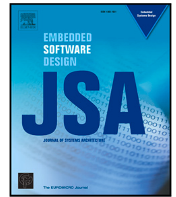
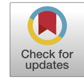
Journal of Systems Architecture 176 (2026) 103811

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc





Scalable transit delay prediction at city scale: A systematic approach with multi-resolution feature engineering and deep learning ☆

Emna Boudabbous [a], Mohamed Karaa [a], Lokman Sboui [a], Julio Montecinos [a,*], Omar Alam [b]

[a] Systems Engineering Department, École de technologie supérieure (ÉTS), 1100 Notre-Dame Street West, Montréal, H3C 1K3, Québec, Canada
[b] Department of Computing and Information Systems, Trent University, 1600 West Bank Drive, Peterborough, K9L 0G2, Ontario, Canada

ARTICLE INFO

Keywords:
Transit delay prediction
GTFS-realtime
Feature engineering
H3 geospatial indexing
Scalability
Urban mobility
Intelligent transportation systems

ABSTRACT

Urban bus transit agencies need reliable, network-wide delay predictions to provide accurate arrival information to passengers and support real-time operational control. Accurate predictions help passengers plan their trips, reduce waiting time, and allow operations staff to adjust headways, dispatch additional vehicles, and manage disruptions. Although real-time feeds such as GTFS-RT are now widely available, most existing delay prediction systems handle only a few routes, rely on hand-crafted features, and offer little guidance on designing a scalable, reusable architecture.

We present a city-scale prediction pipeline that combines multi-resolution feature engineering, dimensionality reduction, and deep learning. The framework systematically generates spatiotemporal features by exploring aggregation combinations over spatial regions (using hexagonal hierarchical indexing), routes, segments, and temporal patterns, then compresses them using Adaptive PCA while preserving 95 % of the variance. To avoid the "giant cluster" problem that occurs when dense urban areas fall into a single spatial region, we introduce a hybrid clustering method that combines geographic and network topology information to yield balanced route clusters and enable efficient distributed training.

We compare five model architectures on six months of bus operations from the Société de transport de Montréal (STM) network in Montréal. A global LSTM with cluster-aware features achieves the best trade-off between accuracy and efficiency (R^2 = 0.7121 at the elementary level), outperforming XGBoost by 9.3 %, xLSTM by 5.3 %, and Autoformer by 43 % in terms of R^2, while achieving comparable accuracy to PatchTST (R^2 = 0.7043) with 77× fewer parameters. LSTM's compact architecture (31,000 parameters) effectively captures short-term temporal dependencies in the compressed feature space, making it more suitable than transformer models, which are overparameterized for this task. We also report multi-level evaluation at the elementary segment, segment, and trip level using walk-forward validation and latency analysis, showing that the proposed pipeline is suitable for real-time, city-scale deployment and can be reused for other networks with limited adaptation.

1. Introduction

Urban public transportation systems are a cornerstone of sustainable mobility in metropolitan areas, carrying millions of passengers daily and helping reduce congestion, pollution, and energy consumption [1]. For bus networks in particular, **reliability**–defined as the consistency of bus service, measured by on-time performance (adherence to scheduled arrival times) and headway regularity (consistent spacing between buses)–is often degraded by multiple factors. Traffic congestion causes variable travel times as buses encounter unpredictable delays at intersections and along congested corridors [2]. Incidents such as accidents, vehicle breakdowns, or road closures create unexpected disruptions that propagate through the network. Adverse weather conditions (snow, rain, ice) affect road conditions and vehicle speeds, leading to increased variability in travel times [3]. Operational constraints, including vehicle availability, driver scheduling, and maintenance requirements, can disrupt scheduled service patterns. These factors interact to create complex, non-linear delay patterns that are difficult to predict using simple rule-based systems.

Accurate real-time delay prediction has therefore become a key function of modern Intelligent Transportation Systems (ITS), as it

☆ This article is part of a Special issue entitled: 'AI4ORC' published in Journal of Systems Architecture.

* Corresponding author.
E-mail addresses: emna.boudabbous.2@ens.etsmtl.ca (E. Boudabbous), mohamed.karaa.1@ens.etsmtl.ca (M. Karaa), lokman.sboui@etsmtl.ca (L. Sboui), julio.montecinos@etsmtl.ca (J. Montecinos), omaralam@trentu.ca (O. Alam).

https://doi.org/10.1016/j.sysarc.2026.103811
Received 19 November 2025; Received in revised form 1 March 2026; Accepted 18 April 2026
Available online 21 April 2026
1383-7621/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

directly affects service quality, passenger satisfaction, and operational efficiency [2]. Reliable predictions enable passengers to plan their trips more effectively and reduce waiting time. They help agencies adjust operations, modify routes and frequencies, and improve service in ways that support the adoption of public transport [2]. As urban mobility demand grows, scalable, reliable delay-prediction systems are increasingly necessary.

While real-time data feeds, such as General Transit Feed Specification Realtime (GTFS-RT), are now widely available, they provide only current vehicle positions and trip updates without predictive capabilities. GTFS-RT does not include predictive models—it only reports where vehicles currently are, requiring additional processing to convert positions into delay predictions. Moreover, GTFS-RT cannot learn from historical patterns and anticipate delays before they occur. It does not provide multi-level predictions (elementary segment, segment, trip) that support operational decision-making. **Artificial intelligence (AI) and machine learning (ML) solutions are therefore essential** to address these limitations: transit delay patterns are complex and non-linear, requiring machine learning to capture intricate spatiotemporal relationships that manual rule-based systems cannot handle. Deep learning models can learn from large-scale historical data to identify patterns that human experts might miss, and scalable AI systems can process millions of observations across entire networks, which is infeasible with manual approaches.

Despite advances in artificial intelligence (AI) and the widespread availability of real-time data through standards such as General Transit Feed Specification Realtime (GTFS-RT), several fundamental challenges still hinder the deployment of reliable, large-scale delay prediction systems [4].

First, feature engineering remains largely ad hoc. Many existing studies rely on manually designed feature sets driven by researcher intuition [5,6]. Such approaches are difficult to reproduce, offer limited coverage of multi-resolution spatiotemporal patterns, and generalize poorly across different transit networks. The lack of a systematic, reproducible framework for feature generation is a central methodological gap.

Second, scalability and computational efficiency pose significant challenges for network-wide transit delay prediction. While many studies report strong performance on individual routes [7,8], few address the complexities of modelling an entire metropolitan transit system comprising hundreds of routes, thousands of stops, and millions of observations [9,10]. Monolithic models that treat the network as a single, undifferentiated unit are both computationally intensive and ill-suited to capturing spatial heterogeneity, particularly the operational differences between dense urban cores and lower-density suburban zones. Spatial clustering has emerged as a practical strategy to address this issue by dividing the network into subregions with similar operational profiles, thereby improving model efficiency and maintaining local delay patterns [11]. However, conventional clustering methods often rely on regular spatial grids that ignore the network topology, leading to severe data imbalances. In particular, high-density urban areas are frequently collapsed into disproportionately large clusters that dominate the dataset. We refer to this as the "giant cluster phenomenon": a spatial-aggregation artifact in which dense regions are represented by oversized spatial units, resulting in spatial homogenization, obscured intra-urban variability, degraded model performance, and inefficient parallel training.

Third, much of the literature focuses on experimental prototypes with limited attention to deployment-oriented evaluation. Essential aspects, such as inference latency, resource usage, systematic architecture comparisons, and computational efficiency, are rarely reported [2]. This makes it difficult for practitioners to select architectures that are feasible for real-time, network-wide deployment. We explicitly address these non-functional requirements in Section 5.2, report measured performance characteristics, and explain how our architecture supports scalability, maintainability, and operational reliability.

Recent technological advances open new opportunities to address these gaps. **Hierarchical spatial indexing systems**, particularly Uber's hexagonal hierarchical spatial indexing (H3) [12], provide a flexible framework for multi-resolution spatial representation. H3 partitions the Earth's surface into nested hexagonal grids across 16 resolution levels (0–15), with decreasing cell diameters, enabling seamless multi-resolution aggregation. Hexagons offer superior geometric properties compared to squares: uniform distance to six neighbours, minimal area distortion, and exact 1:7 parent–child subdivision ratios. At different resolutions, H3 cells naturally align with transit operational scales: neighbourhoods, corridors, and street segments. This hierarchical structure enables consistent aggregation of transit patterns across multiple scales, making H3 particularly well-suited for multi-resolution feature engineering in transit delay prediction. Yet the systematic use of H3 for transit delay prediction, especially in combination with rigorous multi-resolution feature engineering and topology-aware spatial clustering, remains relatively unexplored.

This paper presents an end-to-end pipeline for city-scale delay prediction, developed to meet the operational requirements of bus transit and to support the creation of a reusable architecture that can be readily adapted to other urban transit systems.

Our work pursues three interconnected research objectives: (1) to develop a reproducible framework for exhaustive multi-resolution spatiotemporal feature generation, (2) to design an efficient spatial clustering strategy for data organization and feature engineering that resolves the "giant cluster problem" while preserving network topology, and (3) to perform a rigorous comparative evaluation of multiple deep learning architectures, including computational analysis, on a complete metropolitan bus network.

The main contributions of this work are fourfold:

- **Systematic multi-resolution feature engineering framework.** We introduce a reproducible framework for spatiotemporal feature generation that systematically explores aggregation combinations across multiple spatial resolutions, route identifiers, segments, and temporal dimensions. The framework captures both local segment-level patterns and neighbourhood-level trends. We evaluate multiple dimensionality reduction methods and show that Adaptive PCA effectively compresses the feature space while retaining essential variance, making global model training computationally less demanding.
- **Hybrid H3+topology spatial clustering for data organization.** We propose a route clustering methodology based on weighted Jaccard similarity that combines spatial coverage with topological structure to organize network data and generate cluster-aware features. The method resolves the giant cluster problem by producing balanced clusters that support efficient feature engineering while preserving spatial and topological coherence for global model training.
- **Comparative evaluation of deep learning architectures.** We perform an extensive comparison of five architectures for elementary-segment bus delay prediction: LSTM (long short-term memory), XGBoost (gradient boosting trees), xLSTM (Extended LSTM), PatchTST (patch-based transformer), and Autoformer (autocorrelation transformer). All models share the same preprocessing and feature pipelines and are trained globally on the complete bus network. For each architecture, we report predictive accuracy, training time, inference latency, and memory usage. Results show that an LSTM with compressed features achieves the best overall performance (elementary-level $R^2 = 0.7121$), outperforming XGBoost by 9.3 %, xLSTM by 5.3 %, and Autoformer by 43 % in terms of R^2, while achieving comparable accuracy to PatchTST ($R^2 = 0.7043$) with 77× fewer parameters and a favourable computational and latency profile.

- **End-to-end pipeline for bus delay prediction with multi-level validation.** We implement a complete pipeline for bus delay prediction from data ingestion (GTFS static data, GTFS-RT feeds, and weather data) to distributed feature engineering, dimensionality reduction, and global model training with multi-level prediction aggregation (elementary segment → segment → trip). The system uses walk-forward temporal validation and reports RMSE, MAE, and R^2 at all three levels, along with runtime and resource metrics. Multi-level aggregation exhibits error cancellation: trip-level RMSE (1.85 min for the LSTM model) is lower than segment-level RMSE (2.17 min), demonstrating that random errors partially cancel during hierarchical aggregation and supporting the suitability of the framework for operational deployment.

The remainder of this paper is organized as follows. Section 2 reviews related work in transit delay prediction, feature engineering, and distributed computing for spatiotemporal data. Section 3 presents our methods, including the multi-resolution feature engineering framework, hybrid spatial clustering strategy, and model architectures. Section 4 details the technical implementation, infrastructure choices, hyperparameter configurations, and experimental results. Section 5.3 discusses implications and future research directions.

Motivation and problem setting

The pipeline is demonstrated using data from the Société de transport de Montréal (STM). Control centre operators must identify delayed vehicles, understand where delays accumulate, and decide when to intervene, for example, by holding vehicles, short-turning trips, or adding extra buses. At the same time, passengers rely on predicted arrival times exposed through various information channels. When these predictions are unreliable or unstable, both operators and passengers quickly lose trust in the information system.

Our goal is to provide reliable delay predictions for the entire STM bus network and to make them usable across multiple spatial and temporal granularities. To achieve this, the underlying data and models are organized around a small set of basic units:

- A **trip** is a single scheduled bus journey, as defined by its GTFS `trip_id`, from its origin terminal to its destination terminal.
- A **segment** is a directed pair of consecutive stops $(\text{stop}_i, \text{stop}_{i+1})$ on a route. Segment travel time is computed from GPS observations using geofences around each stop [13].
- Each segment is subdivided into fixed-length **elementary segments**. Predictions are expressed at this elementary level as **pace**, i.e., travel time per metre (seconds/metre), which normalizes for segment length and yields a more homogeneous prediction target across the network [13].

These definitions ensure that delay predictions are comparable across routes of varying lengths and geometries and provide a clear link between the model outputs and quantities of direct operational interest, such as segment- and trip-level delays [13].

A commonly identified requirement for large urban transit systems is a *reusable* architecture that generalizes across networks. Building a bespoke prediction system for a single city or corridor is feasible, but maintaining it is difficult as routes change, new data sources are added, or evaluation requirements evolve. It is also hard for other agencies to reuse such a system if its components are tightly coupled to local assumptions. To address this, our design separates city-specific configurations (GTFS feed, H3 resolution, clustering parameters) from generic components (feature generation, dimensionality reduction, global modelling, and inference). The same pipeline can therefore be adapted to other bus networks by modifying a limited set of configuration files and retraining the models, while preserving the overall structure.

In summary, the motivation for this work is twofold: (i) to support real operational decision making through delay predictions defined consistently at multiple spatial and temporal scales, and (ii) to do so with a scalable, reusable architecture that can be transferred across bus networks with limited engineering effort.

2. Related work

Public transit delay prediction has been extensively studied and remains a relevant problem, given the continuous evolution of transportation systems and their increasing complexity. In this section, we summarize pertinent prior work on delay prediction models, feature engineering methods, and challenges related to scalability and computational costs.

2.1. Transit delay prediction

Early approaches to transit delay prediction relied on simple statistical assumptions, focusing primarily on historical averages and scheduled timetables. With the increasing availability of real-time sensor data and advances in machine learning, models have gradually evolved into data-driven, context-aware frameworks [13].

Statistical approaches include time series analysis using ARIMA [14], Kalman Filters [15], mixture models [16], Bayesian networks [17], and Markov models [18]. Although these methods are easier to implement and interpret, and require less computational resources and data, they often struggle to capture the complex underlying patterns of traffic dynamics and to adapt to irregularities such as service disruptions and sudden events.

The proliferation of real-time transit data from automatic vehicle location (AVL) systems has spurred the development of advanced deep learning models that enable more accurate delay predictions. The models trained on historical GPS data include linear regression [19], support vector machines [20], and gradient-boosting algorithms [21, 22]. However, these methods require meticulous feature engineering and selection for better results.

More recently, deep learning models have become the norm for processing time-series and network data to capture complex, latent spatio-temporal patterns in road networks. Recurrent neural networks (RNNs) have become the go-to model architecture for delay prediction because they can handle complex sequential data. In [3], Alam et al. built a hybrid RNN-LSTM model to predict bus arrival irregularities based on historical GPS positions and weather data for two Toronto bus service routes. Liu et al. [23] employed an LSTM model to predict both short and long-distance arrival times of buses to the station. Another work used BiLSTMs to estimate the bus departure time for each route, then combined them with a DNN to calculate travel time [24]. These works have leveraged the temporal aspect of public transit dynamics to predict delay and arrival times.

In other studies, authors have used spatial patterns in traffic networks with Graph Neural Networks (GNNs) to improve delay prediction modelling. In [25], the authors combined an LSTM with a GNN that captures the static characteristics of the public transit network. This hybrid approach improved the prediction accuracy compared to using temporal and spatial features separately. Sharma et al. proposed a Temporal Graph Convolutional Network that predicts delays across a segmented road network and then aggregates them to estimate the bus arrival time at a given station [26]. Another study proposed a Graph Attention Network that learns dynamic correlations between bus routes and then combines them with spatio-temporal features via an attention mechanism. This significantly improves bus delay estimation, especially for bus routes with fewer road-network representations [27].

2.2. Feature engineering for transit delay prediction

Feature engineering remains the cornerstone of effective predictive modelling for transit systems. While deep learning-based methods often avoid manual feature design through end-to-end learning, empirical evidence consistently shows that well-engineered features substantially improve performance, even for neural architectures.

