UNITE: A Unified Framework for Accurate and Efficient Origin-Destination and Route Travel Time Estimation

Wei Tian
Hong Kong Polytechnic University
Hong Kong SAR, China
wei.tian@connect.polyu.hk

Jieming Shi*
Hong Kong Polytechnic University
Hong Kong SAR, China
jieming.shi@polyu.edu.hk

Man Lung Yiu
Hong Kong Polytechnic University
Hong Kong SAR, China
csmlyiu@comp.polyu.edu.hk

Abstract

Travel Time Estimation (TTE) on road networks is crucial for modern location-based services. There are two main variants: Origin-Destination TTE (ODTTE), which estimates travel time between an origin and destination given a departure time, and Route TTE (RTTE), which estimates travel time along a specified route. Existing approaches typically address ODTTE and RTTE separately. We propose UNITE, a unified framework that efficiently addresses both ODTTE and RTTE with a single training process. UNITE operates in a progressive, segment-wise manner. We first design a Progress State Encoder (PSE) that produces a shared state embedding capturing the origin, destination, departure-time context, sequential dependencies, and dynamic traffic conditions along the partially generated route. Specifically, PSE constructs segment-level travel-time histograms from historical trajectories and applies attention-based sequence modeling. Next, we develop a Segment Travel Time Estimator (mSTE) that leverages the shared state embedding, builds a local spatio-temporal travel graph to capture traffic dynamics, and employs a Mixture-of-Experts architecture to model heterogeneous traffic patterns for segment-level travel time estimation. We further introduce a Next Segment Prediction (NSP) module that predicts the next segment from the outgoing neighbors of the current one based on the shared state embedding, incorporating multi-hop look-ahead neighborhoods. Extensive experiments on large-scale real-world datasets demonstrate that UNITE consistently outperforms state-of-the-art methods in both accuracy and efficiency.

CCS Concepts

• **Information systems → Spatial-temporal systems**.

Keywords

Travel Time Estimation, Spatio-temporal Modeling, Road Networks

ACM Reference Format:
Wei Tian, Jieming Shi, and Man Lung Yiu. 2026. UNITE: A Unified Framework for Accurate and Efficient Origin-Destination and Route Travel Time Estimation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770855.3818012

*Corresponding Author.

KDD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2259-2/2026/08
https://doi.org/10.1145/3770855.3818012

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.20430075.

1 Introduction

Travel time estimation (TTE) aims to predict the time required to travel between two locations and is a core primitive in spatial data management over urban road networks. There are two main TTE problems depending on the available input: *Origin-Destination Travel Time Estimation (ODTTE)*, which takes as input only the origin o, destination d, and departure time t_o, with the actual route from o to d unknown; and *Route Travel Time Estimation (RTTE)*, which takes as input a full route R of road segments from o to d and a departure time t_o to estimate the travel time for traversing R.

Both ODTTE and RTTE are fundamental for a wide range of applications, including navigation [19, 35], on-demand delivery [21, 33], ride-hailing dispatch [13], logistics [45], and urban planning [10, 28, 32]. The ODTTE setting is prevalent in services where only OD pairs and departure times are available (e.g., food delivery platforms that do not record full routes), whereas RTTE queries arise when a route is already specified, such as pre-planned logistics routes.

Despite their closely related nature, existing studies largely investigate ODTTE [26, 27, 31, 51] and RTTE [2, 12, 17, 50] separately. Accurate and efficient TTEs remain challenging due to complex spatio-temporal dependencies and dynamic traffic conditions over large road networks. For ODTTE, the state-of-the-art method DOT [27] adopts a two-stage framework. First, it generates a pixelated trajectory from o to d by treating the map as a grid of pixels and utilizing diffusion models; second, it estimates travel time over the generated trajectory. DOT incurs high inference latency due to the computationally expensive diffusion operations. Moreover, this two-stage approach decouples route generation from travel time estimation, which however are intrinsically interrelated. Overlooking such mutual influence may lead to suboptimal accuracy. Meanwhile, studies like DeepOD [51] directly regress travel time from OD inputs with auxiliary features. They do not explicitly model underlying routes, which may overlook fine-grained patterns inherent in actual trajectories for more accurate estimation. For RTTE, it can be regarded as a special case of ODTTE where the full route is provided, along with the OD pair and departure time. Among [2, 12, 17, 50], the state-of-the-art RTTE method STHR [50] uses a sequence encoder–decoder framework to model heterogeneity, spatio-temporal correlations, and dynamic traffic conditions along the input route. However, RTTE methods do not need to address route generation, and thus may miss intrinsic patterns in route-related traffic dynamics. Besides, full routes may not always be available in practice.

In this paper, we propose UNITE, a UNIfied framework that simultaneously addresses both ODTTE and RTTE and requires only one-time training under the ODTTE setting. UNITE estimates travel time in a progressive, segment-wise manner, tightly coupling route generation and travel time estimation to enable mutual refinement at each step. Through a series of carefully designed technical components, UNITE achieves superior accuracy for both ODTTE and RTTE, while attaining high efficiency in training and inference.

Specifically, after converting the input ODT query $q = \langle o, d, t_o \rangle$ into an embedding $\mathbf{q}$, UNITE begins at the origin segment o and proceeds iteratively, segment by segment. At each current segment e_i, it invokes three modules that jointly perform travel time estimation and route generation. First, we develop the *Progressive State Encoder (PSE)*, which produces a shared latent embedding $\mathbf{h}_i$ that encodes the ODT context, sequential dependencies along the partially generated route, and dynamic traffic conditions on e_i as well as previously visited segments. To achieve this, we construct travel time histograms from historical trajectories to capture segment-level traffic patterns and integrate them with road-network structural embeddings using attention and GRU-based sequence modeling. The embedding $\mathbf{h}_i$ is then utilized by both the route generation and travel time estimation components. Thus, we design a *Segment Travel Time Estimator (mSTE)* that takes $\mathbf{h}_i$ and a local spatio-temporal travel graph centered at e_i as input to estimate the segment travel time ψ_i, which is then aggregated to obtain the overall travel time. The travel graph integrates the spatial neighborhood of e_i and multi-slot, multi-day travel time histograms to capture both short-term variation and periodicity. mSTE adopts a Mixture-of-Experts architecture, employing multiple lightweight experts over this graph. A noisy Top-k gating mechanism dynamically selects the most relevant experts for each segment, enabling the model to efficiently capture heterogeneous traffic patterns. Third, we introduce a *Next Segment Prediction (NSP)* module that predicts the next segment e_{i+1} from the outgoing neighbors of e_i using the progressive state $\mathbf{h}_i$. For effective next-segment selection, NSP considers the structural embedding of each candidate segment and a multi-hop look-ahead neighborhood to capture broader connectivity and directional context toward the destination. With e_{i+1}, UNITE proceeds to the next iteration, invoking PSE, mSTE, and NSP. For RTTE, UNITE bypasses NSP and applies PSE and mSTE along the input route.

We conduct extensive experiments on large real-world trajectory datasets, showing UNITE consistently outperforms existing ODTTE and RTTE methods in both accuracy and efficiency. For example, on XA dataset, UNITE answers 1,000 ODTTE queries in just 0.44 seconds–over 100× faster than the next-best competitor, which takes more than 45 seconds. To sum up, our contributions are:

- We propose UNITE, a unified framework that addresses both ODTTE and RTTE via progressive segment-wise inference, requiring only one-time training.
- We design PSE to generate a shared state embedding that captures sequential dependencies and dynamic traffic conditions, integrating historical travel time histograms.
- We develop mSTE, a segment travel time estimator with noisy Top-k gating and local spatio-temporal graphs to model heterogeneous traffic, and introduce NSP for next-segment prediction.
- Extensive experiments demonstrate that UNITE consistently outperforms existing methods in both accuracy and efficiency.

2 Preliminaries

Road Network. We model a road network as a directed graph $G = (V, E)$, where V is the set of intersections and E is the set of directed edges (road segments). Let $n = |E|$ and $m = |V|$ denote the number of segments and intersections, respectively. For a segment e_i, let $N_O(e_i)$ be the set of its out-going adjacent segments. We treat road segments as basic analysis units and formalize their connectivity via conjugate graph of road network: each node is a segment $e_i \in E$, and there is a directed edge from e_i to e_j iff $e_j \in N_O(e_i)$. This graph has node set E and encodes segment adjacency.

Trajectory and Route. A raw trajectory T is a sequence of GPS points, each denoted by $p_i = (lat, lng, t)$, where (lat, lng) are geographic coordinates and t is the timestamp. We focus on trajectories on a road network, *i.e.*, each GPS point is associated with a road segment. We apply a standard map-matching algorithm [49], widely used in trajectory analytics [19, 42], to project raw trajectories onto the road network. After map-matching, a trajectory T is represented as a route $R = \langle e_1, e_2, \ldots, e_\ell \rangle$ from origin segment $o = e_1$ to destination segment $d = e_\ell$, with departure time t_o and arrival time t_d; the total travel time is $\Phi^* = t_d - t_o$. For each segment e_i in R, we derive its travel time ψ_i from the entry and exit timestamps obtained via map-matching, such that the traveled segments do not overlap temporally. In the following, we use the term trajectory to refer to such map-matched trajectories unless otherwise specified.

Time Slot. To model periodic traffic patterns, we discretize time into fixed-length slots of duration δ. A day has 86400 seconds and thus $\frac{86400}{\delta}$ slots; a week has $7 \times \frac{86400}{\delta}$ slots. Given a timestamp t, we extract its weekday w, hour h, minute m, and second s, and compute its weekly time-slot id as $\lambda = \frac{86400}{\delta} \times w + \left\lfloor \frac{3600h+60m+s}{\delta} \right\rfloor$.

In this paper, we study two closely related TTE problems:

ODTTE. Given historical trajectory data $\mathcal{D}$ on a road network G, the ODT Travel Time Estimation (ODTTE) problem takes as input an origin segment o, a destination segment d, and a departure time t_o—that is, an ODT query $q = \langle o, d, t_o \rangle$—and aims to estimate the travel time Φ from o to d when departing at t_o.

RTTE. Given $\mathcal{D}$ on G, the Route Travel Time Estimation (RTTE) problem takes as input a route R from origin to destination and a departure time t_o—i.e., a route-time query $q_R = (R, t_o)$—and aims to estimate the travel time Φ for traversing R when departing at t_o.

RTTE is a special case of ODTTE where the route R is given; the origin o and destination d are the first and last segments of R, so the input (R, t_o) implicitly defines the ODT tuple $\langle o, d, t_o \rangle$.

