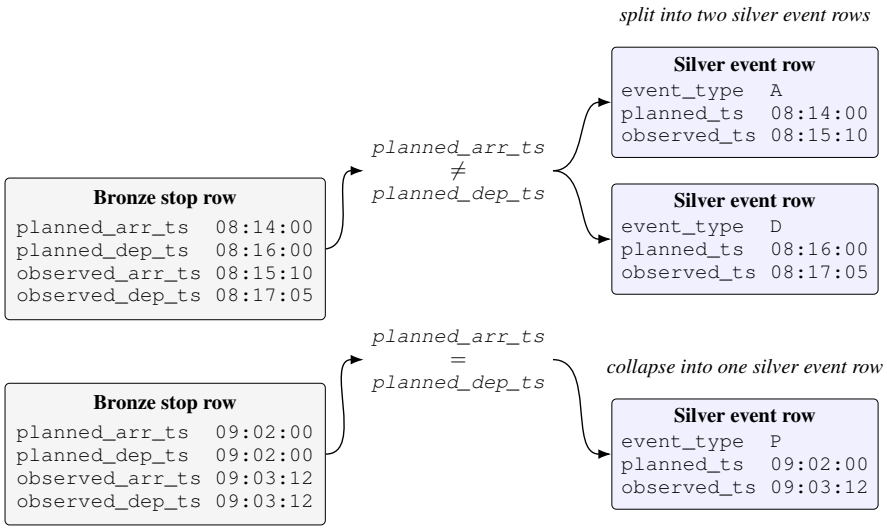

Figure 11: Illustration of event-type separation from bronze stop records into silver event rows.

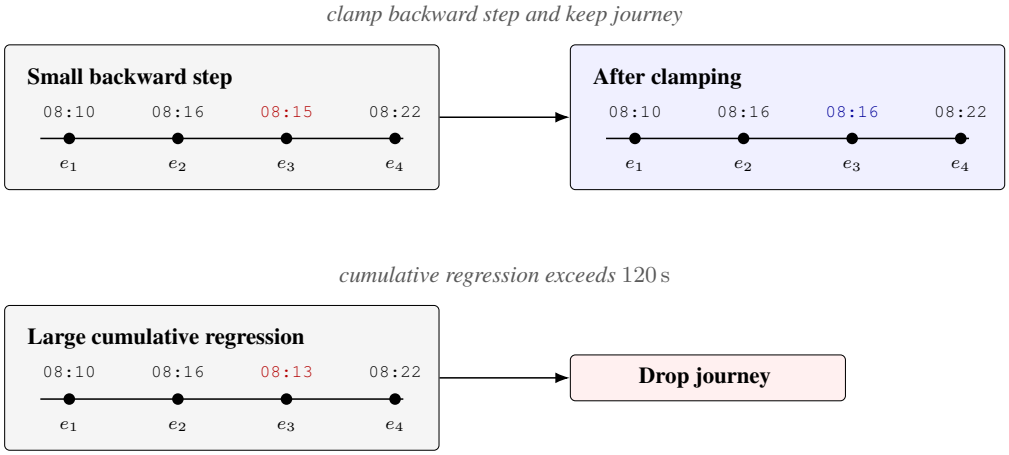

Figure 12: Illustration of local consistency enforcement during event cleaning.

Table 13: Summary statistics for the event-cleaning step.

Operation	Count	Share
Journey-level removals (relative to 3,625,094 bronze journeys)		
Dropped due to local arrival/departure conflicts	207	0.006%
Dropped after planned monotonicity checks	1,821	0.050%
Dropped after observed monotonicity checks	12,063	0.333%
Dropped by delay bounds	130	0.004%
Total removed journeys	14,221	0.392%
Event-level adjustments/removals (relative to 94,492,813 final silver events)		
Arrival/departure pairs collapsed into passage events	6,826	0.007%
Planned timestamps modified for monotonicity	6,767	0.007%
Observed timestamps modified for monotonicity	32,115	0.034%
Duplicate event rows removed	442	0.0005%
Total event-level adjustments/removals	46,150	0.049%

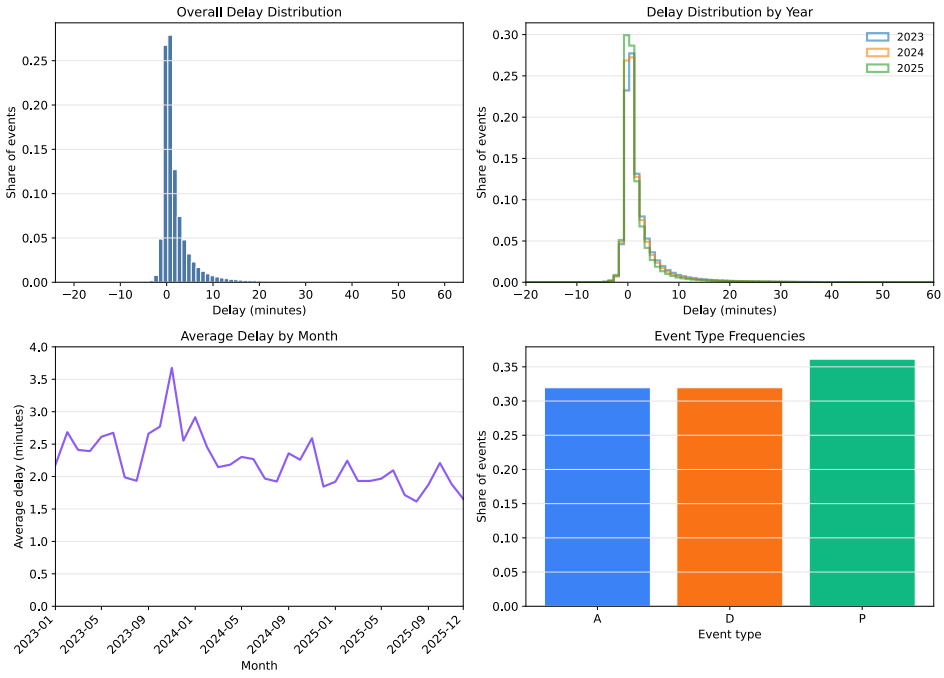

Figure 13: Descriptive statistics for the silver `events` table. The figure shows the overall delay distribution, the delay distribution by year, the average delay by month, and event-type frequencies.

Journey construction. The purpose of the `journeys` construction step is to provide one high-level descriptor per train and service date, aggregating information that would otherwise be repeated across the event table. Starting from the cleaned silver `events` table, the pipeline constructs one row per `(train_id, service_date)` pair and restores journey-level metadata from the corresponding bronze records, including `train_relation`, `operator`, and `relation_direction`. It then aggregates journey-level summary fields from the event sequence, namely the start and end operational points, the start and end planned and observed timestamps, the total number of events, and the maximum and minimum delay observed along the journey. Because many downstream methods may benefit from, or directly require, the inferred edge sequence used by a train, the `journeys` table also stores `deduced_paths`, which records for each consecutive pair of events the sequence of `node_links` traversed between their operational points. Figure 14 shows an illustrative example

of one such journey, where the inferred subpaths between consecutive events may pass through additional operational points without generating new event records. Inferring these paths is not trivial. Consecutive events in the cleaned event sequence provide only an ordered list of operational points, and successive operational points are not necessarily adjacent in the railway graph. Recovering the traversed segment therefore requires resolving a path through the network between each consecutive pair of event locations. This is further complicated by the fact that multiple feasible routes may exist between the same two operational points, especially when parallel or branching line segments are available locally. To resolve this ambiguity, path inference uses two additional sources of information. First, each split silver event retains its arrival and departure line identifiers, so that for a consecutive pair of events we know the line context at both ends of the segment. Second, each `node_link` is associated with a `line_section`, and therefore with a railway `line_id`, as well as a physical along-track distance. We therefore construct the full silver railway graph from `node_links` and resolve each consecutive event pair with a biased Dijkstra search Dijkstra (2022) whose edge costs are the physical link distances multiplied by line-preference coefficients. For a consecutive event pair, let ℓ_{src} denote the departure line identifier of the first event and ℓ_{dst} the arrival line identifier of the second event. The biased Dijkstra search then multiplies each edge length by a line-preference coefficient. Edges incident to the source node prefer ℓ_{src} first and ℓ_{dst} second; edges incident to the destination node prefer ℓ_{dst} first and ℓ_{src} second; and intermediate edges receive a preference when their line identifier matches either ℓ_{src} or ℓ_{dst}. Concretely, the multiplicative coefficients are 0.55 for the strongest source/destination preference, 0.75 for the secondary source/destination preference, 0.65 for intermediate matches, and 1.0 otherwise. The coefficient values were selected empirically to favor line-consistent routes without overwhelming the underlying physical-distance criterion. A sensitivity analysis on all January 2025 silver events, covering 1,856,170 nontrivial consecutive-event segments from 103,852 journeys, showed limited sensitivity to plausible, order-preserving coefficient changes: six of nine alternative configurations produced identical inferred routes, and the most sensitive setting changed 0.214% of segments. Figure 15 illustrates how this bias can favor a slightly longer physical route when it is more consistent with the line identifiers observed at the two ends of the segment. Figure 16 summarizes descriptive statistics of the final silver `journeys` table, including operator and train-relation frequencies, journey size, deduced-path distance, and final-delay behavior across train-relation categories.

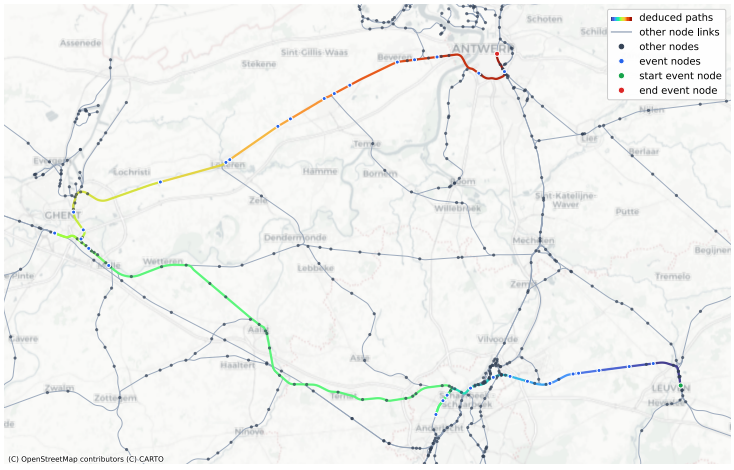

Figure 14: Illustrative example of a journey-level inferred path. Each color corresponds to one inferred subpath between two consecutive event locations. The color therefore remains unchanged when the train passes through intermediate operational points without generating an event, and changes only when the next event location is reached. Blue nodes denote event locations from the silver `events` table, while darker nodes denote other operational points in the surrounding network. Gray lines show the remaining local railway network; the green and red nodes mark the first and last events of the journey.

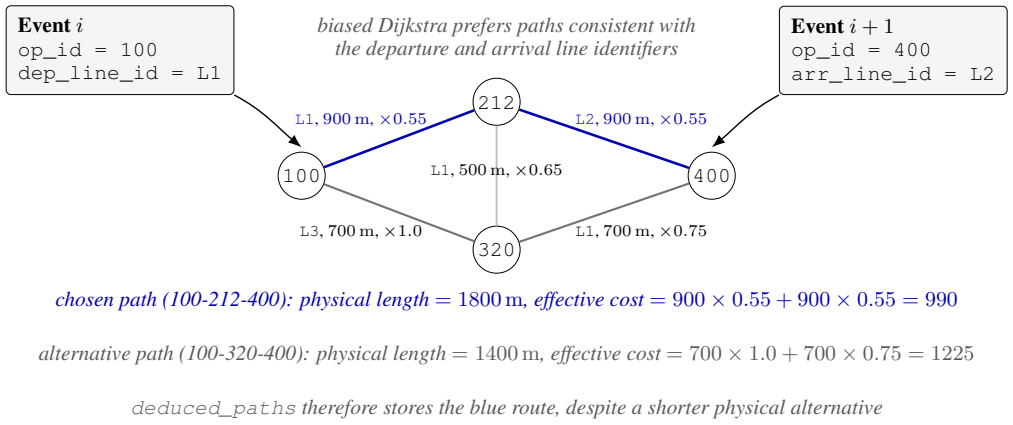

Figure 15: Illustrative example of path inference between two consecutive events. Edge labels indicate the line identifier, physical distance, and multiplicative line-preference coefficient used in path inference.

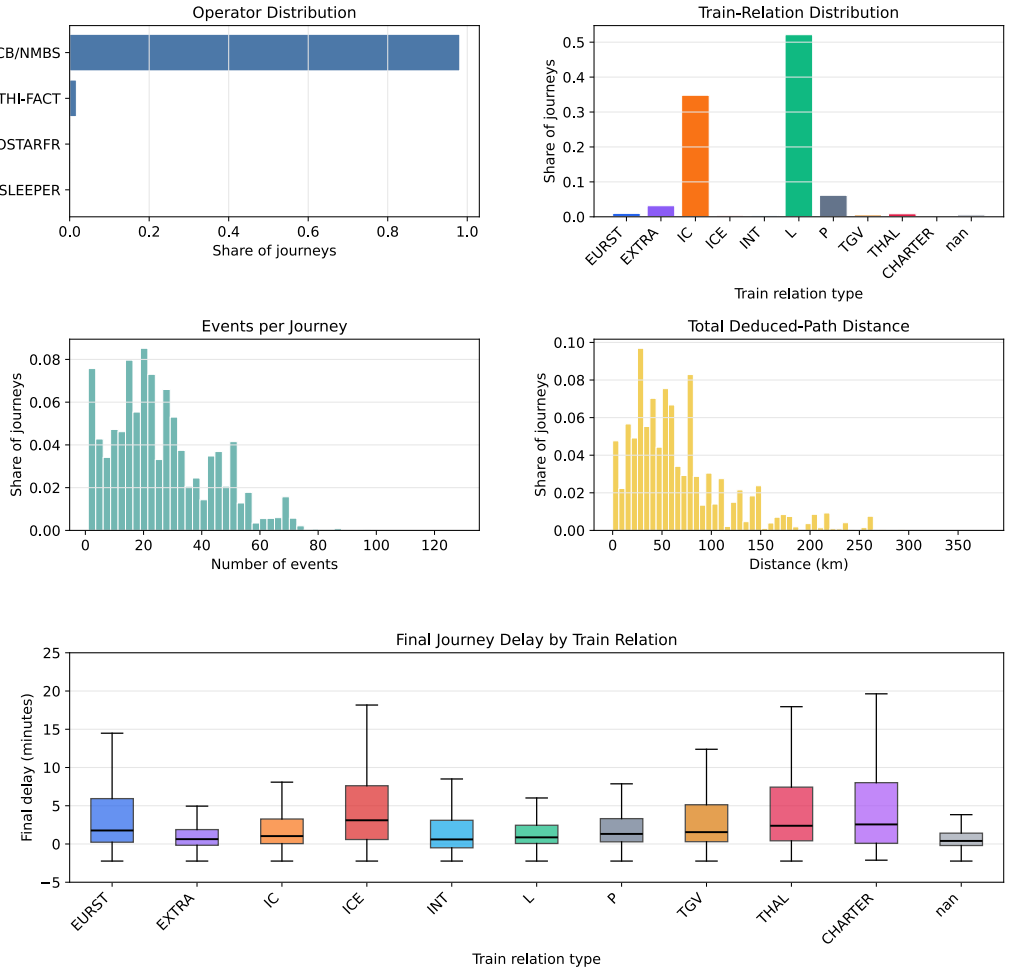

Figure 16: Descriptive statistics for the silver `journeys` table. The figure shows operator frequencies, train-relation frequencies, the distribution of event counts, the distribution of total deduced-path distance, and final journey delay by train-relation category.

Weather construction. The purpose of the `weather` construction step is to provide, for every operational point and every hour from 2022-12-31 to 2026-01-01, covering the 2023–2025 period with one-day padding on both sides, a synchronized set of local weather features that can be aligned with train events and prediction snapshots. Unlike the other silver tables, this step is not executed directly within the main silver pipeline: the raw weather data must first be downloaded from the Open-Meteo archive API after the silver `op_nodes` table has been finalized, since weather extraction requires the final operational-point coordinates. The raw extracts are then concatenated into a single table containing the six retained variables, namely `temperature_2m`, `rain`, `snowfall`, `relative_humidity_2m`, `wind_speed_10m`, and `weather_code`. To obtain a temporally regular representation, the concatenated records are sorted by `op_id` and time, converted to hourly timestamps in Europe/Brussels time, and reindexed independently for each operational point onto a complete hourly grid, with forward filling applied if gaps are present. In the final dataset, this yields 35,706,960 rows, corresponding to 1,355 operational points with a complete hourly series of 26,352 timestamps each. In practice, the downloaded batches already form a complete hourly grid, so reindexing and forward filling act here as validation-preserving safeguards rather than substantive repairs: no rows are added, no duplicate `(op_id, time)` pairs are present, all 1,355 operational points are covered, all six weather variables are fully observed, and every operational point has a strictly hourly time series. Figure 17 shows an illustrative snapshot from the resulting table, while Figure 18 summarizes the distributions of the five scalar weather variables and grouped `weather_code` frequencies.