Conventional feature engineering techniques encompass several categories, including temporal, spatial, and operational (contextual) features. Temporal features such as hours of the day, day of the week,

seasonality, and holiday indicators form the foundation of most prediction models [10,24,28]. Effective temporal encoding requires domain knowledge to identify relevant periodicity and interaction terms. For example, rush-hour behaviour differs fundamentally between weekdays and weekends, necessitating interaction features. On the other hand, spatial information is represented as raw latitude/longitude coordinates [25,28] or higher-level features such as regions, neighbourhoods, and street types [27]. However, these representations suffer from critical limitations: coordinates lack the semantic structure that tree-based models can exploit, while administrative zones exhibit arbitrary boundaries and high spatial heterogeneity.

Operational features provide more context on bus speed, passenger load, and traffic, thereby enriching the input data [23,24,27]. However, this data is often unavailable across different transit agencies. Other works have integrated weather data as input features to improve the prediction, as weather conditions affect road conditions and mobility choices [29]. In [3], the authors improved bus arrival time prediction by 48 % by including weather data to train their LSTM model.

More recent studies have used spatial clustering to generate finer, more robust features for multiple traffic analysis and forecasting tasks [30,31]. In particular, Uber's hexagonal spatial indexing has revolutionized spatial partitioning techniques. H3 partitions the Earth's surface into nested hexagonal grids across 16 resolution levels (0–15) with decreasing diameter size, enabling seamless multi-resolution aggregation [12]. Hexagons offer superior geometric properties compared to squares: uniform distance to six neighbours, minimal area distortion, and exact 1:7 parent–child subdivision ratios. At resolutions 9 (174 m), 10 (66 m), and 11 (25 m), H3 cells naturally align with transit operational scales, namely neighbourhoods, blocks, and street segments, respectively. Several studies have used H3 Indexing for traffic analysis and travel time estimation [26,32,33]. In these works, H3-based clustering helped identify distinctive features of public transportation, such as service quality, density, and frequency.

2.2.1. Critical gap: Ad-hoc feature selection

A fundamental limitation across the existing literature is the **ad hoc, non-systematic nature of feature engineering**. Studies typically report 50 to 200 manually selected features with limited documentation of their selection criteria, making results difficult to compare and build upon [8]. This approach is inherently vulnerable to researcher bias and risks incomplete coverage of critical spatiotemporal interactions. Ultimately, these shortcomings undermine both the **reproducibility** of the findings and the **generalization** of the features to other transit networks, creating a significant barrier to progress in the field.

Despite H3's demonstrated effectiveness in mobility analytics, its systematic exploitation for **multi-resolution feature generation** in transit delay prediction remains largely unexplored. Most studies employ spatial features at a single resolution or rely on ad hoc manual selection, failing to capture the hierarchical nature of delay propagation, in which local segment-level disruptions interact with broader neighbourhood-level congestion patterns. To our knowledge, no prior work systematically explores multi-resolution feature generation across multiple H3 resolutions, combined with comprehensive temporal encoding via automated aggregation.

2.3. Scalability challenges and distributed computing

Operational transit delay prediction for a metropolitan bus network requires processing millions of observations per day across hundreds of routes and thousands of stops. This volume typically exceeds the capacity of single-machine workflows, motivating the use of distributed computing frameworks.

Apache Spark is widely used for large-scale transit data processing because it offers in-memory computation, high-level DataFrame APIs, and distributed machine learning through MLlib [34]—for example, Lv et al. processed 50 million daily taxi GPS trajectories in Beijing with second-level latency using Spark clusters. Other frameworks, such as Dask and Ray, target similar workloads with different design trade-offs, emphasizing tight Python integration or low-latency task scheduling. In this work, we use Spark for feature engineering and model training because it integrates with STM's data infrastructure and can handle the scale of our datasets.

A major scalability challenge for network-wide modelling is spatial heterogeneity: delay patterns differ substantially between dense downtown corridors, suburban residential areas, and industrial zones. Treating the entire network as a single global context leads to (i) computational intractability on multi-million-row datasets with thousands of features, (ii) limited ability to learn location-specific patterns because local signals are drowned in global noise, and (iii) prohibitive training times. Spatial partitioning addresses these issues by decomposing the network into geographic clusters and training specialized models in parallel. However, naive partitioning using regular grids or coarse H3 cells can lead to severe data imbalance. Dense urban cores form a single, large cluster that contains most observations, while peripheral regions form many small clusters with sparse data. We refer to this as the *"giant cluster problem"*. Our empirical analysis in Section 3.6 shows that naive H3 partitioning at resolution 8 yields a coefficient of variation (CV) of 2.0 and an imbalance ratio of 8×, whereas our hybrid H3+topology approach achieves CV = 0.608 and an imbalance ratio of 1.90×.

Topology-aware clustering strategies for transit prediction remain limited. Existing spatial clustering methods typically rely only on geographic proximity, ignoring transit-specific structures such as shared route segments and operational patterns. As a result, geographically close but operationally distinct segments may be grouped, while segments that share the same routes may be split across clusters. To our knowledge, no prior work has combined H3-based spatial partitioning with an explicit topological similarity measure to jointly improve cluster balance and operational coherence in distributed training. This gap motivates the hybrid H3+topology clustering method introduced in Section 3.6.

Finally, production systems must balance prediction accuracy, inference latency, and resource consumption. For real-time passenger information, latency budgets are often on the order of tens of milliseconds, making highly complex models impractical even if they are more accurate. Techniques such as model distillation, feature pruning, quantization, and cascaded architectures have been proposed to manage these trade-offs. Petersen et al. explicitly document this tension: an LSTM model attains lower RMSE than linear regression but at the cost of an order-of-magnitude higher latency; in strict real-time settings, the simpler model may therefore be preferred despite a 26 % accuracy loss [8]. In our evaluation, we report not only accuracy metrics (RMSE, MAE, R^2) but also training time and inference latency to assess the feasibility of candidate architectures for deployment.

2.4. Validation methodologies and production deployment

2.4.1. Evaluation protocols

Most transit delay prediction studies employ some form of cross-validation, but temporal dependencies require specialized protocols. Walk-forward (rolling-origin) validation has been recognized as more appropriate for time-series data because it preserves causal ordering and avoids leaking future information into the training set [35]. Nevertheless, many works still rely on random train–test splits, which can overestimate performance and complicate comparisons across studies. Reported metrics also vary considerably: RMSE, MAE, and MAPE are most common, with occasional use of R^2 and classification-style metrics (e.g., on-time vs. late). This heterogeneity across protocols and metrics limits the ability to compare models fairly and assess whether a given approach is suitable for deployment.

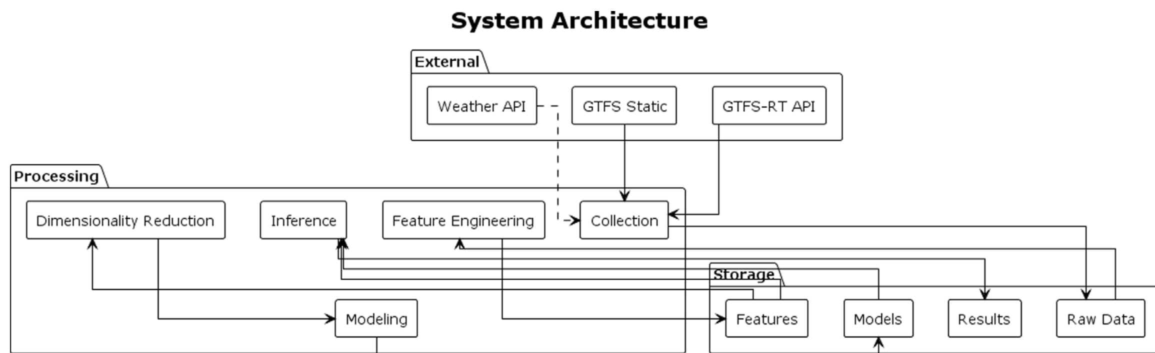


Fig. 1. System architecture: five pipelines with external data sources and storage.

2.4.2. *Production deployment and reusable architectures*

Beyond offline evaluation, a smaller body of work considers deployment aspects of data-driven transit prediction systems. Several papers describe prototype real-time arrival prediction services integrated with passenger information systems or control centres, often using microservice-based backends or streaming frameworks for ingesting AVL and GTFS-RT data (e.g., [8,9]). These systems typically emphasize latency and robustness requirements and sometimes discuss monitoring and retraining strategies. More general ITS frameworks also propose modular architectures for processing streams of transportation data and serving predictions to multiple applications [34,35].

However, most of these deployments are tailored to a single city, operator, or corridor and do not aim to provide a reusable, network-agnostic architecture. The feature engineering pipelines, data organization strategies, and model selection procedures are often tightly coupled to local assumptions. They are rarely documented at a level that would enable other agencies to adopt them directly. In particular, we find little work that (i) treats feature generation, spatial clustering, dimensionality reduction, and model training as components of a configurable architecture, and (ii) evaluates this architecture explicitly in terms of both predictive performance and computational characteristics (training time, inference latency, and resource usage).

This gap motivates our system-level focus. Rather than proposing yet another isolated prediction model, we aim to design and evaluate an end-to-end, reusable architecture for large-scale bus delay prediction that can be adapted to different networks with limited configuration changes.

2.5. *Summary and research gaps*

Our review of the literature highlights four research gaps that motivate this work.

Gap 1: Non-systematic feature engineering. Most existing studies rely on ad-hoc, manually designed feature sets, typically containing 50 to 200 features [8]. The researcher's intuition often drives feature selection and is only briefly documented, limiting reproducibility and transferability. While some recent works use H3 for spatial indexing [26,33], the systematic use of multi-resolution spatial features remains limited. We found no study that combines multi-resolution H3 aggregation (e.g., resolutions 9 and 10) with a structured exploration of temporal encodings via automated aggregation combinations.

Gap 2: Unresolved scalability challenges. Many contributions target a single route or a small subset of the network [7,8]. The few network-wide studies that exist either train a single monolithic global model (which leads to prohibitive training times and spatial dilution of local patterns) or use naive spatial partitioning schemes that introduce severe data imbalance across clusters. In particular, we did not find prior work that combines H3-based geographic partitioning with an explicit topological similarity measure to obtain balanced spatial clusters and to address the "giant cluster problem" observed in dense urban cores.

Gap 3: Limited guidance on operational and reusable architectures. Most papers present experimental prototypes with little discussion of how the proposed methods could be embedded in an operational system. Production-related aspects, such as inference latency, resource constraints, monitoring, model retraining, and integration with existing transit management tools, are rarely addressed. Moreover, architectures are typically described for a single city or corridor, with tightly coupled preprocessing and modelling components, which makes reuse by other agencies difficult. There is a lack of work that treats feature generation, spatial clustering, dimensionality reduction, and model training as configurable components of a reusable architecture for large-scale bus delay prediction.

Gap 4: Insufficient validation rigour. Many studies employ simple temporal holdout splits without walk-forward cross-validation, limited hyperparameter optimization, and few ablation studies. As a result, it is often unclear which feature families or model components drive performance gains. Long-term evaluation, robustness under atypical conditions (e.g., disruptions or extreme weather), and detailed reporting of experimental protocols are also uncommon, making it hard to compare approaches or assess their suitability for deployment.

2.5.1. *Comparative analysis of prior work*

Table 1 summarizes key characteristics of representative prior studies, highlighting their methods, features, scalability characteristics, and limitations. This comparison reveals the diversity of approaches to transit delay prediction and identifies common limitations that motivate our work. The comparison reveals several common limitations across prior work: (1) most studies focus on single routes or small subsets, limiting network-wide scalability, (2) feature engineering remains largely ad-hoc with manual selection and limited documentation, (3) model interpretability is rarely addressed, making it difficult to understand prediction drivers, (4) sensitivity to noisy GPS data and sparsity in low-frequency routes is often not discussed, and (5) production deployment considerations (latency, resource usage, reusable architectures) receive limited attention. Our work addresses these limitations through systematic feature engineering, hybrid clustering for scalability, comprehensive model comparison, and explicit consideration of deployment requirements.

3. Methodology

This section describes the architecture and methods used for bus delay prediction. Our system is organized as five interconnected pipelines: (i) data collection, (ii) feature engineering, (iii) dimensionality reduction, (iv) predictive modelling, and (v) inference. Fig. 1 provides an overview of these components and their interactions with external data sources and data stores.

Table 1
Comparative analysis of prior work in transit delay prediction.

Study	Method	Features	Scalability	Limitations
Alam et al. [3]	Hybrid RNN-LSTM	GPS, weather	2 routes	Limited to few routes; manual feature selection
Liu et al. [23]	LSTM	Temporal, GPS	Single route	Route-specific model; no network-wide evaluation
Sharma et al. [26]	Temporal GCN	Graph structure, temporal	Network-wide	Requires graph construction; limited feature engineering
Petersen et al. [8]	LSTM, Linear	Manual features	Multiple routes	Ad-hoc features; latency-accuracy trade-off
GNN-based [25]	LSTM+GNN	Static network, temporal	Network-wide	Complex architecture; scalability concerns
Graph Attention [27]	GAT with attention	Spatio-temporal, route correlations	Network-wide	Sensitivity to noisy GPS; sparsity in low-frequency routes
H3-based [26,33]	Various with H3	H3 spatial features	Network-wide	Single H3 resolution; limited multi-resolution exploration
Statistical [14,15]	ARIMA, Kalman	Historical averages	Single route	Cannot capture complex patterns; limited adaptability

3.1. Data collection pipeline

The data collection pipeline converts heterogeneous transit data sources into trip-level records suitable for machine-learning models (Fig. 2). The architecture follows a layered design, with an orchestration layer that manages control flow (scheduling, API polling, batch triggers) and a processing layer that handles data transformation. GTFS static schedules provide the network topology (stops, routes, and stop sequences), while GTFS-RT vehicle position updates supply real-time observations. We align these sources in time, associate vehicle positions with scheduled trips, and enrich each record with city-wide weather information. The orchestration layer polls the GTFS-RT API every 10 s and triggers asynchronous batch writes to storage, enabling continuous collection without blocking. Schedule updates are handled by storing successive GTFS snapshots, along with their validity periods, and linking observations to the corresponding schedule version. Details of collection frequency and aggregation procedures are provided in Section 4.3.1.

3.1.1. Trip-level processing

We define a *trip* as a single scheduled bus journey from its origin terminal to its destination terminal, identified by its GTFS `trip_id`. The goal of the trip-level processing step is to transform the raw stream of GTFS-RT vehicle positions into a single record per realized trip, with consistent timestamps and delay values.

Starting from the GTFS static feed and the service calendar, we first expand all scheduled trips for each service day. GTFS-RT vehicle positions are then matched to these planned trips using the vehicle identifier, route, direction, and timestamp. For each matched trip, we construct an ordered sequence of observations along the stop sequence and estimate arrival times at each stop using geofences centred at the stop locations. Delay at a stop is computed as the difference between the observed and scheduled arrival times.

We apply a set of rule-based quality checks to filter unreliable records. Typical checks include discarding trips with too few observations, non-monotonic timestamps, unrealistically high speeds between consecutive points, or locations that deviate excessively from the scheduled route. Records that fail these checks are excluded from subsequent stages. Weather information for the corresponding time interval and area is then joined to each valid trip record. Implementation details for the matching procedure, alignment windows, and filtering thresholds are provided in Section 4.3.1.

3.2. Feature engineering pipeline

The feature engineering pipeline transforms trip-level records into a feature matrix for modelling (Fig. 3). Instead of selecting features manually, we use a fixed procedure that applies the same set of aggregation operations to all experiments, ensuring that the resulting feature set is well-defined and easy to reproduce.

As introduced in Section 3.1.1, each realized trip is represented as an ordered sequence of segments, where a segment is the directed pair of consecutive stops on a route. For modelling, we further subdivide each segment into fixed-length *elementary-segments* and define the prediction target as the elementary-segment *pace* (seconds per metre). Pace is less sensitive to segment length than raw travel time and provides a more homogeneous target across the network.

In addition to this target, we construct spatiotemporal features by aggregating historical values using different grouping strategies. The aggregation framework combines spatial groupings (H3 cells at resolutions 9 and 10, route and segment identifiers), temporal groupings (hour of day, broader time-of-day periods, and per-hour indicator windows), and route-based groupings. In total, 23 distinct combinations of spatial and temporal groupings are considered. For each combination, we compute a small set of summary statistics (mean, standard deviation, minimum, maximum, selected quantiles, count, and sum) over historical delay or pace values. Together, these operations yield 1,683 features per elementary segment, spanning both local segment-level patterns and neighbourhood-scale trends.

The implementation of the aggregation framework, including the exact grouping definitions and a list of statistical functions, is described in Section 4.3.2. Summary statistics on feature counts and the data volume after feature engineering are reported in Section 5.1.1.

