Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Transition-Aware Short-Term Traffic Congestion Prediction: Efficient Gradient Boosting versus Spatio-Temporal Graph Neural Networks

George S. Theodoropoulos

gstheo@unipi.gr

University of Piraeus

Yannis Theodoridis

University of Piraeus

Research Article

Keywords: traffic congestion prediction, XGBoost, spatio-temporal graph neural networks, congestion origins, evaluation metrics, urban traffic management

Posted Date: September 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10839892/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Transition-Aware Short-Term Traffic Congestion Prediction: Efficient Gradient Boosting versus Spatio-Temporal Graph Neural Networks

George S. Theodoropoulos[1*] and Yannis Theodoridis[1]

[1*]University of Piraeus, Piraeus, Greece.

*Corresponding author(s). E-mail(s): gstheo@unipi.gr;
Contributing authors: ytheod@unipi.gr;

Abstract

Short-term traffic congestion prediction matters for effective traffic management, and the field usually tackles it with deep spatio-temporal graph neural networks, judged by aggregate accuracy metrics such as plain $\boldsymbol{F_1}$. Such metrics, however, reward persistence more than genuine early-warning skill, and in this paper we question both practices. We show that plain $\boldsymbol{F_1}$ can rank a trivial current-state persistence baseline similarly to spatio-temporal graph neural networks, even though that baseline has no transition skill by design; motivated by this, we introduce TRSP (Transition-Recall, Stable-Precision), a transition-aware metric that balances performance near state changes with reliability during stable periods. We then show that feature-engineered gradient boosting can match or surpass deep spatio-temporal graph neural networks on this task at a fraction of the computational cost. We frame congestion prediction as binary early-warning classification over horizons $\boldsymbol{h}$ (up to 60 minutes, thus short-term) and propose a reproducible graph-based procedure for selecting congestion-origin sensors as prediction targets. We evaluate our proposal on METR-LA, PEMS-BAY, and a proprietary Dutch dataset supplied by the Dutch National Data Warehouse (NDW). Against Graph WaveNet, MTGNN, DGCRN, and naive persistence baselines, our feature-engineered XGBoost models in global, local, and compact transfer regimes match or surpass the deep baselines on transition-aware accuracy while using substantially less training memory–time and lower per-sensor inference latency. The transfer model gives the strongest accuracy-to-cost trade-off, and because its features are named and physically grounded rather than learned embeddings, it is also easier to inspect than the deep baselines.

Keywords: traffic congestion prediction, XGBoost, spatio-temporal graph neural networks, congestion origins, evaluation metrics, urban traffic management

1 Introduction

Urban traffic congestion remains a persistent challenge for cities around the world, affecting daily mobility and resulting in large economic and environmental costs [1]. As urban demand grows, peak-period gridlocks continue to increase commute times, fuel consumption, and air pollution. For city planners and traffic engineers, the ability to predict when and where congestion will occur is thus central to effective traffic-management strategies. Anticipating traffic jams before they materialize allows for timely interventions such as adjusting traffic signal timings, ramp metering, dynamic lane management, or real-time rerouting of drivers through navigation applications. Accurate forecasting, however, remains challenging. Traffic patterns are shaped by the time of day, day of week, weather, special events, and, above all, the interconnected nature of road networks, where a disturbance at one location quickly propagates to others [2, 3]. Figure 1 illustrates the two characteristic forms this takes. In Figure 1(a), congestion sets in at a bottleneck and then spreads backward along the carriageway: each upstream sensor drops below the congestion threshold a few minutes after the one downstream of it, the signature of a congestion wave travelling against the direction of traffic. On the other hand, in Figure 1(b), the target sensor dips below its threshold for a single short interval and recovers almost immediately, while nearby sensors barely react. A congestion event is usually pictured as the long, sustained jam of panel (a); at the timescales this paper addresses, the brief, easy-to-miss episodes of panel (b) are just as common. Together, the two panels delimit what a useful predictor must do: anticipate the wave early, without raising an alarm at every short-lived dip.

In recent years, the research community has converged on deep spatio-temporal architectures for this problem, with graph neural networks that couple graph convolutions over the road network with temporal sequence models [4–8] being the most prominent. While these models have pushed the state of the art on standard speed-forecasting benchmarks, when the task is recast as the question traffic operators actually care about (*will this location be congested within the next hour?*), two observations emerge that motivate this work.

Observation 1: the standard metric rewards persistence. Congestion is a persistent, relatively rare state. Once a jam has been formed, it tends to last for tens of minutes, while most of the day remains free-flowing. As a consequence, a trivial *persistence* predictor ("the state in h minutes will most probably be equal to the current state") achieves a remarkably high F_1 score at short horizons: in our experiments, at the 10-minute horizon this simple non-predictive technique attains the highest plain F_1 of any method, learned or not, on two of our three datasets, and stays within 0.02 of the best on the third. Plain F_1 is thus dominated by the easy, already-decided portions of the timeline and barely reflects the model behavior that has operational value, i.e., detecting *transitions*, especially congestion onsets and sustained return-to-free-flow windows. Similar concerns about misleading evaluation practice have been raised both in the traffic forecasting literature [9, 10] and in the broader forecasting community [11]. We therefore argue that congestion prediction must be evaluated, and models must be selected, based on transition-aware metrics. To this end, we employ a *transition-F_1* score (an F_1 computed only on the time steps where the future state differs from the present one) and introduce *TRSP* (Transition-Recall,

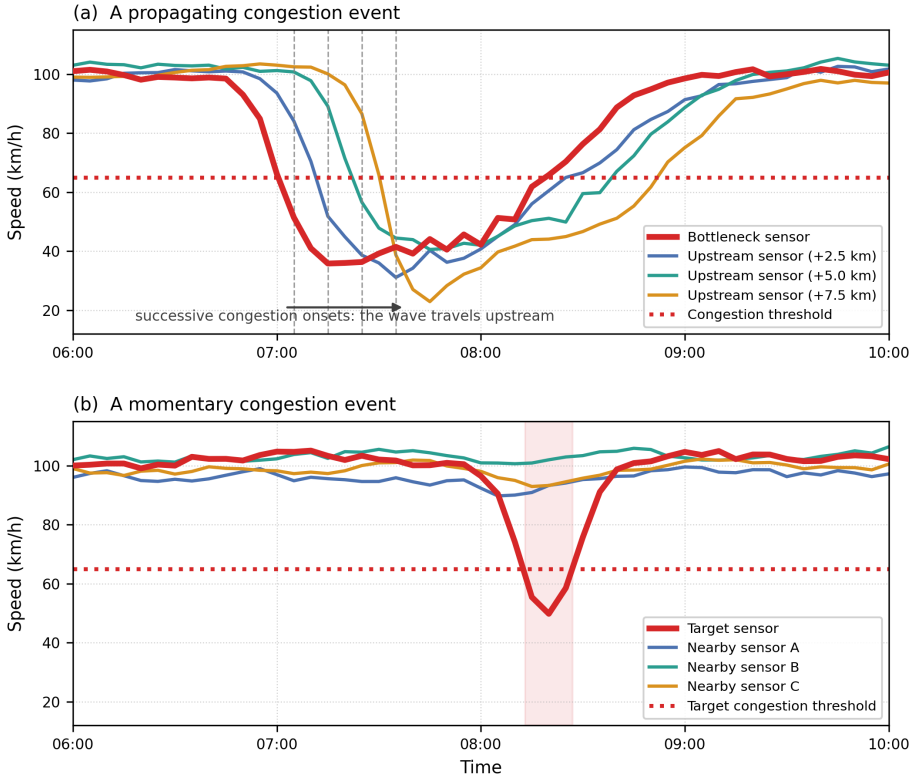


Fig. 1 Two characteristic congestion episodes (illustrative example). (a) A propagating event: successive congestion onsets (grey dashed lines) sweep from the bottleneck sensor to its upstream neighbors within minutes, as the jam grows backward against the direction of travel; the dotted line marks the congestion threshold, drawn at a common level here for clarity. (b) A momentary event: the target sensor falls below its threshold for one brief interval (shaded) and recovers almost immediately, while nearby sensors stay in free flow throughout. Episodes this short and localized are hard to catch and hard to anticipate.

Stable-Precision), a metric that combines performance inside transition windows with reliability during stable periods, and we show that it exposes failure modes that both plain F_1 and transition-F_1 miss when used in isolation.

Observation 2: the computational price of deep models is hard to justify. Spatio-temporal graph neural networks can require hours of GPU training per network and per configuration, occupying several to tens of gigabytes of accelerator memory throughout (Section 5.6 quantifies this for our three baselines), which complicates the frequent retraining that non-stationary traffic demands, and puts them out of reach of smaller road operators. Recent evidence from the tabular learning literature suggests that gradient-boosted trees remain highly competitive with deep architectures when the problem can be expressed through informative features [12, 13], and traffic-specific studies have questioned whether deep learning makes a practical difference over well-tuned simpler models [9]. We build on that view by making the congestion

process explicit in the feature space: multi-scale speed lags, congestion-state dynamics, periodic profiles, and upstream neighborhood aggregates aligned with the backward propagation of congestion waves [14]. XGBoost [15] can then model the remaining nonlinearities efficiently. The result is a family of predictors that are accurate at predicting transitions, train in minutes to a couple of hours on a single machine, and produce single-sensor predictions in the low-microsecond range (and often below one microsecond for the compact transfer variant).

A third, methodological, contribution concerns *where* prediction should be evaluated. Prior work selects target sensors either randomly or by expert judgment, which hampers reproducibility and may bias the evaluation toward easy locations. We instead propose a simple graph-based procedure that mines the historical data for *congestion origins*, i.e., sensors at which congestion systematically emerges while the road downstream of them is still free-flowing. Early warnings have the highest operational value at exactly such origins, which makes them a more meaningful test bed than randomly chosen sensors. The procedure is deterministic, dataset-agnostic, and reproducible; its output serves as the target-sensor set for all evaluated methods.

