TrajFormer: Efficient Trajectory Classification with Transformers

Yuxuan Liang
Kun Ouyang
Yiwei Wang
Xu Liu
National University of Singapore, Singapore
yuxliang@comp.nus.edu.sg

Hongyang Chen
Zhejiang Lab, Hangzhou
h.chen@ieee.org

Junbo Zhang
Yu Zheng
JD Intelligent Cities Research & JD iCity, JD Technology, China
msjunbozhang@outlook
msyuzheng@outlook

Roger Zimmermann
National University of Singapore, Singapore
rogerz@comp.nus.edu.sg

ABSTRACT

Transformers have been an efficient alternative to recurrent neural networks in many sequential learning tasks. When adapting transformers to modeling trajectories, we encounter two major issues. First, being originally designed for language modeling, transformers assume regular intervals between input tokens, which contradicts the irregularity of trajectories. Second, transformers often suffer high computational costs, especially for long trajectories. In this paper, we address these challenges by presenting a novel transformer architecture entitled TrajFormer. Our model first generates continuous point embeddings by jointly considering the input features and the information of spatio-temporal intervals, and then adopts a squeeze function to speed up the representation learning. Moreover, we introduce an auxiliary loss to ease the training of transformers using the supervision signals provided by all output tokens. Extensive experiments verify that our TrajFormer achieves a preferable speed-accuracy balance compared to existing approaches.

CCS CONCEPTS

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

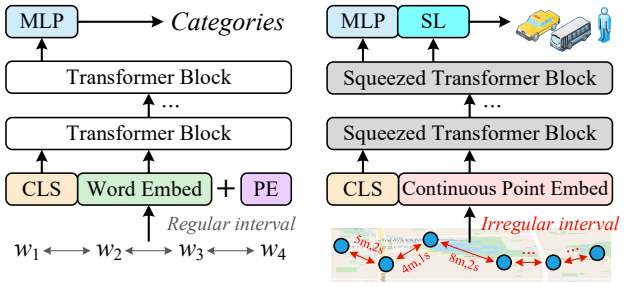
KEYWORDS

Trajectory classification; urban computing; transformer

ACM Reference Format:
Yuxuan Liang, Kun Ouyang, Yiwei Wang, Xu Liu, Hongyang Chen, Junbo Zhang, Yu Zheng, and Roger Zimmermann. 2022. TrajFormer: Efficient Trajectory Classification with Transformers. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3511808.3557481

1 INTRODUCTION

Trajectory classification is one of the fundamental topics in smart cities. It aims to differentiate between trajectories of different aspects, such as motions, activities, and transportation modes. Recurrent Neural Networks (RNNs) have long been the dominant models for trajectory classification [18, 19, 30]. They can easily be applied to trajectories of arbitrary length, thus being more flexible than classical methods such as SVM and decision trees. Though successful, their *efficiency* is far inferior to other deep learning counterparts (e.g., CNNs) because of their inherently sequential nature.



(a) Standard transformer for texts (b) Our trajectory transformer

Figure 1: (a) w_i denotes the i-th word in a sentence. (b) m: meter. s: second. SL: subpath labeling.

Recently, the natural language processing (NLP) community has witnessed a shift from RNNs to self-attention methods, in particular, Transformers [27]. Compared with RNNs, transformers not only enjoy a better parallelization in computation, but also have achieved new state-of-the-art results across various NLP tasks, such as text classification. Considering the intrinsic similarities between natural language and trajectories (e.g., sequential nature, contextual interrelations), one may ask: *can we adapt a transformer as an efficient alternative for trajectory classification?*

To further illustrate, Figure 1(a) depicts a sketch of a transformer for text classification [7]. The input contains word embeddings, position encoding (PE), and an extra learnable class token (CLS). All these tokens go through multiple transformer blocks to learn a representation of the sentence. Finally, the class token is employed to perform classification. Here we highlight two key factors that limit the direct application of transformers to trajectory classification:

- *Position encoding*: Being originally designed for language modeling, a strong assumption of standard transformers is the regular intervals between words. Since self-attention is permutation-invariant, PE is employed to encode the discrete order information of sentences, e.g., 1st, 2nd and 3rd. However, such an encoding cannot capture the irregular nature of trajectories – the variable spatio-temporal intervals between consecutive points, which have been proven in the literature to be an extremely important trait of trajectories [18, 34]. For example, two points in a shorter time period tend to be more related. Failing to capture this characteristic may result in suboptimal performance.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557481

- *Scalability of self-attention*: As the key component of transformers, self-attention induces quadratic computational costs with respect to the number of points within the sequence. Such costs may become unaffordable with the increase of sampling rate, especially for long trajectories with hundreds or even thousands of points. Therefore, how to improve the scalability of self-attention for modeling long trajectories remains an open problem.

To tackle these issues, we present *TrajFormer*, a novel transformer architecture for efficient trajectory classification, as shown in Figure 1(b). Primarily, unlike transformers for NLP, we unify the tasks of learning point embeddings and capturing positional information into a single layer called *Continuous Point Embedding* (CPE). In CPE, embeddings are dynamically generated and conditioned on two crucial factors – the spatio-temporal intervals and the local neighborhood of each input token. Thanks to the locality of CPE, it can generalize to any input length and implicitly provide absolute position information to some extent. Moreover, to capture spatio-temporal dependencies between GPS points, we present a *Squeezed Transformer Block* which can significantly reduce the number of keys and values in attention computations to boost efficiency. By controlling the squeeze rate, it can achieve a preferable trade-off between speed and accuracy against standard transformers.

In addition, high-performance transformers generally require a heavy pre-training step using large corpora, limiting their adoption to relatively smaller trajectory datasets. Inspired by [13], we further enhance TrajFormer with an auxiliary training objective to improve its performance. This new training objective provides additional supervision to the transformers by assigning each subpath (consisting of several points) with an individual label generated by a machine annotator. In summary, our contributions lie in three aspects:

- To the best of our knowledge, we are the first to present a transformer-based approach for efficient trajectory classification. Our transformer model can not only capture the irregularity of trajectories, but is also more efficient than standard transformers by squeezing the key-value space.
- We present a new teacher-forcing strategy for training high-performance transformers on modeling trajectories.
- We conduct extensive experiments on two mobility datasets to validate the superiority of our TrajFormer in terms of accuracy and throughput. Compared to the state-of-the-art RNN, i.e., TrajODE [18], our model can achieve comparable performance while running **26.2**× and **31.1**× faster on the two datasets, respectively.