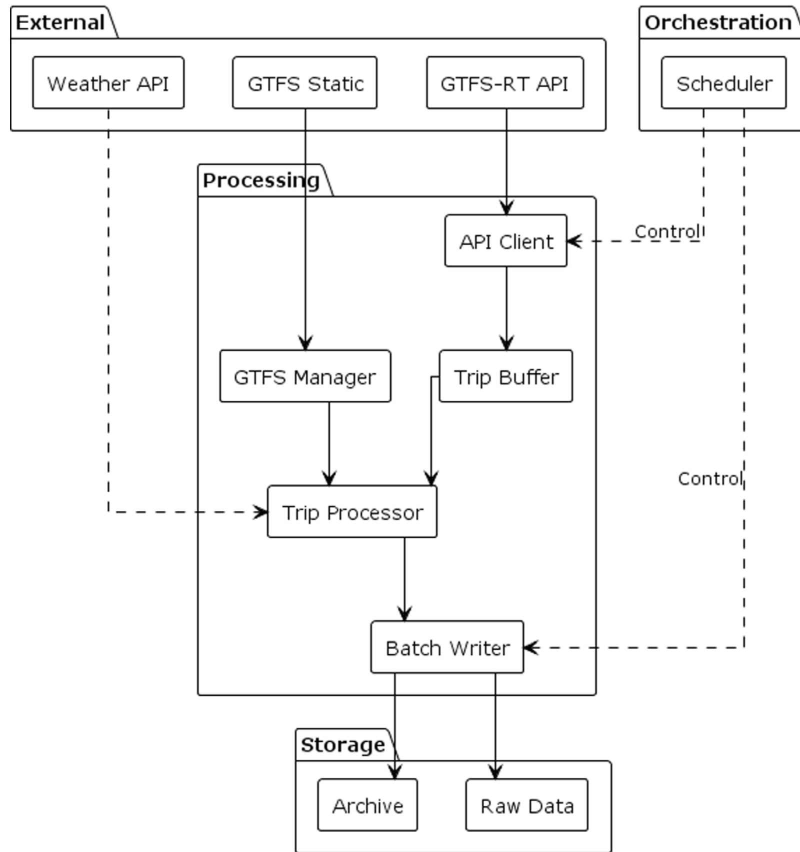


Fig. 2. Data Collection Pipeline: external APIs to storage.

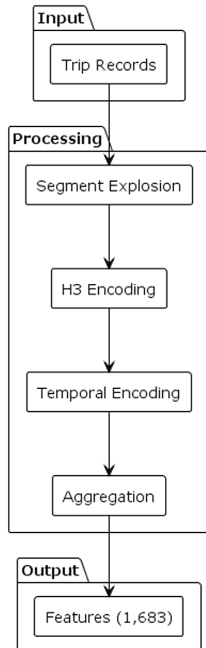


Fig. 3. Feature Engineering Pipeline: trip records to feature matrix.

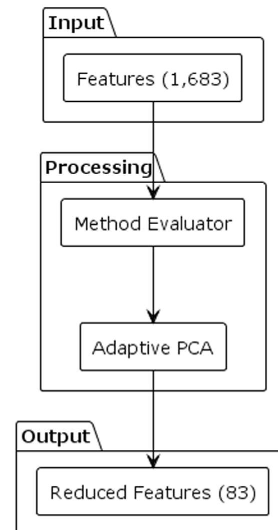


Fig. 4. Dimensionality Reduction: 1,683 features to 83 components (Adaptive PCA).

3.3. Dimensionality reduction pipeline

The dimensionality reduction pipeline maps the 1,683 engineered features (Section 3.2) to a lower-dimensional representation shared

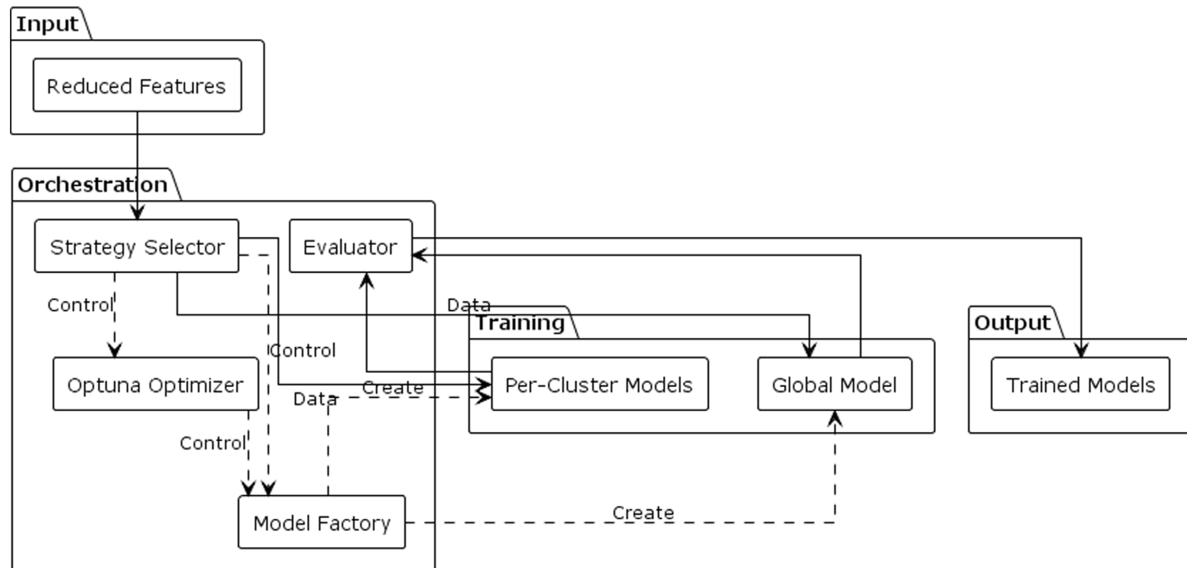


Fig. 5. Predictive Modelling Pipeline: global model training.

by all models (Fig. 4). Reducing the feature space is necessary to keep training and inference times manageable while preserving the information needed for accurate predictions.

Instead of adopting a single method by default, we evaluated a set of 20 dimensionality reduction techniques, following the comparative protocol of Sadegh-Zadeh et al. [36]. The candidate methods include linear techniques (PCA and its variants), supervised projections (e.g. LDA), nonlinear manifold methods (e.g. UMAP), and two-stage compositions. For each method we measured (i) downstream prediction performance of a reference LSTM model, (ii) training time and memory usage, and (iii) the number of components required to reach a given level of explained variance. This evaluation focuses on methods that can be trained on our full-scale feature matrix and applied efficiently at inference time.

Based on these results, we selected Adaptive PCA as the default dimensionality-reduction method for the remainder of the paper. In our setting, it compresses the 1,683 features to 83 components while retaining 95 % of the variance and achieving the best overall trade-off between accuracy and computational cost. All subsequent experiments use these 83 components as inputs to the predictive models. Detailed evaluation results and ablation studies for the candidate methods are presented in Section 4.4.2 and Section 5.1.

3.4. Predictive modelling pipeline

The predictive modelling pipeline trains delay-prediction models on the dimensionally reduced feature matrix produced in Section 3.3. The architecture separates orchestration (strategy selection, hyperparameter optimization, model evaluation) from processing (data partitioning, model training). We adopt a **global modelling strategy**, defined as training a single model across all routes in the network (as opposed to route- or cluster-specific models). In this approach, a single network-wide model is trained on the complete dataset, using the 83 Adaptive PCA components together with additional categorical features such as cluster identifiers. Dimensionality reduction makes training computationally less demanding rather than merely feasible: reducing from 1,683 features to 83 components significantly reduces memory requirements and training time, making it practical to train deep learning models on the full dataset. The orchestration layer manages the Optuna hyperparameter optimization loop (10 trials per architecture) and coordinates parallel training when using per-cluster strategies. Cluster IDs allow the model to learn cluster-specific delay patterns while keeping a unified architecture. This approach avoids the operational overhead of maintaining separate models per cluster and is compatible with recent practices in large-scale traffic forecasting [37,38]. The overall structure of the pipeline is illustrated in Fig. 5.

We evaluate five model architectures, described in Section 3.8: LSTM, XGBoost, xLSTM, PatchTST, and Autoformer. These architectures were selected to represent three major families of time-series models: (1) XGBoost (gradient boosting trees) as a strong tree-based baseline for tabular data, (2) LSTM (recurrent networks) for temporal sequence modelling, and (3) PatchTST and Autoformer (transformer-based) for long-range dependency modelling. This selection covers different architectural paradigms, reflects proven effectiveness in the time-series prediction literature, offers computational diversity (CPU vs. GPU requirements), and addresses practical deployment considerations. All models share the same input representation and prediction target, namely, elementary-segment pace (seconds per metre), as defined in Section 3.2. Training uses a walk-forward temporal cross-validation (Section 3.9.1) to respect causality and to obtain realistic performance estimates for deployment. For each architecture, we report RMSE, MAE, and R^2 at the elementary-segment, segment, and trip level, as well as training time and inference latency. The comparative results of this evaluation are presented in Section 5.1.

3.5. Inference pipeline

The Inference Pipeline (Fig. 6) transforms model predictions into actionable delay estimates suitable for real-time operational use. The orchestration layer manages inference scheduling; the preprocessing and aggregation stages are designed to complete in under 100 ms, while total end-to-end latency depends on the model architecture (100 ms to 800 ms, as reported in Section 5.1). Converting predicted elementary segment pace into delay estimates is essential because pace predictions alone are not directly actionable for transit operations—operators and passengers need delay information relative to scheduled times. Hierarchical aggregation is necessary because transit operations require predictions at multiple granularities: elementary segment predictions enable stop-level adjustments, segment predictions support route-planning decisions, and trip-level predictions enable schedule-adherence monitoring and passenger information systems. The aggregation process exhibits error cancellation, in which random errors

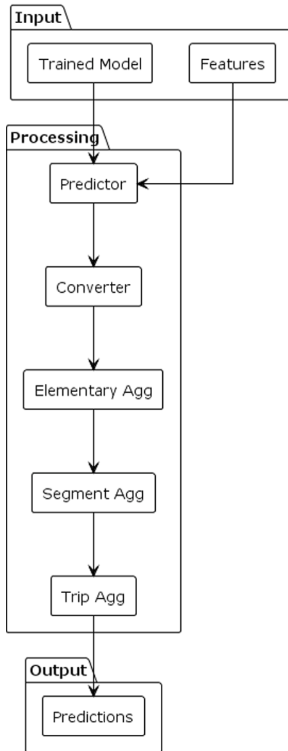


Fig. 6. Inference Pipeline: hierarchical aggregation (elementary → segment → trip).

at the elementary level partially cancel during aggregation, thereby improving R^2 at higher levels despite increasing absolute RMSE. Multi-level output is critical because different operational use cases require predictions at various aggregation levels, as justified in the evaluation protocol (Section 3.9.2). The implementation details of the inference pipeline, including pace-to-delay conversion, aggregation hierarchy, and performance optimizations, are described in Section 4.3.3. The resulting inference latency and error cancellation outcomes are presented in Section 5.1.

3.6. Hybrid H3+topology spatial clustering strategy

A critical challenge in scaling transit prediction is what we term the *"giant cluster" problem*: naive geographic partitioning can produce one or two large clusters that account for 60 % to 80 % of the data, while the peripheral areas remain data-sparse. This imbalance undermines distributed processing and leads to unstable models for regions with little data. For instance, in a naive H3 partitioning at resolution 8, downtown Montréal (a dense urban core) might fall into a single H3 hexagon containing 60% of all bus observations. At the same time, suburban routes are distributed across 20 smaller hexagons with sparse data. This imbalance causes: (1) the downtown cluster to dominate model training, drowning out suburban patterns, (2) inefficient parallel training (one worker handles 60% of data while others are underutilized), and (3) degraded model performance in suburban areas due to insufficient representation. To reduce this effect, we use a hybrid clustering strategy that combines H3-based spatial coverage with network topology information, as illustrated in Fig. 7.

For routes r_i and r_j, we define a combined similarity

$$\text{similarity}_{\text{combined}}(r_i, r_j) = w_{\text{spatial}}\, J_{\text{H3}}(r_i, r_j) + (1 - w_{\text{spatial}})\, J_{\text{segment}}(r_i, r_j), \quad (1)$$

where J_{H3} quantifies spatial overlap via H3 hexagons, J_{segment} measures the proportion of shared route segments, and $w_{\text{spatial}} \in [0, 1]$ balances the two contributions. To compute J_{H3}, H3 hexagons are assigned to each route by: (1) computing the H3 index for each GPS observation (vehicle position) along the route, (2) collecting all unique H3 indices that intersect with the route's trajectory, and (3) using the set of these H3 indices as the route's spatial footprint. The intuition is that routes which operate in similar areas and share many segments should be clustered together so that each cluster corresponds to a coherent group of routes with similar operating conditions.

The clustering procedure follows three stages. First, we establish geographic coherence by assigning all observations to H3 hexagons and grouping routes by their spatial footprints. We evaluate several candidate H3 resolutions (6, 7, and 8); resolution 7 (hexagons of roughly $5\,\text{km}^2$) offers a good compromise between spatial detail and data volume. This initial step reveals large clusters in which dense urban cores dominate the distribution.

Second, we refine the large clusters using topology-aware analysis. For each giant cluster, we construct a route-connectivity graph in which edges represent J_{segment} similarity, and we apply community detection to partition the cluster into subgroups that preserve route relationships while improving balance.

Third, we perform a simple rebalancing step: clusters that are too small are merged with neighbouring clusters with similar spatial and topological profiles, whereas clusters that remain too large are recursively subdivided. The resulting configuration achieves substantially more uniform cluster sizes while maintaining spatial and topological coherence. Algorithmic details of the three stages are given in Section 4.2.2, and quantitative results on cluster balance and prediction performance are reported in Section 5.1.2.

3.7. Hybrid parallelization strategy

The system combines process-based parallelization (CPU-bound tasks: model training with isolated memory) and thread-based concurrency (I/O operations: data loading, model serialization). Memory optimization via lazy loading, selective column reading, and feature streaming ensures scalability. Fault tolerance through checkpoint-based recovery and idempotent operations enables reliable distributed training. Implementation details in Section 4.1.

3.8. Predictive models and architectures

To compare different modelling approaches within the proposed framework, we evaluate five architectures representing three major families of time-series models: XGBoost (gradient-boosting trees), LSTM (recurrent network), xLSTM (extended LSTM), PatchTST (patch-based transformer), and Autoformer (autocorrelation transformer). This selection allows us to contrast tree-based, recurrent, and transformer-based methods under a common preprocessing and evaluation setup.

XGBoost is included as a strong tree-based baseline. It has been widely used for structured spatiotemporal tabular data, where categorical identifiers (e.g., routes, time periods) interact with continuous statistics, such as delays or pace [39]. Tree-based models naturally handle mixed feature types and can capture nonlinear interactions without additional feature engineering, which is attractive for the feature-rich setting considered here.