Concretely, this paper makes the following contributions:

- We formulate short-term congestion prediction as an *early-warning* binary classification task ("will congestion occur within the next h minutes?", $h \leq 60$) and make the case, conceptually and empirically, that it must be evaluated with transition-aware metrics. For this purpose, we introduce the TRSP metric and justify its use over plain F_1 and transition-F_1 (Section 3.1).
- We propose a reproducible, graph-based method for identifying congestion-origin sensors from historical data, and use it to define the evaluation targets on the target datasets (Section 4.1).
- We present a transition-aware XGBoost framework for congestion prediction, combining systematic leakage-safe feature engineering with three deployment regimes: a network-wide *global* model, a per-sensor *local* model, and a compact *transfer* model distilled from the global one via soft labels (Section 4). Because our features are named and tied to physical quantities rather than learned embeddings, the models remain easy to inspect, down to the level of a single target sensor.
- We conduct an extensive experimental study on three datasets (METR-LA, PEMS-BAY, NDW) against three state-of-the-art spatio-temporal graph neural networks (DGCRN, Graph WaveNet, MTGNN) and three naive persistence baselines, covering accuracy, transition behavior, training cost, GPU memory occupancy, and inference latency (Section 5).

The remainder of the paper is structured as follows. Section 2 reviews related work. Section 3 defines the traffic congestion prediction task and introduces the transition-aware evaluation methodology, including the TRSP metric. Section 4 presents the congestion-origin identification procedure and the proposed XGBoost framework. Section 5 describes the baselines and reports the experimental study. Section 6 concludes the paper and sketches future research directions.

2 Related Work

2.1 From Statistical Models to Deep Spatio-Temporal Learning

Early efforts to understand and mitigate urban traffic congestion focused on broad conceptual frameworks, as in Afrin & Yodo's survey [16], which emphasized holistic, data-driven solutions that integrate infrastructure, traffic management, and policy. On the predictive side, classical statistical models were the first methods used to tackle this problem. Alghamdi et al. [17] demonstrated the utility of ARIMA for short-term forecasting of linear, stationary patterns, while also exposing its limitations on the non-linear, non-stationary dynamics of real traffic. These limitations resulted in the adoption of machine learning and, subsequently, deep learning.

Early deep-learning approaches targeted temporal structure, with spatial awareness gradually folded in. Liu et al. [18] embedded convolutional operations within LSTM units (Conv-LSTM), learning spatial correlations alongside temporal dynamics and outperforming pure time-series methods, thereby establishing that effective traffic prediction requires joint modeling of space and time. Graph-based methods then made the road network topology explicit. Li et al. [4] cast traffic flow as a diffusion process on a directed graph (DCRNN) and introduced the METR-LA and PEMS-BAY benchmarks used in virtually all subsequent work. Yu et al. [5] proposed STGCN, a fully convolutional spatio-temporal architecture; Zhao et al. [19] combined graph convolutions with GRUs (T-GCN); and Guo et al. [20] optimized the graph representation itself (OGCRNN).

The three architectures we adopt as baselines represent the maturation of this line. *Graph WaveNet* [6] couples dilated causal convolutions with graph convolutions over both a predefined and a self-learned (adaptive) adjacency matrix, capturing long temporal ranges efficiently. *MTGNN* [7] generalizes this idea to multivariate time series without a given graph, jointly learning the graph structure and the spatio-temporal dynamics through mix-hop propagation and dilated inception layers. *DGCRN* [8] generates a dynamic adjacency matrix at every time step from the evolving traffic state and combines it with a static, distance-based graph inside a recurrent encoder–decoder. All three are trained as speed regressors and are widely regarded as state-of-the-art; Section 5.3 details how we adapt them to congestion classification.

Although these architectures are no longer the most recent in the literature, they remain appropriate comparison points for two reasons. First, independent large-scale benchmark studies find that later spatio-temporal architectures improve on them only marginally once tuning and evaluation are harmonized [21, 22], so they still represent the practical state of the art. Second, they span the main architectural families of the field: recurrent graph convolution with dynamic adjacency (DGCRN), dilated temporal convolution with fixed and self-adaptive graphs (Graph WaveNet), and fully learned graph structure (MTGNN). Conclusions drawn against them are therefore unlikely to be overturned by a further variation on the same components, especially since our claims concern transition-aware skill and computational cost rather than position on the speed-forecasting leaderboards.

2.2 Congestion Prediction as Classification

A parallel thread addresses congestion explicitly as a (typically binary) state. Chen et al. [23] modeled the strong daily and weekly periodicity of congestion through time-series folding (PCNN), treating congestion as a thresholded binary state and prioritizing recall via weighted losses. Ranjan et al. [24] combined CNN, LSTM, and transpose-CNN layers to produce city-wide congestion maps. Nagy & Simon [3] showed that explicitly modeling how congestion *spreads* across the network boosts forecasting accuracy, where traffic must be treated as a connected system in which a jam at one location directly impacts its neighbors. Li et al. [25] tackled sparse sensor coverage with a multi-task GNN that jointly predicts continuous speed and binary congestion, and Liu et al. [26] introduced a learned spatial transfer matrix to propagate congestion information from sensed to unsensed segments, using rank-based speed features to handle non-stationarity. Our windowed early-warning formulation (Section 3) follows this operational tradition, while our origin-mining procedure (Section 4.1) is, to the best of our knowledge, the first to use the propagation structure of congestion to *select the evaluation targets* themselves.

Why we do not compare against these works directly

Although the works above share our classification objective, none of them can serve as a baseline under our protocol. They operate at different spatial granularities (city-region grids or image-like congestion maps [23, 24] rather than individual sensors), they target different inference problems (propagating congestion estimates to *unsensed* segments [25, 26] rather than early warning at monitored locations), and none provides a public implementation that could be retrained under our labels, splits, and threshold tuning. Re-implementing them from their paper descriptions would risk unfair and irreproducible comparisons. We instead follow the evaluation practice of this literature itself [25, 26] and convert state-of-the-art speed forecasters into congestion classifiers via the operational congestion definition of Section 3 (see Section 5.3).

2.3 Congestion Propagation and Bottleneck Identification

The structure of congestion, where it starts and how it spreads, has long been studied in traffic engineering. Chen et al. [27] proposed a systematic procedure for identifying freeway bottlenecks from loop-detector data, based on speed differentials between adjacent stations; such bottleneck locations are prime candidates for ramp metering and other interventions. From a data-mining perspective, Nguyen et al. [2] discovered congestion propagation patterns by mining frequent spatio-temporal congestion subtrees from sensor data. Traffic flow theory provides the physical underpinning: congestion at an active bottleneck propagates backward as a shockwave travelling upstream at a characteristic speed of roughly 15–20 km/h [14]. Our origin-identification procedure operationalizes these ideas on the sensor graph, with an origin being a sensor whose congestion systematically arises while the road downstream of it is still free-flowing and while upstream segments behind it are already congested. The wave-aligned upstream features of our predictor (Section 4) exploit the same physics in the feature space.

2.4 Evaluation Pitfalls and the Case for Strong Simple Baselines

The choice of evaluation metric is more than a formality. Naidu et al. [10] stress that inappropriate metrics (e.g., overall accuracy on imbalanced congestion data) lead to misleading conclusions. In forecasting more broadly, the M-competitions repeatedly found that simple statistical methods beat sophisticated ML models more often than the literature suggests [11], and Manibardo et al. [9] reached a similar conclusion specifically for road traffic forecasting, urging the community to report strong naive baselines. For tabular and feature-based problems, Grinsztajn et al. [12] and Shwartz-Ziv & Armon [13] show that gradient-boosted trees remain superior or competitive to deep models at a fraction of the cost. This work aims to bring these threads together for congestion prediction, reporting persistence baselines, demonstrating that they invalidate plain F_1 as a headline metric, proposing a transition-aware alternative, and showing that a feature-engineered XGBoost [15] matches or beats deep spatio-temporal models under it.

Our concern with metric design also connects to a broader line of work on evaluating detectors of rare, temporally extended events, primarily in time-series anomaly and change-point detection. Tatbul et al. [28] generalized precision and recall from individual points to *ranges*, crediting partial overlap between predicted and ground-truth anomaly intervals and allowing domain-specific weighting of existence, position, and cardinality. The Numenta Anomaly Benchmark [29] introduced a windowed scoring rule for streaming detection that rewards early, timely alarms near a labeled anomaly and penalizes late or spurious ones. In change-point detection, van den Burg and Williams [30] score predictions with an F_1 measure that counts a detection as correct when it falls within a fixed margin of a true change point. Our proposed metric, TRSP (to be introduced in Section 3.1), borrows this same margin-tolerant idea for congestion early-warning: its transition bands act as a tolerance window around each true change in the windowed target label. Its stable-precision term goes further and also penalizes false alarms during steady periods, which event-localized scores largely ignore because they only score the events themselves. TRSP folds transition-localized recall and stable-period reliability into a single harmonic mean tailored to the persistence-dominated structure of congestion labels.

3 Problem Definition

We address **short-term traffic congestion prediction** in road networks monitored by fixed-point speed sensors, formulated as a spatio-temporal binary classification problem with an early-warning target.

Congestion state

Given the speed measurement $v(s,t)$ of sensor s at time t, we define the congestion state as a significant deviation from the typical free-flow conditions of that particular

location:

$$\mathcal{C}(s,t) = \begin{cases} 1 & \text{if } v(s,t) < \tau \cdot \text{quantile}_{\alpha}\big(v(s,\cdot)\big) \\ 0 & \text{otherwise} \end{cases}$$

where $\text{quantile}_{\alpha}(v(s,\cdot))$ is the α-quantile of the historical speed distribution of sensor s, serving as a per-sensor reference of typical fast conditions, and τ is the congestion severity threshold. Following domain experts' feedback, we set $\tau = 0.65$ and $\alpha = 0.9$; that is, congestion corresponds to speed reductions exceeding 35% of the typical (90th-percentile) conditions at each location, as illustrated in Figure 1. Defining the threshold per sensor makes the label robust to heterogeneous speed limits and road geometries across the network.