2 RELATED WORKS

Trajectory Classification

A *trajectory* is a sequence of points $p_1 \rightarrow \cdots \rightarrow p_n$, where each point $p_i = (a_i, b_i, t_i)$ contains longitude a_i, latitude b_i, and a timestamp t_i. Modeling trajectories fosters a wide range of applications in smart cities [2, 4, 8, 16, 17, 22–24, 32], such as trajectory classification. Zheng et al. [35, 36] first manually discovered effective features (e.g., movement distance, velocities, acceleration) that can reflect the semantics of a trajectory, and then applied traditional algorithms (e.g., SVM, decision trees) to detect the transportation modes of a user's trajectory. To reduce human efforts in feature extraction, Wu et al. presented the first RNN scheme for modeling trajectories [30]. Afterward, a line of studies [19, 34, 38] integrated various

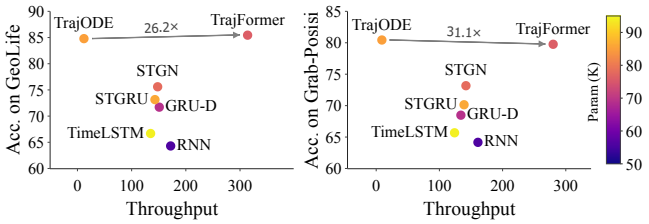


Figure 2: Accuracy (%) vs. throughput on GeoLife [37] and Grab-Posisi [11]. Throughput means the number of trajectories the network can process in a second during inference.

gating mechanisms into RNNs to tackle the irregularity issue. The recent state-of-the-art [18] enhanced RNNs to be a continuous-time model with continuous states obeying ordinary differential equations (ODE) between consecutive points. Despite their success, the sequential nature of these RNN-based models inevitably precludes parallelizing training or inference, leading to extremely low efficiency. In this paper, we perform efficient trajectory classification by taking advantage of the parallelization of transformers.

Transformers & Self-Attention

Self-attention-based architectures, in particular Transformers [27], have become the de facto most popular models for sequential learning tasks by virtue of their strong capability of capturing long-range relations. A transformer block contains a stack of transformer layers, and each layer is characterized by two parts: multi-head self-attention (MSA) and a position-wise feed-forward network (FFN).

MSA is the key operation of transformers to learn an alignment where each token in the sequence learns to gather messages from other tokens. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the input sequence with length n and feature dimension d. The operation of one head is defined as:

$$\mathbf{X}_h = \text{Softmax}\left(\alpha \mathbf{Q}_h \mathbf{K}_h^\top\right)\mathbf{V}_h, \tag{1}$$

where $\mathbf{X}_h \in \mathbb{R}^{n \times d/N_h}$ is the updated features; $\mathbf{Q}_h = \mathbf{X}\mathbf{W}_q$, $\mathbf{K}_h = \mathbf{X}\mathbf{W}_k$ and $\mathbf{V}_h = \mathbf{X}\mathbf{W}_v$ are linear transformations applied on the temporal dimension of the sequence; $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d \times d/N_h}$ are the learnable parameters for the query, key and value projections, and N_h is the number of heads; α is a scaling factor for magnitude control. The computational complexity required in Eq. (1) is quadratic w.r.t the sequence length. When modeling long sequences, some follow-ups have explored more efficient ways to compute attention [14, 20, 29]. Although there are several concurrent works [1, 31] using transformers for trajectory prediction, transformer-based trajectory classification is still less explored in the community.

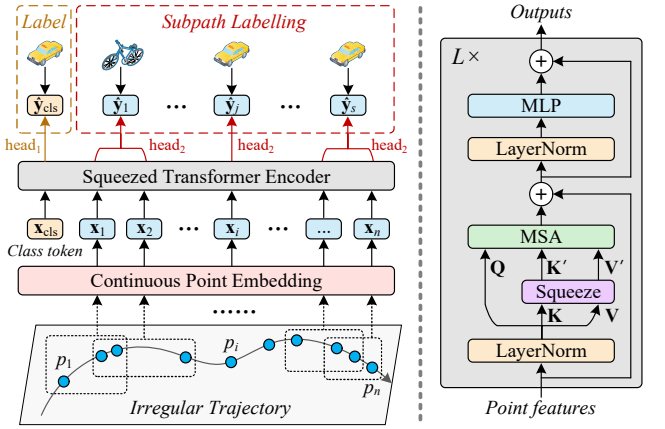
Positional Encoding

As MSA is permutation-equivalent, transformers employ positional encoding (PE) to incorporate the order information of input sequences. Generally, PE can be categorized into two groups: absolute PE and relative PE. Within the former class, PE is complemented to the MSA inputs, and can either be learnable [7] or fixed with some functions [27], such as sinusoidal functions of different frequencies. In contrast, the second group of PE considers the relative distances between tokens, which can be viewed as a bias added to the attention weights [25]. However, both groups assume regular intervals and use discrete values to index the embedding table, which fails to encode the variable spatio-temporal intervals in trajectories.

3 METHODOLOGY

Figure 3 depicts the overall architecture of the proposed TrajFormer to identify the mode of a given trajectory. Our model is composed of three major stages as follows:

- *Continuous point embedding*: At the first stage, the raw trajectory is fed to the CPE to learn the embeddings of each point, where n is the sequence length. The embedding of each point is adaptively generated based on its local neighborhood and the spatio-temporal intervals between tokens.
- *Squeezed Transformer Encoder*: After CPE, the point embeddings $\{\mathbf{x}_i\}_{i=1}^{n}$ are appended with a class token $\mathbf{x}_0$, and go through L squeezed transformer blocks for learning representations. As its name implies, this encoder compresses the dimensions of key $\mathbf{K}$ and values $\mathbf{V}$ before attention computations (see Figure 3(b)), thereby significantly reducing the computational overheads.
- *Prediction & subpath labeling*: At the top of TrajFormer, there are two types of decoders: head_1 performs classification using the class token and obtains the prediction $\hat{\mathbf{y}}_{cls}$; the other decoder head_2 takes advantage of all output tokens (in the red box) by assigning each subpath an individual label generated by a machine annotator as additional supervision.



(a) Overall architecture of Trajormer (b) Squeezed Transformer

Figure 3: The framework of TrajFormer.