LSTM architectures are used to capture sequential temporal dependencies in delay patterns along a route. Recurrent networks are well-suited to modelling stop-to-stop delay propagation, where conditions at one stop influence the next stops in the sequence. Compared with transformer-based models, LSTMs typically require fewer parameters for a given input size, which makes them a natural candidate for our compressed feature space (Section 5.1.1).

xLSTM extends standard LSTM cells with additional gating and memory mechanisms designed to represent longer-range temporal relationships. We include xLSTM to test whether these extensions help

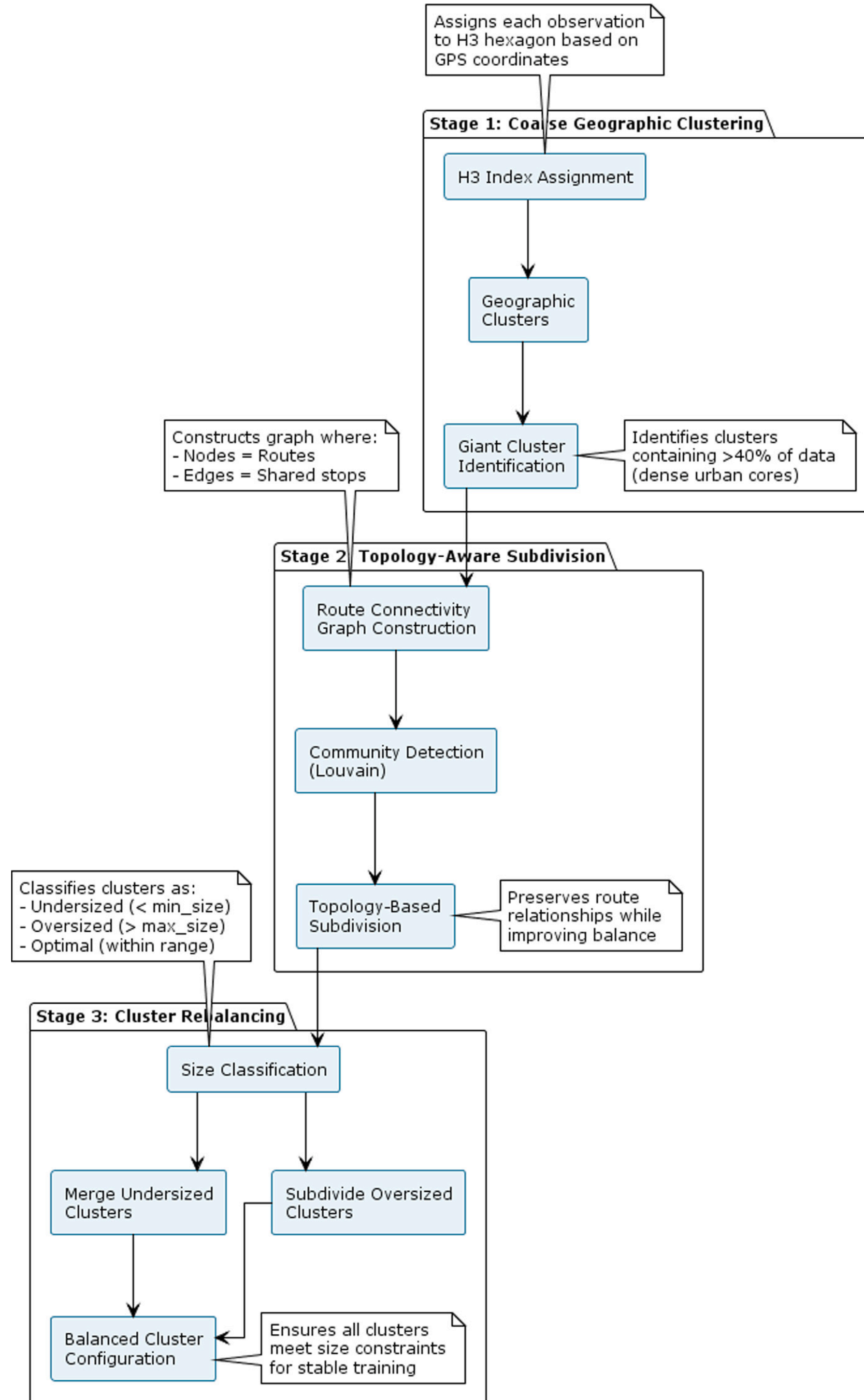


Fig. 7. Three-stage hybrid clustering process: (1) Stage 1: Coarse Geographic Clustering using H3 indexing, (2) Stage 2: Topology-Aware Giant Cluster Subdivision, and (3) Stage 3: Cluster Rebalancing and Optimization.

capture delay propagation across extended time windows, such as interactions between morning peak traffic and afternoon operations.

PatchTST and Autoformer are transformer-based architectures that have shown competitive performance on time-series forecasting benchmarks. PatchTST groups input sequences into patches before applying self-attention, which reduces computational cost while preserving local temporal structure. Autoformer replaces standard attention with an autocorrelation mechanism coupled with series decomposition, thereby modelling seasonal and periodic patterns more explicitly. Evaluating these architectures allows us to assess the benefits and costs of transformer-style models for large-scale bus delay prediction.

All models are trained globally on the complete network dataset using the same input representation (Adaptive PCA components and auxiliary features) and the same training protocol, including walk-forward

temporal validation (Section 3.9.1). This common setup ensures a fair comparison across architectures. Quantitative results and a discussion of accuracy–efficiency trade-offs are presented in Section 5.1.

3.8.1. Hyperparameter optimization strategy

Production deployment requires balancing multiple objectives beyond prediction accuracy alone. Inference latency and memory footprint constrain real-world deployment, particularly for real-time transit systems where sub-second response times are essential. We employ Bayesian optimization with a composite objective function that balances accuracy (70 %), latency (20 %), and memory footprint (10 %), reflecting practical deployment priorities where accuracy matters most but operational constraints cannot be ignored [37,38]. This multi-objective approach ensures that models meeting accuracy targets also satisfy operational constraints essential for production deployment, following recent best practices in deep learning optimization [40]. This optimization strategy is applied during model development to select optimal hyperparameters for each architecture, enabling fair comparison while ensuring production viability.

3.9. Evaluation protocol

The evaluation protocol is designed to provide realistic performance estimates for deployment and to support fair comparisons between models and clustering strategies. It combines walk-forward temporal cross-validation, multi-level evaluation at different aggregation scales, feature-importance analysis, and statistical testing. Implementation details are given in Sections 4.4.4–4.4.6.

3.9.1. Walk-forward temporal cross-validation

Standard k-fold cross-validation is not appropriate for time-series data because it randomly splits data, allowing future observations to appear in training sets when predicting past observations, violating temporal causality. This creates information leakage: a model trained on data from 'next week' could exploit patterns that are unavailable when predicting 'this week', thereby artificially inflating performance. In production, models only have access to historical data, so validation must respect this constraint. Because standard k-fold cross-validation violates temporal causality by allowing future information to leak into training sets, and because transit delay patterns exhibit temporal dependencies that require chronological ordering, we use walk-forward temporal cross-validation. Walk-forward validation: (1) respects temporal ordering by using only past data to predict future data, (2) simulates real-world deployment where models are trained on historical data and applied to future observations, (3) provides realistic performance estimates that reflect production conditions, and (4) enables detection of temporal drift (performance degradation over time) which k-fold cannot capture. The dataset described in Section 5.1.1 is split into five sequential folds, each corresponding to a contiguous time period. For each fold, models are trained on historical data and evaluated on a subsequent validation and test window, with an explicit temporal gap between training and testing to reduce leakage. This rolling-window design preserves chronological order and yields performance estimates consistent with the intended deployment scenario. The detailed procedure for computing time boundaries, performing data splits, and enforcing temporal gaps is described in Section 4.4.4.

3.9.2. Multi-level evaluation strategy

Transit operators use predictions at several levels of aggregation, from local segments to complete trips. We therefore report metrics at three levels: elementary segment, segment, and trip (see Section 3.5 for the corresponding operational use cases). Evaluating all three provides complementary information: elementary metrics reflect local accuracy, segment metrics capture line-level behaviour, and trip metrics reveal how errors accumulate or cancel over the entire journey. The latter is significant for passenger-facing applications, where trip-level delay is the primary quantity of interest.

3.9.3. Feature importance and model comparison

To assess whether the feature engineering framework captures meaningful spatiotemporal patterns, we perform model-agnostic feature-importance analysis using SHAP (Shapley Additive exPlanations). SHAP values are computed for the best-performing models and aggregated to obtain (i) importance scores for individual features and (ii) group-level scores for feature families such as H3 resolutions, route identifiers, and temporal encodings. We also conduct progressive feature ablation experiments to examine how performance changes as feature subsets are removed. Stability of importance rankings across cross-validation folds is used as an additional robustness check. The computational procedure is detailed in Algorithm 5 in Section 4.4.5.

To compare models and clustering strategies, we train all combinations under the same walk-forward protocol and apply statistical tests to their performance metrics. After checking the distributional assumptions, we use appropriate parametric or nonparametric tests for multiple comparisons. We apply corrections to control the family-wise error rate. The whole procedure for model training, metric aggregation, and hypothesis testing is summarised in Section 4.4.6.

3.10. Methodological limitations and assumptions

This section discusses the limitations of our approach and the assumptions underlying our methodology.

Global modelling trade-offs. The global modelling approach, in which a single model is trained on all routes, balances network-wide generalization with local pattern capture through cluster-aware features. However, this approach may dilute highly localized patterns that are specific to individual routes or atypical operational contexts. The model may achieve lower accuracy in low-density regions or in specialized transit corridors with unique operational characteristics than route-specific models. The hybrid clustering strategy mitigates these concerns by creating balanced clusters that preserve spatial and topological coherence, enabling the global model to learn cluster-specific patterns through categorical cluster identifiers while maintaining a unified architecture.

Dimensionality reduction challenges. The feature engineering pipeline yields a high-dimensional input space (1,683 features), necessitating dimensionality reduction. While Adaptive PCA preserves 95% of variance while reducing the dimensionality to 83 components, it may discard subtle but informative patterns not captured by the principal components. We address this challenge through: (1) systematic evaluation of 20 dimensionality reduction methods to select the optimal approach, (2) validation that critical patterns are preserved through downstream model performance, and (3) retention of cluster IDs and other categorical features that capture domain-specific information not captured by PCA.

Implementation complexity. The system's complexity, including hybrid clustering, parallelization, and multi-level inference, could pose implementation and maintenance challenges for smaller transit agencies with limited technical resources. We mitigate these concerns through: (1) modular pipeline design that enables selective adoption of components (agencies can use feature engineering without clustering, or clustering without distributed processing), (2) configuration-based customization that reduces code complexity (most parameters are specified in configuration files rather than hard-coded), (3) comprehensive documentation and reusable components that facilitate deployment, and (4) scalability considerations that allow the system to operate effectively at different scales (from single-machine to distributed clusters).

Sensitivity to hyperparameters and temporal drift. The system's performance depends on the selection of hyperparameters (clustering parameters, model architectures, and dimensionality reduction settings). While we employ systematic optimization procedures (grid search, Bayesian optimization), optimal configurations may vary across different transit networks or over time as operational patterns evolve. Additionally, the model may experience temporal drift as transit patterns change due to network modifications, service adjustments, or external factors (e.g., urban development, policy changes). We address these concerns through: (1) systematic hyperparameter optimization with cross-validation, (2) modular design that enables periodic retraining and configuration updates, and (3) monitoring capabilities that can detect performance degradation over time.

4. Implementation details

4.1. Technology stack

Python implementation with Apache Parquet storage (5:1 compression, columnar layout for 1,600+ features), hierarchical partitioning by `route_id/date` (99 % storage overhead reduction). Hybrid parallelization: `ProcessPoolExecutor` for CPU tasks ($N_{\text{workers}} = N_{\text{cores}} - 1$), `ThreadPoolExecutor` for I/O. Memory optimization via lazy loading and Apache Arrow memory-mapping. H3 geospatial indexing (`H3-py`).

4.2. Clustering implementation details

This subsection details the specific parameter configurations, optimization process, and resulting cluster characteristics that enable efficient spatial partitioning of the transit network.

4.2.1. Optimal clustering configuration

The final clustering configuration was determined through an extensive hyperparameter search that combined a systematic grid search and Bayesian optimization. The grid search explored over 80 configurations spanning H3 resolutions (6, 7, 8), cluster counts (8, 10, 12, 15, 20), spatial weights (0.3, 0.5, 0.7), and linkage methods (Ward, Complete, Average). For each configuration, we computed cluster quality metrics including coefficient of variation (CV), imbalance ratio, silhouette score, and Davies–Bouldin index.

Following the grid search, Bayesian optimization with Optuna refined the continuous parameters (spatial weight, distance threshold) using 50 additional trials. The optimization objective balances cluster variance (minimizing CV) with spatial coherence (maximizing silhouette score) and topological similarity (minimizing inter-cluster route overlap). This multi-objective optimization employed a weighted scalarization approach with empirically tuned weights: 0.5 for CV, 0.3 for silhouette, and 0.2 for topological coherence. The optimization procedure evaluates each configuration by computing cluster-quality metrics and selecting the one that maximizes the composite objective function. The resulting optimal configuration and performance metrics are presented in Section 5.1.2.

4.2.2. Three-stage clustering algorithm implementation

The three-stage clustering methodology (Section 3.6) is implemented as follows. These algorithms translate the conceptual framework into executable steps that systematically resolve the "giant cluster problem". The H3 resolution parameter r is determined through systematic grid search (Section 4.2.1), with resolution 7 identified as optimal for our network. The algorithms below use this optimal resolution value.

Stage 1: Coarse geographic clustering. The initial stage establishes geographic coherence using H3 resolution r (optimal value: $r = 7$, $\sim 5\,\text{km}^2$ hexagons). This algorithm implements the first stage of the hybrid clustering strategy described in Section 3.6. It assigns each transit observation to an H3 hexagon based on its GPS coordinates, groups observations by their H3 indices to form initial geographic clusters. It identifies giant clusters that contain a disproportionate share of the data (exceeding 40% of total observations). The algorithm returns both the geographic cluster assignments and a list of giant clusters that require further subdivision in Stage 2.

Algorithm 1 Stage 1: Coarse Geographic Clustering

Require: Transit observations with GPS coordinates (lat, lon); H3 resolution r (recommended: $r = 7$)
Ensure: Geographic cluster assignments $\text{cluster}_{\text{geo}}$; giant cluster list $\text{giant}_{\text{clusters}}$
1: **H3 Index Assignment**
2: **for** each observation obs_i with coordinates $(\text{lat}_i, \text{lon}_i)$ **do**
3: Compute H3 index: $h3_i \leftarrow \text{geo_to_h3}(\text{lat}_i, \text{lon}_i, r)$
4: $\text{cluster}_{\text{geo}}[\text{obs}_i] \leftarrow h3_i$
5: **end for**
6: **Cluster Validation**
7: Let $\mathcal{H} \leftarrow \text{unique}(\{h3_i\})$
8: **for** each $h \in \mathcal{H}$ **do**
9: $\text{count}_h \leftarrow |\{\text{obs}_i : \text{cluster}_{\text{geo}}[\text{obs}_i] = h\}|$
10: $\text{bounds}_h \leftarrow \text{h3_boundary}(h)$
11: $\text{density}_h \leftarrow \text{count}_h / \text{area}(\text{bounds}_h)$
12: **end for**
13: **Giant Cluster Identification**
14: $\text{total}_{\text{obs}} \leftarrow \sum_{h \in \mathcal{H}} \text{count}_h$
15: $\text{threshold}_{\text{giant}} \leftarrow 0.4$
16: $\text{giant}_{\text{clusters}} \leftarrow \{h \in \mathcal{H} : \text{count}_h > \text{threshold}_{\text{giant}} \cdot \text{total}_{\text{obs}}\}$
17: **return** $\text{cluster}_{\text{geo}}, \text{giant}_{\text{clusters}}$

Stage 2: Topology-aware giant cluster subdivision. Giant clusters are subdivided based on network topology analysis, preserving route connectivity while achieving balanced data distribution. This algorithm implements the second stage of the hybrid clustering strategy, addressing the giant cluster problem identified in Stage 1. For each giant cluster, it constructs a route-connectivity graph in which nodes represent routes and edges connect routes that share at least one stop. The algorithm then applies a community detection method (e.g., the Louvain algorithm) to partition the graph into topologically coherent subgroups. The subdivision assignment proceeds by mapping each observation to its route's community assignment, ensuring that observations from routes in the same community are grouped. This process preserves route relationships while improving cluster balance.

Algorithm 2 Stage 2: Topology-Aware Subdivision (Summarised)

Require: Giant clusters C_{giant}; route network topology
Ensure: Topology-based cluster assignments $\text{cluster}_{\text{topo}}$
1: **for** each cluster $C \in C_{\text{giant}}$ **do**
2: Build connectivity graph $G = (V, E)$ where:
3: V is the set of routes in C and E connects routes sharing at least one stop
4: Partition G into communities by optimizing modularity (e.g., Louvain algorithm)
5: Validate each community for spatial compactness and connectivity
6: Merge communities that fail validation criteria
7: **Subdivision Assignment**
8: **for** each observation $\text{obs} \in C$ **do**
9: $\text{cluster}_{\text{topo}}[\text{obs}] \leftarrow \text{community}(\text{get_route}(\text{obs}))$
10: **end for**
11: **end for**
12: **return** $\text{cluster}_{\text{topo}}$

Stage 3: Cluster rebalancing and optimization. The final stage ensures all clusters meet minimum data requirements for stable model training while maintaining spatial and topological coherence. The size constraints below are implementation parameters chosen to balance computational feasibility with sufficient training data density. This algorithm implements the third and final stage of the hybrid clustering strategy, refining the clusters produced by Stages 1 and 2. It classifies clusters as undersized, oversized, or optimal based on predefined thresholds. Undersized clusters are merged with their nearest topologically compatible neighbours, while oversized clusters are recursively subdivided (e.g., using finer H3 resolutions) until all clusters satisfy size constraints. The algorithm performs final validation to ensure all clusters meet both size and spatial coherence requirements, producing the balanced cluster configuration used for feature engineering and model training.

Algorithm 3 Stage 3: Cluster Rebalancing (Summarised)

Require: Subdivided clusters from Stage 2
Ensure: Balanced cluster assignments $\text{cluster}_{\text{balanced}}$
1: Classify each cluster as undersized, oversized, or optimal using thresholds min_size and max_size
2: **Rebalance Undersized Clusters**
3: **for** each undersized cluster c_{under} **do**
4: Merge c_{under} with its nearest topologically compatible neighbour
5: Ensure $\text{size}(c_{\text{merged}}) \leq max_size$
6: **end for**
7: **Rebalance Oversized Clusters**
8: **for** each oversized cluster c_{over} **do**
9: Recursively subdivide c_{over} (e.g., using finer H3 resolution)
10: Continue until each resulting cluster c satisfies $min_size \leq \text{size}(c) \leq max_size$
11: **end for**
12: Perform final validation to ensure all clusters meet size and spatial coherence constraints
13: **return** $\text{cluster}_{\text{balanced}}$

4.3. Data processing pipelines implementation

This subsection details the practical implementation of the three core data processing pipelines: collection, feature engineering, and dimensionality reduction. Each pipeline presented unique engineering challenges that required careful optimization to achieve production-grade performance and reliability.