Early-warning target

Rather than predicting the state at an exact future instant, we predict whether congestion will be observed *at any point in time within* the next h minutes:

$$\mathcal{Y}(s,t,h) = \max_{0<\delta\leq h} \mathcal{C}(s,t+\delta), \qquad h \leq 60 \text{ minutes (short-term)}.$$

This windowed formulation matches the operational semantics of an early-warning system: a traffic operator who receives an alarm "congestion expected at s within the next 30 minutes" does not care whether the onset occurs at minute 25 or minute 30, but does care that the alarm precedes the onset. It is also better posed for intervention planning than the instantaneous variant, since interventions (signal retiming, ramp metering, rerouting) take effect over a window rather than at an instant. Conversely, when the sensor is congested at time t, a negative label means that the whole forecast window is free-flowing; it should be read as a sustained-clear-window target, not as an estimate of the exact clearance time.

Transitions

The pairs (t,h) for which the windowed future label differs from the present state, i.e., $\mathcal{Y}(s,t,h) \neq \mathcal{C}(s,t)$, are called *transitions*. A transition is either an *onset* ($\mathcal{C}=0$, $\mathcal{Y}=1$; the sensor is currently free-flowing but at least one congested measurement occurs within the forecast window) or a *sustained-clear transition* ($\mathcal{C}=1$, $\mathcal{Y}=0$; the sensor is currently congested but no congested measurement occurs in the forecast window). This is intentionally a window-based definition, i.e., it does not estimate the precise clearance instant, and a congestion episode that persists for any part of the forecast window remains labeled future-congested. Transitions are the operationally valuable part of the prediction problem (a prediction that merely repeats the current state is available for free). Transitions are also rare: in our datasets, congestion itself covers less than $\sim$15% of the timeline, and transitions an even smaller fraction. This combination of persistence and imbalance is what makes the evaluation methodology below necessary.

3.1 Transition-Aware Evaluation: Why TRSP

Evaluating congestion predictors appears straightforward: congestion is binary, so one reports F_1. The temporal structure of the label sequence, however, makes this choice misleading. The label sequence consists of long stable runs (free flow, or established congestion) separated by rare transitions. We argue, and empirically verify in Section 5, that the standard metrics are each incomplete on their own, and we therefore evaluate with three metrics of increasing specificity.

Plain F_1 and its failure mode

The F_1 score over all test records is dominated by the stable majority. A persistence predictor that simply copies the current state ($\hat{\mathcal{Y}}(s,t,h) = \mathcal{C}(s,t)$) is correct on every stable record and wrong on every transition; since transitions are rare, its F_1 is high (in our study, 0.71–0.88 at the 10-minute horizon), which puts it on par with, or above, the best learned models on two of the three datasets. Yet this predictor has, by construction, *zero* predictive skill: it cannot anticipate an onset or identify a sustained-clear forecast window, which is the purpose of the system. Plain F_1 therefore measures overall agreement with the future state; it says nothing about whether the model adds value over doing nothing. We report it for comparability with prior work, but not as a headline metric.

Transition-F_1 and its failure mode

The natural fix is to score only where it is hard. Based on that premise, we define *transition-F_1* as the F_1 computed exclusively on transition records, i.e., those with $\mathcal{Y}(s,t,h) \neq \mathcal{C}(s,t)$. Persistence scores exactly 0 on it, as it should. However, transition-F_1 in isolation has the mirror-image flaw: it is blind to the model's behavior on the stable majority. An over-sensitive model can score well on transition records while producing many false alarms during stable periods, alarms that plain F_1 would catch but transition-F_1 ignores. The stale-pattern persistence baselines illustrate the failure mode concretely: Naive-Yesterday and Naive-LastWeek forecast state changes from a previous day or week, so (unlike Naive-Current) they reach non-trivial transition-F_1 (up to 0.5–0.6, e.g. 0.62 for Naive-LastWeek on PEMS-BAY at h=60) by producing many *mistimed* changes, yet their stable-period precision is among the worst of any method and their TRSP sits at the bottom of every comparative table (Section 5.4). A metric reading only the transition records would rank these stale-pattern predictors above far more reliable models. Neither metric alone can rank such behavior sensibly.

TRSP

We therefore introduce *TRSP* (Transition-Recall, Stable-Precision), which scores the two regimes separately and then combines them. The test timeline of each sensor is partitioned into *transition bands* (the records within $\pm W$ steps of every change point of the true target sequence) and *stable regions* (everything else). We compute:

- **TR (transition recall):** the F_1 score restricted to the transition bands, rewarding the timely detection of onsets and sustained-clear windows.

- **SP (stable precision):** the F_1 score restricted to the stable regions, penalizing false alarms and misses during steady stretches.

For regions whose true labels are single-class, where F_1 degenerates (a perfect all-negative prediction would score 0), the corresponding component falls back to accuracy; empty regions score 1, so an all-stable series is judged purely by its stable component. The combined score is the weighted harmonic mean

$$\mathrm{TRSP} = \left(\frac{\lambda}{\mathrm{TR}} + \frac{1-\lambda}{\mathrm{SP}}\right)^{-1} \in [0, 1],$$

with window $W = 1$ step and $\lambda = 0.5$ (the plain harmonic mean) throughout this paper. Larger λ emphasizes transition performance, and the harmonic mean ensures that a model cannot compensate for a collapsed component with an inflated one.

Why TRSP is the right headline metric for this task

TRSP is constructed so that each of the failure modes above is costly. Persistence has TR ≈ 0 contribution from skill (it scores on transition bands only via the band slack W), so its TRSP is limited regardless of its stable behavior; an over-sensitive model has high TR but a lower SP, and the harmonic mean pulls it down; and a model good at both regimes is rewarded over one excellent at a single regime. TRSP thus asks the operationally complete question of whether the model detects the transitions while remaining reliable between them. For this reason, TRSP is used in this study both for reporting and for *model selection*: the decision thresholds of all learned models (Sections 4 and 5.3) are tuned on the validation split to maximize TRSP, so that every method competes under the same, transition-aware objective. We accompany TRSP with plain F_1 and transition-F_1 in all result tables, both for transparency and because the divergence between the three metrics is itself informative.

4 The Proposed Framework: Transition-Aware Gradient Boosting

4.1 Identifying Congestion Origins

Any empirical comparison of congestion predictors must choose *where* to predict. Evaluating on all sensors is wasteful, since most locations rarely congest (Section 5.1 reports this as a dataset feature), and dilutes the result with trivial cases, whereas hand-picking sensors, whether at random or by expert judgment, is hard to reproduce and justify. We propose instead to mine the historical data for the locations where prediction matters most: *congestion origins*.

Intuition

Congestion in a road network is not spatially uniform, as it systematically emerges at specific locations (bottlenecks induced by merges, lane drops, weaving sections, or demand surges [14, 27]) and then spreads to adjacent segments [2, 3]. A sensor at which congestion repeatedly *begins*, rather than one that merely receives congestion

propagated from elsewhere, is the place where an early warning enables preventive intervention before the event affects the surrounding network.

Origin score

Let $G = (S, E)$ be the directed sensor graph induced by the dataset's adjacency matrix, where an edge (s_i, s_j) denotes that s_i (s_j) is an upstream (downstream) neighbor of s_j (s_i, respectively). Using the congestion indicator $\mathcal{C}(s, t)$ of Section 3, at every timestamp t we call sensor s an *active origin* if (i) s is congested, (ii) *none* of its downstream neighbors is congested, and (iii) *at least one* of its upstream neighbors is congested. Condition (ii) places s at the downstream head of the congested stretch, where traffic still discharges freely ahead, so its congestion marks an active bottleneck rather than a jam inherited from further downstream. Condition (iii) requires that the congestion coexists with congestion on at least one upstream segment behind it, i.e., that it is part of a spatially extended, backward-propagating event rather than an isolated fluctuation. The *origin score* of s is the number of timestamps at which s is an active origin, and the top-k sensors by this score constitute the target set; without loss of generality, we use $k = 10$ per dataset.

Discussion

The procedure is deterministic, requires only the speed matrix and the adjacency matrix of the sensor network, and runs in minutes on commodity hardware. Applied to our three datasets (Section 5.1) it yields the $k = 10$ most probable origins per dataset; all methods in this paper are trained to predict, and are evaluated at, exactly these sensors. Compared to a random selection, the origin set constitutes a *harder* benchmark (origins congest frequently and their onsets are the events that matter), and compared to expert selection, it is reproducible on any dataset, including ones where no expert annotation exists. Section 5.1 reports an external validation of the procedure on the NDW dataset.

Unlike the deep baselines, which learn spatio-temporal representations from raw sensor data, our approach encodes the structure of the congestion process into explicit features and trains gradient-boosted tree ensembles [15] on them. This approach results in increased accuracy at transitions (the features directly expose onset precursors), better efficiency (fast training, low-microsecond inference), and greater interpretability (feature importances are meaningful to traffic engineers). The framework includes a feature-engineering layer, a transition-weighted training procedure, and three deployment regimes.

4.2 Feature Engineering

All features are causal (computed from data up to time t only) and *leakage-safe*, as every component that requires a statistic of the data (congestion thresholds, time-of-week profiles, normalization constants) is computed on the training split only. The features fall into eight families:

1. **Speed history.** The current speed and lags at 5–180 minutes, plus calendar-aligned lags (12h, 24h$-h$, 48h$-h$, 7d$-h$, so that the lagged value refers to the time for which

the prediction is made); rolling means and standard deviations over 15/30/60-minute windows, rolling minima, and exponential moving averages (half-lives of 15 and 60 minutes).