3.1 Continuous Point Embedding

3.1.1 Module Design. In NLP tasks, one of the most influential factors is the quality of word embeddings that are normally pre-trained on a large-scale corpus, such as BERT [7]. Conversely, in the literature of trajectory modeling [18, 28], the embedding of each GPS point is jointly trained with the whole network. Here, we argue that a powerful and efficient point embedding mechanism should meet the following four requirements:

(1) Being trained in a **parallel** manner (i.e., no recurrence).
(2) Being able to model **arbitrarily long** trajectories.
(3) Providing **position information** (either absolute or relative) to a certain degree, which is of great importance to many sequential tasks [12].
(4) Being aware of the **spatio-temporal intervals** between a token and its nearby tokens.

We notice that the convolution-based methods [6, 28] satisfy the first three demands, while they utilize kernels with regular intervals to aggregate neighbors' features without considering variable spatio-temporal intervals. The ODE-based encoding [21] supports the last three requirements but cannot be parallel. Based on these findings, can we combine the parallelization provided by convolutions with the continuity of some functions (e.g., integral) for generating trajectory embeddings?

Here, we give an affirmative answer by *extending convolutions to be continuous in both temporal and spatial domains*, which is sufficient to meet all of the above requirements. Before introducing our method, let us recall the formula of 1D convolutions with discrete kernels:

$$\text{Conv}(x) = (f * p)(x) = \sum_{y=-k}^{k} f(y)p(x+y), \tag{2}$$

where $p : \mathbb{Z} \to \mathbb{R}$ represents the input sequence; $f : \mathcal{F} \to \mathbb{R}$ denotes the discrete kernel applied to the sequence, and $\mathcal{F} = \{-k, -k+1, \cdots, k\}$. Both f and p are functions defined over the support domain of finite integer set. By integrating continuity into Eq. (2), we propose a Continuous Point Embedding (CPE) as:

$$\text{CPE}(x) = (f * p)(x) = \int_{-k}^{k} f(y)p(x+y)dy, \tag{3}$$

where $p : \mathbb{R} \to \mathbb{R}$ and $f : \mathcal{F} \to \mathbb{R}$ are continuous on their whole domain, and $\mathcal{F}$ ranges from $-k$ to k.

However, the form in Eq. (3) requires the integration to be analytically tractable. In practice, objects move continuously throughout a city while their locations can only be updated at discrete times. To address this issue, we approximate the continuous convolution by Monte Carlo integration as:

$$\text{CPE}(x) \approx \sum_{u \in \mathcal{N}(x)} \frac{1}{|\mathcal{N}(x)|} g\left(\Delta t_{u,x}, \Delta d_{u,x}\right) p\left(u\right), \tag{4}$$

where $\mathcal{N}(x)$ is the set of neighbor points within a local window and the window size is k. The continuous function g is parameterized by an MLP that takes the spatio-temporal intervals (time interval Δt and geographic distance Δd) between the anchor point x and the nearby token u as inputs to adaptively generate the corresponding kernel. Generally, shorter spatio-temporal intervals indicate higher correlations. To help better understand how CPE works, Figure 4 shows an example of the pipeline.

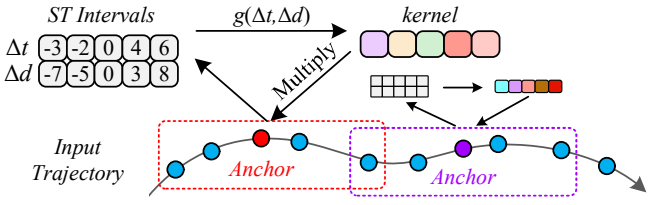


Figure 4: Continuous position embeddings. For each local window (e.g., the red rectangle with $k = 5$), we first compute the spatio-temporal (ST) intervals of each point to the anchor, and then employ an MLP function g to convert them to convolution kernels. Finally, we apply the dynamic kernels to the inputs and update the anchor.

Next, we elaborate on how CPE addresses the above four requirements, respectively. Firstly, CPE does not leverage any recurrent structure and can thus be efficiently trained via modern deep learning tools. Secondly, it is adaptable to sequences with arbitrary length because only the local neighborhoods of a point are involved in the computations. Thirdly, locality-based operations can implicitly learn to encode the absolute position information, which has been verified in [6, 12]. Lastly, our CPE generates dynamic convolution kernels based on the spatio-temporal intervals between the anchor point and its neighbors, thus considering the irregularity of trajectories in a principle way.

3.1.2 Implementation. As shown in Algorithm 1, our CPE can be implemented in just a few lines of code (in a PyTorch-like style). In Line 6-8, we initialize all neural network layers. Following the trajectory feature extraction method in [18], we first extract e-dimensional point features[1] for each point and then feed them into the CPE module. In Line 13-15, we produce the continuous kernels based on the spatio-temporal interval information. Note that we also employ the "multi-head" trick for CPE to improve the ability of representation learning. In Line 16-19, we employ the unfold operation to obtain the neighbor features of each point and perform a linear transformation. Finally, we multiply the features by the derived continuous kernels in Line 20 to obtain the outputs.

Algorithm 1: Implementation of continuous point embeddings in a PyTorch-like style.

```
1  # Variables are illustrated below:
2  # b: batch size, t: sequence length
3  # e: input feature dimension, d: hidden channel number
4  # k: kernel size, h: number of heads
5  # ==================Initialization==================
6  fc1 = nn.Linear(e, d)
7  st_mlp = nn.Sequential(nn.Linear(2, 64), nn.ReLU(),
     nn.Linear(64, 128), nn.ReLU(), nn.Linear(128, h))
8  fc2 = nn.Linear(d, d)
9  # ====================Forward=====================
10 # Inputs are illustrated below:
11 # X: point features with shape [b, t, e]
12 # Delta: spatio-temporal interval information in each
     local window; its shape is [b, t, k, 2]
13 kernel = st_mlp(Delta) # b,t,k,h
14 kernel = kernel.reshape(-1, k, h) # b*t,k,h
15 kernel = kernel.permute(0, 2, 1).unsqueeze(2) #
     b*t,h,1,k
16 X = fc1(X).transpose(1, 2).unsqueeze(-1) # b, d, t, 1
17 tmp = nn.functional.unfold(X, (k, 1), 1, ((k-1)//2, 0),
     1).reshape(b, d, k, -1) # b, d, k, t
18 tmp = tmp.permute(0, 3, 2, 1).reshape(-1, k, d) # b*t,
     k, d
19 v = fc2(tmp).reshape(b*t, k, h, d//h).permute(0, 2, 1,
     4) # b*t, h, k, d/h
20 x = (kernel @ v).squeeze().reshape(b, t, d) # b, t, d
21 return x # output features with shape [b, t, d]
```

[1]Besides latitude and longitude, we follow a prior study [18] to approximate the velocity and acceleration of each point, leading to four attributes per point ($e = 4$).