4.3.1. Collection pipeline details

The data collection pipeline implements a robust, fault-tolerant system for continuously ingesting real-time GTFS-RT feeds. A custom-built collector tool manages API requests, automatically rotating keys to handle rate limiting and ensure uninterrupted data flow. The collector maintains multiple API keys in a rotation pool and automatically switches to the next key when rate limits are reached, ensuring continuous collection without manual intervention.

Data is collected at 10 s intervals, balancing temporal resolution and API load. Each collection cycle fetches vehicle position updates, trip updates, and service alert data, which are stored as Protocol Buffer binary files to minimize storage overhead and preserve the original data structure for future reprocessing.

The trip integrity buffer is a critical component that ensures the completeness of trip data before processing. Since GTFS-RT feeds provide incremental updates, a trip may span multiple collection cycles before completion. The buffer maintains an in-memory state for active trips, accumulating updates until a completion signal is detected. Trip completion is inferred through timeout-based heuristics: if no updates are received for a trip within 1 h, it is considered complete and flushed to persistent storage. This approach handles edge cases such as trips that terminate early or vehicles that go offline without explicit completion messages.

Trip-level processing transforms raw vehicle positions into structured trip records suitable for feature engineering. The processing pipeline aggregates vehicle position updates into trip-level records, reducing data volume while preserving essential information. Temporal alignment uses 2 min windows to match real-time updates to scheduled time points, ensuring accurate delay calculation. Weather data from city-wide airport stations is integrated at the daily level, enriching trip records with contextual information. The delay calculation uses 3-sigma outlier filtering to remove measurement artifacts, ensuring data quality. Segments–defined as consecutive stop pairs $(stop_i, stop_{i+1})$–use 100 m geofences for precise travel time detection. Quality validation evaluates completeness, consistency, and plausibility, assigns composite scores, and flags observations below the threshold for manual review.

After trip-level processing, the archival system implements a simple yet effective file-based state management strategy. Processed files are moved from raw storage to archive storage, serving dual purposes: freeing up working directory space for new data and providing a precise checkpoint mechanism for incremental processing. The system can recover from failures by identifying which raw files have not yet been archived and reprocessing only incomplete work, avoiding redundant computation.

4.3.2. Feature engineering implementation

The feature engineering pipeline transforms trip-level observations into a comprehensive set of spatiotemporal features through systematic aggregation and vectorized computation. The implementation prioritizes memory efficiency and computational throughput, leveraging code optimizations, caching and parallel processing to handle datasets with millions of records.

At the core of the pipeline is the systematic aggregation-combination framework, which computes 23 distinct grouping strategies across the spatial, temporal, and route dimensions. Each combination applies 9 statistical aggregation functions (mean, standard deviation, minimum, maximum, 25th, 50th, and 75th percentiles, count, and sum) to historical delay values, producing rich representations of delay patterns at different scales. The groupby operations are parallelized across clusters using process-based workers, with each worker handling a subset of routes to balance computational load.

A key idea is the elementary segment explosion strategy, which expands each trip observation into multiple segment-level records representing the constituent stop-to-stop movements. This expansion substantially increases data volume while enabling segment-specific feature computation, thereby significantly improving prediction granularity. To mitigate memory impact, segment explosion is performed in a streaming fashion: data is processed in route-level chunks, with intermediate results written to disk immediately after computation rather than being accumulated in memory.

The per-hour boolean interval encoding uses a vectorized approach to representing temporal patterns. For each of the 24 hours in a day, six boolean columns indicate whether the current trip falls into specific time intervals (early morning, morning rush, midday, afternoon rush, evening, night). This encoding produces 144 boolean columns that capture fine-grained temporal patterns while maintaining numerical stability for machine learning models. The implementation uses broadcasting to compute all 144 columns in a single vectorized operation, achieving a significant speedup over iterative column creation.

The output of feature engineering is a comprehensive feature dataset stored in partitioned Parquet format. Feature metadata is embedded in the Parquet schema, including feature-group annotations (spatial, temporal, and statistical) that enable selective loading during model training. This organization reduces memory consumption during processing by 60 % compared to loading the full feature matrix.

4.3.3. Inference pipeline implementation

The inference pipeline implements a hierarchical aggregation chain that transforms predictions of elementary segment pace into actionable delay estimates at multiple aggregation levels. The implementation begins with pace-to-delay conversion, transforming predicted pace values into delay estimates using the formula:

$$\text{Delay} = \left(\frac{\text{Distance}}{\text{Predicted Pace}}\right) - \left(\frac{\text{Distance}}{\text{Observed Pace}}\right), \tag{2}$$

where distance is the segment length, predicted pace is the model output, and observed pace is the scheduled pace. This conversion enables direct comparison with scheduled arrival times, making predictions actionable for transit operations.

The aggregation hierarchy implements a multi-level aggregation chain: elementary segment delays aggregate through **Elementary segment** → **Segment** (sum of elementary delays) → **Trip** (sum of segment delays) → **Schedule Adherence** (trip cumulative segment delays compared to scheduled arrival times at stops). The rationale for multi-level aggregation and operational use cases at each level is described in Section 3.5.

Performance optimization implements a staged, highly-efficient inference process to achieve low end-to-end latency suitable for real-time applications. The pipeline begins by applying precomputed, cached aggregations and a minimal feature subset to the input data, thereby reducing preprocessing overhead. The core model inference uses highly vectorized operations for batch prediction, enabling efficient parallel processing of multiple predictions. Additional optimizations include feature pre-computation to eliminate redundant calculations, model compression to reduce memory footprint, and vectorized batch prediction to maximize throughput. The resulting inference latency and error cancellation outcomes are presented in Section 5.1.

4.4. Predictive modelling

This section details the implementation of model architectures, dimensionality reduction, and hyperparameter optimization procedures.

4.4.1. Model architecture implementations

XGBoost implementation. XGBoost (Extreme Gradient Boosting) is a highly optimized gradient-boosting framework designed for speed and performance on tabular data. The architecture sequentially builds an ensemble of decision trees, with each new tree trained to correct the residual errors of the preceding ensemble. This additive model formulation can be expressed as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i), \tag{3}$$

where $\hat{y}_i^{(t)}$ is the prediction after t iterations, and f_t is the tth decision tree mapping input features x_i to predicted residuals. The objective function at iteration t combines prediction error and regularization:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k), \tag{4}$$

where l is the loss function (squared error for regression), and Ω is the regularization term penalizing model complexity. XGBoost's tree-based structure naturally handles mixed feature types (continuous delay statistics, categorical temporal indicators, discrete route identifiers) without requiring extensive preprocessing. The histogram-based tree construction algorithm reduces complexity from $O(n \cdot d \cdot \log(n))$ to $O(n \cdot d + d \cdot \log(d))$, where n is the sample count and d is the feature dimensionality. Training implements early stopping (after 50 consecutive rounds without improvement), feature caching for rapid histogram construction, and parallel tree construction across features.

LSTM implementation. The LSTM architecture employs a bidirectional encoder-attention-decoder design for sequence-based delay prediction, modelling temporal dependencies in historical delay patterns along route trajectories. The bidirectional architecture captures both forward and backward contexts within the historical observation window, enabling the model to integrate patterns from earlier and more recent observations already recorded. The input sequence consists entirely of historical records available at prediction time, making the architecture fully compatible with real-time deployment. The embedding layer projects features into a dense representation, while stacked Bidirectional LSTM layers process sequences in both forward and backward directions. Bahdanau's attention computes context-weighted representations, and a set of dense layers with dropout produces the final delay predictions. Training combines MSE with a temporal smoothness penalty ($\lambda_{\text{smooth}} = 0.01$) to encourage realistic delay trajectories, using Adam optimization with a learning rate schedule. The temporal smoothness penalty penalizes large changes in predictions between consecutive time steps, reducing noise and improving forecast stability. Dropout layers provide regularization by randomly deactivating units during training, preventing co-adaptation and improving generalization.

xLSTM implementation. The Extended LSTM (xLSTM) architecture incorporates recent advances in LSTM design, including exponential gating mechanisms and enhanced memory structures. Unlike standard LSTMs, xLSTM employs matrix memory (mLSTM) with configurable matrix size, enabling more expressive memory representations. Exponential gating improves gradient flow and long-term dependency modelling, making it particularly effective at capturing extended temporal patterns, such as morning delays that affect afternoon operations. Training uses the Adam optimizer with early stopping on the validation loss and conservative gradient clipping to prevent gradient explosions.

PatchTST implementation. PatchTST (Patch-based Time Series Transformer) employs a patch-based strategy that divides time series into overlapping patches to reduce computational complexity while preserving temporal relationships. The architecture uses channel-independent processing, treating each feature independently to avoid cross-channel noise. Multi-head self-attention captures long-range dependencies efficiently through patch-level interactions. The model employs learned positional encodings and transformer encoder layers to process patch embeddings, enabling effective modelling of temporal patterns at multiple scales. The embedding dimension is determined by $d_{\text{model}} = n_{\text{heads}} \times d_{\text{model_multiplier}}$, where n_{heads} is the number of attention heads and $d_{\text{model_multiplier}}$ scales the dimension per head.

Autoformer implementation. Autoformer replaces standard self-attention with an autocorrelation mechanism that uses the Fast Fourier Transform (FFT) to capture temporal dependencies efficiently. The architecture employs series-decomposition blocks that separate input time series into trend and seasonal components via moving-average filtering. This decomposition enables the model to capture both long-term trends and periodic patterns separately, improving forecasting accuracy for time series with strong seasonal components. The autocorrelation mechanism aggregates information from similar temporal phases, providing an efficient alternative to attention-based mechanisms. The architecture uses autocorrelation heads with embedding dimensions $d_{\text{model}} = n_{\text{heads}} \times d_{\text{model_multiplier}}$.

4.4.2. Dimensionality reduction implementation

The dimensionality reduction pipeline evaluates 20 distinct methods using a systematic, reproducible framework. Seven base methods form the foundation: Standard PCA, Adaptive PCA, Supervised PCA, Hybrid PCA, Sparse PCA, Linear Discriminant Analysis (LDA), and UMAP. These methods represent diverse reduction strategies, ranging from unsupervised linear transformations (Standard PCA) to supervised nonlinear embeddings (UMAP), ensuring comprehensive coverage of the dimensionality-reduction landscape.

Beyond base methods, the pipeline evaluates 13 compositional approaches that combine multiple reduction techniques in sequential or

parallel arrangements. Sequential composition applies one reduction method, then another, enabling staged reduction strategies such as PCA for noise removal followed by LDA for class separation. Parallel composition applies multiple methods independently and concatenates their outputs, creating hybrid feature spaces that capture complementary aspects of the data structure.

The evaluation protocol implements walk-forward temporal cross-validation with careful attention to chronological ordering. Data is split into training and test sets using temporal cutoffs to prevent information leakage from the future to the past. Each method is trained on the training set, evaluated on the test set, and assessed by downstream model performance (RMSE, R^2) and computational efficiency (training time, memory usage). This comprehensive evaluation ensures that the selected method optimizes both prediction accuracy and operational feasibility.

The implementation configures Adaptive PCA to retain 95 % of the total variance threshold, enabling flexible dimensionality reduction based on variance retention criteria. The reduced feature set is cached in Parquet format with standardized column names, enabling seamless integration with downstream modelling pipelines. The method selection results are presented in Section 5.1.

4.4.3. Hyperparameter optimization procedure

We employ Bayesian optimization via Optuna's Tree-structured Parzen Estimator (TPE) to systematically explore the hyperparameter space for each architecture, following recent best practices in deep learning optimization [40]. The optimization procedure minimizes a composite objective function $\text{Objective}(\theta) = 0.7 \cdot \text{RMSE} + 0.2 \cdot \text{Latency} + 0.1 \cdot \text{Memory}$ (rationale for weight selection in Section 3.8.1). We conduct 10 trials per architecture to ensure fair comparison, using early stopping based on the validation loss to prevent overfitting.

Hyperparameter search spaces. Each architecture employs a tailored search space reflecting its computational characteristics:

XGBoost: max_depth $\in [3, 10]$, learning_rate $\in [10^{-4}, 0.3]$ (log scale), n_estimators $\in [50, 500]$, subsample $\in [0.6, 1.0]$, colsample_bytree $\in [0.6, 1.0]$, min_child_weight $\in [1, 10]$, reg_alpha $\in [10^{-6}, 10]$ (log scale), reg_lambda $\in [10^{-6}, 10]$ (log scale).

LSTM: units $\in [32, 256]$ (log scale), layers $\in [1, 3]$, dropout $\in [0.1, 0.5]$, learning_rate $\in [10^{-5}, 10^{-2}]$ (log scale), batch_size $\in \{16, 32, 64, 128\}$, clipnorm $\in [0.5, 2.5]$ (log scale).

xLSTM: units $\in [32, 256]$ (log scale), matrix_size $\in [4, 16]$, learning_rate $\in [10^{-5}, 10^{-2}]$ (log scale), batch_size $\in \{16, 32, 64\}$, clipnorm $\in [0.3, 1.5]$ (log scale). The conservative clipping range reflects xLSTM's sensitivity to gradient explosions.

PatchTST/Autoformer: $n_{\text{heads}} \in \{4, 8, 16\}$, $d_{\text{model_multiplier}} \in [8, 32]$ (log scale), $n_{\text{layers}} \in [2, 4]$, dropout $\in [0.1, 0.3]$, learning_rate $\in [10^{-5}, 10^{-2}]$ (log scale), batch_size $\in \{16, 32, 64\}$, clipnorm $\in [0.3, 1.5]$ (log scale).

Evaluation protocol implementation. Models are trained globally on the complete training set (80 % of 6-month data) and evaluated on the held-out test set (20 %) using three complementary metrics: RMSE (Root Mean Squared Error) as the primary metric penalizing significant errors, MAE (Mean Absolute Error) as a robustness metric less sensitive to outliers, and R^2 (Coefficient of Determination) as a variance explanation metric. Performance is evaluated at three granularity levels: elementary (stop-to-stop predictions), segment (aggregated route segments), and trip (end-to-end trip delays), enabling assessment of error cancellation through hierarchical aggregation.

4.4.4. Temporal cross-validation implementation

The walk-forward temporal cross-validation procedure (Section 3.9.1) is implemented using the following algorithm, which enforces chronological ordering and prevents temporal leakage. This algorithm implements the walk-forward validation strategy described in Section 3.9.1, creating temporally ordered data splits that respect causality. For each fold, it defines training, validation, and test windows as contiguous time periods, ensuring that training data always precedes validation and test data. A temporal gap between training and validation periods reduces potential leakage from edge effects. The algorithm operates on the training portion of the data (80 % of the 6-month dataset), with the remaining 20 % held out as a final test set for model evaluation. This ensures realistic performance assessment while maintaining a strict temporal split.

Algorithm 4 Walk-Forward Temporal Cross-Validation (Summarised)

Require: Time-series dataset $\mathcal{D}_{\text{train}}$; number of folds K
Ensure: K temporally ordered splits $\{(\text{Train}_k, \text{Valid}_k, \text{Test}_k)\}_{k=1}^{K}$
1: **for** fold $k = 1$ to K **do**
2: Define time boundaries for fold k:
3: Valid_k: next month-long block
4: Test_k: month following Valid_k
5: Train_k: fixed-length window (e.g., 6 month) preceding Valid_k
6: Enforce a temporal gap (e.g., 15 min) between the end of Train_k and the start of Valid_k
7: Generate split: $(\text{Train}_k, \text{Valid}_k, \text{Test}_k)$
8: **end for**
9: **return** $\{(\text{Train}_k, \text{Valid}_k, \text{Test}_k)\}_{k=1}^{K}$

4.4.5. Feature importance analysis implementation

The systematic feature importance analysis (Section 3.9.3) is implemented using the following algorithm, which computes SHAP values, performs feature group analysis, and conducts progressive ablation. This algorithm implements the feature importance analysis framework described in Section 3.9.3, providing systematic assessment of which features drive model performance. It ranks individual features by their importance scores (computed from mean absolute SHAP values), aggregates importance at the feature-group level (e.g., H3 resolutions, temporal encodings), and performs progressive ablation by retraining models with increasingly restricted feature sets to assess the impact of feature removal. The algorithm also assesses feature stability by computing the correlation of importance rankings across cross-validation folds, identifying features with consistently high importance.