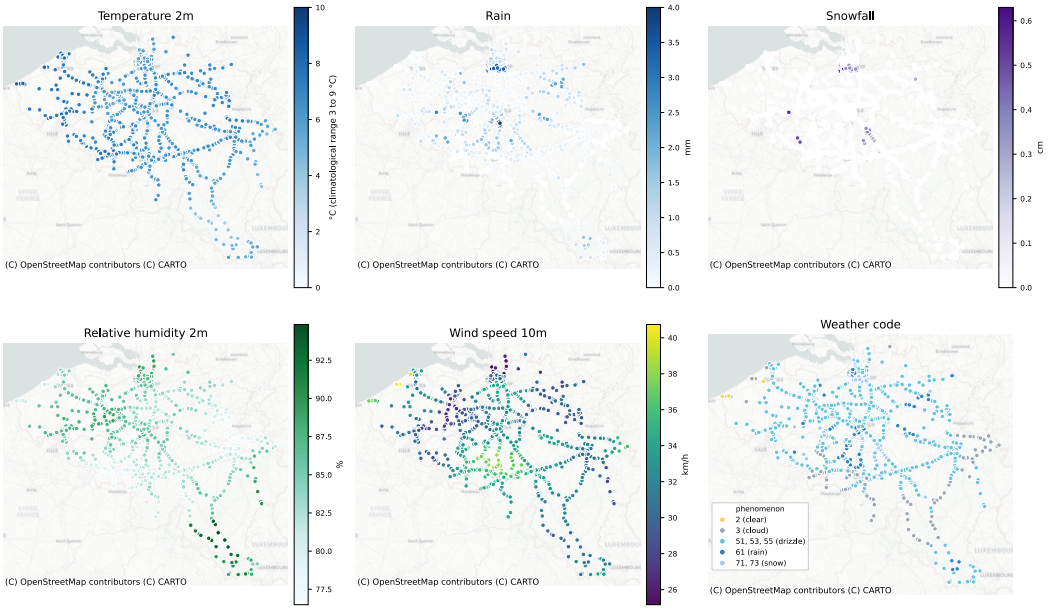

Figure 17: Illustrative weather snapshot from the silver `weather` table, showing the six aligned weather variables for all operational points at a single timestamp.

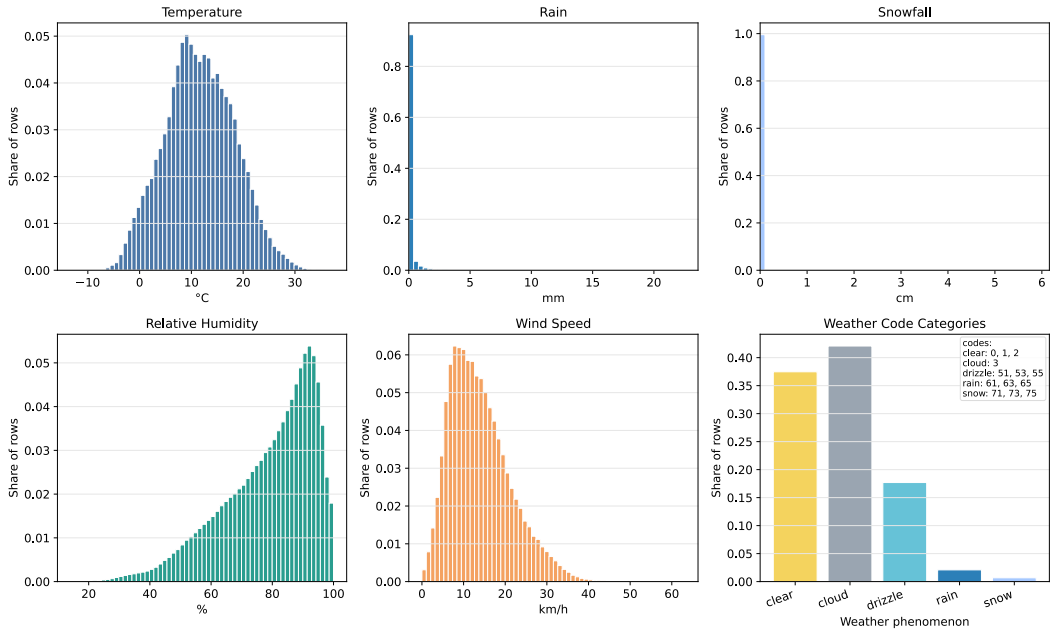

Figure 18: Descriptive statistics for the silver `weather` table. The figure shows the distributions of temperature, rain, snowfall, relative humidity, and wind speed, together with grouped frequencies of the observed `weather_code` categories.

B.5 Gold Datasets Release

The gold release is the model-ready tier of RIDE. It contains two types of artifacts: a shared benchmark core, which defines the common snapshots, splits, targets, and evaluation metadata, and four representation-specific datasets tailored to different prediction approaches. In the current release, these downstream datasets correspond to the tabular, sequential, GNN, and graph-event benchmark variants. In addition, the gold release provides lite and standard versions, which instantiate the same benchmark design at different scales. The following paragraphs first describe the shared core artifact and the lite/standard variants, then the representation-specific datasets built on top of them.

Core dataset. The core gold dataset is the authoritative benchmark artifact for RIDE. It defines the prediction problem through a fixed set of snapshots, train/test splits, active-train instances, and future delay targets, so that downstream users can build new gold datasets, feature constructions, or model families while evaluating on exactly the same prediction instances and target values. This design makes comparisons across papers possible without requiring prior work to be retrained or adapted to a different snapshot definition, split, or target construction. As explained in the main paper, the benchmark is organized around the notion of a snapshot, that is, a timestamped view of the railway network from which active trains are identified, model inputs are constructed, and future delay targets are defined; see Figure 2 for an illustration.

A core dataset is defined by the following components:

- the training and test period bounds $T^{\text{train}}_{\text{start}}$, $T^{\text{train}}_{\text{end}}$, $T^{\text{test}}_{\text{start}}$, and $T^{\text{test}}_{\text{end}}$. These correspond to `start_train_day`, `end_train_day`, `start_test_day`, and `end_test_day`;
- the numbers of sampled training and test snapshots, N_{train} and N_{test}, given by `n_train` and `n_test`;
- the horizon parameter n (`n_future`), giving the number of future events used to define the prediction targets for each active train;

- the parameters Δ_{beg} (`idle_time_beg`) and Δ_{end} (`idle_time_end`), which specify how much time before its first event and after its last event a train is still considered present in a snapshot;
- the additional tail-safety buffer Δ_{tail}, stored as `tail_safety_buffer_min` and used to separate the training and test periods;
- the training snapshot set $\mathcal{S}_{\text{train}}$, consisting of sampled snapshot timestamps used for training;
- the test snapshot set $\mathcal{S}_{\text{test}}$, consisting of sampled snapshot timestamps used for testing.

For a journey with planned start time $t_{\text{start}}^{\text{planned}}$, observed start time $t_{\text{start}}^{\text{obs}}$, and observed end time $t_{\text{end}}^{\text{obs}}$, the core benchmark categorically defines its activity window as

$$t^{\text{appear}} = \min\big(t_{\text{start}}^{\text{planned}} - \Delta_{\text{beg}},\ t_{\text{start}}^{\text{obs}}\big), \qquad t^{\text{disappear}} = t_{\text{end}}^{\text{obs}} + \Delta_{\text{end}}.$$

Given any snapshot time t^{snap}, this definition uniquely determines the active-train set through the following half-open interval condition:

$$t^{\text{appear}} \le t^{\text{snap}} < t^{\text{disappear}}.$$

The release code exposes reusable implementations of both steps. Activity windows are computed by `compute_journey_activity_windows`, and active trains at a snapshot are selected by `get_active_mask_for_snapshot`.

The snapshot sets are sampled from the seconds at which at least one journey is active. Training snapshots are sampled uniformly without replacement from the active seconds in the training period, yielding $\mathcal{S}_{\text{train}}$ with $|\mathcal{S}_{\text{train}}| = N_{\text{train}}$. For the test split, the nominal start time is shifted forward when needed to avoid overlap with trains from the last training service day. Let $t_{\text{end}}^{\text{last}}$ denote the latest observed end time among journeys from the final training day. The effective test start time is

$$\widetilde{T}_{\text{start}}^{\text{test}} = \max\big(T_{\text{start}}^{\text{test}}, t_{\text{end}}^{\text{last}} + \Delta_{\text{end}} + \Delta_{\text{tail}}\big)\,.$$

Test snapshots are then sampled uniformly without replacement from the active seconds in the resulting test period, yielding $\mathcal{S}_{\text{test}}$ with $|\mathcal{S}_{\text{test}}| = N_{\text{test}}$. The benchmark-defining parameters and the realized snapshot sets are stored in the core specification file, `dataset_core_spec.yaml`. Release metadata and output paths are stored[2] separately in `metadata.yaml`.

The `test_eval_table.parquet` file is the central evaluation artifact of the core dataset. Its purpose is to make the test prediction instances and target values fully explicit: downstream users can build any representation or model they want, but evaluation is performed by matching predictions against this fixed table. Consequently, users only need to provide predicted delays in seconds for the valid target slots, and the evaluation code automatically applies the target mask and computes all aggregate and breakdown metrics. The table contains one row per `(snapshot, active train)` pair. Each row records:

- the prediction instance identifiers: the snapshot time `ts`, `train_id`, and `service_date`;
- the current delay state: `last_known_delay`, defined as $d_{s,i}^{\text{last}}$ in Section 4.1;
- the future observation times: `future_obs_ts_1`, …, `future_obs_ts_n`;
- the future operational-point identifiers: `future_op_id_1`, …, `future_op_id_n`;
- the future event types: `future_event_type_1`, …, `future_event_type_n`;
- the future delay targets: `future_delay_1`, …, `future_delay_n`.

The future-delay columns define the common target values against which all downstream gold datasets are evaluated. If fewer than n valid future events remain for an active train, the invalid target slots are encoded with placeholder values: `future_op_id` is set to -1, `future_event_type` is set to -1, `future_delay` is set to `NaN`, and `future_obs_ts` is set to `NaT`.

Lite and standard variants. The gold release is provided in two scales. The lite variant is designed for fast iteration and lower-resource experimentation, and can also serve as the primary benchmark tier for users with limited computational resources. The standard variant is the main benchmark scale used for the main model comparison in this paper. Each variant has its own core artifact, including a separate `dataset_core_spec.yaml`, realized training and test snapshot sets, `test_eval_table.parquet`, and `metadata.yaml`. The core parameters and snapshot sets are stored in `dataset_core_spec.yaml`, while `metadata.yaml` stores release metadata and output paths. The representation-specific tabular, sequential, GNN, and graph-event datasets are then built separately from the corresponding lite or standard core, so that all datasets within a given tier share exactly the same prediction instances and targets.

The same construction code can instantiate additional cores for custom studies with different parameters, but the released lite and standard cores are the fixed reference tiers for comparable reporting across papers.

Both variants use the same training period, test period, horizon, activity-window definition, and tail-safety buffer. They differ only in the number of sampled training and test snapshots. Table 14 reports the core parameters, while Table 15 reports realized scale statistics for each variant.

Table 14: Core parameters for the lite and standard gold variants.

Parameter	Lite	Standard
$T_{\text{start}}^{\text{train}}$	2023-01-01	2023-01-01
$T_{\text{end}}^{\text{train}}$	2024-12-31	2024-12-31
$T_{\text{start}}^{\text{test}}$	2025-01-01	2025-01-01
$T_{\text{end}}^{\text{test}}$	2025-12-31	2025-12-31
N_{train}	15,000	50,000
N_{test}	3,000	10,000
n	15	15
Δ_{beg}	5 min	5 min
Δ_{end}	5 min	5 min
Δ_{tail}	1 min	1 min

Table 15: Realized scale statistics for the lite and standard gold variants.

Statistic	Lite	Standard
training active-train rows	3,153,365	10,554,576
test active-train rows	623,314	2,093,550
valid training targets	32,638,929	109,200,414
valid test targets	6,528,133	21,941,499
mean active trains per training snapshot	210.22	211.09
mean active trains per test snapshot	207.77	209.36
mean valid targets per training row	10.35	10.35
mean valid targets per test row	10.47	10.48

Tabular dataset. The tabular gold dataset is the fixed-vector materialization of the core benchmark. It contains one row per active train at one snapshot and is used by the MLP, XGBoost, and Transformer models in our benchmark. For each split, the dataset stores input features in `x.npy`, target values in `y.npy`, target-validity indicators in `y_mask.npy`, and row metadata in `md.npy`. The row metadata columns are `snapshot_ts`, `train_id`, and `service_date`. The exported column contract is stored in `scheme.yaml`, while the fitted preprocessing statistics are stored in `normalization.yaml`. In the released lite and standard variants, `x.npy` has 452 input features, `y.npy` has 15 future-delay targets, and `y_mask.npy` has 15 corresponding validity indicators.

Table 16 summarizes the input feature groups in `x.npy`. Train metadata come from the silver `journeys` table and include an indicator for SNCB/NMBS-operated services and a one-hot encoding

of a coarse category derived from the journey `train_relation` field. Snapshot-time features are derived from the core snapshot timestamp and encode day of week together with cyclical hour-of-day and day-of-year information. Event-window features come from the silver `events` table. Link features use the inferred `deduced_paths` stored in `journeys` together with silver `node_links`. For each active train, the pipeline estimates its current or next oriented `node_link` along the inferred path, then records up to 10 oriented node links ahead of the train. For each such slot, the tabular vector stores the link distance, a traffic count, an average delay, and a placeholder flag. Weather features come from the silver `weather` table and are computed at the snapshot hour from the weather observations at the previous and next event operational points; the two values are averaged when both sides are available, and the available side is used when one side is padded. Operational-point embeddings are computed from the silver `node_links` graph.

The remaining input features describe event windows around the snapshot. The past window contains 15 previously observed event slots and records, for each slot, the planned-time offset from the snapshot, the observed delay, the event type, and an operational-point embedding. The future window also contains 15 slots, matching the core horizon, but only uses information that is known at prediction time: the planned-time offset, the scheduled event type, and the operational-point embedding. When an event window extends beyond the available journey events, event identifiers are padded with the missing-event value -1; embeddings for missing event identifiers are zero vectors, with padded past event types set to departure and padded future event types set to arrival. Missing link slots are marked by the link placeholder flag, and invalid future target slots are excluded through `y_mask.npy`. The future delays themselves are not included in `x.npy`; they are stored as residual targets in `y.npy`, as summarized in Table 17.

Operational-point embeddings are computed from the silver railway graph. The code builds an undirected adjacency from `node_links`, computes normalized-Laplacian eigenmap coordinates, keeps the first 8 nonzero components, and normalizes each node embedding to unit length. These components provide a compact topological encoding of where an event occurs in the railway network; Figure 19 visualizes the embedding components. Because the railway graph is reconstructed from source files downloaded after the 2025 test period, these embeddings are not a perfect historical snapshot of the infrastructure available at the end of the training period in 2024. In practice, this leakage is limited: only 11 operational points appearing in test events are absent from the training period, out of 1355 operational points overall, and they represent 0.00015% of test events. Its effect is also narrow, since it can only slightly improve the embedding coordinates assigned to these new stations.

All preprocessing statistics are fitted on training samples only and then reused for both training and test arrays, mimicking the information available in a real forward-in-time evaluation. The target array stores preprocessed residual future delays, with `future_delay_delta_i` first defined as the future delay at horizon i minus `last_known_delay`. For each horizon, these target residuals are transformed with signed sqrt scaling and then z-scored using training-split statistics recorded in `normalization.yaml`. At evaluation time, model outputs are transformed back to residual-delay seconds with the same stored statistics and then shifted by `last_known_delay` before being matched against `test_eval_table.parquet`. The corresponding `y_mask.npy` array marks which future event slots are valid and should be included in training losses and evaluation metrics.

Table 16: Input feature groups in the tabular gold dataset's `x.npy` array.