3.2 Squeezed Transformer Encoder

3.2.1 Overview. Given the embeddings of all points $\mathbf{X} \in \mathbb{R}^{n \times d}$, the complexity of MSA is $O(n^2 d)$ where n is the number of points and d is the feature dimension, making it inefficient to accommodate long trajectories. Thus, we propose a squeezed transformer encoder to alleviate this issue. As depicted in Figure 3(b), in each squeezed transformer block, we first normalize the input features and then generate the queries, keys and values using non-shared projections. Once the queries, keys and values are ready, we utilize a squeeze function to reduce the number of keys/values and perform MSA to learn spatio-temporal dependencies. Finally, the features are passed to a layer normalization and an MLP for non-linear transformation.

3.2.2 Squeezed Transformer Block. The major difference between a standard MSA and our Squeezed MSA is the *squeeze function* applied before MSA. Assume that the number of keys/values after reduction is m in each layer, we expect $m < n$ (or even $m \ll n$) to improve the efficiency of MSA. Under this circumstance, the computational complexity has been reduced to $O(mnd)$, $\frac{n}{m}$ times less than standard MSA. For simplicity, we use a *squeeze rate* r to denote this percentage of reduction. By controlling r, we achieve a trade-off between the efficiency and accuracy, e.g., a squeezed transformer with higher r runs faster but may degrade the performance.

We give a unified formulation of the squeeze function by introducing a simple yet effective term – **assignment matrix**. As depicted in Figure 5, the assignment matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$ describes how the points in the original trajectory space are aggregated into the latent space with fewer nodes, where each entry $b_{ij} \geq 0$ denotes the likelihood that the j-th trajectory point belongs to the latent node i. We guarantee $\mathbf{B}$ has *each row summing to zero* and *each non-zero entry in the same row is evenly distributed.*

Given the assignment matrix $\mathbf{B}$, the original keys $\mathbf{K}_h \in \mathbb{R}^{n \times d/N_h}$ and values $\mathbf{V}_h \in \mathbb{R}^{n \times d/N_h}$, the squeeze function of each head is a linear projection between the two spaces, which is defined as:

$$\mathbf{K}'_h = \mathbf{B}^\top \mathbf{K}_h, \quad \mathbf{V}'_h = \mathbf{B}^\top \mathbf{V}_h, \tag{5}$$

where $\mathbf{K}'_h, \mathbf{V}'_h \in \mathbb{R}^{m \times d/N_h}$ are the squeezed keys and values, respectively; N_h is the number of heads. After compressing the key-value space, we perform MSA to capture the spatio-temporal correlations between the points (in the trajectory space) and latent nodes as:

$$\mathbf{X}_h = \text{Softmax}\left(\alpha \mathbf{Q}_h \mathbf{K}'^\top_h\right) \mathbf{V}'_h, \tag{6}$$

where $\mathbf{X}_h \in \mathbb{R}^{n \times d/N_h}$ is the updated features and α is a scaling factor for magnitude control.

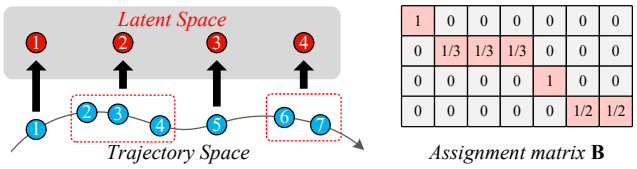


Figure 5: An example to illustrate how the assignment matrix B transforms the input keys and values from the trajectory space to the latent space. The squeeze rate of this example is $(7-4)/7 = 42.9\%$.

3.2.3 Squeeze Function. The assignment matrix can be either fixed based on some algorithms or totally learnable, which is flexible to a variety of trajectory-based applications. We propose several feasible variants of the squeeze function as follows:

Pooling-based squeeze. 1D average pooling [15], which has been widely used for shrinking sequences to reduce costs, can be interpreted as the simplest variant of a squeeze function. For example, we can implement a squeeze function with $r = 2$ by using a 1D average pooling with a kernel size 2 and stride 1. However, such pooling may aggregate consecutive but distant points into the same latent node since it does not leverage the interval information.

Interval-based squeeze. To address the limitations of the above pooling-based method, we propose two methods that consider the time intervals or spatial distance for compression. Concretely, we divide the trajectory at the two points with the largest time interval (or spatial distance) at each iteration, until the squeeze rate is satisfied. Figure 6 illustrates an example of how we divide the trajectory based on the time interval and spatial distance, respectively. It is worth noting that the assignment matrix can be processed and saved in the preprocessing phase, thereby inducing *no extra time* during training or inference.

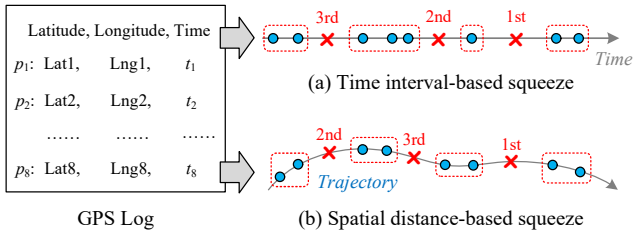


Figure 6: Interval-based squeeze functions. The red "x" cross means the divided location. 1st, 2nd and 3rd denote the position we divide by at the first, second and third iteration.

Projection-based squeeze. We notice that the above squeeze functions are fixed during the training phase. To explore more flexible compression, we further present a projection-based squeeze function with weights that can be jointly learned with the backbone network. Inspired by Linformer [29], we utilize 1D convolutions as a projection to adaptively generate the assignment matrix $\mathbf{B} \in \mathbf{R}^{m \times n}$ from the input features $\mathbf{X} \in \mathbb{R}^{n \times d}$ as follows:

$$\mathbf{X}_m = \text{Softmax}(\text{Conv1D}(\mathbf{X})), \quad \mathbf{B} = \mathbf{X}_m^\top \tag{7}$$

where $\mathbf{X}_m \in \mathbb{R}^{n \times m}$ is the intermediate result and Softmax guarantees the sum of each column of $\mathbf{X}_m$ equals to one. This solution has two major benefits. First, it is easy to implement and friendly to parallelization due to the use of convolutions. Second, the assignment matrix is conditioned on the input features in the trajectory space and can thus consider the latent semantics of points for projection.