2. **Speed dynamics.** First differences at 1 and 5 steps and the second difference (acceleration), capturing the rapid speed drops that precede onsets.
3. **Threshold margin.** The signed margin to the congestion threshold, $v(s,t) - \tau \cdot \text{quantile}_{\alpha}(v(s,\cdot))$, and its relative form. The margin normalizes heterogeneous sensors onto a common scale ("how far from congestion is this location right now?"), which is particularly valuable for the global model, where one tree ensemble serves all sensors.
4. **Congestion-state history.** The current congestion indicator; rolling congestion fractions over 15/30/60 minutes; and recent onset/recovery transition indicators over 10–30-minute windows.
5. **Time-in-state counters.** Minutes spent in the current state, minutes since the last onset, and the duration of the previous congestion episode. Congestion episodes have strong duration structure (they typically last several tens of minutes), so these state-machine features directly inform whether an episode is young or about to dissipate, i.e., the transition cases.
6. **Periodicity.** Cyclical encodings (sine/cosine) of time of day and day of week, weekend/rush-hour indicators, and a holiday flag; and, more powerfully, *empirical time-of-week profiles*, i.e., the historical congestion probability and mean speed of the sensor in each 15-minute bin of the week, looked up both at t and at $t + h$, together with the speed's residual from its profile. The profiles inject exactly the recurring-pattern knowledge that calendar-lag persistence baselines exploit, at the cost of a table lookup.
7. **Naive predictions as features.** The congestion state 24h$-h$ and 7d$-h$ in the past (i.e., the predictions of the calendar persistence baselines) are offered to the model as features, so that the learned model has access, by construction, to everything the naive baselines know.
8. **Upstream spatial aggregates.** For hop distances 1–3 in the sensor graph, inverse-distance-weighted aggregates of the upstream neighbors' speeds, congestion states, and transition indicators. In addition, *wave-aligned* variants shift each upstream aggregate backward in time proportionally to its hop distance, mimicking the known backward propagation speed of congestion waves ($\sim$15–20 km/h [14]); the upstream state then acts as a genuine leading indicator of what will arrive at the target sensor.

Because every feature is named and tied to a physical quantity, the models keep XGBoost's usual gain-based feature importances readable in plain traffic terms, for instance "the congestion state two hops upstream, 10 minutes ago," rather than an abstract embedding coordinate that would need further work to interpret. Since a separate model is fit per target sensor (Section 4.4), these rankings can also be pulled per sensor, so a traffic engineer can check what each individual location's model is actually relying on.

4.3 Transition-Weighted Training and Threshold Selection

Two imbalances must be addressed, namely that congestion records are rare overall, and that transition records are rare among them. The former is handled with the standard positive-class weight (set to the negative-to-positive ratio of the training split). For the latter we introduce *transition weighting*, where every training record whose target differs from the current state (an onset or sustained-clear case) receives sample weight $w \geq 1$, while stable records keep weight 1. We train with $w \in \{1, 3\}$ and report both; w=1 recovers the standard objective, while w=3 explicitly steers the trees toward the transition cases, trading a small amount of plain F_1 for potentially better transition-F_1 and TRSP (Section 5.5).

Since the model outputs probabilities, a decision threshold must be chosen. We sweep thresholds in $[0.02, 0.98]$ (step 0.02) on the validation split and keep the one maximizing TRSP, in line with the evaluation philosophy of Section 3.1. The same procedure is applied to all baselines, ensuring a level playing field.

4.4 Three Deployment Regimes

The framework is trained and deployed in three regimes, which differ in the scope of the training data (the whole network versus a single sensor) and in the size of the resulting model. The three regimes correspond to different operational settings, namely full network access, independently deployed sensors, and resource-constrained installations, respectively; their accuracy is compared in Section 5.5 and their computational profiles in Section 5.6.

Global model

One model per horizon is trained on all sensors' features to predict the target sensors. Because the candidate feature space is large (all sensors $\times$ all families), the global pipeline performs: (i) a univariate pre-filter retaining the 1,000 candidate columns most correlated with the target on the training split; (ii) a gain-based importance ranking using a fixed-parameter XGBoost run; (iii) a feature-budget sweep that always retains the target sensor's own features and adds the best 15 or 30 cross-sensor columns, selecting the budget by validation TRSP; and (iv) a staged hyperparameter search over max depth $\{3, 5, 7\}$, learning rate $\{0.01, 0.03, 0.05\}$, subsampling and column-subsampling rates $\{0.6, 0.8\}$, and L2 regularization $\{0, 1\}$, with early stopping (50 rounds, at most 2,000 boosting rounds) on the validation split. The global model can exploit cross-sensor structure (e.g., an upstream origin's state helping predict a downstream target) and shares statistical strength across locations.

Local model

One model per sensor and horizon, restricted to the target sensor's own features (families 1–7). The local model is the natural choice when no network information is available or when sensors must be deployed independently; it also serves as the ablation that quantifies the value of the spatial features.

Transfer model

A compact per-sensor model distilled from the global one. For each target sensor and horizon, we (i) take the top-20 features of the global model (by gain) that are computable locally, and (ii) train a small model on *soft labels* $\tilde{y} = \alpha\, p_{\text{global}} + (1-\alpha)\, y$ with $\alpha = 0.3$, where p_{global} is the global model's predicted probability on the training rows. The soft labels transfer the global model's cross-sensor knowledge into a model with a strict 20-feature budget; validation and testing always use the hard labels. The result is the smallest and fastest of the three regimes, a property we exploit in the efficiency analysis (Section 5.6) and one that makes it the natural candidate for resource-constrained or edge deployments.

5 Experimental Study

5.1 Datasets

We evaluate on three highway-network datasets spanning two continents (Table 1). Across all three, congestion as defined in Section 3 is a comparatively rare state, since most locations rarely congest at all. For this reason we do not evaluate on every sensor and rely instead on the origin-mining procedure of Section 4.1.

- **METR-LA** [4] contains speed measurements from 207 loop detectors on the highways of Los Angeles County, recorded at 5-minute intervals over four months (March–June 2012). It is the noisiest of the three benchmarks and a de facto standard in the traffic forecasting literature.
- **PEMS-BAY**[1] [4] comprises 325 loop detectors across the San Francisco Bay Area highway network, recording speed at 5-minute intervals over six months (January–June 2017), with precise geocoordinates enabling the construction of the directional sensor graph.
- **NDW**[2] is a proprietary dataset provided by the Dutch National Data Warehouse, containing 5-minute speed measurements from 206 loop detectors along two major highways (A20 and A4) around Rotterdam, collected over 13 months (April 2022–May 2023). It captures complex dynamics across merging zones, bottlenecks, and free-flow segments, encompassing seasonal variation, weather-related disruptions, and incident-induced anomalies. Access is restricted to approved research collaborations with the Dutch transportation authorities.

The locations of the 10 target sensors selected in METR-LA, PEMS-BAY, and NDW are shown in Figure 2. For each dataset, the target sensors are the top-k congestion origins mined by the procedure of Section 4.1; we use $k = 10$ per dataset throughout. We chose this value because it gives a target set large enough to average metrics over reliably, without making the experiments unreasonably expensive: each additional sensor requires rerunning twelve method configurations across three horizons. On NDW, the mined origins independently recover locations consistent with the

[1] https://pems.dot.ca.gov
[2] https://english.ndw.nu

bottlenecks that Dutch traffic engineers had previously identified as locations of interest (merging zones and interchanges on the A4/A20 corridors), which we regard as external validation of the origin-mining procedure.

Table 1 Summary of the datasets used.

Attribute	METR-LA	PEMS-BAY	NDW
Location	Los Angeles, USA	San Francisco, USA	Rotterdam, NL
No. of sensors	207	325	206
Total time	4 months	6 months	13 months
Sampling rate	5 min	5 min	5 min
Total samples	≈7M	≈17M	≈22M
Target sensors	10 (origins)	10 (origins)	10 (origins)
Availability	Open	Open	Proprietary

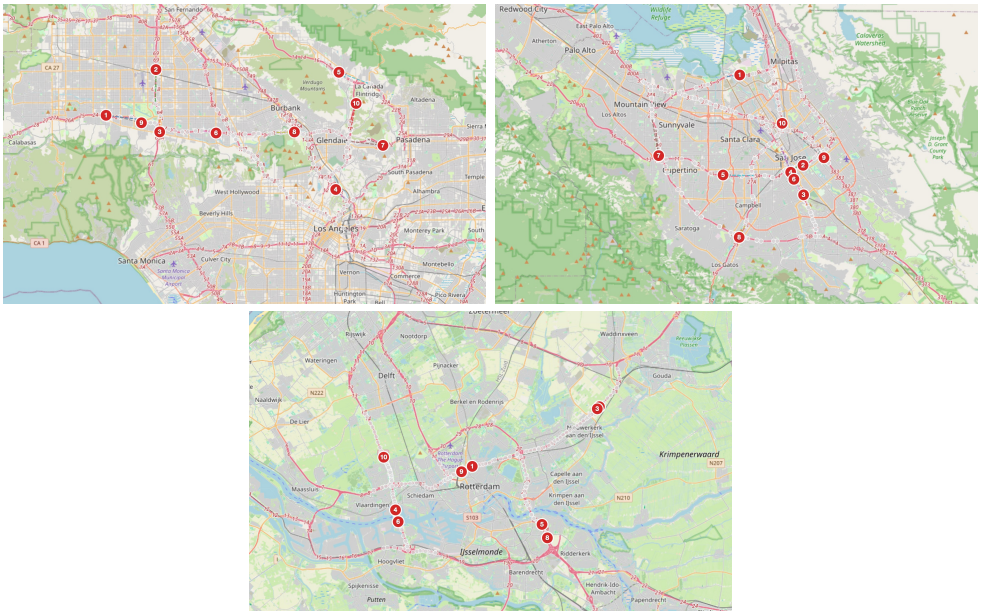


Fig. 2 The locations of the 10 target origin sensors (numbered red markers) in the METR-LA (top-left), PEMS-BAY (top-right), and NDW (bottom) networks.

5.2 Protocol

All data are split chronologically into 70% training, 10% validation, and 20% test; the validation split is used exclusively for hyperparameter, feature-budget, and decision-threshold selection, and the test split exclusively for the reported results. The target sensors are the 10 congestion origins per dataset mined in Section 4.1; the prediction horizons are $h \in \{10, 30, 60\}$ minutes; and all metrics (F_1, transition-F_1, TRSP, precision, recall) are averaged over the 10 target sensors. The complete experimental code is publicly available in our accompanying repository.[3] All experiments were conducted on an Intel Xeon server with an NVIDIA A100 GPU (40 GB); the deep baselines use the GPU for both training and inference, while the XGBoost models use the GPU for training only and run inference on CPU.