Algorithm 5 Systematic Feature Importance Analysis (Summarised)

Require: Trained models; test data $\mathcal{D}_{\text{test}}$; feature groups $\mathcal{G}$
Ensure: Feature importance rankings, group contributions, ablation results, stability metrics
1: **1. Rank Individual Features**
2: **for** each model **do**
3: Compute feature importance scores using mean absolute SHAP values
4: Rank features by their importance
5: **end for**
6: **2. Analyze Feature Groups**
7: For each group $g \in \mathcal{G}$, aggregate the importance scores of its features
8: Determine the relative contribution of each group
9: **3. Perform Progressive Ablation**
10: **for** various thresholds k **do**
11: Select the top-k most important features
12: Retrain and evaluate the model using only these k features
13: **end for**
14: Plot the performance curve to evaluate the effect of feature removal
15: **4. Assess Feature Stability**
16: Compute the correlation of feature importance rankings across folds
17: Identify features with consistently high importance
18: **return** Feature rankings; group contributions; ablation performance curve; stability metrics

4.4.6. Model comparison implementation

The comprehensive model comparison protocol (Section 3.9.3) is implemented using the following algorithm, which performs systematic training, evaluation, and statistical significance testing. This algorithm implements the model comparison framework described in

Section 3.9.3, enabling fair and statistically rigorous comparison of different model architectures and clustering strategies. It systematically trains and evaluates all model-clustering combinations across cross-validation folds, collecting performance metrics as distributions. The algorithm then performs statistical significance testing for each metric and model pair, selecting the appropriate test (paired t-test or Wilcoxon) based on distributional assumptions and applying a Bonferroni correction to control the family-wise error rate. The algorithm returns a matrix of significant differences, effect sizes, and confidence intervals, providing quantitative evidence for model ranking.

Algorithm 6 Comprehensive Model Comparison Protocol (Summarised)

Require: Set of model families $\mathcal{M}$; clustering strategies $\mathcal{S}$
Ensure: Performance ranking matrix with statistical significance
1: **1. Systematic Training and Evaluation**
2: **for** each $(m, s) \in \mathcal{M} \times \mathcal{S}$ **do**
3: Train and evaluate m with clustering strategy s across all cross-validation folds
4: Collect performance metrics (e.g., RMSE, MAE) as distributions
5: **end for**
6: **2. Statistical Significance Testing**
7: **for** each metric **do**
8: **for** each model pair (m_i, m_j) **do**
9: Test for normality of performance distributions
10: Select appropriate test: paired t-test or Wilcoxon
11: Apply Bonferroni correction to p-values
12: Record if the difference is statistically significant
13: **end for**
14: **end for**
15: **return** Matrix of significant differences, effect sizes, and confidence intervals

5. Experimental results and discussion

5.1. Experimental results

This subsection presents comprehensive experimental validation of our transit delay prediction system. The clustering framework developed in Section 3.6 organizes the data and generates cluster-aware features that inform a single global model trained on all network observations. We compare architectures using identical preprocessing pipelines and a systematic hyperparameter optimization process, evaluating performance across three granularity levels (elementary, segment, trip) to identify the best architecture for production deployment.

5.1.1. Experimental setup and data organization

Dataset characteristics. Our evaluation dataset comprises operational data from September 15, 2024, to March 15, 2025 (6 months) from the Société de transport de Montréal (STM), which operates 196 bus routes across diverse urban contexts. After preprocessing and quality filtering, the dataset includes complete GTFS static schedules, GTFS-RT vehicle positions, weather data, and cluster assignments from the hybrid H3+Topology clustering. Trip-level processing (Section 3.1.1) aggregates raw vehicle positions into structured records, reducing data volume by approximately 80 % while preserving essential information. The systematic feature engineering framework (Section 3.2) generates 1,683 spatiotemporal features through exhaustive exploration of aggregation combinations. Elementary Segment Explosion (Section 4.3.2) disaggregates coarse segments into uniform sub-units, increasing resolution 10–15× to approximately 5.5 million observations, enabling fine-grained analysis while maintaining computational feasibility. Data is split chronologically into training (80 %) and test (20 %) sets, preserving temporal order.

Table 2
Optimal clustering configuration and performance metrics.

Parameter	Value
H3 Resolution	7 (~ 5 km^2)
Number of Clusters	12
Spatial Weight	0.5
Linkage Method	Ward
Distance Metric	1 − weighted Jaccard
Performance Metrics	
Coefficient of Variation	0.608
Imbalance Ratio	1.90×

Data organization via clustering. The hybrid H3+Topology clustering (Section 3.6) partitions the 196 routes into balanced clusters based on spatial and topological similarity. Cluster assignments generate cluster-specific features and add a cluster ID as a categorical variable to the global model, enabling the model to learn cluster-specific delay patterns without requiring separate model instances. This organization ensures efficient feature engineering and trains a single global model that generalizes across the entire network.

5.1.2. Clustering results

Systematic hyperparameter search (Section 4.2) identified the optimal clustering configuration through grid search over 80+ configurations and Bayesian optimization with 50 Optuna trials. The optimal configuration achieves balanced cluster partitioning that resolves the "giant cluster problem" while maintaining spatial-topological coherence (see Table 2).

The optimal configuration employs H3 resolution 7 hexagons ($\sim$5 km^2), providing coarse geographic partitioning that balances spatial granularity with cluster size. The choice of 12 clusters strikes a balance between fine-grained spatial specialization and sufficient training data per cluster. With 196 total routes in the network, this configuration yields an average of 16.3 routes per cluster, providing sufficient data density for robust model training while maintaining spatial locality. The spatial weight of 0.5 indicates equal importance assigned to geographic proximity (H3 Jaccard similarity) and topological structure (segment Jaccard similarity) in the weighted similarity metric (rationale in Section 3.6).

The resulting clustering achieves a coefficient of variation (CV) of 0.608, representing a 3.3× improvement over naive geographic partitioning, which produced a CV = 2.0. The imbalance ratio of 1.90× (ratio of the largest to the smallest cluster) demonstrates successful mitigation of the "giant cluster problem", compared to the naive baseline of 8×, a 4.2× improvement.

The final cluster assignment produces the following route distribution across the 12 clusters: [31, 29, 28, 23, 22, 17, 13, 12, 10, 9, 1, 1] routes per cluster. This distribution reveals three distinct cluster size categories: large urban core clusters (31, 29, 28 routes) serving downtown and high-density neighbourhoods, medium suburban clusters (23, 22, 17, 13, 12 routes) covering intermediate-density areas, and small peripheral clusters (10, 9, 1, 1 routes) representing specialized transit corridors or outlying regions. The two single-route clusters represent express specialized routes with unique spatial coverage that do not overlap significantly with other routes. While these clusters have limited training data, they benefit from the global feature engineering framework that incorporates neighbourhood-level aggregations, providing sufficient context for reliable predictions.

Each cluster is associated with a geographic footprint defined as the union of H3 cells covered by its constituent routes. Cluster footprints exhibit minimal overlap (average Jaccard overlap < 0.15), confirming spatial coherence and enabling independent parallel model training.

Preprocessing pipeline. All models share an identical preprocessing pipeline to ensure fair comparison: 1,683 raw features from systematic

Table 3
Optimal configurations and performance for the evaluated models.

(a) XGBoost

Parameter	Value
Max Depth	3
Number of Estimators	368
Learning Rate	0.0685
Subsample	0.892 (89.2 %)
Column Sample by Tree	0.909 (90.9 %)
Min Child Weight	1
L1 Regularization (α)	3.23×10^{-4}
L2 Regularization (λ)	6.47×10^{-6}
Performance	
Parameters	1,100,000 (tree nodes)
Training Time	1.5 min (CPU)
Inference Latency	100 ms

(b) LSTM

Parameter	Value
Layers	3
Units per Layer	32
Dropout	0.383
Learning Rate	1.54×10^{-3}
Batch Size	16
Gradient Clipping (clipnorm)	2.01
Performance	
Parameters	31,000
Training Time	26 min
Inference Latency	300 ms
GPU Memory	2.1 GB

(c) xLSTM

Parameter	Value
Units	Variable (32 to 256)
Matrix Size	4 to 16
Dropout	Variable (0.1 to 0.5)
Learning Rate	Variable (1×10^{-5} to 1×10^{-2})
Batch Size	Variable (16, 32, 64)
Gradient Clipping	Variable (0.3 to 1.5)
Performance	
Parameters	1,850,000
Training Time	31 min
Inference Latency	380 ms
GPU Memory	3.0 GB

(d) PatchTST

Parameter	Value
Attention Heads (n_{heads})	8
d_{model} Multiplier	12
Embedding Dimension (d_{model})	96
Encoder Layers	2
Dropout	0.237
Learning Rate	2.09×10^{-4}
Batch Size	32
Gradient Clipping	1.30
Performance	
Parameters	2,400,000
Training Time	35 min
Inference Latency	800 ms
GPU Memory	4.3 GB

(e) Autoformer

Parameter	Value
Autocorrelation Heads (n_{heads})	4
d_{model} Multiplier	10
Embedding Dimension (d_{model})	40
Encoder Layers	2
Dropout	0.161
Learning Rate	3.75×10^{-4}
Batch Size	64
Gradient Clipping	0.376
Performance	
Parameters	1,900,000
Training Time	41 min
Inference Latency	700 ms
GPU Memory	3.8 GB

multi-resolution framework (Section 3.2), removal of 18 elementary segment features that could leak target information, Adaptive PCA reducing features to 83 components (95 % variance explained), StandardScaler for features and RobustScaler for target, chronological split (80 % train, 20 % test) preserving temporal ordering, and cluster ID (1–12) added as categorical feature.

5.1.3. Optimal model configurations

Bayesian hyperparameter optimization (Section 4.4.3) identified optimal configurations for each architecture, as shown in Table 3 along with their corresponding performance. Optimal hyperparameters were selected based on held-out test performance, minimizing a composite objective function that balances accuracy, latency, and memory footprint.

Computational resources. All experiments are conducted on 3× NVIDIA GeForce RTX 2080 Ti GPUs (11 GB VRAM each) with TensorFlow 2.x and CUDA 11.8. XGBoost runs on CPU (no GPU required). Training uses deterministic settings (random seed = 42) to ensure full reproducibility.

5.1.4. Global model performance comparison

We now present comprehensive performance results. Each model receives identical preprocessing (83 PCA features + cluster ID as a categorical variable) and is evaluated using consistent metrics across three aggregation levels.

Model performance results. Our results reveal a striking pattern: LSTM (3 layers, 32 units, 31,000 parameters) achieves the best performance ($R^2 = 0.7121$, RMSE=0.0297 s m^{-1}), outperforming XGBoost by 9.3 %, xLSTM by 5.3 %, and Autoformer by 43 % in terms of R^2, while achieving comparable accuracy to PatchTST ($R^2 = 0.7043$, RMSE=0.0301 s m^{-1}) with 77× fewer parameters. This challenges the assumption that architectural sophistication correlates with performance. Recurrent architectures excel at capturing short-term temporal dependencies in our 83-component compressed feature space, whereas transformers appear overparameterized for elementary stop-to-stop predictions. XGBoost achieves $R^2 = 0.6513$, providing a strong tree-based baseline (see Table 4 and Fig. 8).

Table 4
Model comparison—Elementary to trip-level performance. xLSTM Segment and Trip RMSE values are reported in Table 5.

Model	Elementary RMSE ($\mathrm{s\,m^{-1}}$)	Elementary MAE ($\mathrm{s\,m^{-1}}$)	R^2	Segment RMSE (min)	Trip RMSE (min)
LSTM (Best)	0.0297	0.0235	0.7121	2.17	1.85
PatchTST	0.0301	0.0278	0.7043	2.45	2.08
xLSTM	0.0314	—	0.6763	—	—
XGBoost	0.0327	0.0257	0.6513	2.30	1.95
Autoformer	0.0391	0.0325	0.4994	2.75	2.35

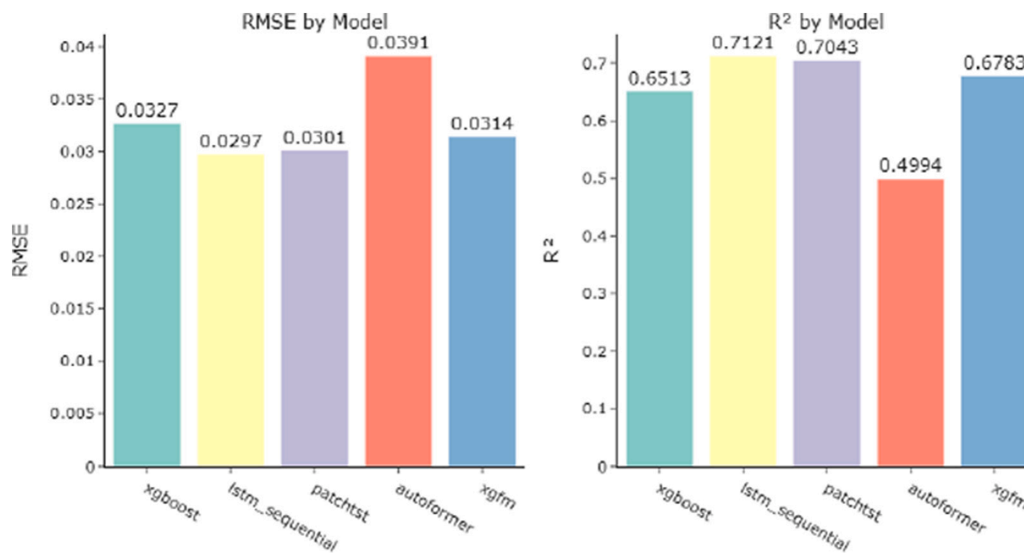


Fig. 8. Model comparison: RMSE and R^2 by architecture.

Intriguingly, trip-level RMSE (1.85 min) improves over segment-level RMSE (2.17 min), demonstrating error cancellation through aggregation. When uncorrelated segment errors e_i aggregate, they partially cancel ($|\sum e_i| < \sum |e_i|$), yielding sublinear variance growth $\mathrm{Var}(E_T) \propto \sqrt{N}$ rather than linear—analogous to variance reduction through averaging.

Error reduction through hierarchical aggregation. The observed error reduction from elementary to trip level has significant practical implications for passenger information systems. At the elementary segment level, predictions exhibit higher variance due to local variability in traffic conditions, passenger boarding times, and signal timing. However, when these elementary segment predictions are aggregated into trip-level estimates, random errors partially cancel out, resulting in more accurate trip-level predictions. This error-cancellation mechanism yields more reliable arrival-time predictions at the trip level–the primary quantity of interest for trip planning–though individual segment predictions may exhibit higher uncertainty. For transit operators, this hierarchical aggregation framework enables accurate trip-level delay estimates that support operational decision-making (e.g., schedule adjustments, vehicle dispatching) while maintaining the granularity needed for segment-level interventions. The practical significance is clear: the system provides actionable, accurate predictions at the aggregation level most relevant to end users, demonstrating the value of multi-level prediction frameworks for real-world transit applications.

Error distribution analysis. To provide deeper insight into model performance, we analyse the distribution of prediction errors. Fig. 9 presents the prediction error distribution and cumulative error distribution for the best-performing LSTM model. The histogram of error distributions reveals a sharply peaked, bell-shaped distribution centred near zero, with the vast majority of errors concentrated within $-0.1\,\mathrm{s\,m^{-1}}$ to $0.1\,\mathrm{s\,m^{-1}}$. The distribution exhibits high kurtosis, indicating that most predictions are highly accurate with rare outliers. The cumulative distribution function demonstrates that approximately 80 % of absolute errors fall below $0.05\,\mathrm{s\,m^{-1}}$, and nearly all errors (98 %) are below $0.1\,\mathrm{s\,m^{-1}}$, confirming the model's reliability for operational deployment.

Predicted vs actual delay analysis. Fig. 10 compares predicted and actual delay distributions at the segment level (in seconds). Both predicted and actual delay distributions are strongly right-skewed, with most delays concentrated near zero and a long tail extending to approximately 40 s. The histogram shows excellent overlap between predicted (blue) and actual (orange) distributions, indicating that the model accurately captures the underlying delay distribution. The scatter plot reveals a strong correlation between predicted and actual delays, with data points clustering tightly around the $y = x$ reference line for delays up to 20 s. For longer delays, the model shows slightly higher variance but maintains directional accuracy.

At the trip level (Fig. 11), the distributions similarly show right-skewed patterns, with most trip delays between −2 min to 4 min. The histogram reveals that predicted delays (blue) closely match the actual delay distribution (orange), with both peaking near zero to 1 min. The scatter plot shows excellent agreement between predicted and actual trip delays, with most predictions within 1 min of the actual values. The model maintains strong predictive performance across the full range of delays from −2 min to 8 min, with only a minor increase in variance for extreme delays.