3 Related Work

ODT Travel Time Estimation. Existing methods leverage historical trajectories for ODTTE [26, 27, 31, 40, 41, 51]. Early approaches, *e.g.*, [40], simply average historical travel times. More recent studies [26, 41, 51] capture complex spatio-temporal patterns to improve accuracy. MURAT [26] adopts multitask learning to jointly predict travel distance and time. DeepOD [51] learns OD–time embeddings and aligns them with trajectory embeddings via an auxiliary loss. However, these methods estimate travel time directly from OD-level features without explicitly modeling the underlying routes, thus missing fine-grained segment-level spatio-temporal patterns. The state-of-the-art method is DOT [27], which adopts a two-stage framework over pixelated trajectories (PiT) on a grid-based map: a

diffusion model first generates PiT conditioned on OD pairs, and a Masked Vision Transformer (MViT) [5] then estimates travel time from the generated PiT. DutyTTE [31] follows the same two-stage paradigm, but uses a reinforcement-learning policy to infer routes for ODT queries before predicting travel time on the inferred routes. The two-stage frameworks in [27, 31] decouple route generation from travel time estimation, thereby ignoring their mutual dependence, and require training separate models for two stages.

Route Travel Time Estimation. RTTE has been extensively studied [2, 8, 11, 12, 17, 38, 44, 50, 52]. The state-of-the-art STHR [50] uses a sequence encoder–decoder architecture to jointly model link heterogeneity, spatial proximity, periodicity, and dynamic traffic from historical trajectories. Earlier regression [18] and decomposition [43] methods have been surpassed by deep models that view a route as a segment sequence and use recurrent networks [3, 16] to capture spatio-temporal dependencies [11, 25, 39, 44, 46, 48], sometimes with auxiliary tasks to further improve accuracy [8, 12, 47]. HetETA [17] models the road network as a multi-relational graph using temporal and graph convolutions, while HierETA [2] leverages multi-view trajectory representations with hierarchical self-attention to model local traffic and trajectory structure. Other works instead predict the travel time of the remaining route given a partially traversed path [6, 7, 52]. Existing RTTE methods assume the full route is given, which is often unrealistic at query time. To our knowledge, UNITE is the first unified model that supports both ODTTE and RTTE, enabling efficient and accurate travel time estimation in both settings.

Route Planning. For ODTTE, a common way is to first generate a route using route-planning algorithms, then estimate its travel time, *e.g.*, by aggregating segment times or applying an RTTE model. Classical methods treat it as a shortest-path problem, minimizing a hand-crafted cost via Dijkstra [20], A* [22], and variants [4, 29, 30]. However, such costs are hard to calibrate and the generated routes often deviate from those actually taken. Recent learning-based methods, e.g., NMLR [19] and DRPK [35], leverage graph neural architectures [24] and problem decompositions to learn transition probabilities and perform greedy or destination-driven route search. Yet these two-step pipelines remain suboptimal in our experiments. In contrast, UNITE jointly models route generation and travel time estimation at the segment level within a single framework, yielding higher accuracy for both ODTTE and RTTE queries.

4 The UNITE Method

UNITE is a unified framework trained once under the ODTTE setting and reused at inference time for both ODTTE and RTTE. Unlike prior methods that decouple route generation and travel time estimation into separate models and predict a scalar for the entire route, UNITE performs progressive, segment-wise inference: at each step, it jointly estimates segment-level travel time and selects next segment, tightly coupling and reinforcing the two tasks.

As shown in Figure 1, given a query $\langle o, d, t_o \rangle$, UNITE first encodes it into an ODT embedding $\mathbf{q}$. Starting from o, UNITE proceeds segment by segment. At step i for the current segment e_i, UNITE invokes three modules: (i) the Progressive State Encoder (PSE) updates the current state embedding $\mathbf{h}_i$ by combining $\mathbf{q}$ with the spatio-temporal context of all previously visited segments; (ii) the Segment Travel Time Estimator (mSTE) predicts the travel time ϕ_i for e_i using $\mathbf{h}_i$, $\mathbf{q}$, and the spatio-temporal traffic context around e_i via a Mixture-of-Experts structure; and (iii) the Next Segment Prediction (NSP) module selects the next segment e_{i+1} from the out-going neighbors of e_i based on $\mathbf{h}_i$ and historical traffic patterns. With e_{i+1} as the current segment, UNITE iterates until reaching the destination. The overall travel time is obtained by summing the segment-level predictions, $\Phi = \sum_i \phi_i$. For RTTE, the route R is given and NSP is bypassed, while PSE and mSTE are applied along the given route in the same way. UNITE is trained end-to-end by jointly optimizing two objectives: a regression loss for segment-level travel time prediction, and a binary cross-entropy loss for next segment prediction.

4.1 ODT Encoding

We encode the origin o, destination d, and departure time t_o into a query embedding vector $\mathbf{q}$, which captures the spatial context of o and d in the road network G and the temporal features of t_o.

To encode o and d, we first obtain pre-trained structural embeddings $\mathbf{W}_G \in \mathbb{R}^{n \times d_0}$ using Node2Vec [15] on the conjugate graph of G, where nodes represent segments. We then use two learnable matrices $\mathbf{W}^o$ and $\mathbf{W}^d$ for origin and destination, both initialized with $\mathbf{W}_G$, and compute

$$\mathbf{e}_o = \mathbf{1}_o^\top \mathbf{W}^o, \quad \mathbf{e}_d = \mathbf{1}_d^\top \mathbf{W}^d, \tag{1}$$

where $\mathbf{1}_o, \mathbf{1}_d$ are one-hot vectors.

Regarding the departure time t_o, treating it as a scalar fails to capture complex periodic patterns. We thus model t_o using both discrete hierarchical semantics and continuous cyclic features. First, we decompose t_o into four discrete components: weekday $\mathbf{1}_w \in \{0,1\}^7$, hour $\mathbf{1}_h \in \{0,1\}^{24}$, minute $\mathbf{1}_m \in \{0,1\}^{60}$, and second $\mathbf{1}_s \in \{0,1\}^{60}$. These one-hot vectors are projected into dense embeddings:

$$\mathbf{t}_w = \mathbf{1}_w^\top \mathbf{W}^w, \mathbf{t}_h = \mathbf{1}_h^\top \mathbf{W}^h, \mathbf{t}_m = \mathbf{1}_m^\top \mathbf{W}^m, \mathbf{t}_s = \mathbf{1}_s^\top \mathbf{W}^s, \tag{2}$$

where $\mathbf{W}^w \in \mathbb{R}^{7 \times d_0}$, $\mathbf{W}^h \in \mathbb{R}^{24 \times d_0}$, and $\mathbf{W}^m, \mathbf{W}^s \in \mathbb{R}^{60 \times d_0}$.

Then, we adopt a continuous cyclic embedding $\mathbf{t}_{cyc}$ using trigonometric functions, inspired by [9, 37]. Instead of fixed periods, we parameterize the embedding with learnable frequencies $[\omega_1, \ldots, \omega_{d_0/2}]$ and phase shifts $[\varphi_1, \ldots, \varphi_{d_0/2}]$ to flexibly capture multi-scale periodicities. The d_0-dimensional cyclic embedding is:

$$\begin{aligned} \mathbf{t}_{cyc}(t_o) = &[\sin(\omega_1 t_o + \varphi_1), \cos(\omega_1 t_o + \varphi_1), \ldots, \\ &\sin(\omega_{d_0/2} t_o + \varphi_{d_0/2}), \cos(\omega_{d_0/2} t_o + \varphi_{d_0/2})] \end{aligned} \tag{3}$$

where sine and cosine functions capture cyclic periodic patterns.

We aggregate the discrete and continuous components to yield the departure time embedding $\mathbf{t}_o = \mathbf{t}_w + \mathbf{t}_h + \mathbf{t}_m + \mathbf{t}_s + \mathbf{t}_{cyc}$. Lastly, we concatenate the origin, destination, and temporal embeddings, projecting them into the final query representation $\mathbf{q}$ via a two-layer Multi-Layer Perceptron (MLP) with ReLU activation [14]:

$$\mathbf{q} = MLP_{\mathbf{q}}(\text{concat}(\mathbf{e}_o, \mathbf{e}_d, \mathbf{t}_o)). \tag{4}$$

4.2 Progressive State Encoding (PSE)

As mentioned, UNITE performs progressive, segment-wise inference: at step i on the current segment e_i, it uses mSTE (Section 4.3) to estimate the segment travel time ϕ_i and NSP (Section 4.4) to predict the next segment e_{i+1}. Both modules share the state representation $\mathbf{h}_i$ produced by PSE.

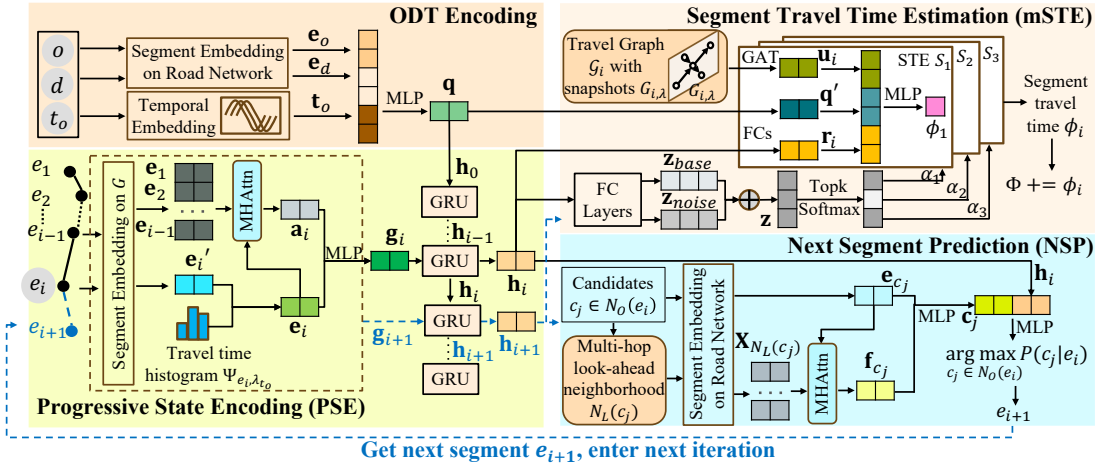


Figure 1: The UNITE Method

Intuitively, $\mathbf{h}_i$ should capture (i) the sequential dependencies along the path from o to e_i, (ii) the dynamic traffic context of e_i from historical trajectories, and (iii) the influence of traffic patterns from previously visited segments, as traffic exhibits strong correlations and propagation effects. For the first aspect, we employ a GRU [3] to model sequential dependencies, initializing $\mathbf{h}_0 = \mathbf{q}$ to incorporate the ODT query context. At each step i, the GRU updates $\mathbf{h}_i$ based on the previous state $\mathbf{h}_{i-1}$ and a current segment representation $\mathbf{g}_i$ for e_i. The construction of $\mathbf{g}_i$ is critical, as it preserves the latter two aspects, which we address as follows. Specifically, we construct a composite embedding $\mathbf{g}_i$ for segment e_i that integrates (i) the topological structure of e_i in the road network, (ii) dynamic traffic patterns on e_i conditioned on t_o via a travel time histogram, and (iii) long-range traffic influence from previously visited segments via multi-head attention. This embedding $\mathbf{g}_i$, together with $\mathbf{h}_{i-1}$, is fed into the GRU to produce the updated state $\mathbf{h}_i$.