5.3 Baseline Methods

We compare against two groups of baselines: three naive persistence predictors, which calibrate what "no model at all" achieves, and three published state-of-the-art deep spatio-temporal architectures.

Naive Persistence Baselines

- **Naive-Current:** $\hat{\mathcal{Y}}(s, t, h) = \mathcal{C}(s, t)$, i.e., the state within the next h minutes will be equal to the current state. This is the canonical persistence forecast.
- **Naive-Yesterday:** $\hat{\mathcal{Y}}(s, t, h) = \mathcal{C}(s,\ t - (24\text{h} - h))$, daily periodicity: the prediction window is matched against the same window yesterday.
- **Naive-LastWeek:** $\hat{\mathcal{Y}}(s, t, h) = \mathcal{C}(s,\ t - (7\text{d} - h))$, weekly periodicity, matching the same window one week earlier.

These baselines cost nothing to train and effectively nothing to run. Following [9, 11], we treat them as the floor that any learned model must clearly beat *on transition-aware metrics* before its complexity is justified.

Deep Spatio-Temporal Baselines

- **DGCRN.** DGCRN [8] embeds graph convolutions inside GRU-style recurrent cells arranged as a sequence-to-sequence encoder–decoder. At every time step, in addition to a static, distance-based graph convolution, a *dynamic* adjacency matrix is generated from the current input and hidden state (masked by the physical topology), allowing the spatial dependencies to vary with the evolving traffic regime (e.g., between peak and off-peak conditions).
- **Graph WaveNet.** Graph WaveNet [6] couples stacked dilated causal convolutions (which expand the temporal receptive field exponentially with depth) with graph convolutions over two graphs, the predefined, distance-based adjacency and a *self-adaptive* adjacency learned end-to-end from node embeddings. The adaptive graph lets the model discover dependencies that the physical topology misses.
- **MTGNN.** MTGNN [7] is a general multivariate-forecasting architecture that learns the graph itself, where a graph-learning layer produces a sparse directed adjacency

[3] https://github.com/giorgostheo/congestion_journal

from node embeddings, processed by mix-hop graph propagation layers interleaved with dilated inception temporal convolutions. It represents the strongest "no prior graph needed" baseline.

Adaptation to congestion prediction

All three models are trained exactly as in their original papers (as multi-step speed regressors) on each dataset, with an input window of 36 steps (180 minutes) and 12 output steps (60 minutes), 100 epochs, batch size 64, and dropout 0.3, using the authors' published implementations. To produce congestion predictions we apply the operational definition of Section 3 to their forecasted speeds. The predicted early-warning label for horizon h is positive iff any forecasted speed within the window falls below the sensor's congestion threshold. Since a fixed threshold may not be optimal under regression bias, we tune a multiplicative threshold scaling factor in $[0.5, 1.5]$ (step 0.01) per model on the validation split, maximizing the same TRSP objective used for our models. This regression-then-threshold protocol follows common practice [25, 26] and gives the deep baselines every advantage we give our own models: identical labels, identical splits, identical target sensors, and identical transition-aware threshold tuning.

5.4 Accuracy Results

Table 2 reports F_1, transition-F_1 (T-F_1), and TRSP at the 10-, 30-, and 60-minute horizons for all twelve method configurations, on all three datasets. Figures 3 and 4 summarize plain F_1 and TRSP across all three horizons by method family. The main findings are discussed below.

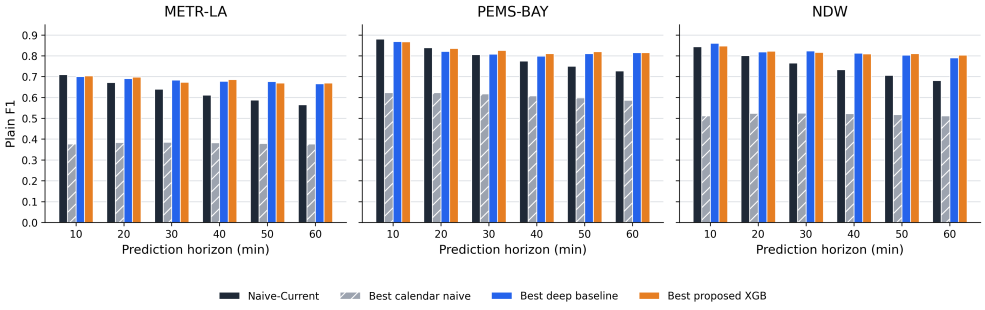


Fig. 3 Plain F_1 across prediction horizons, per dataset. Bars show Naive-Current and the best method within each family (calendar naive, deep baseline, proposed XGBoost) for that dataset and horizon. Note how the persistence baseline remains competitive at short horizons despite having no predictive skill.

(1) Plain F_1 is indeed misleading

Naive-Current attains $F_1 = 0.710$ on METR-LA, 0.881 on PEMS-BAY and 0.843 on NDW at the 10-minute horizon, the *highest* plain F_1 of any method, learned or not, on

Table 2 F_1, transition-F_1 (T-F_1) and TRSP at the 10/30/60-minute horizons, per dataset (mean over the 10 origin sensors; best value per column in bold, within each dataset block). w is the transition training weight of Section 4.3.

		h = 10 min			h = 30 min			h = 60 min		
Dataset	**Method**	F_1	T-F_1	TRSP	F_1	T-F_1	TRSP	F_1	T-F_1	TRSP
METR-LA	Naive-Current	**0.710**	0.000	0.398	0.640	0.000	0.356	0.566	0.000	0.337
	Naive-Yesterday	0.377	0.307	0.324	0.385	0.320	0.290	0.377	0.332	0.273
	Naive-LastWeek	0.111	0.142	0.118	0.117	0.147	0.120	0.119	0.150	0.127
	DGCRN	0.608	0.435	0.574	0.650	0.516	0.511	0.641	0.597	0.516
	Graph WaveNet	0.700	0.390	**0.614**	**0.684**	0.483	0.533	0.667	0.576	0.542
	MTGNN	0.631	0.393	0.578	0.644	0.523	0.506	0.625	0.610	0.512
	XGB-Local (w=1)	0.696	0.489	0.587	0.663	**0.627**	0.550	0.661	0.634	0.551
	XGB-Local (w=3)	0.674	**0.522**	0.576	0.671	0.622	**0.556**	0.663	0.636	**0.553**
	XGB-Global (w=1)	0.664	0.476	0.561	0.662	0.613	0.545	0.654	0.669	0.542
	XGB-Global (w=3)	0.652	0.500	0.560	0.667	0.604	0.541	0.648	0.666	0.538
	XGB-Transfer (w=1)	0.705	0.462	0.585	0.674	0.600	0.535	**0.669**	0.644	0.542
	XGB-Transfer (w=3)	0.665	0.517	0.572	0.658	0.599	0.537	0.653	**0.752**	0.537
PEMS-BAY	Naive-Current	**0.881**	0.000	0.530	0.805	0.000	0.502	0.728	0.000	0.481
	Naive-Yesterday	0.498	0.437	0.428	0.508	0.517	0.421	0.492	0.556	0.387
	Naive-LastWeek	0.623	0.525	0.523	0.617	0.585	0.489	0.587	0.618	0.465
	DGCRN	0.828	0.644	0.725	0.746	0.772	0.652	0.779	0.768	0.638
	Graph WaveNet	0.868	0.574	**0.757**	0.798	0.736	0.675	0.799	0.792	0.676
	MTGNN	0.869	0.583	0.753	0.808	0.728	0.694	0.815	0.773	0.690
	XGB-Local (w=1)	0.867	0.605	0.751	0.820	0.761	0.716	**0.816**	0.829	**0.724**
	XGB-Local (w=3)	0.848	0.629	0.741	0.795	**0.787**	0.707	0.802	**0.846**	0.722
	XGB-Global (w=1)	0.840	0.618	0.739	0.802	0.780	0.716	0.805	0.822	0.711
	XGB-Global (w=3)	0.837	**0.650**	0.742	0.793	0.768	0.704	0.799	0.822	0.715
	XGB-Transfer (w=1)	0.864	0.603	0.749	**0.827**	0.755	**0.719**	0.812	0.841	0.721
	XGB-Transfer (w=3)	0.852	0.638	0.747	0.806	0.773	0.714	0.807	0.833	0.712
NDW	Naive-Current	0.843	0.000	0.485	0.765	0.000	0.462	0.682	0.000	0.447
	Naive-Yesterday	0.445	0.434	0.413	0.453	0.466	0.398	0.439	0.479	0.375
	Naive-LastWeek	0.513	0.480	0.463	0.525	0.522	0.441	0.512	0.527	0.414
	DGCRN	0.805	0.627	0.696	0.747	0.766	0.641	0.725	0.813	0.633
	Graph WaveNet	0.839	0.607	0.727	0.805	0.744	0.681	0.785	0.801	0.670
	MTGNN	**0.861**	0.606	0.746	**0.824**	0.735	0.689	0.791	0.811	0.678
	XGB-Local (w=1)	0.831	0.616	0.728	0.798	0.757	0.684	0.796	0.790	0.681
	XGB-Local (w=3)	0.825	0.625	0.723	0.781	0.776	0.683	0.787	0.790	0.674
	XGB-Global (w=1)	0.847	0.655	0.755	0.816	0.773	**0.708**	**0.804**	**0.813**	0.696
	XGB-Global (w=3)	0.843	**0.688**	**0.760**	0.804	**0.786**	0.705	0.802	0.811	**0.698**
	XGB-Transfer (w=1)	0.845	0.601	0.730	0.796	0.765	0.684	0.802	0.784	0.679
	XGB-Transfer (w=3)	0.825	0.642	0.727	0.789	0.784	0.690	0.796	0.804	0.683

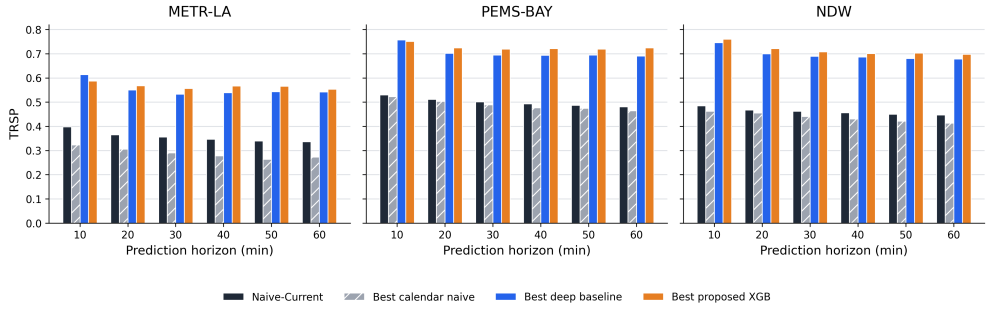


Fig. 4 TRSP across prediction horizons, per dataset. Bars show Naive-Current and the best method within each family (calendar naive, deep baseline, proposed XGBoost) for that dataset and horizon. Under the transition-aware metric the ranking changes substantially relative to Figure 3.