Comprehensive trip-level error analysis. Fig. 12 presents a comprehensive three-panel analysis of trip-level prediction performance. The prediction error distribution (left panel) is symmetric and bell-shaped, centred at zero, with errors ranging from −2 min to 2 min, confirming unbiased predictions. The predicted vs. actual delay distribution (centre panel) shows excellent agreement between the model's predictions and observed delays, with both distributions peaking between 0 min to 1 min. The scatter plot (right panel) provides visual confirmation of the strong linear relationship between predicted and actual trip delays, with points tightly clustered around the $y = x$ reference line across the full delay range from 0 min to 10 min.

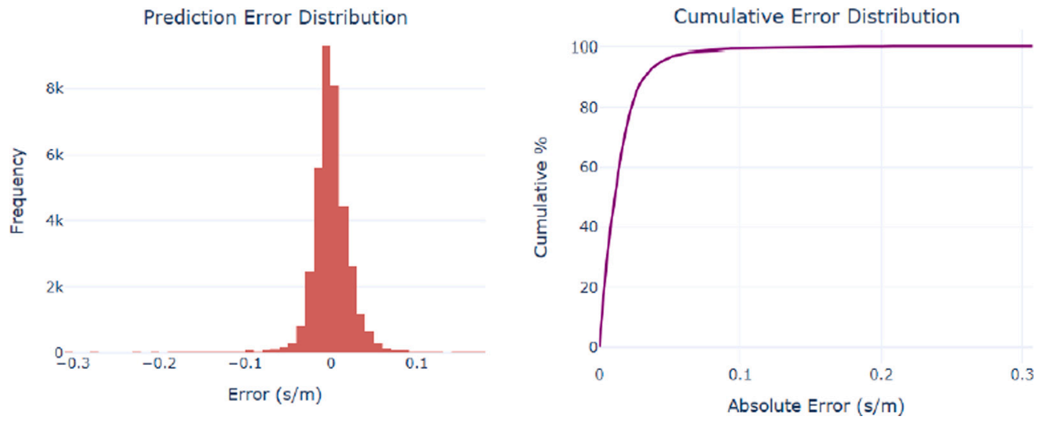


Fig. 9. Error distribution: histogram (left) and cumulative distribution (right).

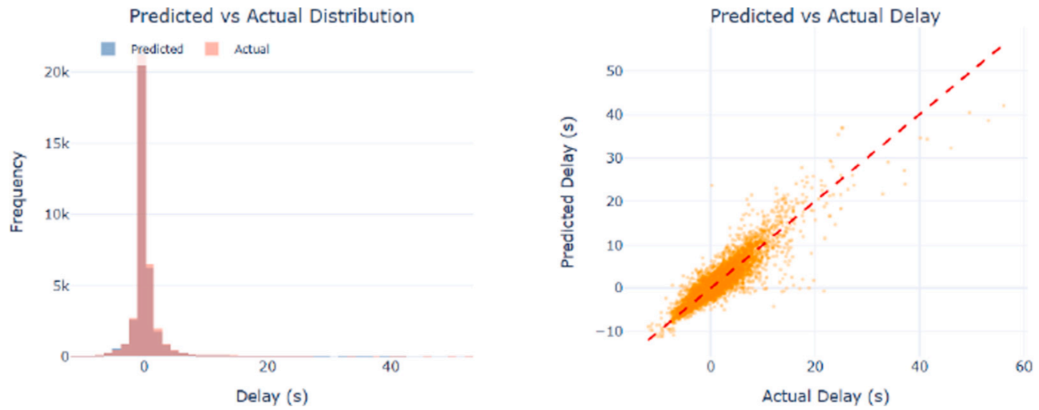


Fig. 10. Predicted vs actual delay at segment level: distributions (left) and scatter plot (right).

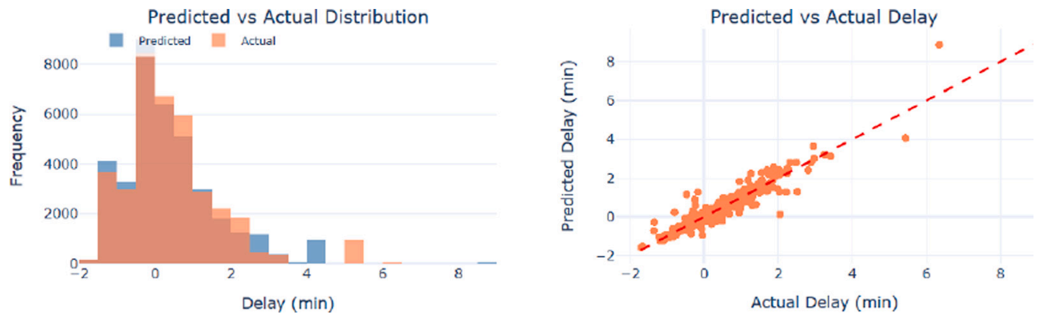


Fig. 11. Predicted vs actual delay at trip level: distributions (left) and scatter plot (right).

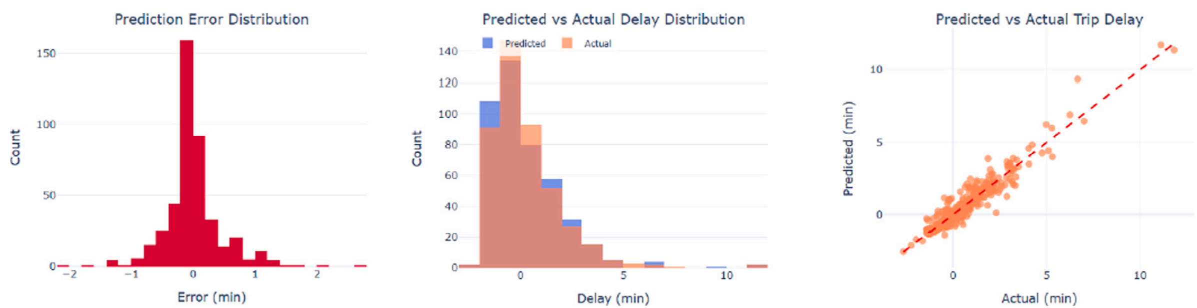


Fig. 12. Trip-level analysis: error distribution, histograms, and scatter plot.

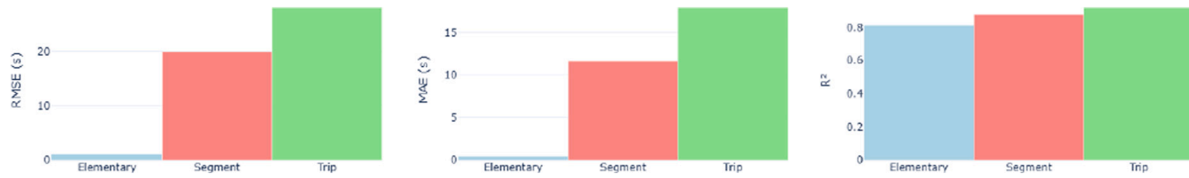


Fig. 13. Multi-level metrics: RMSE, MAE, and R^2 across aggregation levels.

Table 5
Computational complexity.

Model	Parameters	Training	Inference	GPU memory
LSTM	31,000	26 min	300 ms	2.1 GB
xLSTM	1,850,000	31 min	380 ms	3.0 GB
PatchTST	2,400,000	35 min	800 ms	4.3 GB
XGBoost	1,100,000	1.5 min	100 ms	CPU only
Autoformer	1,900,000	41 min	700 ms	3.8 GB

Multi-level performance metrics. Fig. 13 presents a comprehensive analysis of model performance across three aggregation levels: Elementary (stop-to-stop), Segment (route segments), and Trip (end-to-end). The results reveal important patterns in error propagation and variance explanation:

- **RMSE**: Increases across aggregation levels, reflecting the compounding of prediction errors across multiple segments within a trip, as illustrated in Fig. 13.
- **MAE**: Similarly increases across aggregation levels, following the expected pattern of error accumulation through aggregation, as illustrated in Fig. 13.
- **R**^{2}: Interestingly, R^2 improves progressively across aggregation levels, as shown in Fig. 13, indicating that the model explains a larger proportion of variance at higher aggregation levels. This counterintuitive result occurs because random errors at the elementary level partially cancel during aggregation, while systematic patterns become more pronounced at coarser granularities.

This multi-level analysis validates our hierarchical prediction framework and demonstrates that error cancellation through aggregation improves relative performance in trip-level predictions, despite increasing absolute errors.

LSTM achieves the best accuracy with surprisingly moderate resource requirements: 26 min of training time and a 2.1 GB memory footprint (see Table 5). For resource-constrained environments, XGBoost offers a compelling CPU-only alternative—trading just 8.5 % accuracy for dramatically faster training (1.5 min) and zero GPU dependency.

5.2. Non-functional requirements and architecture support

This subsection explicitly addresses how the system architecture addresses key non-functional requirements for production deployment. We report measured performance characteristics and explain architectural decisions that support scalability, maintainability, and operational reliability.

5.2.1. Performance requirements

Inference Latency: Real-time passenger information systems require sub-second prediction latency to maintain responsiveness. Our architecture addresses this through multiple design choices. The LSTM model achieves 300 ms per-prediction inference latency, while XGBoost provides 100 ms latency, suitable for CPU-only deployment. The inference pipeline (Section 3.5) is designed to minimize end-to-end latency through pre-computed feature caching, vectorized batch prediction, and hierarchical aggregation. Per-prediction inference times range from 100 ms (XGBoost) to 800 ms (PatchTST), as reported in Table 5. The orchestration layer manages inference scheduling to ensure latency targets are met under varying load conditions.

Training Efficiency: Daily model retraining requires training to complete within operational windows (typically overnight). LSTM training completes in 26 min on a single GPU, enabling daily retraining without specialized infrastructure. XGBoost provides an even faster alternative (1.5 min CPU training) for resource-constrained environments. The hybrid clustering strategy reduces training complexity by organizing data into 12 balanced clusters (CV = 0.608), enabling efficient distributed processing when needed.

Throughput: The system architecture supports high-throughput prediction serving through parallel processing. Feature engineering employs route-level chunking with ProcessPoolExecutor for CPU tasks and `ThreadPoolExecutor` for I/O operations, enabling distributed processing across worker nodes. The inference pipeline uses vectorized batch prediction to maximize throughput while maintaining latency targets.

5.2.2. Scalability requirements

Data Volume: The system processes 5.5 million observations across 196 routes over 6 months. The architecture scales through several mechanisms: (i) columnar Parquet storage with 5:1 compression and hierarchical partitioning by `route_id/date`, reducing storage overhead by 99 %; (ii) lazy loading and Apache Arrow memory-mapping to minimize memory consumption; (iii) distributed feature engineering via route-level parallelization. The hybrid H3+topology clustering enables efficient data organization, with balanced cluster sizes (imbalance ratio 1.90×) that support parallel model training when using per-cluster strategies.

Feature Dimensionality: The systematic feature engineering framework generates 1,683 features, which would be computationally prohibitive for deep learning models. Adaptive PCA compression to 83 components (95 % variance retained) makes global model training feasible while preserving predictive accuracy. This dimensionality reduction reduces memory requirements by 95 % and enables efficient model training on standard hardware.

Network Growth: The architecture supports network expansion through its modular design. Adding new routes requires only updating the GTFS static feed and retraining the global model with cluster-aware features. The clustering strategy automatically adapts to network changes, and the feature engineering framework applies uniformly across all routes without manual configuration.

5.2.3. Maintainability requirements

Reusability: The architecture separates city-specific configuration (GTFS feeds, H3 resolution, clustering parameters) from generic components (feature generation, dimensionality reduction, global modelling, inference). This design enables adaptation to other bus networks by changing configuration files and retraining models, while preserving the overall pipeline structure. The systematic feature engineering framework ensures reproducibility across different transit networks.

Modularity: The five-pipeline architecture (data collection, feature engineering, dimensionality reduction, predictive modelling, inference) provides clear separation of concerns. Each pipeline has well-defined interfaces and can be modified independently. The orchestration layer manages control flow separately from processing components, making maintenance and updates easier.

Versioning and Reproducibility: The system maintains version metadata for trained models (training date, hyperparameters, feature schema, performance metrics), enabling model versioning and rollback capability. GTFS version management automatically handles schedule updates by storing successive GTFS snapshots with validity periods, ensuring accurate delay calculations against correct baseline schedules.

5.2.4. Reliability considerations

Data Quality: The data collection pipeline implements rule-based quality checks to filter unreliable records (non-monotonic timestamps, unrealistic speeds, route deviations). The trip integrity buffer ensures that complete trip data is available before processing, handling edge cases such as trips that terminate early or vehicles that go offline. Quality validation assigns composite scores and flags observations below the threshold for manual review.

Fault Tolerance: The archival system uses file-based state management, where processed files are moved from raw storage to archive storage. This serves dual purposes: freeing working directory space and providing precise checkpoint mechanisms for incremental processing. The system can recover from failures by identifying which raw files have not been archived and reprocessing only incomplete work, avoiding redundant computation.

Model Robustness: The system demonstrates graceful degradation under extreme conditions. Multi-level aggregation exhibits error cancellation, in which random errors at the elementary level partially cancel during aggregation, thereby improving R^2 at higher levels despite increasing absolute RMSE. This property enhances reliability by reducing the impact of individual prediction errors on trip-level estimates.

5.2.5. Resource efficiency

Memory Requirements: LSTM operates with 2.1 GB GPU memory, while XGBoost provides a CPU-only alternative requiring no GPU resources. The feature engineering pipeline reduces memory consumption by 60 % through selective loading of feature groups during model training. Parquet storage with embedded feature metadata enables selective loading, reducing the memory footprint compared to loading the full feature matrix.

Computational Resources: The architecture supports deployment on standard hardware. LSTM training completes in 26 min on a single consumer-grade GPU (NVIDIA RTX 2080 Ti), enabling daily retraining without specialized infrastructure. For agencies without GPU resources, XGBoost offers a compelling alternative with 1.5 min CPU training time.

5.3. Discussion

This section synthesizes our experimental findings, interprets their significance for transit delay prediction research and practice, acknowledges limitations, and identifies promising directions for future investigation.

5.3.1. Summary of key findings

Our experimental evaluation advances transit delay prediction through three interconnected contributions:

Finding 1: Hybrid H3+Topology Clustering Resolves the "Giant Cluster Problem"

Naive geographic partitioning creates severe data imbalances (CV = 2.0, imbalance ratio = 8×)—what we term the "giant cluster problem". Our hybrid approach achieves CV = 0.608 and an imbalance ratio of 1.90×, representing 3.3× and 4.2× improvements, respectively. By combining H3 hexagonal indexing (geographic coherence) with route-overlap Jaccard similarity (topological awareness), we create 12 balanced clusters that exhibit strong spatial coherence (Jaccard overlap < 0.15 between adjacent clusters).

These clusters represent meaningful operational zones rather than arbitrary geographic divisions. Cluster IDs serve as categorical features for global modelling, enabling the model to learn cluster-specific delay patterns – distinguishing urban congestion from suburban traffic – without the operational complexity of managing multiple model instances.

Finding 2: Simpler Recurrent Architectures Outperform Transformers

LSTM achieves the best performance (R^2 = 0.7121), outperforming XGBoost by 9.3 %, xLSTM by 5.3 %, and Autoformer by 43 %. PatchTST is close behind, with R^2 = 0.7043, demonstrating that transformer architectures can achieve competitive performance but at a significantly higher computational cost. This result challenges conventional wisdom: LSTM's 31,000 parameters are dramatically smaller than PatchTST's 2,400,000, yet it delivers superior accuracy with lower resource requirements.

Stop-to-stop delay patterns exhibit short-term temporal dependencies that are better captured by recurrent gates than by attention mechanisms. Transformers excel at long-range dependencies but overfit our 6-month training dataset, suggesting that architectural sophistication becomes a liability when the parameter count exceeds the data scale. LSTM's compact design proves optimal for our compressed feature space (83 PCA components), where continuous state spaces handle dense embeddings more effectively than XGBoost's discrete tree splits—though XGBoost remains admirably competitive (R^2 = 0.6513).

Multi-level aggregation reveals systematic error cancellation, validating our hierarchical framework across all architectures.

Finding 3: Systematic Feature Engineering with Adaptive PCA Balances Expressiveness and Computational Efficiency

Training on the complete feature set directly is computationally prohibitive for deep learning architectures. Adaptive PCA emerges as the optimal dimensionality reduction strategy, enabling efficient global model training across all architectures while preserving essential delay patterns.

Our cluster-aware feature engineering framework generates features at multiple spatial resolutions (H3 res 9, 10) and temporal granularities (hour-of-day, time periods, 144 per-hour boolean intervals), capturing both local segment-level patterns and regional neighbourhood-level trends. Including cluster ID as a categorical feature enables the global LSTM model to learn cluster-specific patterns – distinguishing urban core congestion from suburban traffic and peripheral express routes – without explicit model partitioning.