3.2.4 Implementation. The assignment matrix varies with different trajectories. To support parallel mini-batch training, we propose an effective implementation of squeeze functions to deal with different trajectories in the same batch. For instance, if a batch contains two samples (T_1 and T_2) with 6 and 10 points, we first compute the assignment matrix of them with a squeeze rate $r = 2$, leading to two matrices ($\mathbf{B}_1$ and $\mathbf{B}_2$) with shape 6×3 and 10×5. We then pad T_1 to 10 points to allow parallel training. Likewise, we pad $\mathbf{B}_1$ to be the same shape as $\mathbf{B}_2$. In this way, the first 6 points in T_1 are real points while the last 4 points are padded points, and the last 2 latent nodes of T_1 are useless. Finally, we *mask* two types of relations in attention computations to remove the impact of padded points/nodes: 1) the real points to the padded latent nodes; 2) the padded points to all latent nodes.

3.3 Prediction & Subpath labeling

3.3.1 Loss Function. The last step is leveraging the class token for making classifications. Further, we introduce an auxiliary objective termed *subpath labeling* which assigns each subpath (derived from the interval-based squeeze) with an individual label generated by a machine annotator. In this way, we can provide *additional supervision* (i.e., not only the mode of the whole trajectory) to the models to ease the training, and the student model can generally outperform the teacher model after training [13]. For example, a car sometimes stops at the traffic control for a while, which makes this subpath analogy to walking or riding a bike. If we force our model to recognize these points as driving, it may cripple the training. Using a teacher model to assign pseudo labels for these subpaths can provide more correct supervision to them. Consequently, we train our model by minimizing the following loss:

$$\mathcal{L} = \mathcal{H}(\hat{\mathbf{y}}_{cls}, \mathbf{y}_{cls}) + \mathcal{L}_{SL}, \tag{8}$$

where the first term is the cross entropy (denoted as $\mathcal{H}$) between the prediction $\hat{\mathbf{y}}_{cls}$ and ground truth $\mathbf{y}_{cls}$; the second term $\mathcal{L}_{SL}$ represents the subpath labeling loss, which is computed as:

$$\mathcal{L}_{SL} = \frac{1}{s} \sum_{i=1}^{s} \mathcal{H}\left(\hat{\mathbf{y}}_i, \mathbf{y}_i\right), \tag{9}$$

where $\mathbf{y}_i$ is the pseudo label of the i-th subpath and s is the number of subpaths. Compared to literature [13] that assigns a pseudo label to each output token within images, our subpath-level assignment is preferable since correctly labeling a single point is far more difficult than a subpath with far more contextual information.

3.3.2 How to Annotate? The machine annotator can be a variety of models, such as RNNs and transformers. Taking our TrajFormer as an example, we first train a TrajFormer without subpath labeling, denoted as the **teacher** model. Based on this teacher model, we utilize the re-labeling method [33] to produce pseudo labels for each subpath of all instances. To be more specific, recall that the teacher model employs the MLP network head_1 (see Figure 3) to perform classification on the class token $\hat{\mathbf{y}}_{cls}$. Following the idea of re-labeling, since each output token has the same dimensionality as the class token, [13] simply utilizes head_1 to give a prediction (i.e., a pseudo label) to all output tokens, and these pseudo labels will provide additional supervision to the student model to ease the training. However, the information of one token is usually limited, making it difficult to label these tokens one by one. Accordingly, we first employ the interval-based squeeze to aggregate the output tokens into some subpaths[2] which have richer contextual information, and then produce pseudo labels to these subpaths via head_1. After labeling all subpaths, we start to train a new TrajFormer as the student model using the knowledge provided by the teacher.

[2]Each subpath's representation is the average of each point that belongs to this subpath.

4 EXPERIMENTS

4.1 Datasets

We evaluate our approach over two public trajectory datasets of different sizes. We briefly introduce them as follows:

- **GeoLife** [37]: This GPS trajectory dataset was collected in the Geolife project by 182 users from 2007 to 2012. There are 17,621 trajectories with a total distance of 1,292,951 km in this dataset. 40% of the users have labeled their trajectories with transportation modes, such as driving, taking a bus, riding a bike and walking. We follow [19] to choose the typical four modes (including walking, bus, bike and driving) for the classification task. However, we notice that the number of labeled trajectories is very limited (only around 2,000) compared to the unlabeled trajectories. To overcome this problem, we split each labeled trajectory into sub-trajectories using Algorithm 2 (we set $M = 10$). This partition can be regarded as data augmentation. As a result, we obtain 15,639 instances ranging from 5 to 10 minutes from the GeoLife dataset. Each of them possesses 20 to 100 GPS points. Finally, we split the data into non-overlapping training, validation and test data by a ratio of 8:1:1.
- **Grab-Posisi** [11]: Grab-Posisi is a large-scale dataset collected by Grab[3] drivers in two metropolises – Singapore and Jakarta. The collection dates range from 2019-04-08 to 2019-04-21 with 6,000 trajectories gathered per day. In particular, the trajectories in Jakarta have been labeled with two modes, i.e., driving a car or riding a motorcycle. We follow [18] to conduct a binary classification task over one-week data (2019-04-10 to 2019-04-16). Algorithm 2 with $M = 30$ is utilized to partition the trajectories into many instances. After data preprocessing, we obtain 319,587 instances, each of which ranges from 5 minutes to 30 minutes and consists of 20 to 178 points. Similar to GeoLife, we use the ratio of 8:1:1 to partition the instances.

More details about the datasets can be found in Table 1.

Algorithm 2: Trajectory partition

Input: $\{T_i\}_{i=1}^{N_T}$: trajectories with labels.
N_m: maximum points within a trajectory;
M: maximum time range of an instance (in minutes).
Output: Instances set $\mathcal{S}$.