METR-LA and PEMS-BAY, and within 0.018 of the best on NDW (0.843 vs. 0.861 for MTGNN). Its transition-F_1, on the other hand, is exactly 0 everywhere, since it never anticipates a transition under the windowed target. A ranking led by short-horizon plain F_1 would therefore suggest that current-state persistence is competitive with learned models, even though it contributes no transition skill. This behavior is not incidental: since every learned model's decision threshold is tuned for the transition-aware objective, each of them deliberately gives up a little plain F_1 in exchange for transition skill, a trade the persistence baseline never makes. Plain F_1 alone therefore gives an incomplete ranking, one that is only corrected when it is read alongside transition-F_1 and TRSP.

(2) Under TRSP, the proposed models match or surpass the deep baselines on every dataset

Aggregated over horizons, the proposed models hold the best TRSP on all three datasets, clearly on the two cleaner networks and narrowly on the noisier METR-LA. NDW shows the clearest short-range advantage, where XGB-Global (w=3) reaches TRSP 0.760 at h=10 versus 0.746 for the best deep baseline (MTGNN), and the global model (in one weighting or the other) holds the best TRSP at every horizon (mean over horizons: 0.715 vs. 0.697). On PEMS-BAY, the cleanest, most predictable dataset, the deep baselines are at their strongest at the shortest horizon, where Graph WaveNet edges ahead at h=10 (0.757 vs. 0.751 for XGB-Local w=1), but from 20 minutes onward every proposed regime overtakes the deep field (0.704–0.719 vs. 0.694 at h=30; 0.711–0.724 vs. 0.690 at h=60), yielding a mean-over-horizons TRSP of 0.726 (XGB-Local w=1) against 0.705 for MTGNN. METR-LA is the closest comparison. The best proposed and best deep models are nearly tied on the mean (0.557 for XGB-Local w=1 vs. 0.554 for Graph WaveNet), with Graph WaveNet ahead at h=10 (0.614 vs. 0.587) and the proposed models ahead from h=30 onward (0.556 vs. 0.533). Thus, the deep architectures' advantage is concentrated at the shortest horizons, while the proposed models remain competitive at a substantially lower computational cost (Section 5.6). The proposed models also dominate the naive baselines on every transition-aware metric, which, following the argument presented in Section 5.3, is the minimum bar for justifying any learned model.

(3) The proposed models are consistent across datasets

The proposed models' transition-F_1 improves markedly with horizon (e.g., NDW: 0.69 at h=10 to 0.81 at h=60 for XGB-Global w=3; PEMS-BAY: 0.61 to 0.83 for XGB-Local w=1), reflecting that longer windows make onsets predictable from recurring patterns, which the profile and calendar-lag features capture directly. The deep baselines show more dataset-specific variation. Graph WaveNet is the best deep model on the noisy METR-LA (mean TRSP 0.554 vs. 0.530 for DGCRN and 0.529 for MTGNN), whereas MTGNN tops the deep field on PEMS-BAY and NDW (mean TRSP 0.705 and 0.697) but is tied for worst on METR-LA; DGCRN is the weakest deep model on both cleaner datasets. The proposed regimes keep essentially the same ranking and the same accuracy profile on all three networks, which matters for practitioners who cannot afford a per-dataset architecture search.

5.5 Ablations: Transition Weighting and Deployment Regimes

Transition weight

Across the 54 dataset/regime/horizon configurations, moving from w=1 to w=3 raises transition-F_1 in 43 of 54 cases (mean +0.016, up to +0.107) at a mean cost of 0.010 plain F_1 (Figure 5). The net effect on TRSP, however, is essentially neutral (improvement in 25 of 54 cases, mean −0.001). Because the decision threshold of every model is already tuned to maximize validation TRSP, much of what reweighting used to buy is now absorbed by the threshold selection itself. Transition weighting thus no longer changes the headline ranking, but it remains a useful operational knob that shifts the operating point toward transition recall at a given threshold, and we recommend w=3 whenever missed onsets are costlier than false alarms, and w=1 otherwise.

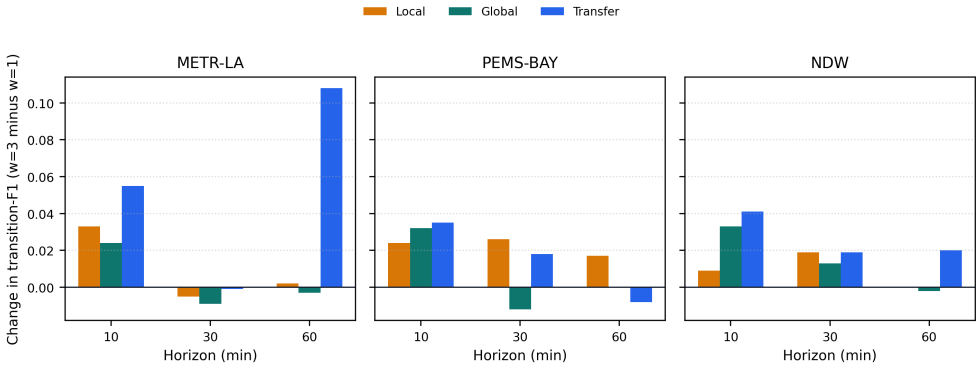


Fig. 5 Effect of the transition training weight (w=1 vs. w=3) on transition-F_1 at the reported 10/30/60-minute horizons.

Global vs. local vs. transfer model

The most accurate regime depends on the network. On NDW the global model leads clearly (mean-over-horizons TRSP 0.715, a +0.023 margin over the local model), confirming that on its corridor topology the cross-sensor features carry real, upstream-propagating signal. On METR-LA and PEMS-BAY the local model is level with or marginally ahead of the global one (METR-LA 0.557 vs. 0.550; PEMS-BAY 0.726 vs. 0.720), indicating that on those networks the target sensors' own histories already saturate the achievable accuracy. The transfer model retains essentially all of the best regime's accuracy (within 0.002–0.021 mean TRSP on every dataset) with a hard 20-feature budget and, as shown next, the lowest computational footprint of any learned method in this study.

5.6 Efficiency

Table 3 reports the training times and per-sensor inference latencies, Table 4 reports the peak memory and the memory–time occupancy of training, and Figure 6 places

Table 3 Computational cost: total training time (minutes, all horizons and target sensors) and mean per-sensor inference latency (μs) per dataset (best value per dataset and metric in bold). XGBoost training is timed over the full pipeline and covers both w settings jointly (the base feature ranking and selection are shared across w); inference is reported separately for w=1 and w=3, whose selected tree counts differ.

	METR-LA		**PEMS-BAY**		**NDW**	
Method	Train (min)	Inf. (μs)	Train (min)	Inf. (μs)	Train (min)	Inf. (μs)
DGCRN	61.1	3.15	192.2	2.75	175.5	3.25
Graph WaveNet	202.9	7.40	517.4	8.39	636.3	7.36
MTGNN	**49.4**	1.69	112.1	1.76	146.6	1.55
XGB-Local (w=1)	80.2	1.46	69.2	0.78	120.8	0.77
XGB-Local (w=3)		2.34		1.38		0.90
XGB-Global (w=1)	87.6	1.41	81.1	0.86	157.4	0.76
XGB-Global (w=3)		1.61		0.94		1.05
XGB-Transfer (w=1)	55.4	**0.87**	**49.1**	**0.58**	**62.4**	**0.34**
XGB-Transfer (w=3)		1.14		0.59		0.45

every method on the accuracy–latency plane. We discuss training time, inference latency, memory occupancy, and the overall cost–accuracy trade-off in turn.

Regarding *training time* (see Table 3, 'Train' columns): a separate model is trained per horizon (and, for XGB-Local and XGB-Transfer, per target sensor as well), which mirrors the setup of the deep baselines, since each of them is trained as one multi-step model per configuration and run through a single forward pass per configuration at inference. The XGBoost training time is shared across w=1 and w=3 because the common feature construction, ranking, and selection stages are run once for both transition-weight settings. The proposed pipeline, including its feature construction, feature selection, and staged hyperparameter search, trains all of these in 49–157 minutes per dataset, versus 49–636 minutes for the deep baselines. Against the two heavier architectures the advantage is large: the transfer regime trains 3.7–10.5× faster than Graph WaveNet and 1.1–3.9× faster than DGCRN. MTGNN is the one deep baseline with comparable wall-clock cost (the transfer regime is still 0.9–2.4× faster), but, as reported in Section 5.4, it trails the proposed models on mean TRSP on every dataset, and, as the memory-occupancy point below shows, equal wall-clock on shared hardware conceals very unequal resource consumption. Fast training has direct practical value here: traffic is non-stationary, so any deployed system must be retrained periodically, and the training cost recurs.

As for *inference* (see Table 3, 'Inf.' columns): per-sensor latencies of the XGBoost models are in the sub-microsecond to low-microsecond range, with the transfer model at 0.34–1.14 μs, i.e., 1.5–22× faster than the deep baselines, and achieved on CPU-class resources rather than a GPU.