First, we encode segment e_i into $\mathbf{e}'_i$ following Eq. (1). We project the one-hot vector $\mathbf{1}_{e_i}$ via $\mathbf{W}^R \in \mathbb{R}^{n \times d_0}$ initialized with $\mathbf{W}_G$ as follows. $\mathbf{W}^R$ is shared by all visited segments in the predicted route.

$$\mathbf{e}'_i = \mathbf{1}_{e_i}^\top \mathbf{W}^R. \tag{5}$$

Next, we capture the dynamic traffic patterns for segment e_i at its corresponding time slot λ, which is determined by the departure time t_o and the cumulative travel time up to segment e_i, i.e., $\sum_{j=1}^{i-1} \phi_j$. Specifically, we construct a travel time histogram $\Psi_{e_i,\lambda}$ for e_i in the time slot λ, which summarizes the empirical distribution of travel times on e_i during that slot. This histogram is then concatenated with $\mathbf{e}'_i$ to form the enriched segment embedding $\mathbf{e}_i$.

The travel time histogram $\Psi_{e_i,\lambda}$ is constructed as follows. For a given segment e_i and the time slot λ, we first retrieve all trajectories from $\mathcal{D}$ that traverse segment e_i within the time slot λ and extract the corresponding multiset of observed travel times (in seconds), denoted as χ. For (e_i, λ) pairs with few observations, we average over its spatial neighbors and adjacent time slots to impute missing values. If these are insufficient, we extend to 3-hop neighbors and 3-hop time slots. To characterize the traffic distribution, we compute the frequency of each unique travel time in χ and identify the top-κ most frequent values. Formally, the travel time histogram vector is defined as $\Psi_{e_i,\lambda} = [\psi_i^1, \eta_1, \ldots, \psi_i^\kappa, \eta_\kappa, \text{mean}(\chi)]$, where ψ_i^j and η_j represent the j-th most frequent travel time and its normalized frequency on segment e_i, and mean(χ) denotes the average travel time. For the κ-th most frequent travel time, if multiple travel times share the same frequency, we break ties by selecting the value closest to the most frequent travel time. If all frequencies are equal, we select the travel time closest to the median of all travel times of all trajectories over e_i sorted in ascending order, to build the histogram. We set $\kappa = 5$ by default. The ratio of pairs with $\kappa < 5$ distinct travel times in training is 18.8% for CD, 19.3% for XA, and 37.2% for HRB, showing UNITE can handle diverse data sparsity. Intuitively, the travel time histogram $\Psi_{e_i,\lambda}$ captures the empirical distribution of travel times on e_i during the specified time slot.

Then we obtain the segment embedding $\mathbf{e}_i$ by integrating the spatial embedding $\mathbf{e}'_i$ (Eq. (5)) with the travel time histogram $\Psi_{e_i,\lambda}$. Specifically, we concatenate these vectors and project them through a two-layer MLP to capture non-linear interactions between topological structure and traffic dynamics, as shown in Eq. (6).

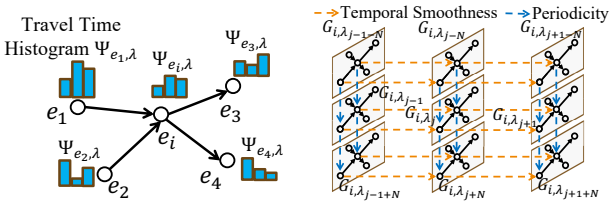
$$\mathbf{e}_i = MLP_R(\text{concat}(\mathbf{e}'_i, \Psi_{e_i,\lambda})) \tag{6}$$

To capture the long-range influence of traffic patterns from previously visited segments on e_i, we employ a multi-head attention mechanism (MHAttn) that integrates the traffic contexts from all preceding segments to form a context vector $\mathbf{a}_i$ for e_i. Specifically, we utilize $\mathbf{e}_i$ as the query, and the sequence of embeddings from all preceding segments, denoted as $\mathbf{X}_i = \text{stack}[\mathbf{e}_1, ..., \mathbf{e}_{i-1}]$, as both keys and values. The resulting context vector $\mathbf{a}_i$ is computed as:

$$\mathbf{a}_i = \text{MHAttn}(\mathbf{e}_i, \mathbf{X}_i, \mathbf{X}_i), \tag{7}$$

With $\mathbf{e}_i$ capturing the dynamic traffic context and topological features of segment e_i, and $\mathbf{a}_i$ integrating the impact of traffic patterns from all previously visited segments, we proceed to construct the composite embedding $\mathbf{g}_i$. Specifically, we concatenate these two representations and project them through an MLP to yield the composite embedding vector $\mathbf{g}_i$, as formalized in Eq. (8). This vector $\mathbf{g}_i$ thus encapsulates both the intrinsic spatio-temporal attributes of the current segment e_i and the relevant historical context aggregated from the route traversed so far.

$$\mathbf{g}_i = MLP_G(\text{concat}(\mathbf{e}_i, \mathbf{a}_i)) \tag{8}$$



(a) Graph Snapshot $G_{i,\lambda}$ **(b) Travel Graph $\mathcal{G}_i$**

Figure 2: Graph Snapshot $G_{i,\lambda}$ and Travel Graph $\mathcal{G}_i$.

Finally, at each step i, we employ a GRU to update the hidden state $\mathbf{h}_i$ for the current segment e_i, taking the composite embedding $\mathbf{g}_i$ and the previous state $\mathbf{h}_{i-1}$ as inputs. The recurrence is initialized with the query embedding $\mathbf{q}$ to ensure that the global ODT context informs the entire sequence:

$$\mathbf{h}_0 = \mathbf{q}, \quad \mathbf{h}_i = GRU(\mathbf{h}_{i-1}, \mathbf{g}_i). \tag{9}$$

PSE generates a comprehensive state representation $\mathbf{h}_i$ that integrates the static topology of the road network, the dynamic traffic patterns on e_i conditioned on the departure time t_o, and the sequential dependencies accumulated along the traversed route. This embedding $\mathbf{h}_i$ serves as the basis for subsequent joint tasks of segment-level travel time estimation and next segment prediction.

4.3 Segment Travel Time Estimation (mSTE)

With the progressive state $\mathbf{h}_i$ available, we address the regression task of estimating the travel time ϕ_i for the current segment e_i. This is challenging due to heterogeneous traffic conditions arising from complex road topology and dynamic, time-varying patterns. A single regressor cannot capture such diversity.

To model these patterns, we propose the Segment Travel Time Estimator (mSTE), which adopts a Mixture-of-Experts architecture. mSTE aggregates multiple Segment Travel Time Experts (STEs), each trained to predict travel time for a segment. A noisy Top-k gating mechanism dynamically selects and combines the most relevant STEs for the current context. Crucially, each STE leverages the spatio-temporal traffic context around e_i, as travel time depends on historical traffic states of e_i and its spatial neighbors across relevant time slots (including recent and periodic slots). To capture these dependencies, we construct a travel graph $\mathcal{G}_i$ centered at e_i, which encodes both spatial structure and temporal traffic distributions, as illustrated in Figure 2. We next detail a single STE and the construction of $\mathcal{G}_i$, followed by the complete mSTE module.

A Single Segment Travel Time Expert (STE). Formally, an STE S is a regression model that predicts the travel time ϕ_i for a target segment e_i. The prediction integrates three complementary contexts: (i) the global query embedding $\mathbf{q}$ from the ODT input; (ii) the sequential trajectory context $\mathbf{h}_i$ produced by the PSE; and (iii) the spatio-temporal traffic context specific to e_i.

To capture the traffic context, we explicitly construct an *offline* travel graph $\mathcal{G}_i$ centered at e_i. Since real-world road networks are generally stable, we preprocess these graphs for efficient retrieval during inference. The *offline* construction proceeds in three steps. First, we create a graph snapshot $G_{i,\lambda}$ centered at e_i for its corresponding time slot λ, adopting a star topology that includes e_i and its immediate neighbors, with each node annotated by its travel time histogram $\Psi_{e,\lambda}$ as described in Section 4.2. Second, to model temporal evolution, we generate such snapshots for a set of relevant time slots Λ around λ, capturing both short-term variations and daily periodicity. Third, we integrate these snapshots into the travel graph $\mathcal{G}_i$ by adding inter-slot edges that link corresponding segments across time slots. An example is shown in Figure 2.

Specifically, for segment e_i at time slot λ, we first retrieve its immediate in-going and out-going adjacent segments from the road network G, forming a star graph centered at e_i with these segments as nodes. For example, in Figure 2a, segment e_1 has in-going adjacent segments $\{e_2, e_3\}$ and out-going adjacent segments $\{e_4, e_5\}$. For each segment e in this star graph, we construct its travel time histogram $\Psi_{e,\lambda}$. This yields the graph snapshot $G_{i,\lambda}$ for segment e_i at time slot λ, capturing both the spatial context and traffic patterns at that time. In Figure 2a, each segment is annotated with its travel time histogram for time slot λ.

Traffic patterns on a segment often exhibit similarity across adjacent time slots and daily periodicity. To incorporate these temporal dynamics, we extend the single snapshot across a set of relevant time slots Λ. Let p be the window size. We define Λ to include the short-term window $[\lambda-p, \lambda+p]$ within the same day, as well as the corresponding windows from the previous and subsequent days. Thus, Λ contains $3(2p+1)$ time slots in total. For each time slot $\lambda \in \Lambda$, we create a graph snapshot $G_{i,\lambda}$ mirroring the star structure of $G_{i,\lambda}$, where nodes are annotated with the travel time histograms $\Psi_{e,\lambda}$ for that specific time slot. For example, in Figure 2b, for segment e_1 with $p = 1$, we illustrate its travel graph $\mathcal{G}_i$ as a sequence of graph snapshots $G_{i,\lambda}$ for the time slots in Λ.