Feature group	Content	Encoding	Normalization
train operator	indicator equal to one for SNCB/NMBS-operated services	binary	none
train relation type	coarse category derived from the journey `train_relation` field	one-hot	none
day of week	snapshot day-of-week indicators	one-hot	none
hour of day	sine/cosine hour-of-day features at frequencies 1, 2, and 4	6 scalars	none
day of year	sine/cosine day-of-year features at frequencies 1, 2, and 4	6 scalars	none
link # distance	distance of oriented node link # ahead of the train	scalar	min–max
link # traffic count	number of active trains on oriented node link # ahead of the train	scalar	min–max
link # average delay	average delay of active trains on oriented node link # ahead of the train	scalar	signed sqrt, z-score
link # placeholder flag	indicator that oriented node link # ahead of the train is missing or padded	binary	none
temperature	average temperature at the previous and next event operational points at the snapshot hour	scalar	z-score
rain	average rain at the previous and next event operational points at the snapshot hour	scalar	signed sqrt, z-score
snowfall	average snowfall at the previous and next event operational points at the snapshot hour	scalar	signed sqrt, z-score
relative humidity	average relative humidity at the previous and next event operational points at the snapshot hour	scalar	z-score
wind speed	average wind speed at the previous and next event operational points at the snapshot hour	scalar	z-score
past event # timing	planned-time offset of past event #	scalar	signed sqrt, z-score
past event # delay	observed delay of past event #	scalar	signed sqrt, z-score
past event # type	event type of past event #	one-hot	none
past event # embedding	8-dimensional operational-point embedding of past event #	8 scalars	none
future event # timing	planned-time offset of future event #	scalar	signed sqrt, z-score
future event # type	event type of future event #	one-hot	none
future event # embedding	8-dimensional operational-point embedding of future event #	8 scalars	none

indexes repeated slots: $\# = 1, \ldots, 10$ for node links ahead of the train, and $\# = 1, \ldots, 15$ for past and future event slots. Weather features fall back to the available side when one of the two event operational points is padded.

Table 17: Target and mask arrays in the tabular gold dataset.

Array	Columns	Description
`y.npy`	`future_delay_delta_1`, ..., `future_delay_delta_15`	normalized residual future-delay targets, defined from future delays minus `last_known_delay`
`y_mask.npy`	`is_target_valid_1`, ..., `is_target_valid_15`	boolean indicators specifying which future event slots are valid and should be included in training losses and evaluation metrics

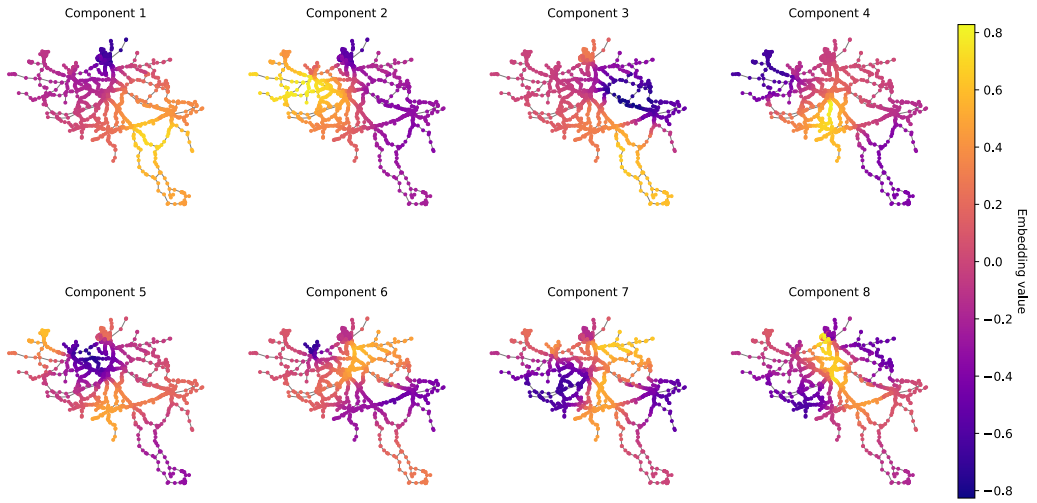

Figure 19: The 8 normalized-Laplacian embedding components used as operational-point features in gold datasets. Each panel colors the silver railway graph by one embedding component.

Sequential dataset. The sequential gold dataset uses the same sample set as the tabular dataset: each sample corresponds to one active train at one snapshot, with the same metadata keys, target column names, target-validity mask, and residual target definition. It uses the same underlying 452 input columns listed in `scheme.yaml`. It reorganizes the tabular input vector into static features, past-event sequences, and future-known-event sequences, and is used by the LSTM model in our benchmark. The resulting arrays are summarized in Table 18. The static array contains train metadata, snapshot-time features, link features, and weather features. The past-event sequence contains the 15 past event slots in chronological order, from older events toward the most recent event before the snapshot. Each past step contains 13 features: planned-time offset, observed delay, three event-type indicators, and eight operational-point embedding components. The future-known sequence contains the 15 future event slots in horizon order. Each future step contains 12 features: planned-time offset, three event-type indicators, and eight operational-point embedding components. Future delays are not included in the input sequence; they remain targets in `y.npy`.

As in the tabular dataset, preprocessing statistics are fitted on training samples only and stored in `normalization.yaml`. The sequential dataset differs in one normalization detail: for repeated event-window quantities, the statistics are shared across slots for `past_planned_delta_*`, `future_planned_delta_*`, `past_delay_sec_*`, and `future_delay_delta_*`, so that the same transformation is applied at each time step of the corresponding sequence.

Table 18: Arrays in the sequential gold dataset.

Array	Shape/row	Description
`x_static.npy`	77	static input features: train metadata, snapshot-time features, link features, and weather features
`x_past_seq.npy`	15×13	past-event input sequence; each step contains planned-time offset, observed delay, event type, and operational-point embedding features
`x_future_known_seq.npy`	15×12	future-known input sequence; each step contains planned-time offset, event type, and operational-point embedding features
`y.npy`	15	normalized residual future-delay targets
`y_mask.npy`	15	Boolean indicators specifying which future target slots are valid
`md.npy`	3	row metadata: snapshot timestamp, train identifier, and service date

GNN dataset. The GNN gold dataset represents each benchmark snapshot as a heterogeneous graph rather than as one fixed-size vector per active train. This graph view keeps the main entities of the prediction problem explicit. Active trains are represented as train nodes carrying train metadata and snapshot-time features (Table 19). Operational points are represented as station nodes carrying position, local weather, operational-point embeddings, and stopped-train traffic features (Table 20). Railway infrastructure is represented by station-to-station edges carrying physical link distance and direction-specific `uv`/`vu` angle, traffic-count, and average-delay features (Table 21). Finally, each train's observed and scheduled itinerary context is represented by train-to-station edges: past edges encode previous events through planned offset, observed delay, event type, and recency rank (Table 22), while future edges carry scheduled future-event inputs and the associated residual-delay target (Table 23). This representation attaches network-context features such as distances, train counts, and average delays directly to the station nodes and railway links where they are measured, rather than folding them into a train-level vector. It therefore preserves the distinction between train-specific itinerary context and shared network state within each snapshot. An illustrative example of this representation is given in Figure 20, and a full-network snapshot produced by the dataset construction is shown in Figure 21.

The released artifact stores PyTorch Geometric graph objects for the training and test splits as `train/graphs_part_*.pt` and `test/graphs_part_*.pt`. These graph objects include reverse relations for all edge types. Past and future train-to-station reverse relations reuse the same edge attributes, while station-to-station reverse relations swap the `uv_*` and `vu_*` attributes so that direction-specific features remain aligned with the direction of the relation. The node and edge feature columns used in these graph objects are recorded in `feature_spec.yaml`, and preprocessing statistics are stored in `normalization.yaml`. As in the tabular and sequential datasets, normalization statistics are fitted on the training split only. The target on each future train-to-station edge is the normalized residual future delay, defined from the future delay minus `last_known_delay`; the horizon rank is retained so that edge predictions can be mapped back to the corresponding core target slots. At evaluation time, predictions are transformed back to delay seconds and compared against the core evaluation table.

Table 19: Train-node input feature groups in the GNN gold dataset.

Feature group	Content	Encoding	Normalization
train operator	indicator equal to one for SNCB/NMBS-operated services	binary	none
train relation type	coarse category derived from the journey `train_relation` field	one-hot	none
snapshot time of day	sine/cosine snapshot-hour features at frequencies 1, 2, and 4	6 scalars	none
snapshot day of year	sine/cosine day-of-year features at frequencies 1, 2, and 4	6 scalars	none
snapshot day of week	snapshot weekday category	one-hot	none

Table 20: Station-node input feature groups in the GNN gold dataset.

Feature group	Content	Encoding	Normalization
latitude	station latitude	scalar	min–max
longitude	station longitude	scalar	min–max
temperature	hourly temperature at the station and snapshot hour	scalar	z-score
rain	hourly rain at the station and snapshot hour	scalar	signed sqrt, z-score
snowfall	hourly snowfall at the station and snapshot hour	scalar	signed sqrt, z-score
relative humidity	hourly relative humidity at the station and snapshot hour	scalar	z-score
wind speed	hourly wind speed at the station and snapshot hour	scalar	z-score
embedding	normalized-Laplacian operational-point embedding components	8 scalars	none
stopped traffic count	number of active trains currently stopped at the station	scalar	min–max
stopped traffic delay	average current delay of active trains currently stopped at the station	scalar	signed sqrt, z-score

Table 21: Station-to-station edge input feature groups in the GNN gold dataset.

Feature group	Content	Encoding	Normalization
link distance	physical distance of the railway node link	scalar	min–max
`uv` angle	sine and cosine of the node-link angle from u to v	2 scalars	none
`vu` angle	sine and cosine of the node-link angle from v to u	2 scalars	none
`uv` traffic count	number of active trains currently occupying the node link from u to v	scalar	min–max
`vu` traffic count	number of active trains currently occupying the node link from v to u	scalar	min–max
`uv` average delay	average current delay of active trains currently occupying the node link from u to v	scalar	signed sqrt, z-score
`vu` average delay	average current delay of active trains currently occupying the node link from v to u	scalar	signed sqrt, z-score

Table 22: Train-to-past-station edge input feature groups in the GNN gold dataset.

Feature group	Content	Encoding	Normalization
planned offset	planned time of the past event relative to the snapshot time	scalar	signed sqrt, z-score
observed delay	observed delay at the past event	scalar	signed sqrt, z-score
past rank	recency rank of the past event in the train itinerary window	scalar	min–max
event type	arrival, departure, or passage indicator	one-hot	none

Table 23: Train-to-future-station edge input and target groups in the GNN gold dataset.

Feature group	Content	Encoding	Normalization
planned offset	planned time of the future event relative to the snapshot time	scalar	signed sqrt, z-score
future rank	horizon rank of the future event in the prediction window	scalar	min–max
event type	arrival, departure, or passage indicator	one-hot	none
future delay residual	target delay at the future event minus `last_known_delay`	target	signed sqrt, z-score

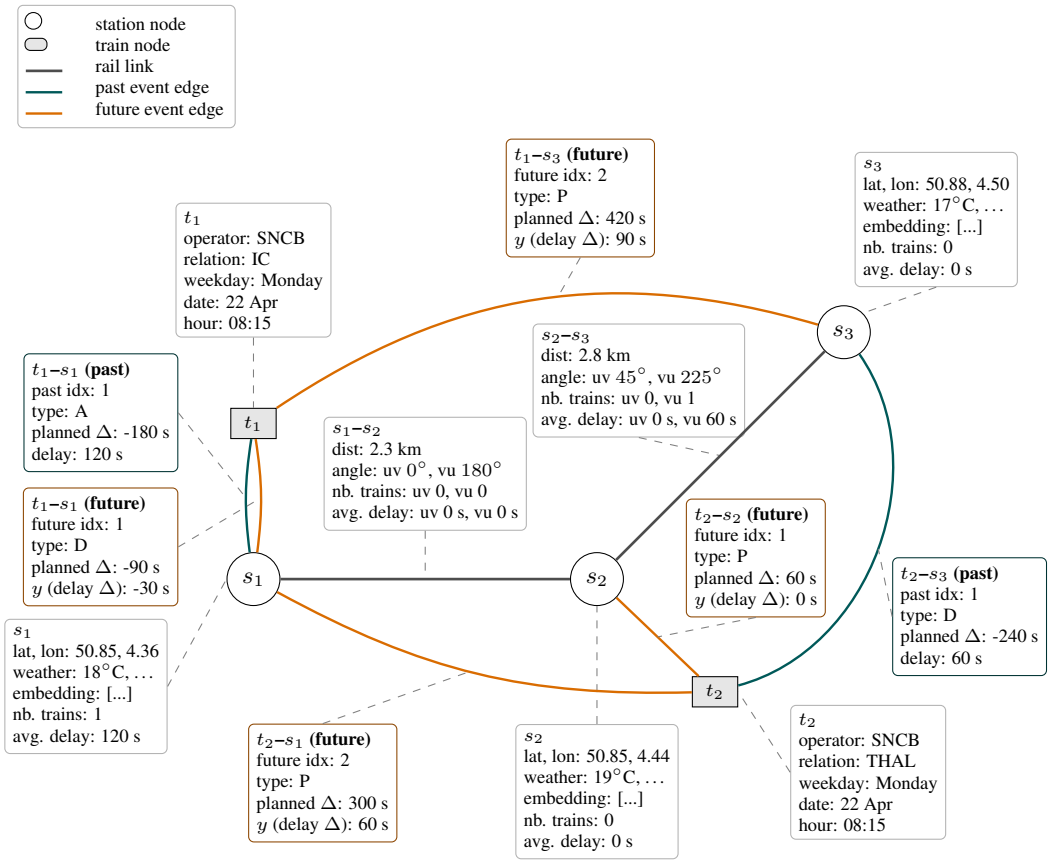

Figure 20: Toy railway layout used to illustrate the heterogeneous GNN snapshot structure and features. The graph contains three consecutive station nodes together with two active trains: one stopped at station s_1 and one moving from station s_3 toward station s_2.

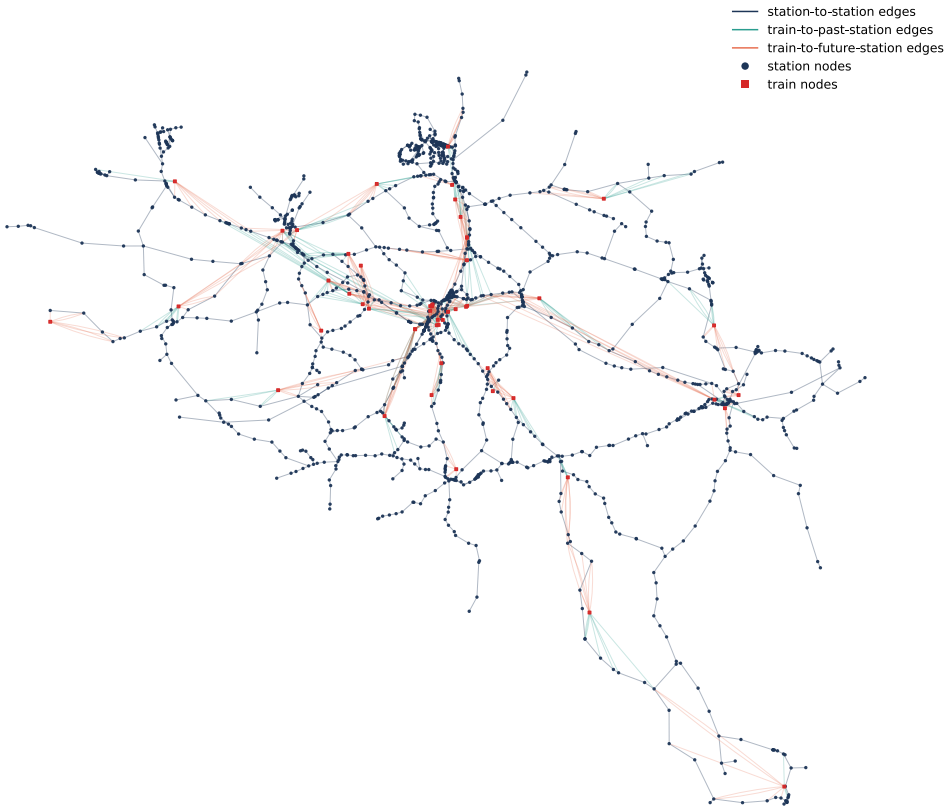

Figure 21: Visualization of a full-network GNN snapshot. Station nodes are placed at their geographic locations, rail links define the infrastructure graph, and train nodes are connected to the operational points associated with the 7 closest past and future events in their itinerary. We thin the window for readability; the released GNN dataset uses 15 past and 15 future event slots.

Graph-event dataset. The graph-event gold dataset provides the model-ready inputs for the deterministic graph-event benchmark described in Appendix C.2. Unlike the learning-based gold datasets, it does not construct normalized feature tensors. Instead, it exports the event, journey, infrastructure, and travel-time lookup artifacts needed to run the graph-event simulation on the core test prediction instances. The released files are summarized in Table 24. The `node_links.parquet` file is copied from the silver release and keeps the columns `line_section_id`, `u_node_id`, `v_node_id`, `distance_m`, and `link_id`. The `test/events.parquet` file contains all event rows for trains appearing in the graph-event test journeys, with columns `train_id`, `service_date`, `op_id`, `event_type`, `planned_ts`, `line_dep`, and `line_arr`. The `test/journeys.parquet` file contains one row per core test `(snapshot, active train)` pair and stores `ts`, `train_id`, `service_date`, `current_event_idx`, `last_known_delay`, `train_relation`, and `path`; here `current_event_idx` identifies the last event before the snapshot, `train_relation` is the coarse category derived from the journey `train_relation` field, and `path` contains the journey-level inferred `deduced_paths`. Finally, `travel_time_samples.pkl` stores travel-time statistics estimated from the full event sequences of trains active in the sampled training snapshots. These statistics are keyed by origin operational point, origin event type, departure line, destination operational point, destination event type, arrival line, and train-relation category, and store the number of samples, minimum, maximum, mean, median, and lower quantiles from 5% to 45%. They are used by the graph-event model to estimate expected travel times between consecutive events without using test-period observations.