$\mathcal{S} = \{\emptyset\}$ // *Initialize a set for storing instances.*
for $i = 1 \dots N_T$ **do**
  if get_time_range(T_i) $> M$ **then**
    Partition T_i into M-minute segments, denoted by $\mathcal{P}$
    for $j = 1 \dots |\mathcal{P}|$ **do**
      N_p = get_num_points($\mathcal{P}_j$)
      if $N_p \geq 20$ *and* $N_p \leq N_m$ **then**
        Add $\mathcal{P}_j$ to $\mathcal{S}$
      else if $N_p > N_m$ **then**
        Sample N_m points from $\mathcal{P}_j$, denoted by $\mathcal{P}'_j$
        Add $\mathcal{P}'_j$ to $\mathcal{S}$
    end
  else if get_time_range(T_i) > 5 **then**
    Add T_i to $\mathcal{S}$
end

[3]Grab is a Singaporean multinational ride-hailing company

4.2 Experimental Settings

4.2.1 Implementation Details. TrajFormer is implemented with PyTorch 1.7 and all experiments are conducted on Nvidia Quadro RTX 6000. We train our method via the Adam optimizer. The learning rate is initially set to 0.005, and reduced by 1/10 every 30 epochs. The batch size is 128 and 512 over the two datasets, respectively. The kernel size k in CPE is 9, which works well in our experiments. As the lengths of trajectories in a batch are inconsistent, we pad them to the maximum length in the same batch. To achieve a better trade-off between accuracy and efficiency, we tune a series of hyperparameters including the number of transformer layers L, the hidden dimensionality d, the number of heads N_h, and the squeeze rate r. The hyperparameter study will be detailed in Section 4.4.

4.2.2 Baselines. We compare TrajFormer with two classes of baselines: 1) traditional methods, i.e., SVM and Random Forest (RF); 2) RNN-based models: RNN, TimeLSTM [38], GRU-D [3], STGN [34], STGRU [19], and TrajODE [18]. The baseline settings are identical to a recent work [18]. We delineate these baselines below:

- **SVM** and **RF**: Following previous studies [35, 36], we utilize SVM or RF to perform classification based on the extracted metadata, such as the mean and variance of velocity and acceleration.
- **RNN**: A vanilla RNN model.
- **TimeLSTM**: It equips LSTM [10] with time gates to model time intervals, so as to better capture short- and long-term patterns.
- **GRU-D**: A variant of GRU [5], which proposes a new time gate to capture irregularity.
- **STGN**: It enhances LSTM by introducing spatio-temporal gates to capture spatio-temporal relationships between successive points.
- **STGRU**: STGRU develops a convolution layer to capture short-term spatial dependencies and a temporal gate to control the information flow related to the temporal interval information.
- **TrajODE**: The state-of-the-art method, which couples the continuity of Neural ODE and the robustness of latent variables.

4.2.3 Evaluation Metrics. We use *accuracy* (Acc) and *throughput per second* (TPS) to evaluate the effectiveness and efficiency of our model, respectively. In particular, TPS indicates the average number of trajectories the network can process in one second during inference, which is more popular than processing time per trajectory in evaluating real-world systems. We train each model five times on both datasets and report their mean performance.

Table 1: Dataset statistics of the two datasets. MBR indicates minimum bounding rectangle.

Dataset	GeoLife	Grab-Posisi
# instances	12,684	319,587
# points	20~100; Avg: 94	20~178; Avg: 84
Time range	5~10 minutes	5~30 minutes
Date	2007-04-01	2019-04-10
End date	2012-08-31	2019-04-16
Class ratio	Walk: 31.77% Bike: 16.39% Bus: 31.82% Car: 20.02%	Car: 50.01% Motorcycle: 49.99% - -
MBR	39.68°N~40.13°N 116.09°E~116.60°E	5.93°S~6.84°S 105.88°E~107.41°E

4.3 Model Comparison

In this section, we perform model comparison in terms of accuracy and throughput. For a fair comparison, we follow the baseline settings of [18] and ensure that each method has similar parameter sizes. The experimental results are presented in Table 2. The default settings of TrajFormer are $L = 2$, $d = 64$, $N_h = 4$, $r = 2$ and the squeeze function is time interval-based squeeze. We also report a larger version of our model with $L = 3$, denoted as TrajFormer*.

As the baselines are not teacher-forcing, to be fair, we mainly compare them with TrajFormer instead of its subpath labeling version (TrajFormer-SL). First of all, TrajFormer obtains the highest accuracy on GeoLife, while achieving **26**× higher TPS than the state-of-the-art method TrajODE. By stacking more squeezed transformer blocks, TrajFormer* sets new state-of-the-art accuracy (81.08%) on Grab-Posisi with up to 248 TPS. These facts demonstrate the practicality of our model in time-sensitive applications. Compared to TrajODE which computes the instantaneous rate of change at every time step, our TrajFormer takes advantage of the parallelization and jointly learns the spatio-temporal dependencies within *all* time steps. Second, TrajFormer clearly outperforms other RNN-based models on both metrics, e.g., it brings ∼10% gains in accuracy on both datasets and runs 2× faster than STGRU.

From this table, we can also observe: 1) By using subpath labeling, TrajFormer can achieve around 0.5% higher accuracy on both datasets without any extra costs. 2) The significant improvement of STGN against RNN reveals the importance of jointly capturing the spatial and temporal intervals. 3) Traditional methods such as SVM and RF are not competitive with the deep-learning-based approaches due to their limited model capacity. 4) FLOPs cannot reflect the "real" running speed of each method because it ignores the effects of parallelization. Hence, we use a more reasonable metric (TPS) to measure the model efficiency in this study. 5) Deep-learning-based methods outperform the traditional models by a considerable margin thanks to their preferable model capacity and admirable ability in representation learning.

Table 2: Model comparison. Param denotes the number of learnable parameters and FLOPs means floating point operations per trajectory with 100 points; The magnitudes are Kilo (10^3), Mega (10^3) and % for #Param, FLOPs, and Acc. The bold and underlined font mean the best and the second best metrics, respectively.

Method	Param	FLOPs	GeoLife		Grab-Posisi	
			TPS↑	Acc↑	TPS↑	Acc↑
SVM	-	-	-	49.88	-	51.17
RF	-	-	-	56.16	-	55.68
RNN	56	0.32	172	64.30	161	64.16
TimeLSTM	94	0.78	135	66.68	124	65.68
GRU-D	69	0.54	151	71.71	134	68.48
STGN	78	0.62	148	75.60	142	73.17
STGRU	86	0.70	143	73.15	139	70.14
TrajODE	85	2.37	12	<u>85.25</u>	9	<u>80.44</u>
TrajFormer	76	2.18	**314**	**85.45**	**280**	79.76
TrajFormer-SL	76	2.18	314	85.92	280	80.29
TrajFormer*	105	2.50	<u>271</u>	84.90	<u>248</u>	**81.08**

4.4 Ablation Studies

We have verified the superiority of TrajFormer against various baselines on the two real-world mobility datasets. After that, we conduct a series of ablation studies to evaluate each model component in the following parts.