Finally, we integrate these snapshots into a unified travel graph $\mathcal{G}_i$ by introducing inter-slot edges to model temporal dependencies. Specifically, we construct two types of edges: (i) Temporal Smoothness edges, which connect corresponding segments in adjacent time slots (λ_j and λ_{j+1}) to capture short-term traffic evolution; and (ii) Periodicity edges, which link the same time slot across consecutive days (λ_j and $\lambda_{j\pm N}$, where N is the number of slots per day) to encode daily recurrent patterns. As a result, $\mathcal{G}_i$ provides a holistic representation of traffic dynamics, enabling the model to jointly reason over spatial topology and multi-scale temporal variations.

Given the travel graph $\mathcal{G}_i$, each STE employs a Graph Attention Network (GAT) [36] to encode the spatio-temporal traffic context into a latent representation $\mathbf{u}_i$ for the central segment e_i. To align the feature spaces of the query embedding $\mathbf{q}$ and the hidden state $\mathbf{h}_i$ with the context $\mathbf{u}_i$, we use fully connected (FC) layers to project $\mathbf{q}$ and $\mathbf{h}_i$ into $\mathbf{q}'$ and $\mathbf{r}_i$, respectively. These representations are concatenated to integrate global, sequential, and local contexts, and then fed into an MLP to predict the scalar travel time. The inference process of an STE $S(\mathbf{q}, \mathbf{h}_i, \mathcal{G}_i)$ is formalized as:

$$\begin{aligned} \mathbf{u}_i = GAT(\mathcal{G}_i), \quad \mathbf{q}' = FC_q(\mathbf{q}), \quad \mathbf{r}_i = FC_r(\mathbf{h}_i), \\ \phi_i = MLP_\phi(\text{concat}(\mathbf{q}', \mathbf{r}_i, \mathbf{u}_i)), \end{aligned} \tag{10}$$

where FC_q and FC_r denote the FC layers, and MLP_ϕ outputs the predicted travel time $\phi_i \in \mathbb{R}$.

MoE-based STE (mSTE). We now present the complete mSTE module with Mixture-of-Experts (MoE) [34] on multiple STEs to model diverse spatio-temporal traffic patterns for travel time estimation. This design enables adaptive expert selection for each segment and context, improving accuracy and robustness. The mSTE module comprises K STEs $\{S_j\}_{j=1}^{K}$ and a gating network α that determines

the contribution of each STE for segment e_i. Given $\mathbf{h}_i$, α computes a probability distribution over the STEs, where $\alpha(\mathbf{h}_i)[j]$ is the gating weight for STE S_j. As shown in (11), the estimated travel time ϕ_i for e_i by mSTE is computed as a weighted sum of the predictions from the STEs in Eq. (10):

$$\phi_i = \sum_{j=1}^{K} \alpha(\mathbf{h}_i)[j] \cdot S_j(\mathbf{q}, \mathbf{h}_i, \mathcal{G}_i) \tag{11}$$

A key challenge in designing the gating network α is to ensure that STE expert selection is efficient and diverse, avoiding the pitfall of expert collapse, where only a few STEs dominate the predictions. Thus, we adopt a noisy Top-k Gating mechanism that enforces sparsity and promotes balanced expert utilization.

Given the state $\mathbf{h}_i$, the gating network computes a weight vector $\alpha(\mathbf{h}_i) \in \mathbb{R}^K$ to aggregate the outputs of the K experts. Unlike standard softmax gating—which requires evaluating all experts and often leads to redundancy and collapse—we introduce two key modifications: (i) sparsity, by activating only the top-k experts for each input, and (ii) stochasticity, by injecting noise during training to encourage exploration and load balancing.

Specifically, $\mathbf{h}_i$ is projected through two linear FC layers to produce a base logit vector $\mathbf{z}_{\text{base}}$ and a noise scaling vector $\mathbf{z}_{\text{noise}}$. The final logits $\mathbf{z}$ are computed by adding scaled Gaussian noise to $\mathbf{z}_{\text{base}}$, where the scaling is determined by $\mathbf{z}_{\text{noise}}$. The Top-$k$ operator then retains only the k largest values in $\mathbf{z}$, masking the rest with $-\infty$ before applying softmax. This process is formalized as:

$$\begin{aligned} &\mathbf{z}_{\text{base}} = FC_{base}(\mathbf{h}_i), \quad \mathbf{z}_{\text{noise}} = FC_{noise}(\mathbf{h}_i), \\ &\mathbf{z} = \mathbf{z}_{\text{base}} + \mathcal{N}(0,1) \cdot \log(1 + e^{\mathbf{z}_{\text{noise}}}), \\ &\alpha(\mathbf{h}_i) = \text{Softmax}(\text{Topk}(\mathbf{z}, k)), \end{aligned} \tag{12}$$

where FC_{base} and FC_{noise} are FC layers, $\mathcal{N}(0,1)$ denotes standard Gaussian noise, and k controls the number of active experts.

4.4 Next Segment Prediction (NSP)

For ODTTE queries, at each step for the current segment e_i, UNITE jointly performs two tasks: estimating the segment travel time ϕ_i and selecting the next segment e_{i+1} to extend the route. The NSP module utilizes the progressive state $\mathbf{h}_i$ generated by the PSE, which encodes the accumulated spatio-temporal and structural context up to e_i, as well as traffic patterns.

Given the current segment e_i, NSP determines the next segment e_{i+1} by evaluating the set of out-going neighbors $N_O(e_i)$ in the road network G. This selection is non-trivial since multiple neighbors may offer plausible paths to the destination.

We formulate NSP as a binary classification task over $N_O(e_i)$. For each candidate $c_j \in N_O(e_i)$, the model estimates a transition probability $P(c_j|e_i)$, representing the likelihood of c_j being the true successor. During inference, the candidate maximizing this probability is selected as e_{i+1}. Formally, $P(c_j|e_i)$ is derived by modeling the compatibility between the current progressive state $\mathbf{h}_i$ and a comprehensive representation of the candidate, denoted as $\mathbf{c}_j$.

To effectively guide routing, $\mathbf{c}_j$ must capture more than the immediate spatial identity of c_j; it requires a look-ahead capability over the downstream topology to assess potential progress toward the destination. Consequently, we construct $\mathbf{c}_j$ by integrating the candidate's intrinsic structural features with a context-aware representation of its multi-hop outgoing neighborhood.

First, similar to Section 4.1, we derive a structural embedding $\mathbf{e}_{c_j}$ for the candidate segment c_j by projecting its one-hot vector $\mathbf{1}_{c_j}$ through a learnable matrix $\mathbf{W}^C \in \mathbb{R}^{n \times d_0}$, initialized with $\mathbf{W}_G$ to capture topological properties:

$$\mathbf{e}_{c_j} = \mathbf{1}_{c_j} \mathbf{W}^C. \tag{13}$$

Next, we extract a multi-hop look-ahead neighborhood $N_L(c_j)$ for candidate segment c_j, which captures downstream segments likely to contribute to progress toward the destination. To construct $N_L(c_j)$, we traverse outgoing segments from c_j up to three hops. For each segment e_k reachable within three hops, we evaluate its directional relevance to the destination d by computing the angle between the vector of e_k and the vector from the exit of c_j to the entrance of d. If this angle is less than 135°, e_k is included in $N_L(c_j)$. This directional filtering ensures that only segments plausibly contributing to progress toward the destination are retained in $N_L(c_j)$. Each segment e_k in $N_L(c_j)$ is embedded into a dense vector $\mathbf{e}_k$ via a projection matrix $\mathbf{W}^L \in \mathbb{R}^{n \times d_0}$, initialized with $\mathbf{W}_G$:

$$\mathbf{e}_k = \mathbf{1}_{e_k} \mathbf{W}^L, \quad \forall e_k \in N_L(c_j). \tag{14}$$

To synthesize these neighbor representations into a coherent vector $\mathbf{f}_{c_j}$, we employ the Multi-Head Attention mechanism. We use the candidate's intrinsic embedding $\mathbf{e}_{c_j}$ as the query to attend over the stacked embeddings $\mathbf{X}_{N_L(c_j)} = \text{stack}[\mathbf{e}_k]_{e_k \in N_L(c_j)}$, which serve as both keys and values. This attention-based aggregation dynamically highlights the downstream segments most relevant to the candidate, yielding the context vector $\mathbf{f}_{c_j}$:

$$\mathbf{f}_{c_j} = \text{MHAttn}(\mathbf{e}_{c_j}, \mathbf{X}_{N_L(c_j)}, \mathbf{X}_{N_L(c_j)}). \tag{15}$$

We obtain the representation $\mathbf{c}_j$ for candidate c_j by integrating its embedding $\mathbf{e}_{c_j}$ with the look-ahead context $\mathbf{f}_{c_j}$ via an MLP:

$$\mathbf{c}_j = MLP_c(\text{concat}(\mathbf{e}_{c_j}, \mathbf{f}_{c_j})). \tag{16}$$

To determine the likelihood of traversing c_j next, we model the interaction between the current trajectory state $\mathbf{h}_i$ (capturing historical context) and the candidate representation $\mathbf{c}_j$ (capturing future structural context). Specifically, we concatenate $\mathbf{h}_i$ and $\mathbf{c}_j$ and pass the result through a scoring MLP followed by a sigmoid activation to compute the transition probability $P(c_j|e_i)$:

$$P(c_j|e_i) = \text{sigmoid}(MLP(\text{concat}(\mathbf{c}_j, \mathbf{h}_i))). \tag{17}$$

During online inference, the next segment e_{i+1} is selected greedily by identifying the candidate in outgoing neighbors $N_O(e_i)$ with the highest transition probability $e_{i+1} = \arg\max_{c_j \in N_O(e_i)} P(c_j|e_i)$.

4.5 Objective Function

UNITE is trained end-to-end by jointly optimizing two objectives: (i) a regression loss for segment-level travel time prediction (mSTE) in Section 4.3 , and (ii) a binary cross-entropy loss for next segment prediction (NSP) in Section 4.4.