Turning to *memory occupancy* (see Table 4, 'Peak' columns): wall-clock alone implicitly assumes the underlying hardware is equal and free, and this is where the comparison on a shared 40 GB A100 misleads. We therefore also report the *memory–time occupancy* of training, GB · h = peak device memory × training wall-clock (the

same unit in which cloud providers bill resource use), measured with short instrumented probes on the same machine as the timing runs (one epoch for the deep baselines and one prediction horizon for the proposed pipeline, since peak memory is reached within the first epoch and is identical across horizons, respectively). As reported in Table 4, the deep baselines allocate 2.1–14.3 GB at peak (and reserve up to 20.8 GB through the allocator, more than half the A100), whereas every proposed regime stays below 0.8 GB, of which roughly 0.5 GB is the fixed CUDA runtime context. The resulting occupancies differ by up to two orders of magnitude, from 0.42–1.83 GB·h for the proposed regimes to 1.7–123.4 GB·h for the deep baselines. The comparison is sharpest where wall-clock had suggested parity, since MTGNN's training occupies 2.1–6.1× the memory–time of XGB-Global on every dataset, and Graph WaveNet occupies up to 294× that of XGB-Transfer. In practice, the proposed models fit in the memory envelope of any entry-level accelerator, while Graph WaveNet's PEMS-BAY run requires a device of at least the 24 GB class at the published batch size.

Finally, the *transfer* regime sits on the cost–accuracy Pareto frontier (see Table 4, 'GB·h' columns, as well as Figure 6). It is the cheapest of the proposed regimes to train (0.42–0.54 GB·h), the fastest learned method at inference on every dataset, the smallest in memory (20 features, hundreds of shallow trees), and within 0.002–0.021 mean TRSP of the best model on every dataset. For resource-constrained deployments (roadside units, municipal hardware without GPUs), it is the configuration we recommend.

Table 4 GPU memory occupancy of training: peak device memory (GB) and memory–time occupancy (GB·h = peak memory × training wall-clock from Table 3; best value per dataset and metric in bold). Deep-baseline peaks come from PyTorch's `torch.cuda.max_memory_allocated` call, which reports the peak bytes actually allocated to tensors; the number in parentheses is the caching-allocator (reserved) peak, i.e., the larger pool PyTorch holds back from the driver so it doesn't have to keep asking for more. XGBoost peaks are per-process totals from `nvidia-smi`, which include the ∼0.5 GB CUDA runtime context, an asymmetry that favors the baselines. XGBoost peaks are shown for w=1; w=3 peaks are identical.

	METR-LA		**PEMS-BAY**		**NDW**	
Method	Peak (GB)	GB·h	Peak (GB)	GB·h	Peak (GB)	GB·h
DGCRN	3.45 (3.86)	3.5	5.81 (6.61)	18.6	3.45 (3.86)	10.1
Graph WaveNet	9.13 (18.61)	30.9	14.30 (20.83)	123.4	9.09 (14.43)	96.4
MTGNN	2.07 (2.88)	1.7	3.23 (4.51)	6.0	2.05 (3.63)	5.0
XGB-Local	**0.52**	0.69	**0.52**	0.60	0.53	1.07
XGB-Global	0.56	0.82	0.73	0.99	0.70	1.83
XGB-Transfer	**0.52**	**0.48**	**0.52**	**0.42**	**0.52**	**0.54**

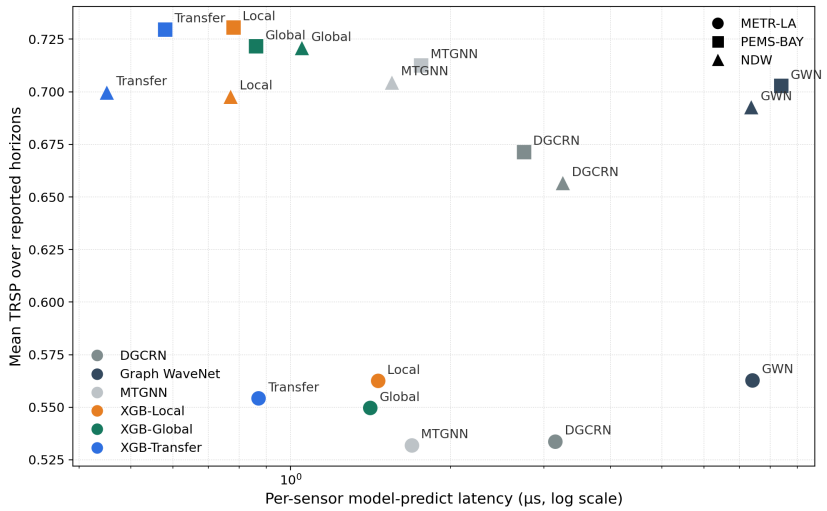


Fig. 6 Accuracy (mean TRSP over the reported 10/30/60-minute horizons) versus per-sensor inference latency (log scale, Table 3); for the proposed models, the transition weight (w=1 or w=3) with the better mean TRSP is shown per regime. The proposed models, and especially XGB-Transfer, dominate the Pareto frontier.

5.7 Streaming feature generation and end-to-end inference cost

The inference latencies of Table 3 measure only the *model-predict* step. For an edge deployment that scores a live sensor stream, the per-timestep cost also includes *feature generation*, and for the proposed models this is the dominant term, since the prediction itself is only a shallow tree-ensemble traversal. We therefore microbenchmark the complete per-timestep cost, feature construction plus prediction, and place it on the same footing as the deep baselines.

Streaming feature extractor

The batch feature construction used during training, vectorized over the entire series, does not reflect edge cost, because at inference one receives a single new observation per sensor per timestep and must update the feature vector incrementally rather than recompute windows. We therefore reimplement the inference-time feature set as an $O(1)$*-per-timestep streaming extractor* that maintains, per sensor, only bounded state: ring buffers for the calendar lags (a 2016-step buffer covers the 7-day lag, ≈8 KB per sensor), running sums and sums-of-squares for the rolling mean and standard deviation, a monotonic deque for the rolling minimum, exponential-moving-average recurrences, $O(1)$ counters for the time-in-state features, and constant-time table lookups for the time-of-week profiles; the global model's spatial features reduce to an inverse-distance–weighted sum over the target's hop-neighbors and their wave-aligned lags. The extractor is JIT-compiled to native code (via `numba`) and runs single-threaded on CPU; its output is validated column-for-column against the batch builders (maximum absolute deviation $< 10^{-4}$, attributable to the summation order of the rolling standard deviation). For the deep baselines the analogous per-timestep preprocessing

(assembling the $T \times N \times C$ sequence window and applying the input normalization) is JIT-compiled and timed in the same way.

Protocol and caveats

Table 5 reports per-(node, timestep) costs, measured on one of our three networks (PEMS-BAY) as a representative case; the mechanics of the benchmark, and the ordering it produces across methods, do not depend on which network is used, since they follow from the shape of each architecture (whole-graph forward pass versus per-sensor feature lookup) rather than from any dataset-specific property. Feature generation is measured on a single CPU thread; model-predict reuses the per-sensor latencies of Section 5.6 (the GNN values are those of Table 3; XGBoost values use the mean of the w=1 and w=3 latencies reported there). Two caveats should be kept in mind when interpreting these figures. (i) The two cost columns are measured on different devices (CPU feature generation; model-predict on the GPU for the GNNs and on CPU for the XGBoost models, from the timing runs), so the totals are *indicative* rather than a single-device measurement. (ii) Costs are normalized *per node*. A GNN forward pass and its window assembly serve the whole graph at once, so their per-node cost is the whole-graph cost divided by N, whereas the single-sensor XGBoost cost is incurred per target sensor and does not amortize. The per-node figures therefore represent the deep baselines' *best case*; in the common edge regime where only a handful of origin sensors are monitored, a GNN still pays its full whole-graph window and forward, while the XGBoost models pay only for the sensors of interest.

Table 5 Indicative end-to-end per-timestep inference cost, measured on PEMS-BAY as a representative network: JIT streaming feature generation (single CPU thread) plus model prediction (best value per metric in bold, excluding Naive). Costs are per (node, timestep); GNN feature generation and prediction are amortized over all N=325 nodes of one forward pass (see caveats). The relative ordering of methods follows from each architecture's structure and is not specific to this dataset.

Method	Feat.-gen (CPU, μs)	Predict (μs)	$\approx$ Total (μs)	Bottleneck
Naive	0.005	0.00	0.005	n/a
DGCRN	0.10	2.75	2.9	model
Graph WaveNet	0.10	8.39	8.5	model
MTGNN	**0.10**	**1.76**	**1.9**	model
XGB-Local	2.26	1.08	3.3	feature
XGB-Global	2.61	0.90	3.5	feature
XGB-Transfer	2.17	0.59	2.8	feature

The table supports a few observations. JIT compilation keeps XGBoost feature generation in the low-microsecond range, at approximately 2.2–2.6 μs per target and timestep, depending on the feature subset. Once both stages are counted, the transfer model remains competitive with DGCRN and substantially below Graph WaveNet,

while the local and global regimes pay extra for their larger generated feature sets. Note also that this accounting is still the deep baselines' most favorable regime, as the GNN costs are amortized across the full graph, whereas in the realistic edge setting of monitoring a few origin sensors, a GNN still pays its whole-graph forward pass and the XGBoost models pay only for the sensors of interest. None of these observations depends on which of our networks we measure on, since they follow from how each architecture is built rather than from any property specific to PEMS-BAY.