4.4.1 Effects of Continuous Position Embedding. To examine the effectiveness of the proposed continuous position embedding (CPE), we compare it with several variants that are integrated with various position encoding methods:

- **Base**: This variant denotes TrajFormer without CPE, i.e., no position encoding in this base model.
- **Base+APE**: We substitute CPE with learnable absolute position encoding (APE) which is widely used in natural language processing [7] and computer vision tasks [9].
- **Base+CAPE**: We concatenate the information of spatio-temporal intervals with the input features to consider the absolute position as well as the continuity of trajectories.
- **Base+RPE**: In this variant, we replace CPE with the relative position encoding (RPE) [25] to provide position information. It can captures the relative relationships between tokens.
- **Base+Conv**: CPE is replaced by a conditional position encoding which is implemented by 1D convolutions [6]. Such convolutions can provide absolute position information to some extent [12].

Table 3 illustrates the comparison results, where we omit the throughput as this metric is not distinctive in this comparison. Primarily, TrajFormer with CPE can jointly capture the node features as well as the spatio-temporal intervals, yielding 1∼2% gains on accuracy against the base model. We also witness a clear accuracy drop by replacing CPE with APE which only encodes the order of GPS points. Moreover, a simple concatenation of the spatio-temporal interval information only brings marginal improvement (see base vs. base+CAPE), and we have similar observations in base vs. base+RPE. These findings verify the importance of capturing variable spatio-temporal intervals as well. In contrast to Base+Conv, our TrajFormer surpasses it by a large margin on accuracy because the convolution operation assigns static weights to nearby tokens while neglecting the important characteristic of trajectories – irregularity. Lastly, we notice that the APE-based variant cannot even outperform the base model, which indicates that an unsuitable position encoding would cripple the model performance.

Table 3: Results of model variants with different types of position encodings (or point embeddings), where Param means the number of learnable parameters. Δ denotes the relative improvement over the base model. The setting of TrajFormer is identical to that in Section 4.3.

Variant	Param	GeoLife		Grab-Posisi	
		Acc↑	Δ	Acc↑	Δ
Base	59	83.42	0.00	78.59	0.00
Base+APE	65	82.38	-1.04	77.02	-1.57
Base+CAPE	59	83.51	+0.09	78.71	+0.12
Base+RPE	61	83.72	+0.30	78.65	+0.06
Base+Conv	61	83.05	-0.37	79.20	+0.61
TrajFormer	76	**85.45**	**+2.03**	**79.76**	**+1.17**

4.4.2 Effects of Squeezed Transformer Block. As the basic building block, our squeezed transformer block allows us to efficiently learn the spatio-temporal dependencies within a trajectory. Here, we evaluate its effects from various perspectives.

Given a squeeze rate $r = 2$, we first test how the number of transformer blocks L affects the model performance. Figure 7 depicts the experimental results, in which we have the following observations. Firstly, removing the squeezed transformer encoder (i.e., when $L = 0$, we only use CPEs for classification) significantly degrades the performance, which indicates the effectiveness of transformers for learning trajectory representations. Secondly, the trend of the four squeeze functions is similar over both datasets, where $L = 2$ or $L = 3$ mostly obtain the best results on these datasets. Thirdly, the time-interval-based squeeze consistently achieves the highest accuracy among these functions, motivating us to utilize it as the default option. Fourthly, the projection-based squeeze is worse than the others in most cases, revealing the difficulty in learning the assignment matrix from scratch. Lastly, using more transformer layers does not always bring remarkable gains and instead slows down our model. We thereby choose $L = 2$ as our default setting.

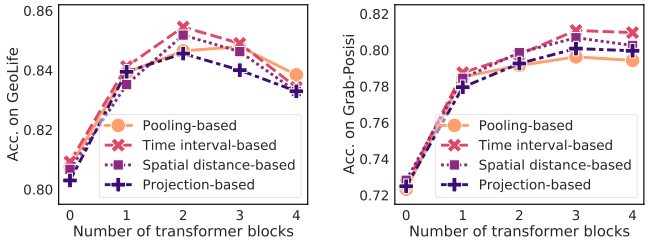


Figure 7: Effects of the number of transformer blocks L, where the colored lines indicate squeezed transformers with different squeeze functions. Left: results on GeoLife. Right: results on Grab-Posisi.

After discussing how the number of blocks L affects the model performance, we further study the effects of the hidden dimensionality d ranging from 16, 32, 64, 128 to 256. As shown in Figure 8, a very small hidden dimensionality (e.g., $d = 16$) significantly restricts model capacity, whereas a very large value (such as $d = 256$) may cause the overfitting problem. We also show that setting $d = 64$ obtains the best or the second best accuracy across all kinds of squeeze functions.

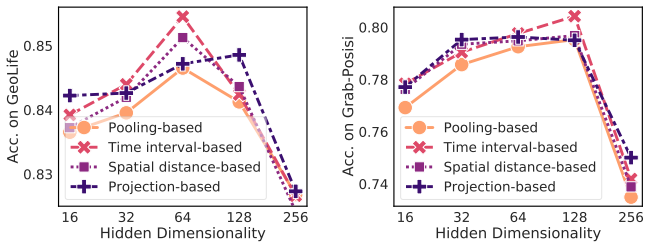


Figure 8: Effects of the number of hidden units d, where the colored lines mean different squeeze functions. Left: results on GeoLife. Right: results on Grab-Posisi.

Next, we investigate the trade-off between efficiency and accuracy by tuning the squeeze rate $r \in \{1, 2, 4, 8\}$. It can be seen from Table 4 that the accuracy gradually drops with the growth of r, since compressing the key-value space inevitably leads to a bit of information loss. However, such slightly degenerated performance seems acceptable in many real-world systems. For example, we prefer using a larger squeeze rate such as $r = 8$ in a time-sensitive system (38.5% faster than $r = 1$) even though it degrades the performance by 0.6%. From Table 4, we can also observe that TrajFormer with $r = 2$ runs 17.6% faster than $r = 1$ but only decreases the accuracy by 0.1%∼0.2% on both datasets. This fact reveals the *information redundancy* in the original trajectory space, motivating us to reduce the key-value space for improving efficiency.

In this paper, we mainly report the results of $r = 2$ (e.g., in Table 2) since it achieves a preferable balance between speed and accuracy. In particular, when $r = 2$, our model obtains very close accuracy to the standard transformer ($r = 1$), while being 18% faster on GeoLife where the maximum length of trajectories is 100. For Grab-Posisi with maximum 178 points in a trajectory, it can run 31% faster. Such efficiency improvement would further become more valuable in modeling very long trajectories, e.g., with thousands of points.