For mSTE, we use the Mean Squared Error (MSE) loss. Given a training T with ground-truth route R, where each segment $e_i \in R$ has ground-truth travel time ψ_i, the regression loss in Eq. (18) is the sum of squared differences between the predicted travel time ϕ_i (Eq. (11)) and the ground truth ψ_i for each segment e_i.

$$L_{\text{mSTE}}(T) = \textstyle\sum_{i=1}^{|R|} (\phi_i - \psi_i)^2 \tag{18}$$

For NSP, we treat the route generation as a sequence of binary classification problems. For each step i (where $i < |R|$), the NSP

module aims to predict the true next segment e_{i+1} from the set of out-going neighbors $N_O(e_i)$. Let $y_c \in \{0, 1\}$ be the ground-truth label for a candidate $c \in N_O(e_i)$, where $y_c = 1$ if c is indeed the e_{i+1} segment in R and 0 otherwise. The classification loss is the sum of binary cross-entropy losses over all steps in the route:

$$L_{\text{NSP}}(T) = -\sum_{i=1}^{|R|-1} \sum_{c \in N_O(e_i)} [\, y_c \log P(c|e_i) + (1 - y_c) \log(1 - P(c|e_i)) \,] \tag{19}$$

The objective for a trajectory T linearly combines these losses:

$$\mathcal{L}(T; \Theta) = L_{\text{mSTE}}(T) + L_{\text{NSP}}(T) \tag{20}$$

The detailed processes of offline training and online inference are presented in Appendix § A.1 and § A.2, respectively.

5 Experiments

All experiments are conducted on a Linux machine powered by Intel Xeon® Gold 6226R 2.90GHz CPU and NVIDIA GTX 3090 GPU with 24GB video memory. Our methods are implemented in Python 3.12 with PyTorch 2.4. Source codes of all competitors are obtained from the respective papers in Python. Our implementation is at https://github.com/derekwtian/UNITE.

5.1 Experimental Setup

Datasets. We conduct experiments on three real-world trajectory datasets collected from the road networks of Chengdu (CD), Xi'an (XA) [1], and Harbin (HRB) [25]. The statistical information of all datasets is summarized in Table 1. The CD and XA datasets contain millions of trajectories from DiDi ride-sharing, while HRB consists of taxi trajectories. For each raw trajectory of GPS points, we apply map-matching [49] to align them with road segments, thereby obtaining the corresponding routes and travel times. Table 1 also reports the average length and travel time of trajectories, as well as the number of segments and intersections in the respective road networks. For each dataset, we sort all trajectories by their departure time, and then split them into training, validation, and test sets with ratios of 40%, 30%, and 30%, respectively. We use the training set for model learning, the validation set for parameter tuning, and the testing set for performance evaluation. We also conduct experiments varying the training data size to assess the data efficiency of different methods.

Competitors. For ODTTE, we use: (i) five native ODTTE methods: DOT [27], DeepOD [51], DutyTTE [31], MURAT [26], and TEMP [40]; (ii) two adapted RTTE methods, STHR [50] and WDDRA [12], where we first generate a route using NMLR [19] and then estimate its travel time; and (iii) a routing-based baseline, Dijkstra, which finds the fastest route and sums the historical average segment travel times. For RTTE, we use: (i) four native RTTE methods: STHR [50], HierETA [2], HetETA [17], and WDDRA [12]; (ii) two adapted ODTTE methods, DOT and DutyTTE, where their travel time estimation modules are applied on input routes; and (iii) baseline HA that sums the historical average travel times of all segments on each route.

Implementations. We set the embedding dimension $d_0 = 64$ for Eq.(1), (2), (5), and (13). For the MLPs in Eq.(4), (6), (8), (10), and (16), the hidden dimension is 256 and the output dimension is 128. The number of STEs in mSTE is set to 8, with $k = 4$ activated per inference. The time slot size δ is set to 900s. The window size p in the travel graph $\mathcal{G}_i$ is set to 2, covering half an hour. We stack two GRU layers in Eq.(9), and use four heads for MHAttn in Eq.(7) and (15) in PSE and NSP, respectively. UNITE applies the teaching force ratio to balance the use of ground-truth and predicted segments and travel times during training, which is initialized as 1.0 and reduced by 0.1 per epoch. The learning rate and batch size for training UNITE are set to 1e-3 and 512, respectively. For all competitors, we follow their recommended settings for parameter tuning. All methods are trained until convergence.

Table 1: Dataset Statistics

	Chengdu (CD)	Xi'an (XA)	Harbin (HRB)
# of trajectories	2,382,422	1,426,950	496,288
Avg length (m)	1,825.41	2,449.27	4030.13
Avg travel time (s)	850.17	1,092.02	1482.36
Time period	2016/10/1-2016/10/31	2016/10/1-2016/10/31	2015/1/3-2015/1/7
# of segments	9,255	5,699	11,644
# of intersections	3,973	2,579	4,917

Evaluation Metrics. For effectiveness, we adopt the same metrics as in [27, 50] for ODTTE and RTTE, including Mean Square Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). Given the ground-truth travel time Φ^* of a trajectory T with predicted travel time Φ, the metrics are defined as follows: $\text{MSE}(\Phi, \Phi^*) = \frac{1}{|\mathcal{D}|} \sum_{\forall T \in \mathcal{D}} (\Phi - \Phi^*)^2$, $\text{MAE}(\Phi, \Phi^*) = \frac{1}{|\mathcal{D}|} \sum_{\forall T \in \mathcal{D}} |\Phi - \Phi^*|$, and $\text{MAPE}(\Phi, \Phi^*) = \frac{1}{|\mathcal{D}|} \sum_{\forall T \in \mathcal{D}} |\frac{\Phi - \Phi^*}{\Phi^*}|$. The lower the values of these metrics, the better the performance. For efficiency, we measure the inference time to estimate the travel time of 1,000 queries and the training time per epoch.

5.2 Effectiveness Evaluation

ODTTE Effectiveness. Table 2 reports the MSE, MAE, and MAPE of all methods across all datasets for ODTTE. Our UNITE consistently achieves the best performance on all datasets and evaluation metrics, outperforming all existing methods by a substantial margin. This includes both native ODTTE methods, adapted RTTE methods utilizing route planning (e.g., STHR), and routing baselines based on historical statistics (e.g., Dijkstra). For example, on XA, UNITE achieves an MSE of 12.44, while the best competitor, DOT, reports 18.40, yielding an absolute improvement of 5.96 (32.39% relative improvement). These results validate the effectiveness of our technical designs in achieving accurate travel time estimation when only the origin, destination, and departure time are available.

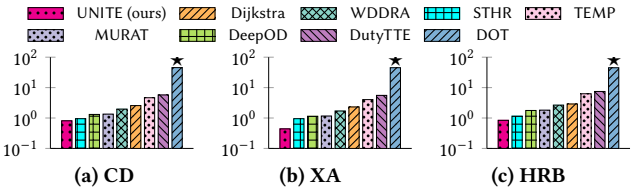
RTTE Effectiveness. For RTTE, Table 3 reports the MSE, MAE, and MAPE. UNITE consistently achieves the best performance across all datasets and metrics. For example, on CD, UNITE achieves an MSE of 3.15, significantly lower than the best competitor STHR (5.19), yielding an absolute improvement of 2.04. Notably, the improvements of UNITE are even more pronounced on XA and HRB. These results demonstrate the effectiveness of UNITE in accurately estimating travel times when complete route information is provided. Importantly, UNITE is trained only once under the ODTTE setting, yet it is capable of handling both ODTTE and RTTE settings during online inference. This validates our unified design, where training

Table 2: Overall Effectiveness of ODTTE. Lower MSE, MAE (in minutes), and MAPE (in percentage %) indicate better performance. The best result is in bold and the second best is underlined.

Methods	CD			XA			HRB		
	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE
Dijkstra	15.42	2.60	27.73	33.44	3.71	29.31	31.92	4.01	35.35
WDDRA	8.53	1.87	23.91	21.62	2.87	27.28	29.05	3.62	33.25
STHR	6.61	1.64	22.20	18.58	2.68	26.42	27.71	3.57	32.49
TEMP	15.11	2.50	27.44	30.58	3.58	28.93	31.63	3.89	34.88
MURAT	13.76	2.48	27.13	29.71	3.43	28.49	31.25	3.68	33.73
DeepOD	7.73	1.65	22.71	19.27	2.73	27.12	30.91	3.67	33.64
DutyTTE	7.18	1.64	22.27	18.92	2.71	26.98	30.14	3.65	33.59
DOT	6.56	1.62	22.13	18.40	2.64	25.86	25.81	3.48	32.13
UNITE	**5.16**	**1.37**	**18.55**	**12.44**	**2.14**	**19.64**	**15.40**	**2.45**	**22.83**

Table 3: Overall Effectiveness of RTTE. Lower MSE, MAE (in minutes), and MAPE (in percentage %) indicate better performance.

Methods	CD			XA			HRB		
	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE
HA	10.39	2.16	24.22	24.81	3.15	25.93	25.92	3.61	27.15
HetETA	8.64	2.12	23.67	22.66	2.93	25.38	16.97	3.13	26.97
HierETA	6.71	2.03	22.99	21.34	2.72	24.90	16.24	2.87	25.72
WDDRA	5.90	1.70	22.05	18.49	2.66	24.38	10.60	2.26	21.13
DutyTTE	5.41	1.54	21.65	16.08	2.54	24.23	9.58	2.22	20.21
DOT	5.38	1.45	21.36	19.09	2.67	24.54	11.97	2.28	23.57
STHR	5.19	1.43	19.63	15.28	2.42	23.89	12.31	2.48	24.93
UNITE	**3.15**	**1.17**	**17.13**	**9.39**	**1.92**	**18.22**	**2.86**	**1.19**	**12.70**

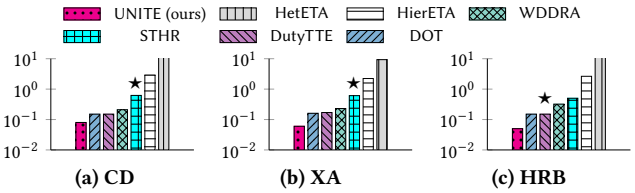


(a) CD (b) XA (c) HRB

Figure 3: Inference Time of ODTTE (in seconds) (★ marks the best competitor in Table 2).

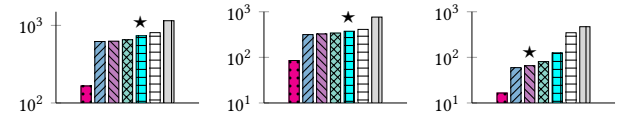
(a) CD (b) XA (c) HRB

Figure 4: Training Time per Epoch of ODTTE Methods.



(a) CD (b) XA (c) HRB

Figure 5: Inference Time of RTTE (in seconds) (★ marks the best competitor in Table 3).



(a) CD (b) XA (c) HRB

Figure 6: Training Time per Epoch of RTTE Methods.

without explicit routes enables UNITE to learn more comprehensive representations. Furthermore, UNITE operates in a progressive segment-wise manner, jointly predicting travel time and the next segment, allowing the two tasks to mutually enhance each other. The results in Table 3 confirm the efficacy of this unified approach.