Table 24: Artifacts in the graph-event gold dataset.

Artifact	Contents
`node_links.parquet`	silver railway node-link table, with link identifiers, endpoint operational points, line-section identifiers, and link distances
`test/events.parquet`	event rows for trains appearing in the graph-event test journeys, including operational point, event type, planned timestamp, and arrival/departure line context
`test/journeys.parquet`	one row per core test (`snapshot, active train`) pair, including current event index, last known delay, train relation, and inferred path information
`travel_time_samples.pkl`	training-snapshot travel-time statistics for event-pair contexts, used as lookup values by the graph-event simulation

C Benchmark Protocol, Implementations and Additional Experiments

This section provides additional details on the benchmark protocol, model implementations, training procedure, and supplementary experiments. We first specify the evaluation setup in more detail, including target masking, aggregate metrics, and the horizon- and delay-delta-based breakdowns used in the main paper. We then describe the benchmark model implementations and training procedure, report the hyperparameter search spaces and selected configurations, and provide additional results on the lite benchmark tier together with the feature-family ablation study.

C.1 Prediction Task and Metrics Details

For a given snapshot s at time t^s, the goal is to predict, for every active train, the delay at the next n scheduled events, where each event corresponds to a scheduled arrival, departure, or passage at an operational point (see Table 1). In this appendix subsection, we provide additional detail on the practical evaluation protocol. Conveniently, the core dataset's `test_eval_table` contains all metadata required to instantiate the full evaluation procedure, so users only need to provide predicted delays in seconds indexed by the core key columns.

Each snapshot contains target slots for the next 15 scheduled events of each active train, but in practice not every train has 15 valid future targets, since some journeys end earlier. We therefore construct an explicit mask indicating which target slots correspond to valid future events. During training, masked targets are excluded from the training objectives of learning-based models. During evaluation, masked targets are automatically excluded from all reported metrics and breakdowns.

For stochastic learning-based models, all reported metrics are first computed independently for each test-evaluation run and then summarized as mean and standard deviation across the fixed seeds $\{0, 1, \ldots, 9\}$. Deterministic models are evaluated once, so only a single metric value is reported.

Aggregate Metrics. We compute aggregate metrics over the explicit set of valid target slots. Let $\mathcal{E}$ denote the set of all valid prediction targets in the evaluation split after masking. Each element $(s, i, j) \in \mathcal{E}$ corresponds to one active train instance i in snapshot s and one future event slot j. The aggregate mean absolute error and root mean squared error are then defined as

$$\mathrm{MAE} = \frac{1}{|\mathcal{E}|} \sum_{(s,i,j)\in\mathcal{E}} \left|\hat{d}_{s,i,j} - d_{s,i,j}\right|, \qquad \mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{E}|} \sum_{(s,i,j)\in\mathcal{E}} \left(\hat{d}_{s,i,j} - d_{s,i,j}\right)^2}.$$

Prediction Horizon. For each valid target $(s, i, j) \in \mathcal{E}$, let

$$h_{s,i,j} = \frac{t^{\mathrm{obs}}_{s,i,j} - t^s}{60}$$

denote its prediction horizon in minutes. Horizon-based breakdowns are obtained by restricting evaluation to targets whose horizons fall within a given bin $[a, b)$. For such a bin, let

$$\mathcal{E}^{\mathrm{hor}}_{[a,b)} = \{\, (s, i, j) \in \mathcal{E} \mid a \le h_{s,i,j} < b \,\}.$$

The corresponding bin-specific metrics are then defined as

$$\mathrm{MAE}^{\mathrm{hor}}_{[a,b)} = \frac{1}{|\mathcal{E}^{\mathrm{hor}}_{[a,b)}|} \sum_{(s,i,j)\in\mathcal{E}^{\mathrm{hor}}_{[a,b)}} \left|\hat{d}_{s,i,j} - d_{s,i,j}\right|,$$

$$\mathrm{RMSE}^{\mathrm{hor}}_{[a,b)} = \sqrt{\frac{1}{|\mathcal{E}^{\mathrm{hor}}_{[a,b)}|} \sum_{(s,i,j)\in\mathcal{E}^{\mathrm{hor}}_{[a,b)}} \left(\hat{d}_{s,i,j} - d_{s,i,j}\right)^2}.$$

We use bin edges at $0, 5, 10, 15, 20, 25, 30, 35, 40, 45, \infty$.

The final open-ended 45+ bin is used because target events become progressively scarcer at larger horizons under the fixed 15-event prediction window, which corresponds on average to about 40 minutes into the future. This effect is visible in the overall horizon distribution (Figure 22), while the per-target breakdown further shows how these bins are distributed across the 15 future-event slots (Figure 23).

An important caveat is that the latest horizon bins are progressively biased toward trains that accumulate delay. This effect is especially pronounced in the final 45+ bin, because under the fixed future-event prediction window the later time bins become thinner by construction, as illustrated in Figure 24. As a result, differences across later horizon bins should not be interpreted as reflecting horizon length alone, but rather as a mixture of horizon and delay regime.

Delay Delta. For each valid target $(s, i, j) \in \mathcal{E}$, let

$$\Delta d_{s,i,j} = \frac{d_{s,i,j} - d^{\mathrm{last}}_{s,i}}{60}$$

denote its delay delta in minutes relative to the last known delay $d^{\mathrm{last}}_{s,i}$ defined in Section 4.1. Delay-delta-based breakdowns are obtained by restricting evaluation to targets whose delay deltas fall within a given bin $[a, b)$. For such a bin, let

$$\mathcal{E}^{\Delta}_{[a,b)} = \{\, (s,i,j) \in \mathcal{E} \mid a \le \Delta d_{s,i,j} < b \,\}.$$

The corresponding bin-specific metrics are then defined as

$$\mathrm{MAE}^{\Delta}_{[a,b)} = \frac{1}{|\mathcal{E}^{\Delta}_{[a,b)}|} \sum_{(s,i,j)\in\mathcal{E}^{\Delta}_{[a,b)}} \left|\hat{d}_{s,i,j} - d_{s,i,j}\right|,$$

$$\mathrm{RMSE}^{\Delta}_{[a,b)} = \sqrt{\frac{1}{|\mathcal{E}^{\Delta}_{[a,b)}|} \sum_{(s,i,j)\in\mathcal{E}^{\Delta}_{[a,b)}} \left(\hat{d}_{s,i,j} - d_{s,i,j}\right)^2}.$$

We use bin edges at $-\infty, -5, -2, -1, -0.5, 0, 0.5, 1, 2, 5, 10, \infty$.

The outer bins are wider because extreme delay recovery and accumulation are much less frequent than small local changes around zero, as illustrated by the overall delay-delta bin counts (Figure 25). Since these bins do not have equal width, this plot should not be interpreted as a density over delay delta, but only as a visualization of support across the evaluation bins. The per-target breakdown further shows that this support becomes progressively more spread out across later future-event slots, which is expected since trains have more time to recover or accumulate delay as the prediction target moves further ahead (Figure 26).

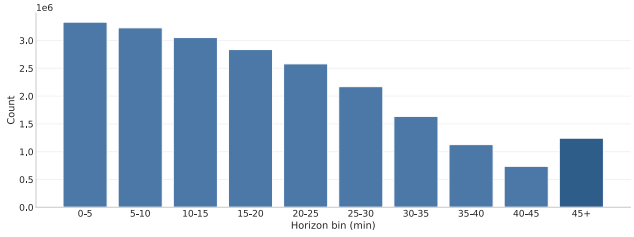

Figure 22: Overall distribution of valid targets across horizon bins on the standard tier.

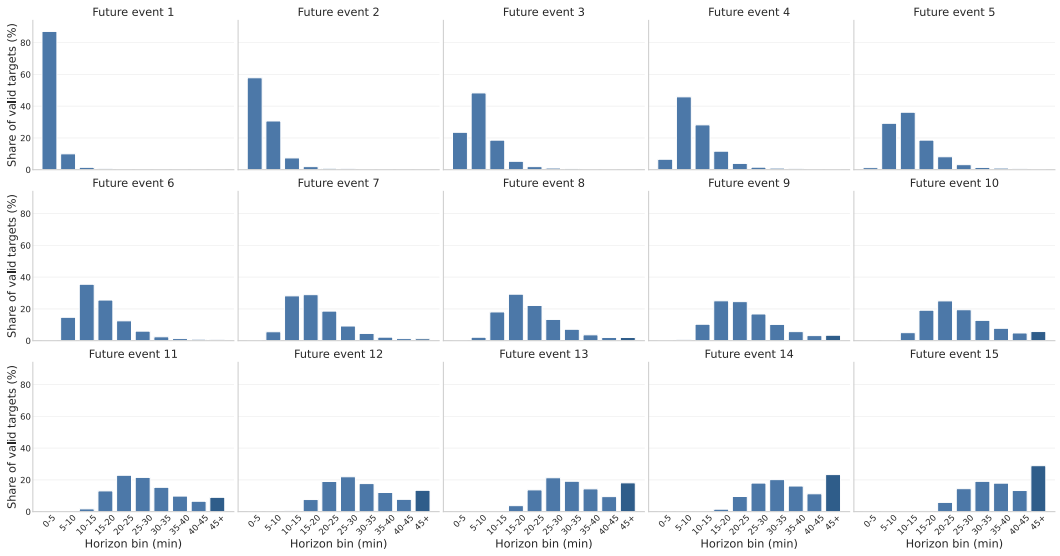

Figure 23: Distribution of valid targets across horizon bins broken down by future-event slot on the standard tier.

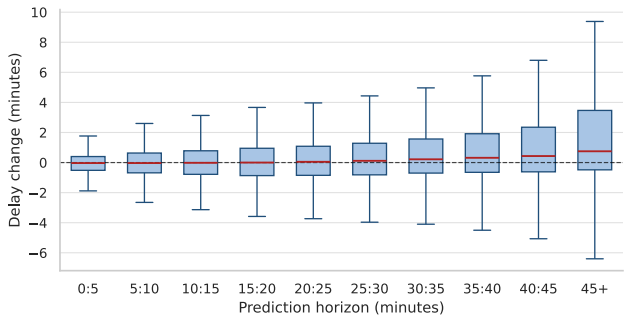

Figure 24: Delay-delta distribution by horizon bin on the standard tier. Boxes show the interquartile range with the median indicated by the horizontal line; whiskers extend to 1.5 times the interquartile range, and outliers are omitted for readability.

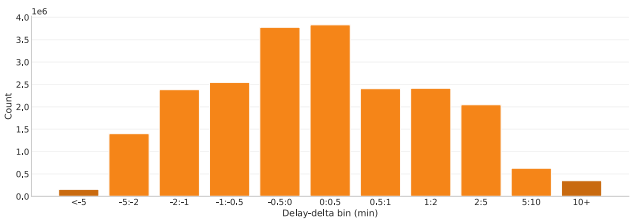

Figure 25: Overall distribution of valid targets across delay-delta bins on the standard tier.

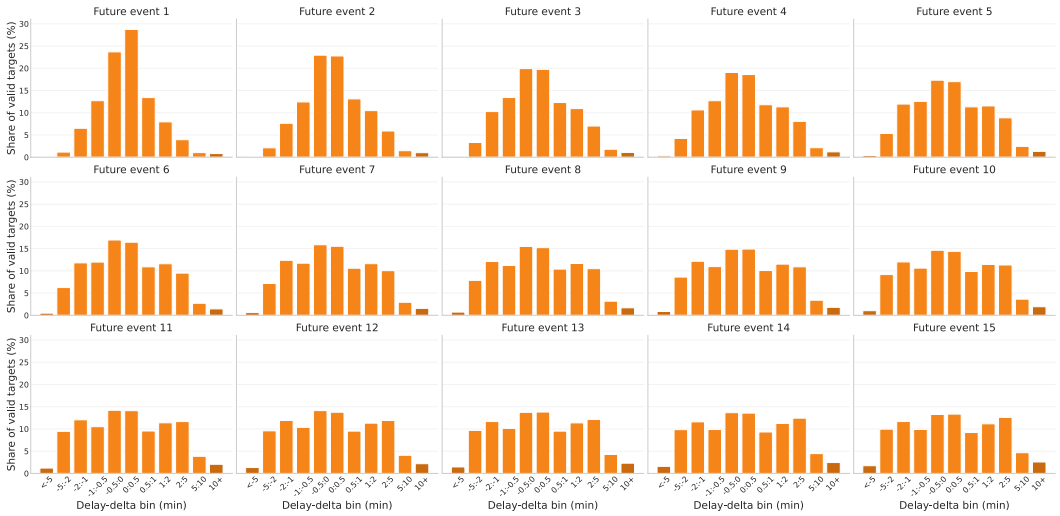

Figure 26: Distribution of valid targets across delay-delta bins broken down by future-event slot on the standard tier.

C.2 Model Details

C.2.1 Translation

We include a simple translation baseline, which is used in similar forms in operational railway settings Arthaud et al. (2024); Wang & Work (2015). For every future target event (s, i, j), it predicts

$$\hat{d}_{s,i,j} = d^{\text{last}}_{s,i},$$

where $d^{\text{last}}_{s,i}$ is defined in Section 4.1. Despite its simplicity, this baseline provides a strong reference point for evaluating the added value of more complex models.

C.2.2 Graph-event

The graph-event model is a deterministic, non-learning reference model inspired by event-based train delay propagation approaches Goverde (2010); Wei et al. (2015). Its input artifacts are described in Appendix B.5. Its purpose in the benchmark is to provide a graph-based model that explicitly simulates train movements and simple infrastructure interactions without relying on learned representations. Starting from each prediction snapshot, it propagates the currently active trains forward over the railway network using their current states, inferred paths, empirical travel-time statistics estimated from the training data, and a simple precedence constraint for trains sharing infrastructure. Future delays are then read out event by event from the resulting simulated evolution. Figure 29 provides an overview of the full pipeline, including the initialization stage and the subsequent graph event processing.

Railway Network We represent the railway network as a graph $G = (V, E)$, where each node $v \in V$ corresponds to an operational point and each edge $e \in E$ corresponds to a rail link between two operational points. Each edge e is associated with a physical length $\ell(e) \in \mathbb{R}_{>0}$ measured in meters. For the simulation, we use a directed version of this graph in which each rail link is duplicated in both directions, allowing train movements and link occupancy constraints to be modeled directionally. This graph provides the spatial support on which the graph-event model simulates train movements. Figure 3 illustrates the resulting network structure on the Belgian railway system.

Train Itinerary For a given train, we represent its itinerary as an ordered sequence of events

$$\big((v_0, \tau_0, t^{\text{sched}}_0),\ (v_1, \tau_1, t^{\text{sched}}_1),\ \ldots,\ (v_m, \tau_m, t^{\text{sched}}_m)\big),$$

where $v_k \in V$ is the operational point of event k, τ_k its event type (arrival, departure or passage), and t_k^{sched} its scheduled time. Consecutive itinerary events are not necessarily adjacent in the railway network, and even when two operational points are directly connected they may be linked by more than one rail edge due to potential parallel tracks. The model therefore uses the inferred path information stored in the `deduced_paths` field of the `journeys` table to recover the exact ordered link-level route followed between successive events. More precisely, for each consecutive event pair $(k, k+1)$, we define a subpath

$$P_k = (e_{k,1}, e_{k,2}, \ldots, e_{k,r_k}),$$

where each $e_{k,r} \in E$ and the ordered sequence P_k connects v_k to v_{k+1} in the railway network. These subpaths form the link-level support on which the event-driven simulation is later carried out.

Snapshot Initialization. At a prediction snapshot with time t^{snap}, each active train is initialized from its current itinerary position, while its initial delay state is set to the last known delay d^{last}. More precisely, let k denote the index of the most recent attained event before the snapshot. The initial delay of the train is set to d^{last}. If $k < 0$, the train is treated as being before its first event and initialized in a special departing state. If $k \geq m$, it is treated as having completed its route and initialized in a special arrived state. Otherwise, the train is initialized relative to the subpath P_k connecting events k and $k+1$.