6 Conclusions & Future Work

In this paper, we studied short-term traffic congestion prediction from an operational, early-warning perspective, and questioned how models for this task are usually built and evaluated. We proposed TRSP, a transition-aware metric that exposes the failure modes of plain and transition-only F_1. We also proposed a reproducible, graph-based procedure for mining congestion-origin sensors as evaluation targets, and a feature-engineered, transition-weighted XGBoost framework that matches or surpasses state-of-the-art spatio-temporal graph neural networks at a fraction of the computational cost. The main lessons of this study are the following. Evaluation practice is central to the problem: plain F_1 is dominated by state persistence and can make trivial baselines appear competitive, while transition-only scores move too far in the other direction by ignoring stable-period reliability. The proposed TRSP metric, which harmonically combines transition detection with stable-period reliability, exposes both failure modes at once and is therefore a useful reporting and model-selection metric for congestion classification studies. Moreover, *where* one evaluates can also be made principled, as congestion origins mined from the data via a simple graph criterion provide reproducible, operationally meaningful target sensors, validated on NDW against independent expert knowledge. Finally, with systematic, leakage-safe feature engineering and transition-weighted training, gradient-boosted trees match or surpass state-of-the-art spatio-temporal graph neural networks on transition-aware accuracy across three diverse datasets, while training up to an order of magnitude faster than the heavier architectures, occupying up to two orders of magnitude less GPU memory–time (0.42–1.83 versus 1.7–123 GB·h), and predicting up to 22× faster per sensor; the distilled 20-feature transfer model brings this within reach of GPU-less municipal hardware. Because the underlying features are named and physically grounded rather than learned embeddings, the resulting models are also easier for a traffic engineer to inspect than the deep baselines, down to what a single sensor's model is relying on.

We plan to extend this work in several directions. One is edge deployment: building on the preliminary streaming microbenchmark of Section 5.7, we intend to compile the transfer model's trees to native code and benchmark single-sample latency, memory, and energy on representative roadside hardware under a unified single-device protocol, across both the whole-graph and the realistic few-target monitoring regimes. Another is sustained operation under drift: NDW provides online access to its measurements, so with suitable processing multi-year datasets can be assembled, long enough to study seasonal periodicity and long-term drift and to evaluate periodic and incremental retraining policies, for which the framework's low training cost is a key

enabler. We also aim to broaden the validation, applying the origin-mining procedure and the transition-aware protocol to additional networks, including urban arterials, and extending the framework with cost-sensitive variants of TRSP that weigh onsets and sustained-clear transitions according to intervention costs.

Acknowledgements. This work was supported partially by the Horizon Europe R&I programme EMERALDS under the GA No. 101093051 and the University of Piraeus Research Center (UPRC). The authors also acknowledge Mr. Erik-Sander Smits (ARANE, NL) for his expert feedback.

Declaration on Generative AI

During the preparation of this work, the authors used LLMs in order to conduct grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Mokbel, M., Sakr, M., Xiong, L., Züfle, A., Almeida, J., Anderson, T., Aref, W., Andrienko, G., Andrienko, N., Cao, Y., Chawla, S., Cheng, R., Chrysanthis, P., Fei, X., Ghinita, G., Graser, A., Gunopulos, D., Jensen, C.S., Kim, J.-S., Kim, K.-S., Kröger, P., Krumm, J., Lauer, J., Magdy, A., Nascimento, M., Ravada, S., Renz, M., Sacharidis, D., Salim, F., Sarwat, M., Schoemans, M., Shahabi, C., Speckmann, B., Tanin, E., Teng, X., Theodoridis, Y., Torp, K., Trajcevski, G., Kreveld, M., Wenk, C., Werner, M., Wong, R., Wu, S., Xu, J., Youssef, M., Zeinalipour, D., Zhang, M., Zimányi, E.: Mobility data science: Perspectives and challenges. ACM Transactions on Spatial Algorithms and Systems **10**(2) (2024) https://doi.org/10.1145/3652158

[2] Nguyen, H., Liu, W., Chen, F.: Discovering congestion propagation patterns in spatio-temporal traffic data. IEEE Transactions on Big Data **3**(2), 169–180 (2017) https://doi.org/10.1109/TBDATA.2016.2587669

[3] Nagy, A.M., Simon, V.: Improving traffic prediction using congestion propagation patterns in smart cities. Advanced Engineering Informatics **50**, 101343 (2021) https://doi.org/10.1016/j.aei.2021.101343

[4] Li, Y., Yu, R., Shahabi, C., Liu, Y.: Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting (2018). https://doi.org/10.48550/arXiv.1707.01926

[5] Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, pp. 3634–3640. International Joint Conferences on Artificial Intelligence Organization, Stockholm, Sweden (2018). https://doi.org/10.24963/ijcai.2018/505

[6] Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp. 1907–1913. International Joint Conferences on Artificial Intelligence Organization, Macao, China (2019). https://doi.org/10.24963/ijcai.2019/264

[7] Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., Zhang, C.: Connecting the dots: Multivariate time series forecasting with graph neural networks. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 753–763. ACM, Virtual Event, CA, USA (2020). https://doi.org/10.1145/3394486.3403118

[8] Li, F., Feng, J., Yan, H., Jin, G., Yang, F., Sun, F., Jin, D., Li, Y.: Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. ACM Transactions on Knowledge Discovery from Data **17**(1), 1–21 (2023) https://doi.org/10.1145/3532611

[9] Manibardo, E.L., Lana, I., Del Ser, J.: Deep learning for road traffic forecasting: Does it make a difference? IEEE Transactions on Intelligent Transportation Systems **23**(7), 6164–6188 (2022) https://doi.org/10.1109/TITS.2021.3083957

[10] Naidu, G., Zuva, T., Sibanda, E.: A review of evaluation metrics in machine learning algorithms. In: Silhavy, R., Silhavy, P. (eds.) Artificial Intelligence Application in Networks and Systems, pp. 15–25. Springer, Cham, Switzerland (2023). https://doi.org/10.1007/978-3-031-35314-7_2

[11] Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The m4 competition: Results, findings, conclusion and way forward. International Journal of Forecasting **34**(4), 802–808 (2018) https://doi.org/10.1016/j.ijforecast.2018.06.001

[12] Grinsztajn, L., Oyallon, E., Varoquaux, G.: Why do tree-based models still outperform deep learning on typical tabular data? In: Advances in Neural Information Processing Systems, vol. 35. New Orleans, LA, USA, pp. 507–520 (2022). https://doi.org/10.48550/arXiv.2207.08815

[13] Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you need. Information Fusion **81**, 84–90 (2022) https://doi.org/10.1016/j.inffus.2021.11.011

[14] Treiber, M., Kesting, A.: Traffic Flow Dynamics: Data, Models and Simulation. Springer, Berlin, Germany (2013). https://doi.org/10.1007/978-3-642-32460-4

[15] Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. ACM, San Francisco, CA, USA (2016). https://doi.org/10.1145/2939672.2939785

[16] Afrin, T., Yodo, N.: A survey of road traffic congestion measures towards a sustainable and resilient transportation system. Sustainability **12**(11) (2020) https://doi.org/10.3390/su12114660

[17] Alghamdi, T., Elgazzar, K., Bayoumi, M., Sharaf, T., Shah, S.: Forecasting traffic congestion using arima modeling. In: 2019 15th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 1227–1232. IEEE, Tangier, Morocco (2019). https://doi.org/10.1109/IWCMC.2019.8766698

[18] Liu, Y., Zheng, H., Feng, X., Chen, Z.: Short-term traffic flow prediction with conv-lstm. In: 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–6. IEEE, Nanjing, China (2017). https://doi.org/10.1109/WCSP.2017.8171119

[19] Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., Li, H.: T-gcn: A temporal graph convolutional network for traffic prediction. IEEE Transactions on Intelligent Transportation Systems **21**(9), 3848–3858 (2020) https://doi.org/10.1109/TITS.2019.2935152

[20] Guo, K., Hu, Y., Qian, Z., Liu, H., Zhang, K., Sun, Y., Gao, J., Yin, B.: Optimized graph convolution recurrent neural network for traffic prediction. IEEE Transactions on Intelligent Transportation Systems **22**(2), 1138–1149 (2021) https://doi.org/10.1109/TITS.2019.2963722

[21] Shao, Z., Wang, F., Xu, Y., Wei, W., Yu, C., Zhang, Z., Yao, D., Sun, T., Jin, G., Cao, X., Cong, G., Jensen, C.S., Cheng, X.: Exploring progress in multivariate time series forecasting: Comprehensive benchmarking and heterogeneity analysis. IEEE Transactions on Knowledge and Data Engineering **37**(1), 291–305 (2025) https://doi.org/10.1109/TKDE.2024.3484454

[22] Liu, X., Xia, Y., Liang, Y., Hu, J., Wang, Y., Bai, L., Huang, C., Liu, Z., Hooi, B., Zimmermann, R.: LargeST: A Benchmark Dataset for Large-Scale Traffic Forecasting (2023). https://doi.org/10.48550/arXiv.2306.08259

[23] Chen, M., Yu, X., Liu, Y.: Pcnn: Deep convolutional networks for short-term traffic congestion prediction. IEEE Transactions on Intelligent Transportation Systems **19**(11), 3550–3559 (2018) https://doi.org/10.1109/TITS.2018.2835523

[24] Ranjan, N., Bhandari, S., Zhao, H.P., Kim, H., Khan, P.: City-wide traffic congestion prediction based on cnn, lstm and transpose cnn. IEEE Access **8**, 81606–81620 (2020) https://doi.org/10.1109/ACCESS.2020.2991462

[25] Li, J., Li, J., Gong, Y.-j.: Multi-task learning for sparse traffic forecasting. arXiv preprint arXiv:2211.09984 (2022) arXiv:2211.09984 [cs.LG]

[26] Liu, Z., Jiang, Z., Xu, Z., Xiao, T., Xiao, Z., zhang, Y., Wang, H., Chen, S.: Spatio-Temporal Partial Sensing Forecast for Long-term Traffic (2025). https:

//doi.org/10.48550/arXiv.2408.02689

[27] Chen, C., Skabardonis, A., Varaiya, P.: Systematic identification of freeway bottlenecks. Transportation Research Record: Journal of the Transportation Research Board **1867**(1), 46–52 (2004) https://doi.org/10.3141/1867-06

[28] Tatbul, N., Lee, T.J., Zdonik, S., Alam, M., Gottschlich, J.: Precision and Recall for Time Series (2019). https://doi.org/10.48550/arXiv.1803.03639

[29] Lavin, A., Ahmad, S.: Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pp. 38–44. IEEE, Miami, FL, USA (2015). https://doi.org/10.1109/ICMLA.2015.141

[30] Burg, G.J.J., Williams, C.K.I.: An Evaluation of Change Point Detection Algorithms (2022). https://doi.org/10.48550/arXiv.2003.06222