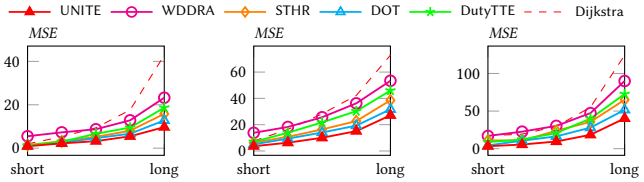
5.3 Efficiency Evaluation

ODTTE Efficiency. We report inference time per 1,000 ODTTE queries in seconds in Figure 3 and training time per epoch in Figure 4, where the strong competitor in Table 2 is marked by ★. As shown in Figure 3, UNITE consistently achieves the fastest inference speed across all datasets. In particular, compared with the best competitor DOT, UNITE is significantly faster, often by orders of magnitude. For example, on XA, to process 1,000 ODTTE queries, UNITE only needs 0.44 seconds, while DOT needs 45.06 seconds, indicating that UNITE is 102× faster. The primary efficiency bottleneck of DOT stems from its reliance on vision-based techniques to generate pixelated trajectories, which is computationally intensive. Other competitors, while relatively fast, are still less efficient than UNITE; more importantly, they yield inferior ODTTE quality as shown in Table 2. We further provide a breakdown of the inference time of UNITE in § B.3. Regarding training efficiency, Figure 4 demonstrates that UNITE achieves faster or comparable training time per epoch compared to all competitors. For instance, on HRB, UNITE takes 81.99 seconds per epoch, whereas DOT takes 113.75 seconds and DutyTTE even requires 1603.96 seconds. The high efficiency of UNITE stems from its technical design in Section 4. Moreover, a key advantage of UNITE is that it requires training only once under the ODTTE setting, yet it can handle both ODTTE and RTTE tasks during inference, whereas existing methods are typically designed for only one setting.

RTTE Efficiency. We report the inference and training times in Figure 5 and Figure 6, respectively, where the strong competitor in Table 3 is marked by ★. Consistently, UNITE achieves the fastest inference and training efficiency across all datasets for RTTE. For example, in Figure 5, on CD, UNITE only costs 0.08 seconds to infer 1,000 RTTE queries. This is approximately 7× faster than the most effective competitor, STHR, which requires 0.62 seconds.

Table 4: Ablation Study by MSE

Variant	CD	XA	HRB
UNITE	**5.16**	**12.44**	**15.40**
UNITE-att	6.20	12.89	16.00
UNITE-hist	5.24	13.62	16.48
UNITE-$\mathcal{G}_i$	5.48	13.18	16.81
UNITE-N_L	6.50	13.84	16.73
UNITE-SR	5.81	15.60	16.32
UNITE-SG	5.33	14.15	15.87



(a) CD (b) XA (c) HRB

Figure 7: MSE of ODTTE with different route lengths

Moreover, UNITE is orders of magnitude faster than competitors such as HierETA and HetETA. Although DOT and DutyTTE are relatively fast, they yield lower effectiveness, as shown in Table 3. Regarding training efficiency, Figure 6 shows that UNITE achieves significantly faster per epoch training time than all competitors. For instance, on CD, UNITE takes only 165.75 seconds per epoch, whereas STHR requires 735.17 seconds.

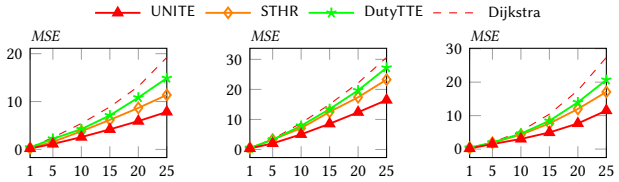
5.4 Model Analysis

Ablation Study. We ablate the techniques in UNITE. UNITE-att removes the long-range traffic pattern influence from previously visited segments ($\mathbf{a}_i$ in Eq. (7)); UNITE-hist omits the travel time histogram $\Psi_{e_i,\lambda_{t_o}}$ (Eq. (6)); UNITE-$\mathcal{G}_i$ excludes the travel graph $\mathcal{G}_i$ in mSTE; UNITE-N_L does not consider the multi-hop look-ahead neighborhood $N_L(c_j)$ in Eq. (15) of NSP; UNITE-SR ablates MoE with a Single regressor; UNITE-SG replaces noisy Top-k gating with a Simple, noise-free gating. Table 4 presents MSE results of these ablated versions across all datasets. The complete UNITE consistently outperforms all ablated variants, underscoring the necessity of each component. Table 4 confirms the effectiveness of all these techniques in UNITE.

Impact of Route Length. We evaluate method performance across varying ground-truth route lengths for ODTTE queries. Test queries are grouped by route segment count into five bins. Figure 7 shows MSE for all methods from shortest to longest routes on three datasets. All methods exhibit higher MSE for longer routes due to error accumulation, but UNITE consistently achieves the lowest MSE across all groups, demonstrating robustness to route length. Inference time results are provided in § B.3.

Accumulated Errors in Long ODTTE. We further analyze error accumulation for long routes by selecting test queries with ground-truth routes exceeding 25 segments. The MSE for every five segments from the origin is shown in Figure 8. All methods exhibit increasing errors with route length, but UNITE achieves the lowest rate of increase, confirming its robustness for long-route ODTTE.

Generalization Capability. Table 5 shows MSE of UNITE and the strongest baselines (DOT and STHR) on unseen OD pairs for ODTTE and RTTE, respectively. All methods exhibit higher MSE on unseen OD pairs in Table 5 than overall MSE in Tables 2, 3.

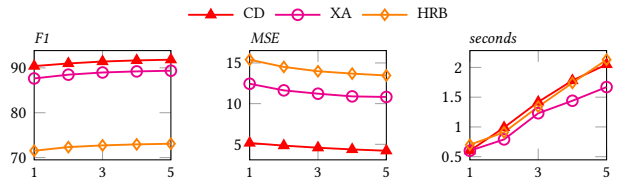


(a) CD (b) XA (c) HRB

Figure 8: Accumulated Errors for Long ODTTE Queries

Table 5: MSE on Unseen OD Pairs

Data		CD	XA	HRB
ODTTE	UNITE	9.68	16.52	16.89
	DOT	11.69	23.37	27.53
RTTE	UNITE	4.78	10.94	3.05
	STHR	8.55	18.11	13.36



(a) Route Quality (b) ODTTE Errors (c) Inference Time

Figure 9: UNITE with Beam Search Mechanism

Notably, UNITE consistently surpasses baselines on unseen OD pairs, validating its strong generalization. Moreover, our design preserves dynamic traffic patterns via PSE, captures temporal features via ODT Encoding, and incorporates road network in NSP. By jointly modeling these factors, UNITE generalizes well with strong performance across cities and time periods.

Beam Search Mechanism. UNITE can support beam search (BS), as shown in Figure 9. Compared to $width = 1$ (Greedy), Beam Search with a larger width improves effectiveness, but increases inference time. With Greedy, UNITE takes < 1 second per 1,000 queries, faster than baselines(Figure 3), but BS costs over 1 second.

6 Conclusion

We present UNITE, a unified framework for ODTTE and RTTE that performs progressive, segment-wise inference to jointly model route generation and travel time, capturing fine-grained and dynamic traffic conditions. UNITE integrates a PSE module for shared context, an mSTE module based on MoE for handling traffic heterogeneity via travel graphs, and an NSP module for accurate route inference. Experiments demonstrate UNITE outperforms existing methods in both accuracy and efficiency. However, our current focus is on city-level travel time estimation, addressing extra-long nationwide routes will be left for future work.

Acknowledgments

This work is supported by grants from the Research Grants Council of Hong Kong Special Administrative Region, China (No. PolyU 15205224), funding from ByteDance, and Smart Cities Research Institute (SCRI) P0051036-P0050643. Man Lung Yiu is supported by Hong Kong Research Grants Council (GRF 152043/23E).