The expected total traversal time of this event pair, denoted by S_k, is estimated from empirical event-pair travel-time statistics computed on the training split, keyed by the start and end operational points, event types, line information, and train relation. More precisely, the model uses the first finite positive estimate among the 0.40 quantile, the 0.45 quantile, and the median of the matching empirical travel-time statistics, in that order, and falls back to the scheduled travel time if no such estimate is available. These quantile choices and their ordering were selected based on MAE measured on the training set. To ensure that the train cannot reach event $k+1$ before its scheduled time, the expected total traversal time is lower-bounded by the remaining scheduled travel time measured from the heuristic start time of the event pair, i.e.

$$S_k \leftarrow \max\big(S_k,\ t_{k+1}^{\text{sched}} - \big(t_k^{\text{sched}} + d^{\text{last}}\big)\big)\,.$$

If P_k is empty, the train is initialized in a stopped-at-station state, corresponding to a stop at node v_k between the current event and the next one, with expected state duration S_k. Otherwise, under a constant-speed assumption, the total time S_k is distributed across the directed links $(e_{k,1}, \ldots, e_{k,r_k})$ of the subpath proportionally to their lengths, i.e.

$$S_{k,r} = S_k \frac{\ell(e_{k,r})}{\sum_{s=1}^{r_k} \ell(e_{k,s})}, \qquad r = 1, \ldots, r_k.$$

For the current event pair, we define the elapsed time since the predicted start of the event pair as

$$\epsilon_k = t^{\text{snap}} - \big(t_k^{\text{sched}} + d^{\text{last}}\big).$$

The train is then initialized on the directed link reached after elapsed time ϵ_k along the subpath. Equivalently, this corresponds to the index r such that

$$\sum_{s=1}^{r-1} S_{k,s} \leq \epsilon_k < \sum_{s=1}^{r} S_{k,s},$$

up to clipping at the ends of the subpath if needed, as illustrated in Figure 27. In this way, each active train is initialized either at a station stop or on a directed rail link, together with its current delay and the remaining future itinerary needed for simulation. When several trains occupy the same directed link, their entry times determine their ordering on that link and therefore the order in which precedence constraints are applied during simulation.

At the end of this step, each active train is represented by its current itinerary event index, its current occupancy state (directed link, stopped-at-station, departing, or arrived), the expected finish time of that state, and its current delay.

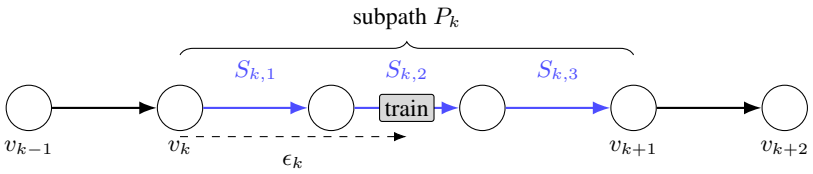

Figure 27: Illustration of snapshot initialization in the graph-event model. The train is initialized on the directed subpath P_k between itinerary events k and $k+1$. The elapsed time ϵ_k is used to infer the current position along the subpath, while the expected link traversal times $S_{k,1}, S_{k,2}, S_{k,3}$ distribute the total event-pair travel time over the directed links proportionally to their lengths.

Event-Driven Propagation. A key modeling assumption is that precedence constraints are enforced only on directed links, not on station states. In other words, trains are not allowed to overtake one another while traversing the same directed rail link, but they may effectively reorder while stopped at nodes between successive itinerary events. This keeps the interaction model simple and local to shared infrastructure occupancy.

For an event pair $(k, k+1)$, if the corresponding subpath P_k is empty, the train occupies a stopped-at-station state with expected finish time

$$t_k^{\text{end}} = t_k^{\text{entry}} + S_k.$$

If instead the train occupies the directed link $e_{k,r}$ of a non-empty subpath, its expected finish time is

$$t_{k,r}^{\text{end}} = t_{k,r}^{\text{entry}} + S_{k,r}.$$

As illustrated in Figure 28, when several trains occupy the same directed link, they are ordered by their entry times, and precedence is enforced in that order by replacing each predicted finish time with

$$t_{k,r}^{\text{end}} \leftarrow \max\big(t_{k,r}^{\text{end}},\ t_{\text{prev}}^{\text{end}} + 40\big),$$

where $t_{\text{prev}}^{\text{end}}$ denotes the finish time of the preceding train on the same directed link, thereby enforcing a fixed 40-second separation between successive trains on that link. The choice of a 40-second separation is based on empirical testing on the training data. These same rules are used both to build the initial ordered queues from the snapshot state and to insert new train states later during simulation.

The global evolution is implemented with a priority queue over predicted state-finish times. At each step, the train with smallest current state finish time t is removed from its current occupancy state and its state is consumed. If the consumed state is the directed link $e_{k,r}$ with $r < r_k$, a new state is created for the next directed link $e_{k,r+1}$ of the same subpath, with entry time t. If the consumed state is the final directed link of the subpath, or a stopped-at-station state, the train reaches event $k+1$, its delay is updated to

$$d_{k+1} = t - t_{k+1}^{\text{sched}},$$

and a new state is constructed toward the next itinerary event. In all cases, the resulting state is inserted into its corresponding occupancy queue, the precedence rule is applied if needed, and its predicted finish time is pushed back into the global priority queue. Whenever this transition increases the itinerary event index, the updated delay is recorded as the prediction for the corresponding future target event.

When a train reaches the end of its itinerary, it enters a terminal arrived state and is no longer reinserted into the priority queue. The simulation ends once this queue becomes empty.

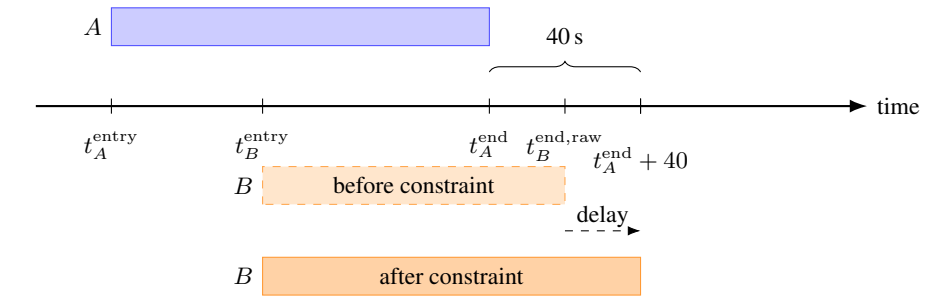

Figure 28: Illustration of the precedence rule in the graph-event model. Train A enters a directed link before train B. If the unconstrained finish time of B occurs less than 40 seconds after the finish time of A, it is shifted forward so that $t_B^{\text{end}} \geq t_A^{\text{end}} + 40$.

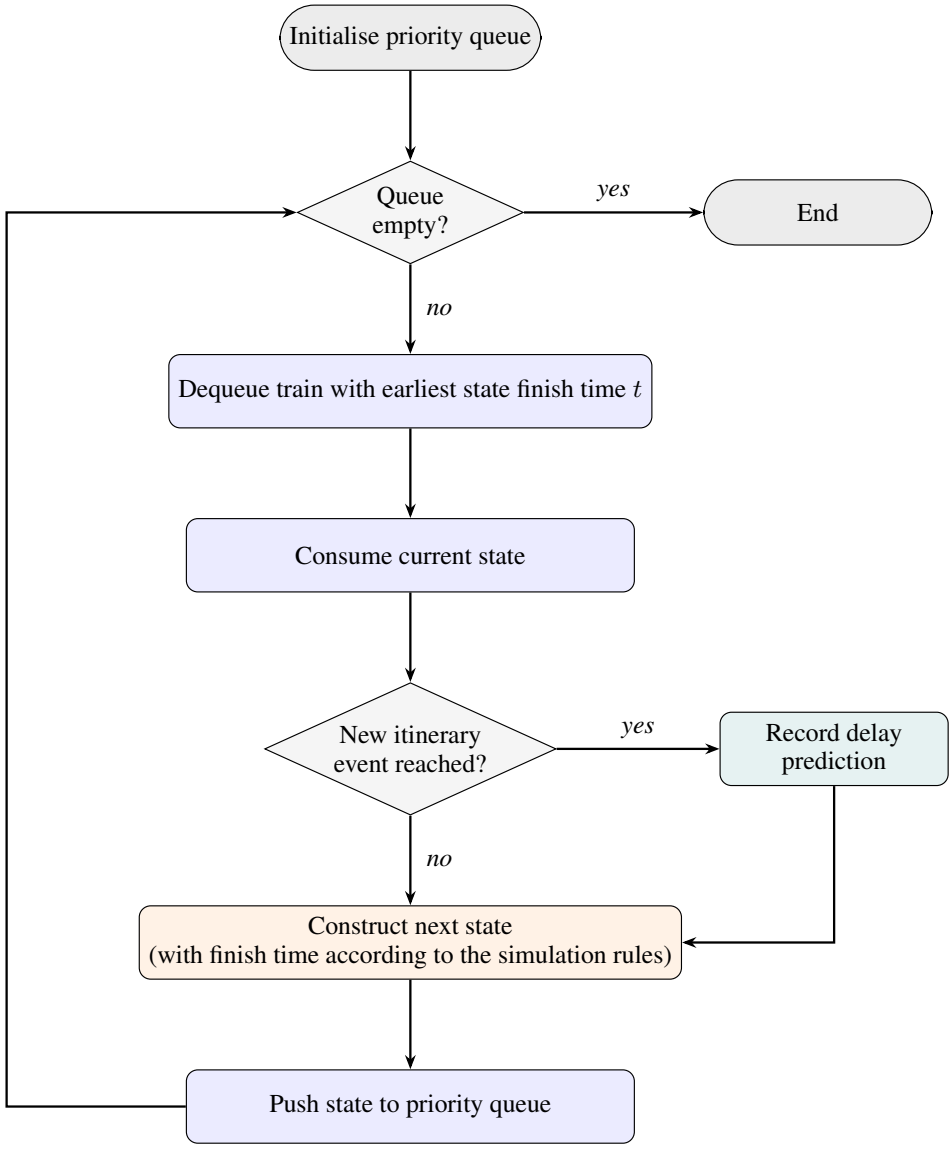

Figure 29: Illustration of the flowchart for the graph-event model.

C.2.3 XGBOOST

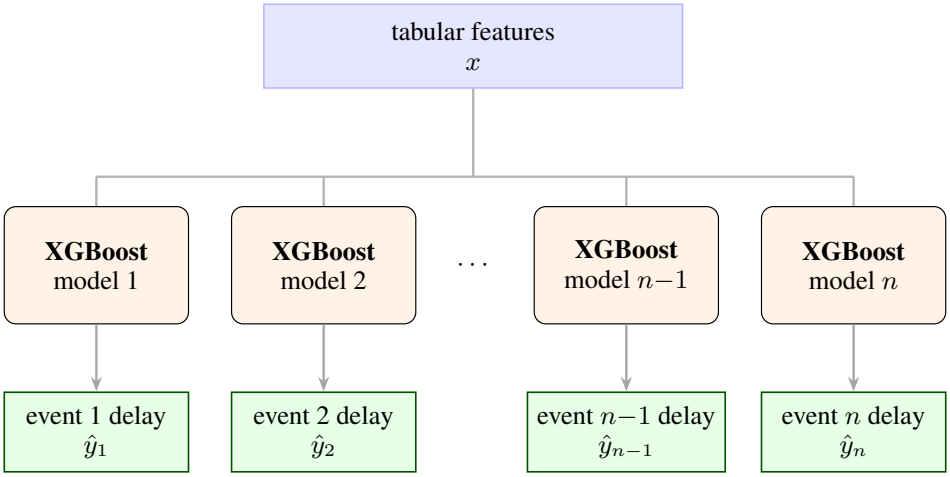

Figure 30: Illustration of the XGBoost architecture.

The XGBoost model uses the same tabular representation as the MLP, namely a single fixed-dimensional feature vector for each train-snapshot pair; this input data format is described in Appendix B.5. As illustrated in Figure 30, it trains one independent gradient-boosted tree regressor per prediction rather than predicting all targets jointly. Each regressor therefore maps the shared tabular input to the normalized residual future delay at one specific event horizon, and the final prediction vector is obtained by concatenating the outputs of the n target-specific models.

C.2.4 MLP

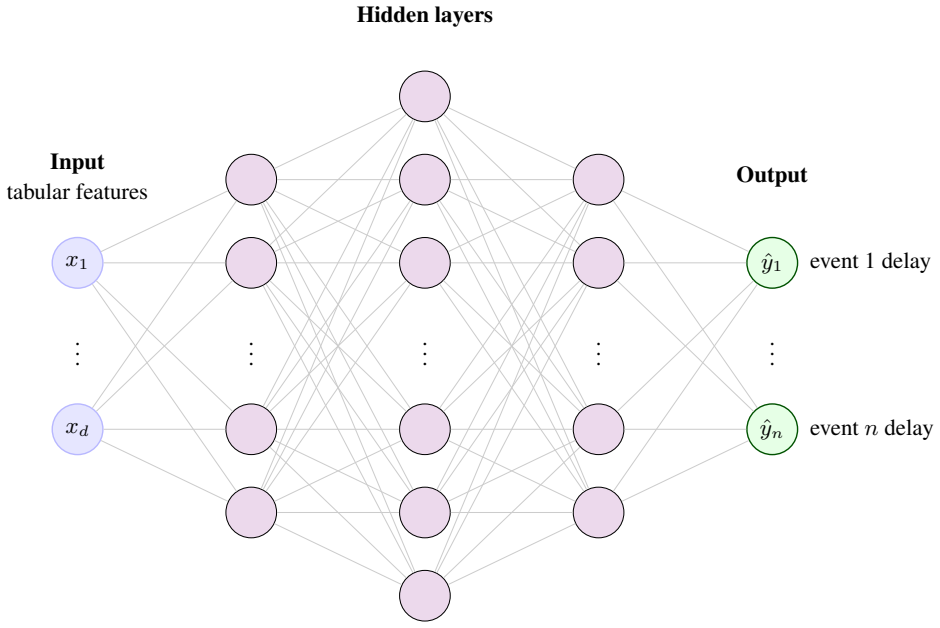

Figure 31: Illustration of the MLP architecture.

The MLP model operates on the tabular representation, where all input features for a train at a given snapshot are concatenated into a single fixed-dimensional vector; this input data format is described

in Appendix B.5. As illustrated in Figure 31, the model is a standard feed-forward multilayer perceptron composed of a stack of fully connected hidden layers with ReLU activations and dropout regularization. Its output layer has one scalar neuron per prediction target, so that the model directly predicts one normalized residual future-delay target for each of the next n events in a single forward pass.

C.2.5 LSTM

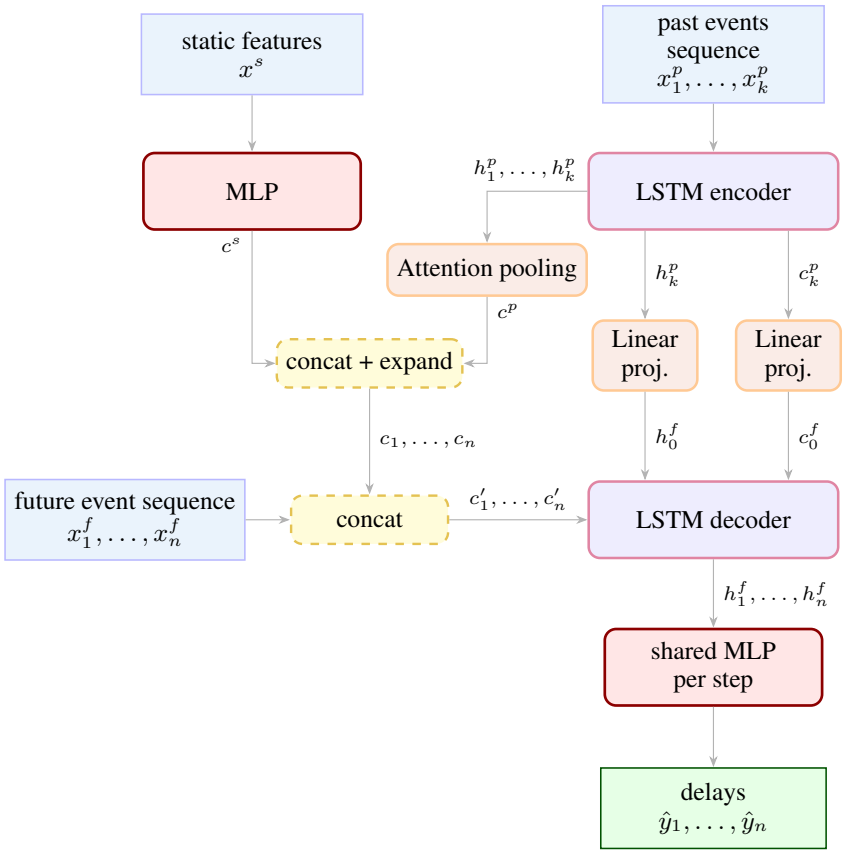

Figure 32: Illustration of the LSTM sequence-to-sequence architecture.

Our prediction setting naturally has a sequential structure: each instance contains a past event sequence describing the recent observed progression of a train up to the prediction snapshot, together with a future event sequence corresponding to the downstream events of the same train for which delays must be predicted. The sequential input data format used by this model is described in Appendix B.5. This makes recurrent architectures a relevant family to consider, in line with prior work on train delay prediction using LSTMs Huang et al. (2020); Yu et al. (2023). However, existing LSTM-based formulations do not directly match our snapshot-based benchmark setting, which requires predicting the next sequence of scheduled events for each active train. We therefore adopt a tailored encoder-decoder LSTM sequence-to-sequence architecture for this model, allowing the model to encode the recent history of the train and decode it into successive delay predictions along its future itinerary.