5.4. Limitations and generalizability

Our validation focuses on a single network (STM, Montréal, 6 months). While the experimental validation is robust within the Montréal network context, the pipeline's generalizability claims would be significantly strengthened by more extensive experiments across diverse transit networks. We acknowledge this limitation and discuss how the pipeline can generalize to other cities. The pipeline's design emphasizes generalizability through several mechanisms: (1) modular architecture that separates city-specific configuration (GTFS feed structure, H3 resolution selection, clustering parameters) from generic components (feature generation, dimensionality reduction, global modelling, inference), (2) configuration-based customization that enables adaptation to different network characteristics without code modifications, and (3) systematic feature engineering framework that captures universal spatiotemporal patterns applicable across transit networks. However, specific components would require adaptation for other

cities: H3 resolution selection may need to be adjusted based on network density and geographic scale, clustering parameters (number of clusters, spatial weights) should be optimized for each network's spatial structure, and model retraining is necessary to capture network-specific delay patterns. Future work should validate the pipeline across multiple cities with varying characteristics (network size, density, operational patterns) to establish generalizability more definitively.

Additional limitations include: weather data rely on daily city-wide airport observations, which miss sub-daily temporal variations and spatial microclimate patterns; hourly, spatially distributed weather stations or 1 km radar could improve extreme-weather performance. Special event detection remains manual (47 events); automatic anomaly detection would enable adaptive responses. GTFS-RT lacks passenger counts (APC deployment < 30 % of agencies), limiting dwell-time modelling. Multi-level aggregation excludes explicit cross-route propagation; GNN approaches offer potential but face scalability challenges.

6. Conclusion

This work presents a scalable transit delay prediction engine that transforms transit operations from *reactive* management (“The bus is late”) to *prescriptive* decision-making (“Hold the bus for 30 s at the terminal to prevent bunching”). By bridging descriptive diagnostics with scalable deep learning, we provide the foundational technology for truly intelligent transit networks.

6.1. Technical contributions

We address three critical gaps in transit delay prediction:

1. **Systematic Multi-Resolution Feature Engineering:** Our framework generates 1,683 spatiotemporal features by exhaustively exploring 23 aggregation combinations across H3 resolutions, compressed to 83 components via Adaptive PCA (95 % variance retained).
2. **Hybrid H3+Topology Clustering:** We resolve the “giant cluster problem” through a novel clustering approach that combines geographic coherence with topological awareness, achieving balanced partitioning (CV = 0.608, imbalance 1.90×) while preserving spatial and route connectivity.
3. **Architecture Evaluation:** Rigorous comparison across five architectures reveals that LSTM achieves optimal performance ($R^2 = 0.7121$, RMSE = $0.0297\,\mathrm{s\,m^{-1}}$), outperforming XGBoost by 9.3 %, xLSTM by 5.3 %, and Autoformer by 43 %, while achieving comparable accuracy to PatchTST with 77× fewer parameters—challenging assumptions that architectural sophistication correlates with accuracy.

6.2. Operational applications

The prediction engine enables four categories of operational improvements:

- **Simulation & Capacity Planning:** Predict velocity drops along routes to proactively inject gap-filler buses before service degradation occurs, reducing passenger wait times and improving schedule adherence.
- **Anti-Bunching Interventions:** Accurately predict arrival times at downstream stops to suggest holding a bus at terminals or timing points, preventing the cascade of delays that leads to bus bunching.
- **Emissions Targeting:** Identify high-friction segments where buses experience persistent delays due to traffic conditions, enabling targeted green infrastructure investments (signal priority, dedicated lanes) where they deliver maximum impact.
- **Dynamic Accessibility:** Generate real-time isochrones for journey planning that reflect current network conditions, enabling passengers to make informed travel decisions based on predicted rather than scheduled arrival times.

6.3. Broader impact

Our finding that simpler recurrent models outperform transformers for short-term temporal patterns has implications beyond transit prediction. The systematic feature engineering framework, hybrid clustering methodology, and multi-level aggregation strategy apply directly to related urban mobility challenges, including traffic flow forecasting, ride-hailing demand prediction, bike-share rebalancing, and freight logistics optimization. By demonstrating that production-ready accuracy is achievable with modest computational resources (a single GPU and 26 min of training time), we lower the barrier for transit agencies worldwide to deploy intelligent prediction systems.

CRediT authorship contribution statement

Emna Boudabbous: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Mohamed Karaa:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Data curation. **Lokman Sboui:** Writing – review & editing, Validation, Supervision, Methodology. **Julio Montecinos:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization. **Omar Alam:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Elassy, M. Al-Hattab, M. Takruri, S. Badawi, Intelligent transportation systems for sustainable smart cities, Transp. Eng. 16 (2024) 100252, http://dx.doi.org/10.1016/j.treng.2024.100252.
[2] B.A. Kumar, R. Singh, H.E. Shaji, L. Vanajakshi, Bus arrival time prediction: A comprehensive review, IEEE Trans. Intell. Transp. Syst. 26 (6) (2025) 7362–7379, http://dx.doi.org/10.1109/TITS.2025.3545695.
[3] O. Alam, A. Kush, A. Emami, P. Pouladzadeh, Predicting irregularities in arrival times for transit buses with recurrent neural networks using GPS coordinates and weather data, J. Ambient. Intell. Humaniz. Comput. 12 (7) (2021) 7813–7826, http://dx.doi.org/10.1007/s12652-020-02572-8.
[4] N. Singh, K. Kumar, A review of bus arrival time prediction using artificial intelligence, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 12 (4) (2022) e1457, http://dx.doi.org/10.1002/widm.1457.
[5] Ł. Pałys, M. Ganzha, M. Paprzycki, Machine learning for bus travel prediction, in: D. Groen, C. de Mulatier, M. Paszynski, V.V. Krzhizhanovskaya, J.J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2022, in: Lecture Notes in Computer Science, vol. 13351, Springer International Publishing, Cham, 2022, pp. 703–710, http://dx.doi.org/10.1007/978-3-031-08754-7_72.
[6] Y. Huo, W. Li, J. Zhao, S. Zhu, Modelling bus delay at bus stop, Transport 33 (1) (2018) 12–21, http://dx.doi.org/10.3846/16484142.2014.1003324, URL: https://journals.vilniustech.lt/index.php/Transport/article/view/124.
[7] A.B. Subramaniyan, C. Wang, Y. Shao, W. Li, H. Wang, G. Zhang, T. Ma, Hybrid recurrent neural network modelling for traffic delay prediction at signalized intersections along an urban arterial, IEEE Trans. Intell. Transp. Syst. 24 (1) (2023) 1384–1394, http://dx.doi.org/10.1109/TITS.2022.3201880.
[8] N.C. Petersen, F. Rodrigues, F.C. Pereira, Multi-output bus travel time prediction with convolutional LSTM neural network, Expert Syst. Appl. 120 (2019) 426–435, http://dx.doi.org/10.1016/j.eswa.2018.11.028.
[9] Z. Li, P. Wolf, M. Wang, ArrivalNet: Predicting city-wide bus/tram arrival time with two-dimensional temporal variation modelling, 2024, http://dx.doi.org/10.48550/ARXIV.2410.14742, CoRR abs/2410.14742, arXiv:2410.14742.
[10] H. Rodriguez-Deniz, M. Villani, Robust real-time delay predictions in a network of high-frequency urban buses, IEEE Trans. Intell. Transp. Syst. 23 (9) (2022) 16304–16317, http://dx.doi.org/10.1109/TITS.2022.3149656.
[11] H.E. Shaji, A.K. Tangirala, L. Vanajakshi, Evaluation of clustering algorithms for the prediction of trends in bus travel time, Transp. Res. Rec.: J. Transp. Res. Board 2672 (45) (2018) 242–252, http://dx.doi.org/10.1177/0361198118791365.
[12] I. Brodsky, H3: A hexagonal hierarchical geospatial indexing system, 2018, Uber Technologies. https://eng.uber.com/h3/.

[13] E. Boudabbous, M. Karaa, L. Sboui, J. Montecinos, O. Alam, Analyzing public transit schedule deviations: A case study on Montreal using real-time data, in: 2024 IEEE 27th International Symposium on Real-Time Distributed Computing, ISORC, IEEE, 2024, pp. 1–6, http://dx.doi.org/10.1109/ISORC61049.2024.10551354, Exploratory case study on which current systematic methodology builds.
[14] W. Suwardo, M. Napiah, I. Kamaruddin, ARIMA models for bus travel time prediction, J. Inst. Eng. Malays. 71 (2) (2010) 49–58, Paper based on data from Ipoh–Lumut corridor, Malaysia.
[15] A. Achar, D. Bharathi, B.A. Kumar, L. Vanajakshi, Bus arrival time prediction: A spatial Kalman filter approach, IEEE Trans. Intell. Transp. Syst. 21 (3) (2020) 1298–1307, http://dx.doi.org/10.1109/TITS.2019.2909314.
[16] X. Chen, S. Saidi, L. Sun, Understanding bus delay patterns under different temporal and weather conditions: A Bayesian Gaussian mixture model, Transp. Res. Part C: Emerg. Technol. 171 (2025) 105000, http://dx.doi.org/10.1016/j.trc.2025.105000.
[17] B. Büchel, F. Corman, Probabilistic bus delay predictions with Bayesian networks, in: 2021 IEEE International Intelligent Transportation Systems Conference, ITSC, 2021, pp. 3752–3758, http://dx.doi.org/10.1109/ITSC48978.2021.9564537.
[18] Y. Sun, J. Spall, W. Wong, X. Zhao, Real-time bus travel time prediction and reliability quantification: A hybrid Markov model, 2025, URL: https://arxiv.org/abs/2503.05907, arXiv:2503.05907.
[19] M. Sinn, J.W. Yoon, F. Calabrese, E. Bouillet, Predicting arrival times of buses using real-time GPS measurements, in: 2012 15th International IEEE Conference on Intelligent Transportation Systems, 2012, pp. 1227–1232, http://dx.doi.org/10.1109/ITSC.2012.6338767.
[20] N. Marković, S. Milinković, K.S. Tikhonov, P. Schonfeld, Analyzing passenger train arrival delays with support vector regression, Transp. Res. Part C: Emerg. Technol. 56 (2015) 251–262, http://dx.doi.org/10.1016/j.trc.2015.03.027.
[21] S. Chtioui, S. Mouelhi, S. Saudrais, T. Azib, M. Ille, M. Morel, F. Oru, Xgboost in public transportation for multi-attribute data: Delay prediction in railway systems in real-time, IEEE Access 12 (2024) 143327–143342, http://dx.doi.org/10.1109/ACCESS.2024.3463022.
[22] A. Warnakulasuriya, C. Weerasinghe, H. Wickramarathna, S. Ratneswaran, U. Thayasivam, Explainable bus arrival time prediction model with improved features related to topography and points of interest, in: 2024 IEEE 27th International Conference on Intelligent Transportation Systems, ITSC, 2024, pp. 2131–2136, http://dx.doi.org/10.1109/ITSC58415.2024.10920146.
[23] H. Liu, H. Xu, Y. Yan, Z. Cai, T. Sun, W. Li, Bus arrival time prediction based on LSTM and spatial-temporal feature vector, IEEE Access 8 (2020) 11917–11929, http://dx.doi.org/10.1109/ACCESS.2020.2965094.
[24] Y. Rong, Z. Xu, J. Liu, H. Liu, J. Ding, X. Liu, W. Luo, C. Zhang, J. Gao, Du-bus: A realtime bus waiting time estimation system based on multi-source data, IEEE Trans. Intell. Transp. Syst. 23 (12) (2022) 24524–24539, http://dx.doi.org/10.1109/TITS.2022.3210170.
[25] P.P. Lopes, G. Gramaglia, D. Bacciu, H.T. Marques-Neto, Towards forecasting bus arrival thorough a model based on GNN+LSTM using GTFS and real-time data, in: Proceedings of the 4th International Conference on AI-ML Systems, AIMLSystems '24, Association for Computing Machinery, New York, NY, USA, 2025, pp. 1–9, http://dx.doi.org/10.1145/3703412.3703417.
[26] S. Sharma, N. Mawane, C.K. Kuraganti, D.G. M, M. Taware, Y.C. Dixit, S. Mishra, R. Krishnapuram, R. Ramesh, Enhanced ETA predictions with T-GCN on optimized road segments, in: 2024 IEEE International Smart Cities Conference, ISC2, 2024, pp. 1–6, http://dx.doi.org/10.1109/ISC260477.2024.11004294.
[27] Y. Rong, J. Yao, J. Liu, Y. Fang, W. Luo, H. Liu, J. Ma, Z. Dan, J. Lin, Z. Wu, Y. Zhang, C. Zhang, GBTTE: Graph attention network based bus travel time estimation, in: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 4794–4800, http://dx.doi.org/10.1145/3583780.3614730.
[28] C. Li, S. Lin, H. Zhang, H. Zhao, L. Liu, N. Jia, A sequence and network embedding method for bus arrival time prediction using GPS trajectory data only, IEEE Trans. Intell. Transp. Syst. 24 (5) (2023) 5024–5038, http://dx.doi.org/10.1109/TITS.2023.3237320.
[29] O. Kaya, M. Utku Kalay, Spatio-temporal forecasting of bus arrival times using context-aware deep learning models in urban transit systems, IEEE Access 13 (2025) 161423–161435, http://dx.doi.org/10.1109/ACCESS.2025.3609530.
[30] H.E. Shaji, A.K. Tangirala, L. Vanajakshi, Joint clustering and prediction approach for travel time prediction, PLoS One 17 (9) (2022) e0275030, http://dx.doi.org/10.1371/journal.pone.0275030.
[31] M. Cebecauer, E. Jenelius, W. Burghout, Spatio-temporal partitioning of large urban networks for travel time prediction, in: 2018 21st International Conference on Intelligent Transportation Systems, ITSC, 2018, pp. 1390–1395, http://dx.doi.org/10.1109/ITSC.2018.8569648.
[32] P. Gramacki, S. Woźniak, P. Szymański, Gtfs2vec: Learning GTFS embeddings for comparing public transport offer in microregions, in: Proceedings of the 1st ACM SIGSPATIAL International Workshop on Searching and Mining Large Collections of Geospatial Data, GeoSearch '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 5–12, http://dx.doi.org/10.1145/3486640.3491392.
[33] C. Wang, F. Zhao, H. Luo, Y. Fang, H. Zhang, H. Xiong, Towards effective transportation mode-aware trajectory recovery: Heterogeneity, personalization and efficiency, IEEE Trans. Mob. Comput. 24 (4) (2025) 2832–2846, http://dx.doi.org/10.1109/TMC.2024.3501280.
[34] B. Chambers, M. Zaharia, Spark: The Definitive Guide: Big Data Processing Made Simple, O'Reilly Media, Sebastopol, CA, 2018.
[35] E.I. Vlahogianni, M.G. Karlaftis, J.C. Golias, Short-term traffic forecasting: Overview of objectives and methods, Transp. Rev. 34 (1) (2014) 4–24, http://dx.doi.org/10.1080/01441647.2013.951992.
[36] S. Sadegh-Zadeh, et al., Comparative analysis of dimensionality reduction techniques: PCA, Laplacian score, and chi-square on EEG classification, Sci. Rep. 14 (2024) 21456, http://dx.doi.org/10.1038/s41598-024-71234-9.
[37] A. Chang, Y. Ji, Y. Bie, Transformer-based short-term traffic forecasting model considering traffic spatiotemporal correlation, Front. Neurorobotics (ISSN: 1662-5218) 19 (2025) 1527908, http://dx.doi.org/10.3389/fnbot.2025.1527908.
[38] A. Ghose, Y. Ren, Y. Cui, Neural network optimization and performance analysis for real-time object detection at the edge, in: Proceedings of the SC24 International Conference for High Performance Computing, Networking, Storage and Analysis – Research Poster Archive, Atlanta, GA, USA, 2024, 1–1. Research poster (ACM Student Research Competition). URL: https://sc24.supercomputing.org/proceedings/poster/poster_pages/post172.html.
[39] J. Xue, R. Tan, J. Ma, S.V. Ukkusuri, Data mining in transportation networks with graph neural networks: A review and outlook, 2025, http://dx.doi.org/10.48550/arXiv.2501.16656, CoRR abs/2501.16656, URL: https://arxiv.org/abs/2501.16656, arXiv:2501.16656.
[40] D. Kartini, et al., Dimensionality reduction using principal component analysis and genetic algorithm for microarray classification, Indones. J. Electron. Électroméd. Eng. Med. Inform. 7 (1) (2025) 23–35, http://dx.doi.org/10.17815/jelectrmed.2025.01.