References

[1] [n. d.]. Gaya. https://outreach.didichuxing.com/research/opendata/.
[2] Zebin Chen, Xiaolin Xiao, Yue-Jiao Gong, Jun Fang, Nan Ma, Hua Chai, and Zhiguang Cao. 2022. Interpreting Trajectories from Multiple Views: A Hierarchical Self-Attention Network for Estimating the Time of Arrival. In *SIGKDD*. 2771–2779.
[3] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *EMNLP*. 103–111.
[4] Jian Dai, Bin Yang, Chenjuan Guo, and Zhiming Ding. 2015. Personalized route recommendation using big trajectory data. In *ICDE*. 543–554.
[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
[6] Yu Fan, Jiajie Xu, Rui Zhou, Jianxin Li, Kai Zheng, Lu Chen, and Chengfei Liu. 2022. MetaER-TTE: An Adaptive Meta-learning Model for En Route Travel Time Estimation. In *IJCAI*. 2023–2029.
[7] Xiaomin Fang, Jizhou Huang, Fan Wang, Lihang Liu, Yibo Sun, and Haifeng Wang. 2021. SSML: Self-Supervised Meta-Learner for En Route Travel Time Estimation at Baidu Maps. In *SIGKDD*. 2840–2848.
[8] Xiaomin Fang, Jizhou Huang, Fan Wang, Lingke Zeng, Haijin Liang, and Haifeng Wang. 2020. ConSTGAT: Contextual Spatial-Temporal Graph Attention Network for Travel Time Estimation at Baidu Maps. In *SIGKDD*. 2697–2705.
[9] Ziquan Fang, Yuntao Du, Xinjun Zhu, Danlei Hu, Lu Chen, Yunjun Gao, and Christian S. Jensen. 2022. Spatio-Temporal Trajectory Similarity Learning in Road Networks. In *KDD*. 347–356.
[10] Ziquan Fang, Lu Pan, Lu Chen, Yuntao Du, and Yunjun Gao. 2021. MDTP: A Multi-source Deep Traffic Prediction Framework over Spatio-Temporal Trajectory Data. *PVLDB* 14, 8 (2021), 1289–1297.
[11] Kun Fu, Fanlin Meng, Jieping Ye, and Zheng Wang. 2020. CompactETA: A Fast Inference System for Travel Time Prediction. In *SIGKDD*. 3337–3345.
[12] Yunchong Gan, Haoyu Zhang, and Mingjie Wang. 2021. Travel Time Estimation Based on Neural Network with Auxiliary Loss. In *SIGSPATIAL*. 642–645.
[13] Nandani Garg and Sayan Ranu. 2018. Route Recommendations for Idle Taxi Drivers: Find Me the Shortest Route to a Customer!. In *KDD*. 1425–1434.
[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *AISTATS*. 315–323.
[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[17] Huiting Hong, Yucheng Lin, Xiaoqing Yang, Zang Li, Kun Fu, Zheng Wang, Xiaohu Qie, and Jieping Ye. 2020. HetETA: Heterogeneous Information Network Embedding for Estimating Time of Arrival. In *KDD*. 2444–2454.
[18] Aiqing Huang, Linli Xu, Yitan Li, and Enhong Chen. 2014. Robust Dynamic Trajectory Regression on Road Networks: A Multi-task Learning Framework. In *ICDM*. 857–862.
[19] Jayant Jain, Vrittika Bagadia, Sahil Manchanda, and Sayan Ranu. 2021. NeuroMLR: Robust & Reliable Route Recommendation on Road Networks. In *NeurIPS*. 22070–22082.
[20] Donald B. Johnson. 1973. A Note on Dijkstra's Shortest Path Algorithm. *J. ACM* 20, 3 (1973), 385–388.
[21] Manas Joshi, Arshdeep Singh, Sayan Ranu, Amitabha Bagchi, Priyank Karia, and Puneet Kala. 2021. Batching and Matching for Food Delivery in Dynamic Road Networks. In *ICDE*. 2099–2104.
[22] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. 2006. Finding Fastest Paths on A Road Network with Speed Patterns. In *ICDE*. 10–10.
[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*. 15 pages.
[24] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. 14 pages.
[25] Xiucheng Li, Gao Cong, Aixin Sun, and Yun Cheng. 2019. Learning Travel Time Distributions with Deep Generative Model. In *WWW*. 1017–1027.
[26] Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. 2018. Multi-task Representation Learning for Travel Time Estimation. In *SIGKDD*. 1695–1704.
[27] Yan Lin, Huaiyu Wan, Jilin Hu, Shengnan Guo, Bin Yang, Youfang Lin, and Christian S. Jensen. 2023. Origin-Destination Travel Time Oracle for Map-based Services. *SIGMOD* 1, 3 (2023), 217:1–217:27.
[28] Todd Litman. 2009. Transportation cost and benefit analysis. *Victoria Transport Policy Institute* 31, 1 (2009), 9.
[29] Wuman Luo, Haoyu Tan, Lei Chen, and Lionel M. Ni. 2013. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*. 713–724.
[30] Zihan Luo, Lei Li, Mengxuan Zhang, Wen Hua, Yehong Xu, and Xiaofang Zhou. 2022. Diversified Top-k Route Planning in Road Network. *PVLDB* 15, 11 (2022), 3199–3212.
[31] Xiaowei Mao, Yan Lin, Shengnan Guo, Yubin Chen, Xingyu Xian, Haomin Wen, Qisen Xu, Youfang Lin, and Huaiyu Wan. 2025. DutyTTE: Deciphering Uncertainty in Origin-Destination Travel Time Estimation. In *AAAI*. 12390–12398.
[32] Can Rong, Jingtao Ding, and Yong Li. 2025. An Interdisciplinary Survey on Origin-destination Flows Modeling: Theory and Techniques. *ACM Comput. Surv.* 57, 1 (2025), 4:1–4:49.
[33] Sijie Ruan, Zi Xiong, Cheng Long, Yiheng Chen, Jie Bao, Tianfu He, Ruiyuan Li, Shengnan Wu, Zhongyuan Jiang, and Yu Zheng. 2020. Doing in One Go: Delivery Time Inference Based on Couriers' Trajectories. In *SIGKDD*. 2813–2821.
[34] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *ICLR*.
[35] Wei Tian, Jieming Shi, Siqiang Luo, Hui Li, Xike Xie, and Yuanhang Zou. 2023. Effective and Efficient Route Planning Using Historical Trajectories on Road Networks. *PVLDB* 16, 10 (2023), 2512–2524.
[36] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[37] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. 2021. On Position Embeddings in BERT. In *ICLR*.
[38] Chenxing Wang, Fang Zhao, Haichao Zhang, Haiyong Luo, Yanjun Qin, and Yuchen Fang. 2022. Fine-Grained Trajectory-Based Travel Time Estimation for Multi-City Scenarios Based on Deep Meta-Learning. *IEEE Trans. Intell. Transp. Syst.* 23, 9 (2022), 15716–15728.
[39] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. 2018. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. In *AAAI*. 2500–2507.
[40] Hongjian Wang, Xianfeng Tang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. 2019. A Simple Baseline for Travel Time Estimation using Large-scale Trip Data. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019), 19:1–19:22.
[41] Hongjun Wang, Zhiwen Zhang, Zipei Fan, Jiyuan Chen, Lingyu Zhang, Ryosuke Shibasaki, and Xuan Song. 2023. Multi-Task Weakly Supervised Learning for Origin-Destination Travel Time Estimation. *IEEE Trans. Knowl. Data Eng.* 35, 11 (2023), 11628–11641.
[42] Jingyuan Wang, Ning Wu, Wayne Xin Zhao, Fanzhang Peng, and Xin Lin. 2019. Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. In *KDD*. 539–547.
[43] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *SIGKDD*. 25–34.
[44] Zheng Wang, Kun Fu, and Jieping Ye. 2018. Learning to Estimate the Travel Time. In *SIGKDD*. 858–866.
[45] Haomin Wen, Youfang Lin, Fan Wu, Huaiyu Wan, Shengnan Guo, Lixia Wu, Chao Song, and Yinghui Xu. 2021. Package Pick-up Route Prediction via Modeling Couriers' Spatial-Temporal Behaviors. In *ICDE*. 2141–2146.
[46] Fan Wu and Lixia Wu. 2019. DeepETA: A Spatial-Temporal Sequential Neural Network Model for Estimating Time of Arrival in Package Delivery System. In *AAAI*. 774–781.
[47] Jia-Jie Xu, Saijun Xu, Rui Zhou, Chengfei Liu, An Liu, and Lei Zhao. 2021. TAML: A Traffic-aware Multi-task Learning Model for Estimating Travel Time. *ACM Trans. Intell. Syst. Technol.* 12, 6 (2021), 75:1–75:14.
[48] Saijun Xu, Jiajie Xu, Rui Zhou, Chengfei Liu, Zhixu Li, and An Liu. 2020. TADNM: A Transportation-Mode Aware Deep Neural Model for Travel Time Estimation. In *DASFAA*, Vol. 12112. 468–484.
[49] Can Yang and Gyözö Gidófalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *Int. J. Geogr. Inf. Sci.* 32, 3 (2018), 547–570.
[50] Haitao Yuan, Guoliang Li, and Zhifeng Bao. 2022. Route Travel Time Estimation on A Road Network Revisited: Heterogeneity, Proximity, Periodicity and Dynamicity. *PVLDB* 16, 3 (2022), 393–405.
[51] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *SIGMOD*. 2135–2149.
[52] Zhihan Zheng, Haitao Yuan, Minxiao Chen, and Shangguang Wang. 2025. RLER-TTE: An Efficient and Effective Framework for En Route Travel Time Estimation with Reinforcement Learning. *SIGMOD* 3, 1 (2025), 71:1–71:26.

A Online and Offline Processing

A.1 Offline Training

Algorithm 1 outlines the offline training procedure. Initially, we preprocess $\mathcal{D}$ to construct training instances comprising the ODT query q, ground-truth route, and segment travel times (Lines 1–4). We initialize learnable parameters Θ, utilizing pre-trained Node2Vec embeddings and orthogonal initialization for the experts in mSTE (Lines 5–8). The training process iterates over at most I epochs

Algorithm 1: UNITE Offline Training

Input: Training Data $\mathcal{D}$
Output: Trained Model Parameters Θ

1 **foreach** *trajectory* $T \in \mathcal{D}$ **do**
2 $q \leftarrow \langle T.o, T.d, T.t_o \rangle$;
3 $R \leftarrow$ ground-truth route from $T.o$ to $T.d$;
4 $\langle \psi_1, \ldots, \psi_{|R|} \rangle \leftarrow$ Get ground-truth segment travel times;
5 $\mathbf{W}_G \leftarrow \text{Node2Vec}(G)$;
6 Initialize $\mathbf{W}^o, \mathbf{W}^d, \mathbf{W}^R, \mathbf{W}^C, \mathbf{W}^L$ with $\mathbf{W}_G$;
7 Orthogonally initialize mSTE experts with distinct seeds;
8 Initialize remaining parameters in Θ via normal distribution;
9 **for** *epoch* $\leftarrow 1$ ***to*** I **do**
10 Shuffle $\mathcal{D}$ and partition into batches $\mathcal{B}$;
11 **foreach** *training trajectory* T *in each batch in* $\mathcal{B}$ **do**
12 **foreach** $e_i \in$ *ground-truth* R *of* T **do**
13 Forward execution of mSTE module (Lines 5-8 in Algorithm 2) to get estimated travel time ϕ_i;
14 Forward execution of NSP module (Lines 13-16 in Algorithm 2) to get $P(c|e_i), \forall c \in N_O(e_i)$;
15 $L_{\text{mSTE}}(T) \leftarrow$ Compute MSE loss by Eq. (18);
16 $L_{\text{NSP}}(T) \leftarrow$ Compute BCE loss by Eq. (19);
17 $\mathcal{L}(T;\Theta) \leftarrow$ Compute loss for T by Eq. (20);
18 Get batch loss $\mathcal{L}_{batch}(\Theta)$;
19 Compute gradients $\nabla_\Theta \mathcal{L}_{batch}$;
20 Update $\Theta \leftarrow \text{Adam}(\Theta, \nabla_\Theta \mathcal{L}_{batch})$;
21 **return** UNITE with Θ;

Algorithm 2: UNITE Online Inference

Input: ODTTE: $q = \langle o, d, t_o \rangle$ or RTTE: $q_R = (R, t_o)$
Output: Estimated Travel Time Φ

1 $\mathbf{q} \leftarrow$ Encode query q via ODT Encoder (Eq. (4));
2 $\mathbf{h}_0 \leftarrow \mathbf{q}$; $\Phi \leftarrow 0$; $i \leftarrow 1$; if ODTTE: $R \leftarrow \{o\}$;
3 **while** $i \le |R|$ ***and*** $i \le$ *MaxSteps* **do**
4 Current segment $e_i \leftarrow R[i]$;
// 1. Progressive State Encoding (Sec. 4.2)
5 $\mathbf{g}_i \leftarrow$ Generate composite embedding from e_i, travel time histogram, and previously visited segments (Eqs. (6)–(8));
6 $\mathbf{h}_i \leftarrow$ Update state with $\mathbf{g}_i$ of e_i and $\mathbf{h}_{i-1}$ via GRU (Eq. (9));
// 2. Segment Travel Time Estimation (Sec. 4.3)
7 $\mathcal{G}_i \leftarrow$ Construct the travel graph for e_i at t_o;
8 $\phi_i \leftarrow$ Estimate time via mSTE($\mathbf{q}, \mathbf{h}_i, \mathcal{G}_i$) (Eq. (11) (10));
9 $\Phi \leftarrow \Phi + \phi_i$;
10 **if** e_i *is* d **then**
11 **break**
// 3. Next Segment Prediction (Sec. 4.4)
12 **if** *ODTTE* **then**
13 **foreach** $c_j \in N_O(e_i)$ **do**
14 $\mathbf{c}_j \leftarrow$ Generate candidate embedding (Eqs. (13)–(16));
15 $P(c_j|e_i) \leftarrow$ Compute transition probability (Eq. (17));
16 $e_{i+1} \leftarrow \arg\max_{c \in N_O(e_i)} P(c|e_i)$;
17 R.append(e_{i+1});
18 $i \leftarrow i + 1$;
19 **return** Φ;