The input is organized into three feature families: static features, past event features, and future event features. Static features encode train- and snapshot-level context together with local rail-context and weather information. Past event features encode the observed train history through schedule offsets, observed delays, event types, and station embeddings. Future event features encode the known scheduled structure of the upcoming itinerary through schedule offsets, event types, and

station embeddings. In the benchmark, these inputs are instantiated over a fixed context of past and future events and a fixed look-ahead over upcoming rail links.

As illustrated in Figure 32, the model processes the three feature families through dedicated components. First, the past event sequence $(x_1^p, \ldots, x_k^p)$ is fed to an LSTM encoder, producing hidden states $(h_1^p, \ldots, h_k^p)$ together with final hidden and cell states (h_k^p, c_k^p). A learned attention-pooling module then scores the encoder hidden states and forms a context vector c^p as a weighted sum over the past sequence, allowing the decoder to focus on the most relevant parts of the observed history rather than relying only on the encoder's last state. Second, the static features x^s are passed through a small MLP to obtain a static embedding c^s. The two context sources are concatenated and then expanded across the prediction horizon to form a repeated context sequence $(c_1, \ldots, c_n)$, so that each future decoding step has access to the same global information about the train and its recent history. Third, each future target step contributes its future known features $(x_1^f, \ldots, x_n^f)$, and the model appends the normalized position of the step within the prediction horizon. These are concatenated with the repeated context sequence to form decoder inputs $(c'_1, \ldots, c'_n)$.

The decoder is itself an LSTM initialized from transformed versions of the encoder final states: linear projection layers map (h_k^p, c_k^p) to (h_0^f, c_0^f) before decoding begins. This bridge lets the decoder start from a representation shaped by the observed past while still allowing a learned adaptation between encoder and decoder dynamics. The decoder then processes the full future input sequence and returns hidden states $(h_1^f, \ldots, h_n^f)$. Finally, a shared step-wise MLP is applied independently to each decoder state h_i^f to predict the corresponding delay $\hat{y}_i$ for future event i. In this way, the model combines sequential memory from the past, static train-level context, and known future covariates to produce a coherent sequence of downstream delay predictions.

C.2.6 TRANSFORMER

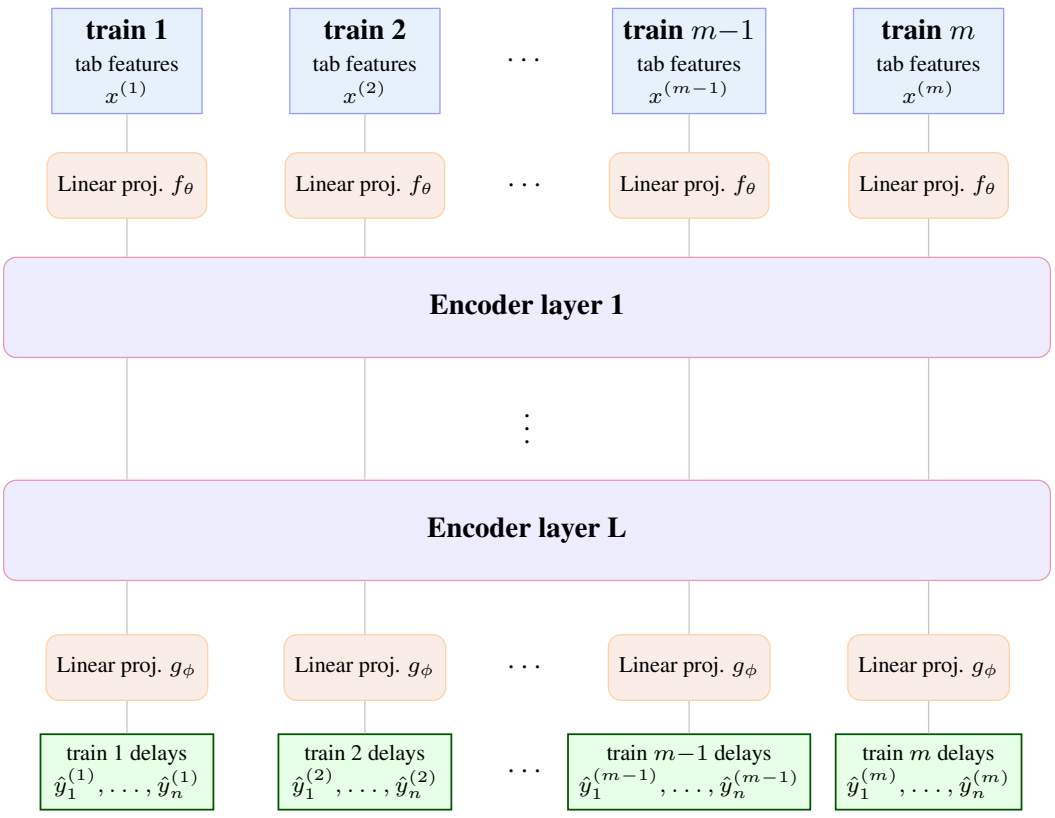

Figure 33: Illustration of the Transformer architecture.

The Transformer model from Arthaud et al. (2024) operates on the set of active trains present in a prediction snapshot. It uses the same tabular feature representation as the MLP and XGBoost models, whose input data format is described in Appendix B.5, but instead of processing each train independently, it groups all train feature vectors from a given snapshot into a variable-length sequence. As illustrated in Figure 33, each train vector is first projected into a shared embedding space and then processed by a stack of Transformer encoder layers with self-attention, ReLU feed-forward blocks, and dropout regularization, allowing every train representation to attend to all others in the snapshot. A shared linear projection head is applied independently to the contextualized representation of each train, producing one normalized residual future-delay target per future event.

C.2.7 GNN

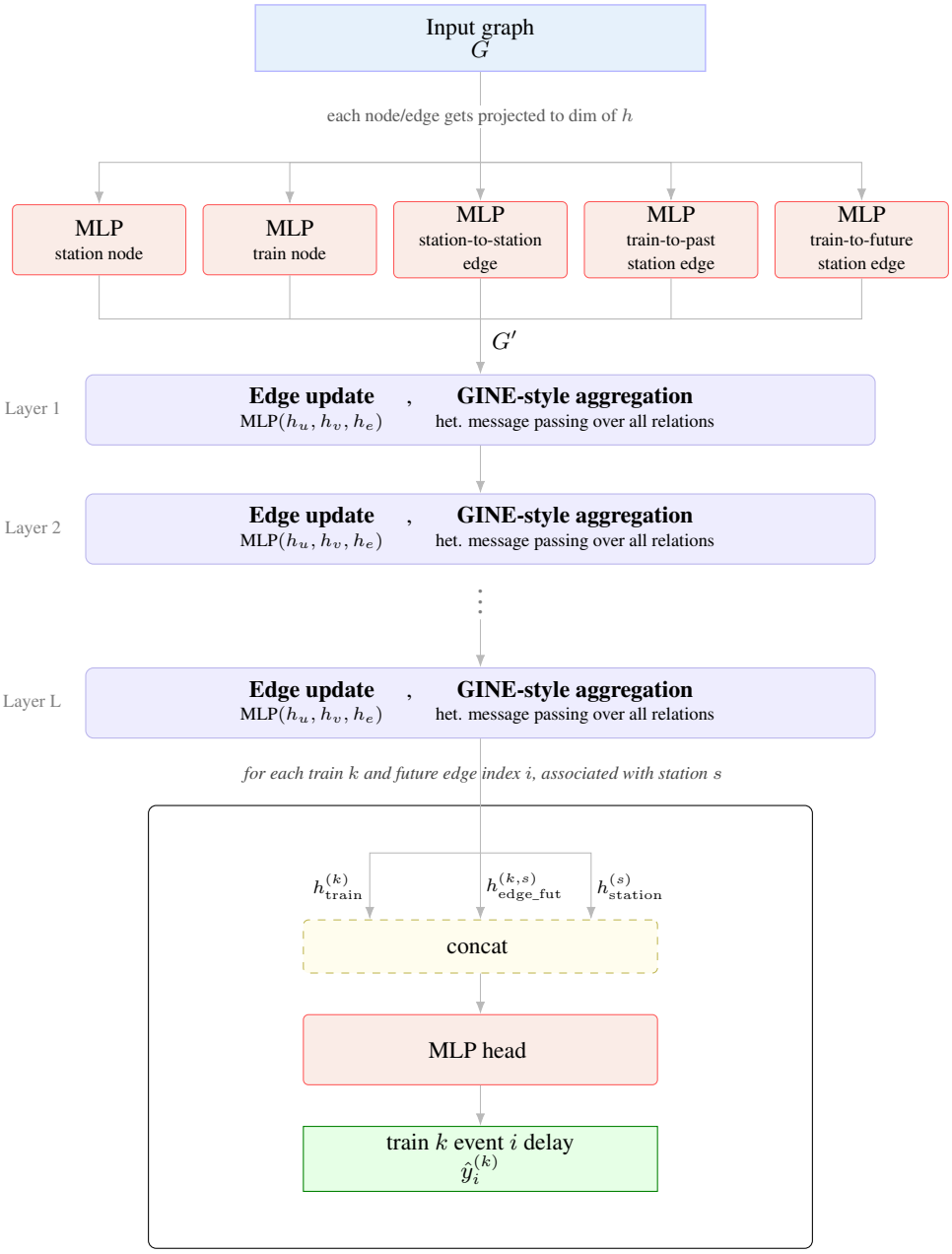

Figure 34: Illustration of the GNN architecture.

In our benchmark, each prediction snapshot can be viewed not only as a collection of train-level features, but as a structured system of interactions between active trains, stations, and railway links. This makes graph neural networks a natural model family for RIDE, as they can operate directly on relational structure rather than relying only on flattened or sequentialized representations. Following recent graph-based approaches to train delay prediction Li et al. (2024); Huang et al. (2024), we design a heterogeneous GNN adapted to our snapshot-based forecasting setting. In particular, existing graph formulations do not directly match our task, which requires predicting delays for the next sequence of scheduled events of each active train within a shared network snapshot. We therefore introduce a graph construction and prediction setup tailored to this benchmark.

The input graph representation is described in Appendix B.5 and illustrated in Figures 20 and 21. The GNN consumes this heterogeneous snapshot graph with train and station nodes, station-to-station edges, train-to-station edges for past and future events, and the reverse relations provided by the dataset. Future delay residuals are not included in the future-edge input attributes; they are stored as edge targets and used only to compute the training loss. The normalized future-event rank remains an input feature, while the restored rank metadata is used to align edge predictions with evaluation horizons.

The GNN architecture is illustrated in Figure 34. All node and edge features are first projected to a common hidden dimension. The model then applies several layers of heterogeneous GINE-style message passing over the station-to-station, train-to-past-station, and train-to-future-station relations, together with their reverse relations. At each layer, each edge embedding is updated from the states of its incident nodes together with its previous edge state, node states are updated through heterogeneous message aggregation, and residual connections with optional layer normalization are applied. Finally, each future train-to-station edge defines a prediction target. For each such edge, the corresponding train representation, station representation, and future-edge embedding are concatenated and passed through an MLP head to predict the normalized residual future delay. This design allows the model to combine network topology, local infrastructure context, and train-specific event history within a unified graph representation.

For illustration, Figure 21 shows a full-network GNN snapshot. To keep the visualization readable, we display at most the 7 closest past and 7 closest future train-to-station edges for each train, and we select a snapshot containing only 53 active trains to avoid overloading the figure.

C.3 Training Procedure

All learning-based models were trained on the training split, with hyperparameter selection based on validation MAE computed on the last 10% of training snapshots in temporal order. After hyperparameter selection, final benchmark results were obtained by retraining each model on the full training split and evaluating on the common 2025 test split through the shared `test_eval_table`. For stochastic learning-based models, final test results were aggregated over seeds $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Table 25 summarizes the main model-specific training choices.

Table 25: Training settings used during hyperparameter-search trials for learning-based models.

Aspect	XGBoost	Neural models
Objective	`reg:absoluteerror`	L1 loss
Optimizer	native XGBoost boosting	AdamW
Max trial length	determined by `n_estimators`	75 epochs
Early stopping	25 rounds	6 epochs

All XGBoost and MLP runs use fp32 precision, while LSTM, Transformer, and GNN use fp32 on the lite tier and bf16 on the standard tier.

C.4 Hyperparameter Search and Best Configurations

Hyperparameter search was performed independently for each model family and benchmark tier using Optuna Akiba et al. (2019), optimizing validation MAE on the last 10% of training snapshots in

temporal order. Search spaces were chosen to reflect each model family's architectural requirements while remaining as comparable as possible across tiers; shared settings are merged across tiers when identical. The best configuration per model family and tier retained for final retraining and test evaluation is reported alongside each search space for reproducibility.

C.4.1 XGBoost

Table 26: XGBoost hyperparameter search space for the lite and standard benchmark tiers.

Hyperparameter	Type	Standard	Lite
`n_estimators`	int	$\{200, 400, 800, 1200\}$	
`max_depth`	int	$\{6, 8, 10, 12\}$	
`learning_rate`	float	log-uniform $[5 \times 10^{-3}, 10^{-1}]$	
`subsample`	float	$\{0.6, 0.7, 0.8, 0.9, 1.0\}$	
`colsample_bytree`	float	$\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$	
`min_child_weight`	int	$\{1, 5, 10, 20\}$	
`reg_alpha`	float	log-uniform $[10^{-3}, 10^{1}]$	
`reg_lambda`	float	log-uniform $[10^{-3}, 10^{1}]$	

Table 27: Best XGBoost configurations selected for the lite and standard benchmark tiers.

Hyperparameter	Standard	Lite
`n_estimators`	1200	1200
`max_depth`	12	12
`learning_rate`	8.13e-2	6.62e-2
`subsample`	1.0	1.0
`colsample_bytree`	0.6	0.7
`min_child_weight`	1	20
`reg_alpha`	4.09e-2	9.70e-2
`reg_lambda`	9.05e-1	1.31e-1

C.4.2 MLP

Table 28: MLP hyperparameter search space for the lite and standard benchmark tiers.

Hyperparameter	Type	Standard	Lite
`hidden_dims`	list[int]	3/5/7/9-layer pyramidal templates, widths from 256 up to 4096[1]	
`dropout`	float	$\{0.05, 0.10, 0.15, 0.20, 0.25\}$	
`batch_size`	int	$\{512, 1024, 2048, 4096\}$	
`lr`	float	log-uniform $[10^{-4}, 3 \times 10^{-3}]$	
`weight_decay`	float	log-uniform $[10^{-6}, 10^{-3}]$	

[1]The exact templates are [256, 512, 256], [256, 512, 1024, 512, 256], [256, 512, 1024, 2048, 1024, 512, 256], and [256, 512, 1024, 2048, 4096, 2048, 1024, 512, 256].

Table 29: Best MLP configurations selected for the lite and standard benchmark tiers.

Hyperparameter	Standard	Lite
hidden_dims	pyramidal, peak 2048	pyramidal, peak 4096
dropout	0.10	0.15
epochs	64	72
batch_size	4096	1024
lr	1.31e-4	1.09e-4
weight_decay	1.68e-6	1.09e-5

C.4.3 LSTM

Table 30: LSTM hyperparameter search space for the lite and standard benchmark tiers.

Hyperparameter	Type	Standard	Lite
batch_size	int	$\{512, 1024, 2048\}$	
lr	float	log-uniform $[5 \times 10^{-5}, 2 \times 10^{-3}]$	
weight_decay	float	log-uniform $[10^{-6}, 10^{-4}]$	
hidden_dim	int	$\{128, 256, 512\}$	
num_layers	int	$\{1, 2, 3\}$	
dropout	float	$\{0.05, 0.10, 0.15, 0.20, 0.25\}$	
static_hidden_dim	int	set to hidden_dim	
static_out_dim	int	set to $\lfloor$hidden_dim$/2\rfloor$	
head_hidden_dim	int	set to hidden_dim	

Table 31: Best LSTM configurations selected for the lite and standard benchmark tiers.

Hyperparameter	Standard	Lite
batch_size	2048	512
lr	1.99e-3	1.32e-3
weight_decay	3.41e-5	1.72e-5
hidden_dim	512	512
num_layers	3	3
dropout	0.20	0.15
static_hidden_dim	512	512
static_out_dim	256	256
head_hidden_dim	512	512
epochs	13	12

C.4.4 TRANSFORMER

Table 32: Transformer hyperparameter search space for the lite and standard benchmark tiers.

Hyperparameter	Type	Standard	Lite
d_model	int	{128, 256, 512, 1024}	{128, 256, 512}
nhead	int	$\{1, 2, 4, 8\}$	
num_layers	int	$\{2, 3, 4, 5, 6\}$	
dim_feedforward	int	set to $4 \times$ d_model	
dropout	float	$\{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35\}$	
batch_size	int	{32, 64, 128, 256}	{16, 32, 64}
lr	float	log-uniform $[5 \times 10^{-5}, 5 \times 10^{-3}]$	
weight_decay	float	log-uniform $[10^{-6}, 10^{-4}]$	

Table 33: Best Transformer configurations selected for the lite and standard benchmark tiers.

Hyperparameter	Standard	Lite
d_model	512	256
nhead	1	1
num_layers	6	6
dim_feedforward	2048	1024
dropout	0.35	0.30
batch_size	128	32
lr	6.58e-5	1.27e-4
weight_decay	2.44e-5	1.85e-6
epochs	60	58

C.4.5 GNN

Table 34: GNN hyperparameter search space for the lite and standard benchmark tiers.

Hyperparameter	Type	Standard	Lite
batch_size	int	$\{16, 32, 64\}$	
lr	float	log-uniform $[5 \times 10^{-5}, 2 \times 10^{-3}]$	
weight_decay	float	log-uniform $[10^{-6}, 10^{-4}]$	
hidden_dim	int	$\{64, 128, 256\}$	
num_layers	int	$\{2, 3, 4, 5, 6\}$	
hetero_aggr	string	{sum, mean}	
use_layer_norm	bool	{true, false}	
gnn_dropout	float	$\{0.00, 0.01, 0.05, 0.10, 0.15\}$	
head_dropout	float	$\{0.00, 0.01, 0.05, 0.10, 0.15\}$	
edge_head_hidden_dim	int	$\{128, 256, 512\}$	

Table 35: Best GNN configurations selected for the lite and standard benchmark tiers.

Hyperparameter	Standard	Lite
batch_size	16	32
lr	1.17e-4	7.90e-4
weight_decay	9.35e-6	6.86e-5
hidden_dim	128	128
num_layers	5	6
hetero_aggr	sum	mean
use_layer_norm	false	false
gnn_dropout	0.15	0.15
head_dropout	0.00	0.10
edge_head_hidden_dim	512	256
epochs	49	34

C.5 Computational Resources and Search Budgets

The experimental setup tables report per-worker resources and budgets; each Optuna job launches 8 workers in parallel, so the aggregate worker-hour budget is 8 times the reported per-worker wall-clock budget.

Table 36: Experimental setup for the lite tier.

Setting	Value
GPU per worker	NVIDIA V100-32GB
CPU resources per worker	10 × Intel(R) Xeon(R) Gold 6248 @ 2.50GHz
RAM	40 GB
Optuna workers per job	8
Max trials per worker	50
Optuna budget per worker	10h for MLP and XGBoost; 19h for other models

Table 37: Experimental setup for the standard tier.

Setting	Value
GPU per worker	NVIDIA A100
CPU resources per worker	8 × EPYC 7543 Milan AMD processors
RAM	64 GB
Optuna workers per job	8
Max trials per worker	50
Optuna budget per worker	10h for MLP and XGBoost; 19h for other models

C.6 Additional Experiments

C.6.1 Standard-Tier Confidence Intervals

Table 38 reports the mean MAE over 10 random seeds together with 95% confidence intervals computed using the Student-t distribution. These intervals quantify the uncertainty associated with the reported Standard-tier mean MAEs and provide an indication of their stability across random initializations. The intervals are narrow for all learning-based models, showing that the corresponding mean estimates are relatively stable across seeds, although the degree of variability differs between models. Pairwise differences between models are assessed separately using paired statistical significance tests in Section C.6.2.

Table 38: Standard-tier mean MAE and 95% confidence intervals over 10 random seeds.

Model	Mean MAE (s)	95% CI (s)
GNN	73.621	[73.477, 73.765]
Transformer	74.537	[74.345, 74.730]
LSTM	74.622	[74.418, 74.826]
XGBoost	76.583	[76.574, 76.592]
MLP	77.202	[77.173, 77.231]

C.6.2 Standard-Tier Statistical Significance Tests

To assess whether the Standard-tier performance differences between the GNN and the other learning-based models are consistent across random seeds, we perform two-sided paired Wilcoxon signed-rank tests over the 10 runs matched by seed. For each comparison, the null hypothesis is that the distribution of paired MAE differences is centered at zero, against the two-sided alternative that it is not centered at zero. Four comparisons are specified: GNN versus Transformer, LSTM, XGBoost, and MLP. The resulting p-values are adjusted jointly using the Holm procedure to control the family-wise error rate at $\alpha = 0.05$.

Table 39 reports the median paired MAE difference, the Wilcoxon signed-rank statistic W, the unadjusted p-value, and the Holm-adjusted p-value for each comparison. The paired difference is defined as $\Delta\mathrm{MAE} = \mathrm{MAE}_{\mathrm{GNN}} - \mathrm{MAE}_{\mathrm{other}}$, such that negative values indicate lower MAE for the

GNN. As none of the Standard-tier paired differences are exactly zero, exact two-sided Wilcoxon p-values are used.

Table 39: Paired Wilcoxon signed-rank tests for Standard-tier MAE across 10 matched random seeds. Holm-adjusted p-values account for the four planned comparisons.

Comparison	Median ΔMAE (s)	W	p-value	Holm-adjusted p-value
GNN vs. Transformer	-0.986	0	0.00195	0.00781
GNN vs. LSTM	-0.962	0	0.00195	0.00781
GNN vs. XGBoost	-2.944	0	0.00195	0.00781
GNN vs. MLP	-3.528	0	0.00195	0.00781

All four comparisons remain statistically significant after Holm correction. In addition, the paired MAE difference is negative for every seed in all four comparisons, indicating that the GNN achieves lower MAE than each comparator in every matched Standard-tier run. Thus, the aggregate Standard-tier differences are consistent across the evaluated random seeds rather than being driven by a small subset of runs.

C.6.3 Standard-Tier Figures

This subsection provides complementary visualizations of the standard-tier benchmark results. Figure 35 summarizes aggregate performance, while Figures 36 and 37 show regime-wise MAE gaps across prediction horizon and delay-delta bins.

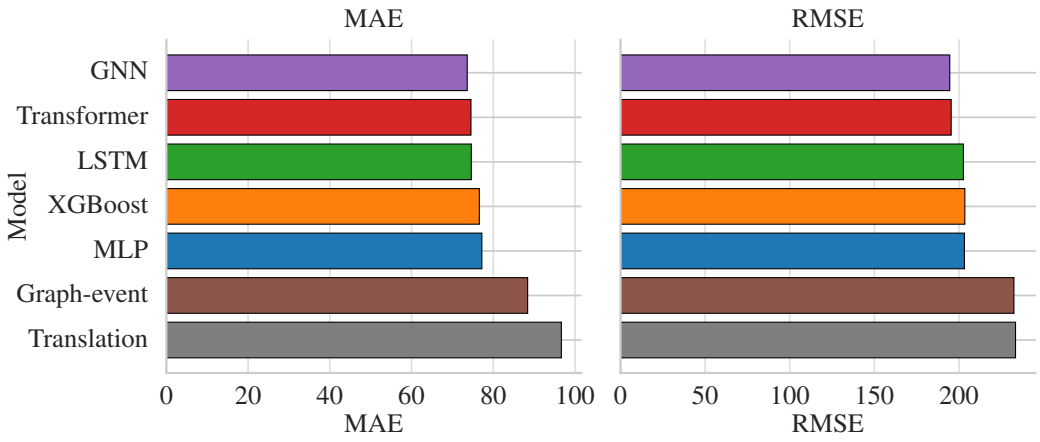

Figure 35: Aggregate standard-tier benchmark performance across models.

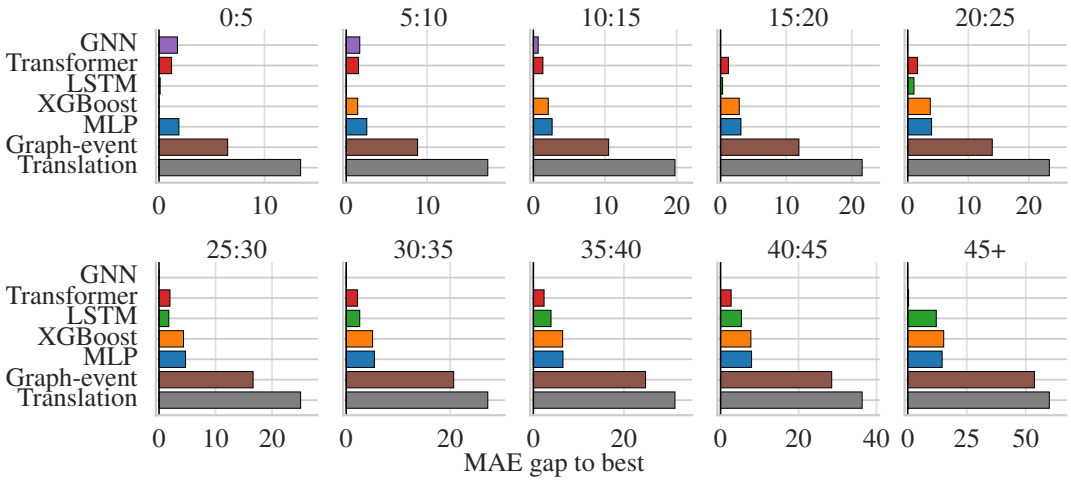

Figure 36: Standard-tier prediction-horizon breakdown, shown as MAE gaps relative to the best model in each horizon bin.

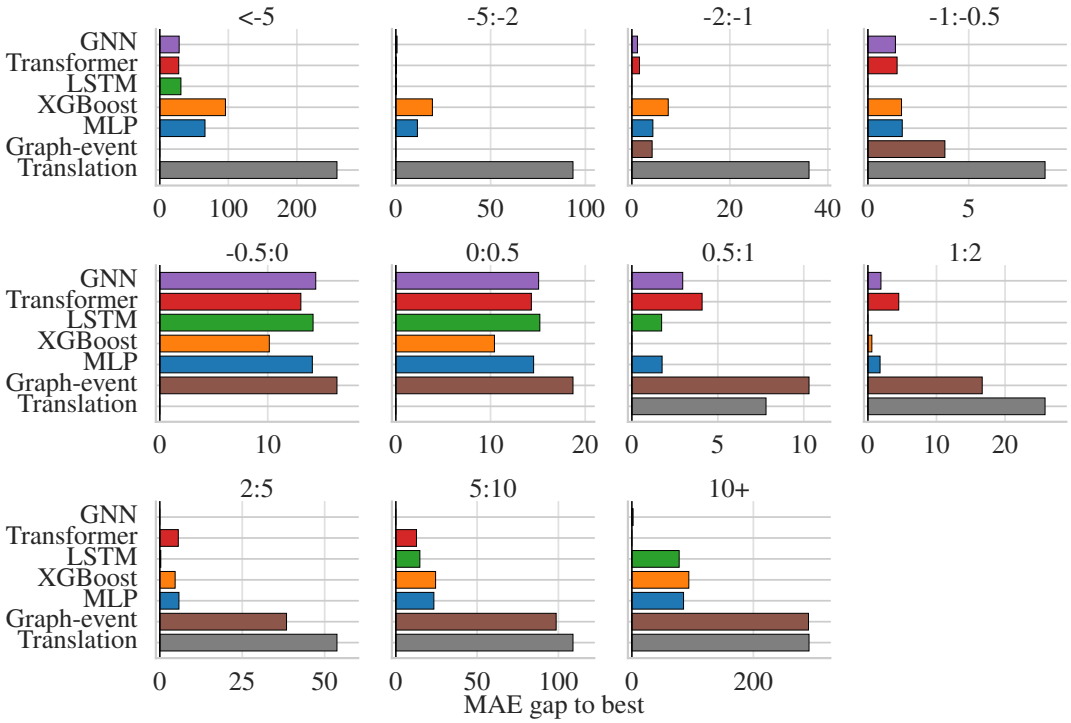

Figure 37: Standard-tier delay-delta breakdown, shown as MAE gaps relative to the best model in each delay-delta bin.

C.6.4 Benchmark on the Lite Dataset

Aggregate Results. Table 40 summarizes results on the lite benchmark tier, which is intentionally smaller and uses 15,000 training snapshots and 3,000 test snapshots. The GNN again attains the best mean performance (76.78 MAE, 199.84 RMSE), followed closely by the LSTM (77.26 MAE)

and Transformer (77.72 MAE). Among tabular models, XGBoost outperforms the MLP (78.59 vs. 79.73 MAE). The graph-event model improves clearly over the simple translation rule (89.45 vs. 97.48 MAE), yet remains well behind all learning-based approaches, so the gap between learning and non-learning methods persists in this lower-data regime. Compared to the standard tier, the learning-based models are more gradually separated, and one shift in the relative ordering is worth noting. The Transformer drops below the LSTM, consistent with the expectation that attention-based architectures are less sample-efficient. The GNN's advantage over the second-best model has also narrowed to a smaller margin (76.78 vs. 77.26 MAE), while its variance across runs is higher than any other learning-based model. That the LSTM remains this competitive despite operating on one train at a time and lacking any explicit network-level delay propagation beyond local network context features is a notable result. As on the standard tier, no single architecture dominates convincingly, suggesting that further progress is as likely to come from stronger feature design and problem-specific modeling choices as from architectural changes alone. Confidence intervals and statistical significance tests for the Lite-tier results are reported in Appendices C.6.5 and C.6.6, respectively. The latter confirms significant differences between the GNN and Transformer, XGBoost, and MLP, while the difference with the LSTM is not statistically significant after Holm correction. Figure 38 provides a compact visual summary of these aggregate lite-tier comparisons.

Table 40: Lite-tier test performance. MAE/RMSE in seconds; ±: std. over 10 seeds.

Model	MAE	RMSE
Translation	97.48	236.87
Graph-event	89.45	236.30
MLP	79.73 ± 0.10	209.44 ± 1.06
XGBoost	78.59 ± 0.02	207.97 ± 0.05
Transformer	77.72 ± 0.18	201.81 ± 0.74
LSTM	77.26 ± 0.26	207.07 ± 0.68
GNN	**76.78 ± 0.68**	**199.84 ± 1.06**

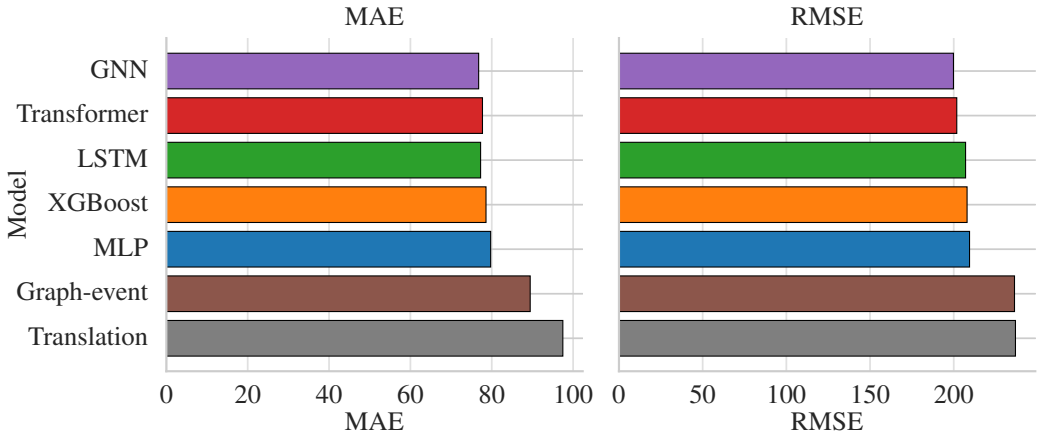

Figure 38: Aggregate lite-tier benchmark performance across models.

Results by Prediction Horizon. Table 41 breaks down performance by prediction horizon. A broadly similar pattern emerges on the lite tier, although with a few notable differences. In the very shortest 0:5 minute bin, XGBoost performs best, consistent with its strong performance in the dominant regime of small delay changes, as explained in the delay change paragraph below. Beyond that, the LSTM is the strongest model from 5 to 20 minutes, so its short-horizon advantage extends further than on the standard tier, where it is strongest from 5 to 15 minutes after XGBoost takes the very shortest 0:5 minute bin. The GNN then becomes the best-performing approach from 20 minutes

onward. The Transformer again remains competitive across all horizons and, as on the standard tier, tends to outperform the LSTM at longer horizons, but here this separation emerges more gradually and only becomes clear later in the horizon range. As on the standard tier, one important caveat is that the latest horizon bins are progressively biased toward trains that accumulate delay, with this effect being particularly pronounced in the final 45+ bin, since these bins become thinner by construction under the fixed future-event prediction window. As a result, performance differences across horizon bins should not be interpreted as a pure horizon effect alone, but rather as reflecting a mixture of horizon length and delay regime. Figure 39 visualizes the corresponding per-bin MAE gaps relative to the best model in each horizon regime.

Table 41: Mean test-set MAE by prediction horizon on the lite tier. Bins are expressed in minutes.