(Lines 9–20). In each epoch, training data is shuffled and divided into mini-batches (Lines 9–10). For every trajectory T within a batch, we traverse ground-truth route R. At each segment e_i, we perform the forward propagation of mSTE module (Sec. 4.3) to predict travel time ϕ_i, and the NSP module (Sec. 4.4) to calculate the transition probabilities for the next segment (Lines 11–14). Based on these outputs, we compute the regression loss L_{mSTE} (Eq. (18)) and the classification loss L_{NSP} (Eq. (19)), which are combined into a trajectory loss $\mathcal{L}(T;\Theta)$ as per Eq. (20) (Lines 15–17). Finally, the model parameters Θ are updated via the Adam optimizer [23] by minimizing the average batch loss (Lines 18–20). Lastly, the trained UNITE with optimized parameters Θ is returned (Line 21).

A.2 Online Inference

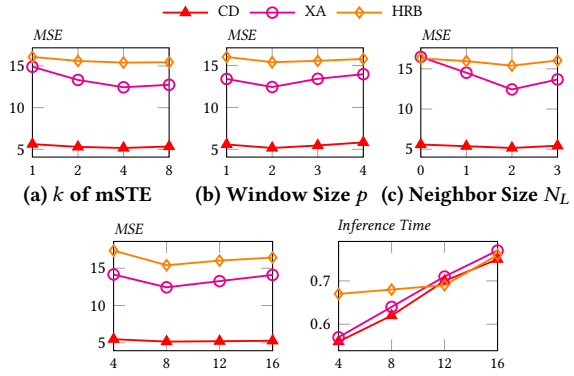
Algorithm 2 outlines the online inference procedure of UNITE, which handles both ODTTE and RTTE queries in a unified manner. The process begins by encoding the origin o, destination d, and departure time t_o into $\mathbf{q}$ (Line 1) using the ODT Encoder (Sec. 4.1). At Line 2, the state $\mathbf{h}_0$ is initialized with $\mathbf{q}$, and the cumulative travel time Φ is set to zero. For ODTTE queries, the route R is initialized with the origin segment o. In each iteration (Lines 3–18), UNITE performs three key operations for current segment e_i retrieved at Line 4. First, the PSE module (Sec. 4.2) generates the composite embedding $\mathbf{g}_i$ and updates the hidden state $\mathbf{h}_i$ (Lines 5, 6). Second, the mSTE module (Sec. 4.3) constructs the travel graph $\mathcal{G}_i$ and estimates the segment travel time ϕ_i on segment e_i (Lines 7, 8) and accumulates it into Φ (Line 9). If the current segment e_i matches the destination d, the loop breaks (Lines 10, 11). Third, for ODTTE tasks, the NSP module (Sec. 4.4) predicts the next segment e_{i+1} by selecting the candidate with the highest probability and appending e_{i+1} to the route R (Lines 12–17). For RTTE, this NSP is skipped as the route is provided. Then the iteration index i is incremented (Line 18) and the loop continues. The iteration continues until the route is fully traversed (for RTTE) or the destination is reached, or a maximum step limit is met (for ODTTE) in Lines 3 and 10. Finally, UNITE returns the total estimated travel time Φ (Line 19).

Complexity. Let d be the dimension of embeddings involved, $\tilde{deg}$ be the max value of $|N_O(e_i)|$ for any e_i in G, ℓ_N be the max value of multi-hop look-ahead neighborhood size $|N_L(e_i)|$ for any e_i in G, and $\ell_{\mathcal{G}}$ be the number of edges in travel graph $\mathcal{G}$. UNITE in Algorithm 2 iterates at most ℓ steps. In the PSE process, we compute attention between current segment and previously visited segments in $O(\ell d)$ time, and the time of MLPs is $O(d^2)$. In mSTE module, it contains GAT which costs $O(k\ell_{\mathcal{G}} d)$ time, while NSP module runs in $O(\tilde{deg}\ell_N d)$ time. Thus the overall time complexity of UNITE is $O(\ell^2 d + \ell d^2 + k\ell\ell_{\mathcal{G}} d + \tilde{deg}\ell\ell_N d)$, where k is constant, $\tilde{deg}$, $\ell_{\mathcal{G}}$ and ℓ_N are usually small, indicating the efficiency of UNITE.

B Additional Evaluations

B.1 The Effect of Hyper-parameters

Effectiveness Study on mSTE. In mSTE, we activate the Top-k STEs for travel time estimation to reduce inference cost (Sec. 4.3). We vary k in $\{1, 2, 4, 8\}$ and report the MSE on all three datasets in Figure 10(a). As shown in Figure 10(a), MSE decreases as k increases from 1 to 4, indicating improved effectiveness, and then



(a) k of mSTE (b) Window Size p (c) Neighbor Size N_L

(d) Number of Experts (e) Number of Experts

Figure 10: Effectiveness of Hyper-parameters

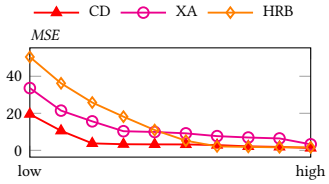


Figure 11: MSE on Different F1-score of Routes

remains relatively stable from 4 to 8. This demonstrates that using an appropriate number of STEs improves performance, thus we set $k = 4$ as the default in mSTE. When $k = 1$, mSTE reduces to a single STE, which yields the worst performance, validating the effectiveness of the noisy Top-k selection in mSTE.

Temporal Window Size. In mSTE, we introduce window size p to capture temporal dynamics (Sec. 4.3). We vary p from 1 to 4 and report MSE on all three datasets in Figure 10(b). As shown in Figure 10(b), MSE decreases when varying p from 1 to 2, and slightly increases from 2 to 4. Thus, it is optimal to set p to 2.

Neighborhood Size. In NSP, we design multi-hop look-ahead neighborhood N_L to enhance representation for candidate segment (Sec. 4.4). Figure 10(c) shows performance of UNITE on all three datasets with different neighborhood size. Compared to $N_L = 0$ (no multi-hop), increasing N_L from 0 to 2 improves MSE, which then stabilizes with slight increase for 2 to 3. This shows the necessity of multi-hop neighbor mechanism, and we set $N_L = 2$.

Number of Experts. We vary the number of experts in the mSTE MoE module, with MSE and inference time per 1,000 queries in Figure 10(d,e). Increasing the number of experts from 4 to 8 improves MSE, after which the gains plateau or slightly degrade; inference time increases only slightly. This shows the MoE design is effective and necessary, while the inference overhead is manageable. Thus, we set the number of experts to 8.

B.2 Quality of Predicted Routes

As UNITE predicts intermediate routes via the NSP module, we compare its route quality with recent route planning methods, further validating the joint design of UNITE.

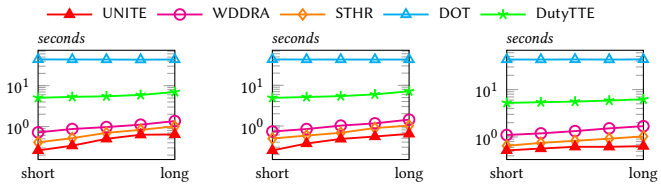
Our method UNITE, equipped with the NSP module, generates predicted routes for ODTTE queries. We evaluate the quality of these predicted routes using the F1-score, comparing against state-of-the-art route planning methods, including NASR [42], NMLR [19], and DRPK [35]. The F1-score is a widely adopted metric for assessing the similarity between predicted and ground-truth routes [19, 35]; a higher F1-score indicates better route prediction quality. The results in Table 6 show that our method UNITE consistently achieves the highest F1-score across all datasets, outperforming all competitors by notable margins. This demonstrates the effectiveness of our NSP module. Besides, we partition ODTTE queries into 10 equal-sized groups by F1 of NSP routes, and report MSE of UNITE in Figure 11. As the F1-score of routes increases, MSE decreases, confirming that accurate route yields better MSE.

Table 6: Effectiveness of NSP by F1-score

Method	CD	XA	HRB
NASR	78.27	75.22	57.43
NMLR	87.25	84.01	65.65
DRPK	88.35	85.18	68.36
UNITE	**90.39**	**87.64**	**71.53**

Table 7: Inference Time Breakdown per 1k Queries (seconds)

Component	CD	XA	HRB
PSE	0.10	0.09	0.08
mSTE	0.26	0.24	0.25
NSP	0.23	0.21	0.24



(a) CD (b) XA (c) HRB

Figure 12: Inference Time of ODTTE with different lengths

B.3 Inference Time Exploration

Breakdown of Components. We break down inference time of UNITE in Table 7, including inference time of PSE at Lines 5–6 of Algorithm 2 (Sec. 4.2), mSTE at Lines 7–11 of Algorithm 2 (Sec. 4.3), and NSP at Lines 12–17 of Algorithm 2 (Sec. 4.4). First, the generation of progressive state representation is highly efficient. For example, on HRB, PSE costs only 0.08 seconds per 1,000 queries. Second, mSTE and NSP are the main overhead in UNITE. We also break down the inference time for the rightmost bin in Figure 7, showing PSE is still not the main cost. Nevertheless, as reported in Figure 3, UNITE needs the shortest total inference time.

Impact of Different Lengths. To explore the inference speed of methods with different route lengths, we also evenly divide the test ODTTE queries into five groups based on the number of segments in their ground-truth routes. Figure 12 displays the inference time per 1,000 queries of all methods under the five groups from short to long over three datasets. As the route length increases, DOT maintains relatively stable inference time, while the inference time of other methods all rise. This is because DOT predicts the route in pixelated format, which is not related to the length between the origin and destination. Among the rest of methods, UNITE exhibits the flattest rise. Nevertheless, UNITE costs the least seconds than all competitors on all datasets with different route lengths, demonstrating the superiority of the unique design in UNITE.