Model	0:5	5:10	10:15	15:20	20:25	25:30	30:35	35:40	40:45	45+
Translation	37.4	57.2	72.3	85.0	95.6	107.5	122.4	142.0	166.2	324.5
Graph-event	30.6	48.7	63.2	75.6	86.6	99.4	116.1	136.0	158.6	318.1
MLP	27.1	43.6	56.6	68.0	77.7	88.8	102.3	119.4	140.6	283.5
XGBoost	**25.0**	42.2	55.8	67.4	77.2	88.0	101.3	118.5	139.6	281.1
Transformer	26.9	43.0	55.8	66.7	76.1	86.7	99.8	116.3	136.8	270.4
LSTM	25.6	**41.5**	**54.4**	**65.6**	75.3	86.2	99.7	116.7	137.6	277.5
GNN	27.5	43.5	55.6	66.0	**74.9**	**85.0**	**97.5**	**113.5**	**133.1**	**266.3**

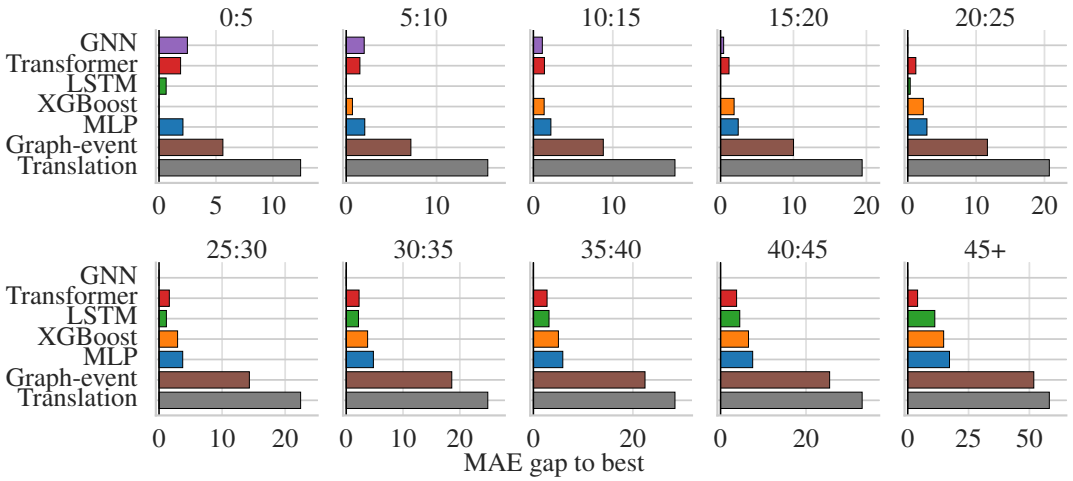

Figure 39: Lite-tier prediction-horizon breakdown, shown as MAE gaps relative to the best model in each horizon bin.

Results by Delay Change. Table 42 reports performance by delay-delta bin. As on the standard tier, the translation baseline is strongest around zero delay change, where simply carrying forward the current delay provides a strong approximation. Excluding this trivial baseline, XGBoost is notably strongest in the central delay-delta regime around zero, from -0.5 to 1 minute, which corresponds to the most typical delay-change regime in the data. This suggests, again, that XGBoost has a stronger tendency than the other learning-based models to stay close to the dominant regime of small delay changes. The LSTM again performs particularly well on moderate negative delay-delta bins, indicating that sequential structure alone already provides a strong signal for modeling local delay recovery. For moderate delay accumulation, between 0.5 and 5 minutes, the strongest models are again tabular or sequential rather than graph-based, with the LSTM now performing best in the 2 to 5 minute range. At the opposite end, the graph-event model performs best on large negative delay deltas, i.e. strong delay recovery, which is consistent with its tendency to predict travel times that are on average faster than the scheduled ones. In contrast, the GNN performs best across the large positive

delay-delta bins and is now also clearly strongest in the most extreme 5+ minute accumulation regime. As on the standard tier, this suggests that models with learned interaction patterns between trains are better able to capture substantial delay increases, likely because these are tied to propagation effects in the network. Figure 40 visualizes these regime-wise differences as MAE gaps relative to the best model in each delay-delta bin.

Table 42: Mean test-set MAE by delay-delta bin on the lite tier. Bins are expressed in minutes; negative values correspond to delay recovery and positive values to delay accumulation.

Model	<-5	-5:-2	-2:-1	-1:-0.5	-0.5:0	0:0.5	0.5:1	1:2	2:5	5:10	10+
Translation	431.4	174.4	85.2	44.3	**14.7**	**13.5**	43.3	85.0	184.8	409.1	1262.7
Graph-event	**171.2**	**82.3**	53.7	39.7	31.5	32.3	45.6	75.6	169.3	398.5	1259.1
MLP	251.0	97.3	55.6	38.2	28.9	27.7	37.4	62.2	140.0	329.2	1083.1
XGBoost	281.5	104.3	58.2	37.9	25.0	23.6	**35.4**	60.5	138.0	328.7	1079.0
Transformer	212.0	85.5	53.0	38.4	28.6	28.4	40.6	65.6	140.4	320.4	1006.5
LSTM	214.0	86.3	**51.6**	**37.0**	30.0	29.4	38.1	**60.5**	**133.8**	318.8	1056.9
GNN	218.5	88.4	52.9	38.1	30.3	29.4	39.2	62.9	134.9	**306.6**	**991.3**

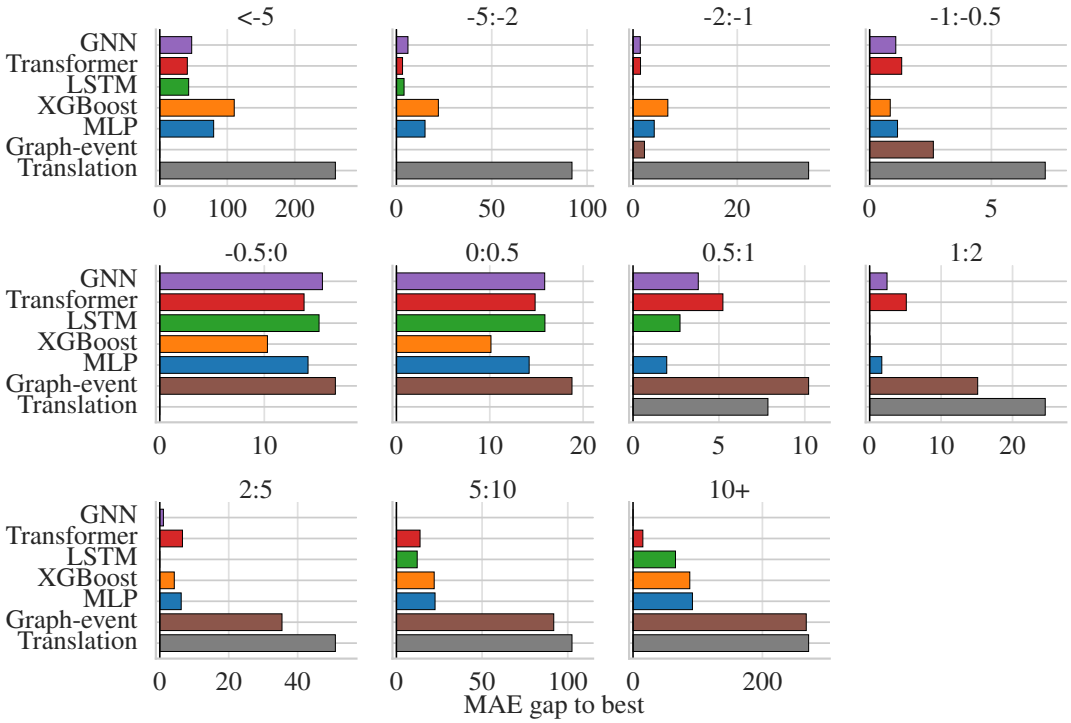

Figure 40: Lite-tier delay-delta breakdown, shown as MAE gaps relative to the best model in each delay-delta bin.

Together, these two evaluation breakdowns again provide a more informative view of model behavior than aggregate MAE and RMSE alone. They make it possible to identify which models perform best in specific forecasting regimes, such as short- versus long-horizon prediction or delay recovery versus delay accumulation, and thereby support broader insights into the dynamics of train delay prediction, even in the smaller-data regime of the lite tier.

C.6.5 LITE-TIER CONFIDENCE INTERVALS

Table 43 reports the mean MAE over 10 random seeds together with 95% confidence intervals computed using the Student-t distribution. These intervals quantify the uncertainty associated with the reported Lite-tier mean MAEs and provide an indication of their stability across random initializations. The degree of variability differs between models, with the GNN exhibiting the widest confidence interval among the learning-based models. Pairwise differences between models are assessed separately using paired statistical significance tests in Section C.6.6

Table 43: Lite-tier mean MAE and 95% confidence intervals over 10 random seeds.

Model	Mean MAE (s)	95% CI (s)
GNN	76.779	[76.264, 77.294]
LSTM	77.256	[77.058, 77.454]
Transformer	77.718	[77.581, 77.854]
XGBoost	78.585	[78.573, 78.598]
MLP	79.729	[79.650, 79.808]

C.6.6 LITE-TIER STATISTICAL SIGNIFICANCE TESTS

To assess whether the Lite-tier performance differences between the GNN and the other learning-based models are consistent across random seeds, we perform two-sided paired Wilcoxon signed-rank tests over the 10 runs matched by seed. For each comparison, the null hypothesis is that the distribution of paired MAE differences is centered at zero, against the two-sided alternative that it is not centered at zero. Four comparisons are specified: GNN versus LSTM, Transformer, XGBoost, and MLP. The resulting p-values are adjusted jointly using the Holm procedure to control the family-wise error rate at $\alpha = 0.05$.

Table 44 reports the median paired MAE difference, the Wilcoxon signed-rank statistic W, the unadjusted p-value, and the Holm-adjusted p-value for each comparison. The paired difference is defined as $\Delta\mathrm{MAE} = \mathrm{MAE}_{\mathrm{GNN}} - \mathrm{MAE}_{\mathrm{other}}$, such that negative values indicate lower MAE for the GNN. As none of the Lite-tier paired differences are exactly zero, exact two-sided Wilcoxon p-values are used.

Table 44: Paired Wilcoxon signed-rank tests for Lite-tier MAE across 10 matched random seeds. Holm-adjusted p-values account for the four planned comparisons.

Comparison	Median ΔMAE (s)	W	p-value	Holm-adjusted p-value
GNN vs. LSTM	-0.436	9	0.06445	0.06445
GNN vs. Transformer	-1.029	3	0.00977	0.01953
GNN vs. XGBoost	-1.877	0	0.00195	0.00781
GNN vs. MLP	-2.996	0	0.00195	0.00781

The comparisons with Transformer, XGBoost, and MLP remain statistically significant after Holm correction, whereas the comparison with LSTM does not reach statistical significance. The paired MAE difference is negative for 8 of the 10 seeds against LSTM, 9 of the 10 seeds against Transformer, and all 10 seeds against XGBoost and MLP. Thus, while the GNN obtains the lowest mean MAE in the Lite tier, the seed-level results provide weaker evidence for a systematic difference between the GNN and LSTM than for the other learning-based models.

C.6.7 ABLATION STUDY ON FEATURE FAMILIES

To better understand which information sources drive performance in the tabular setting, we perform a feature-family ablation study on the lite-tier MLP model. Starting from the best-performing full-feature lite MLP configuration, we remove one feature family at a time, retrain the model, and compare the resulting test-set performance against the full-feature model. Hyperparameters are kept fixed to the selected full-feature configuration, while the number of training epochs for each ablation

is chosen in a preliminary validation phase using the last 10% of training snapshots in temporal order. Final results are then obtained on the lite test split using the selected epoch count and aggregating over three fixed seeds, $\{0, 1, 2\}$.

Table 45 shows that the most important feature families for the lite-tier MLP are past delay features and planned timing features. Removing past delay features produces the largest MAE degradation (+4.26), while removing planned timing features is nearly as harmful (+3.83) and leads to the largest RMSE increase (+9.63). Unsurprisingly, this indicates that the MLP depends strongly on past itinerary delay information and schedule-relative information. Interestingly, the larger RMSE increase when removing planned timing features suggests that schedule-relative information may be more important for limiting larger prediction errors tied to more extreme cases. Local network context also contributes meaningfully, though to a lesser extent (+1.47 MAE). This is notable in light of the broader benchmark results: the overall gap between the GNN and the MLP on the lite tier is only about 3 MAE, so a 1.47 MAE degradation from removing local network context highlights just how informative these engineered features already are for summarizing nearby structural effects. Snapshot-time features have a smaller but still visible effect (+0.44 MAE), suggesting that coarse temporal context beyond the immediate event sequence remains useful. Event-type indicators and node embeddings also have a modest but measurable contribution (+0.45 MAE in both cases). For the MLP, node embeddings can plausibly act mainly as a way to memorize operational-point-specific patterns associated with particular parts of the network, whereas more expressive architectures may be better able to exploit such representations for modeling interactions between trains and events, for instance through attention mechanisms. In contrast, removing train information or weather features has almost no effect, with changes close to zero. For weather, this weak effect may reflect redundancy with other inputs: temporal features already capture strong seasonal and time-of-day patterns, including some of the variation associated with temperature, while delay history may already absorb part of the operational impact of weather conditions. Interestingly, RMSE actually improves slightly when removing train information (-0.46), node embeddings (-0.34), or weather features (-0.13). Figure 41 visualizes these ablation effects relative to the full-feature MLP.

A general caveat in feature ablation studies is that removing a feature family may change not only the information available to the model, but also the effective input dimensionality, model capacity, and regularization regime when the rest of the hyperparameters are kept fixed. In our setup, we reselect only the number of training epochs for each ablation, while keeping all other hyperparameters fixed to the full-feature configuration. This issue is therefore most relevant for node embeddings and, to a lesser extent, event-type indicators, since these are among the highest-dimensional components of the tabular representation. As a result, their ablations should not be interpreted purely as information removal.

Overall, the ablation results indicate that, for the MLP, predictive performance depends most strongly on past itinerary delay information, schedule-relative timing, and local network context, while the remaining feature families appear either secondary or partly redundant in the lite setting.

Table 45: Main results of the lite-tier MLP feature-family ablation study. The first row shows the full-feature model, and each subsequent row removes one feature family. MAE and RMSE are reported as mean ± standard deviation across runs, with Δ values measured relative to the base model.

Setting	MAE	ΔMAE	RMSE	ΔRMSE
MLP (all features)	79.73 ± 0.10	—	209.44 ± 1.06	—
w/o train information	79.70 ± 0.04	-0.03	208.98 ± 0.37	-0.46
w/o snapshot-time features	80.17 ± 0.12	+0.44	209.95 ± 0.67	+0.51
w/o planned timing	83.56 ± 0.21	+3.83	219.07 ± 0.96	+9.63
w/o past delay features	83.99 ± 0.04	+4.26	213.78 ± 0.30	+4.34
w/o event types	80.18 ± 0.10	+0.45	209.75 ± 0.60	+0.31
w/o node embeddings	80.18 ± 0.07	+0.45	209.10 ± 1.06	-0.34
w/o local network context	81.20 ± 0.07	+1.47	210.80 ± 0.36	+1.36
w/o weather features	79.76 ± 0.05	+0.03	209.32 ± 0.25	-0.13

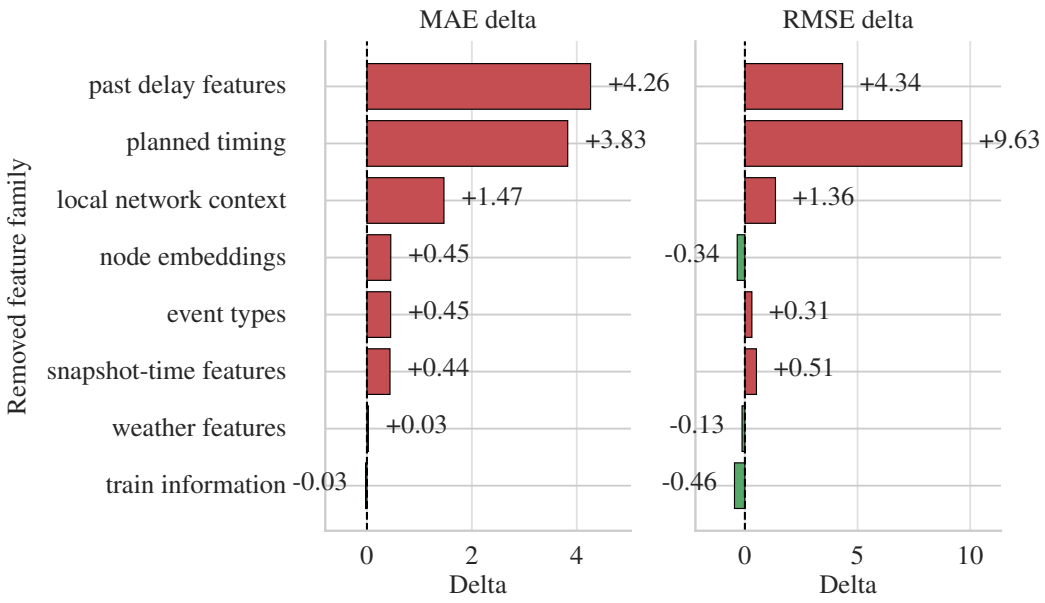

Figure 41: Lite-tier MLP feature-family ablation results, shown as MAE and RMSE changes relative to the full-feature model.