Table 4: Effects of r in a time-interval-based squeeze.

r	FLOPs	GeoLife		Grab-Posisi	
		TPS↑	Acc↑	TPS ↑	Acc↑
1	2.39	267	**85.59**	214	**80.02**
2	2.18	314	85.45	280	79.76
4	2.08	340	85.20	296	79.35
8	2.02	**356**	84.97	**304**	79.07

4.4.3 Effects of Subpath Labeling. Finally, we compare our subpath labeling (**SL**) with two other teacher-forcing schemes below:

- **Distillation (DL)**: This method proposes a new *distillation token*, which serves the same purpose as the class token, except that it aims at reproducing the label estimated by the teacher [26].
- **Token Labeling (TL)**: TL is widely applied in computer vision tasks, which assigns each token with individual location-specific supervision generated by a machine annotator [13].

Besides, we denote TrajFormer with no teacher forcing as **Base**. The teacher model is a pretrained TrajFormer for all schemes. As shown in Table 5, we first test TrajFormer with different schemes in a fully supervised learning setting (see the Fully column). TL, which leverages the rich information of all output tokens, slightly outperforms DL which only utilizes the extra class token as additional supervision. However, TL is still inferior to SL because correctly labeling a single point is more complicated than labeling a subpath with more contextual information.

We also investigate *semi-supervised learning* via these teacher-forcing schemes. We randomly remove 90% of the samples from the training set and set them as unlabeled data. During the training phase, we produce the pseudo labels for the unlabeled data via the above teacher schemes, which provides additional supervision to the model. For the validation and test set, we keep them unchanged. From Table 5, the improvements of the three methods over Base (ignoring unlabeled data) verify the advantage of leveraging the unlabeled data. More importantly, SL significantly surpasses DL and TL in this setting, e.g., with ∼2% higher accuracy on Grab-Posisi.

Table 5: Accuracy of various teacher-forcing schemes in fully-supervised (Fully) and semi-supervised (Semi) learning.

Scheme	GeoLife		Grab-Posisi	
	Fully	Semi	Fully	Semi
Base	85.45 (+0.00)	80.29 (+0.00)	79.76 (+0.00)	73.95 (+0.00)
Base+DL	85.35 (-0.10)	82.47 (+2.18)	79.81 (+0.05)	75.18 (+1.23)
Base+TL	85.50 (+0.05)	82.76 (+2.47)	79.91 (+0.15)	75.33 (+1.38)
Base+SL	**85.92 (+0.47)**	**83.95 (+3.66)**	**80.29 (+0.53)**	**77.48 (+3.53)**

5 CONCLUSION AND FUTURE WORK

We have presented a transformer-based architecture for efficient trajectory classification. Our approach has two major advantages against standard transformers. First, we propose continuous point embeddings to capture the irregularity of trajectories. Second, it speeds up self-attention by compressing the key-value space, resulting in a more efficient way to capture global context. Compared to the state-of-the-art method, our model achieves comparable performance while running at least 26× faster. In the future, we plan to explore self-supervised trajectory classification with our model.

ACKNOWLEDGMENTS

We thank the reviewers for their constructive comments. This study is supported by Singapore Ministry of Education Academic Research Fund Tier 2 under MOE's official grant number T2EP20221-0023, National Natural Science Foundation of China (62172034), Beijing Nova Program (Z201100006820053), and key research project of Zhejiang Lab (No. 2022PI0AC01).

REFERENCES

[1] Zain Ul Abideen, Heli Sun, Zhou Yang, Rana Zeeshan Ahmad, Adnan Iftekhar, and Amir Ali. 2021. Deep Wide Spatial-Temporal Based Transformer Networks Modeling for the Next Destination According to the Taxi Driver Behavior Prediction. *Applied Sciences* 11, 1 (2021), 17.

[2] Xin Cao, Gao Cong, and Christian S Jensen. 2010. Mining significant semantic locations from GPS data. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1009–1020.

[3] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific reports* 8, 1 (2018), 1–12.

[4] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 900–911.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[6] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. 2021. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882* (2021).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Xin Ding, Lu Chen, Yunjun Gao, Christian S Jensen, and Hujun Bao. 2018. UlTraMan: A unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment* 11, 7 (2018), 787–799.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[11] Xiaocheng Huang, Yifang Yin, Simon Lim, Guanfeng Wang, Bo Hu, Jagannadan Varadarajan, Shaolin Zheng, Ajay Bulusu, and Roger Zimmermann. 2019. Grab-posisi: An extensive real-life gps trajectory dataset in southeast asia. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Prediction of Human Mobility*. 1–10.

[12] Md Amirul Islam, Sen Jia, and Neil DB Bruce. 2020. How much position information do convolutional neural networks encode? *arXiv preprint arXiv:2001.08248* (2020).

[13] Zihang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Xiaojie Jin, Anran Wang, and Jiashi Feng. 2021. Token labeling: Training a 85.5% top-1 accuracy vision transformer with 56m parameters on imagenet. *arXiv preprint arXiv:2104.10858* (2021).

[14] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2019. Reformer: The Efficient Transformer. In *ICLR*.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[16] Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. 2015. Rank-geofm: A ranking based geographical factorization method for point of interest recommendation. In *SIGIR*. 433–442.

[17] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 617–628.

[18] Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann. 2021. Modeling Trajectories with Neural Ordinary Differential Equations. In *IJCAI*. 1498–1504.

[19] Hongbin Liu, Hao Wu, Weiwei Sun, and Ickjai Lee. 2019. Spatio-temporal GRU for trajectory classification. In *ICDM*. IEEE, 1228–1233.

[20] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by Summarizing Long Sequences. In *ICLR*.

[21] Xuanqing Liu, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2020. Learning to encode position for transformer with continuous dynamical model. In *ICML*. 6327–6335.

[22] Cheng Long, Raymond Chi-Wing Wong, and HV Jagadish. 2014. Trajectory simplification: on minimizing the direction-based error. *Proceedings of the VLDB Endowment* 8, 1 (2014), 49–60.

[23] Sijie Ruan, Cheng Long, Jie Bao, Chunyang Li, Zisheng Yu, Ruiyuan Li, Yuxuan Liang, Tianfu He, and Yu Zheng. 2020. Learning to generate maps from trajectories. In *AAAI*, Vol. 34. 890–897.

[24] Simonas Šaltenis, Christian S Jensen, Scott T Leutenegger, and Mario A Lopez. 2000. Indexing the positions of continuously moving objects. In *SIGMOD*. 331–342.

[25] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *NAACL (Short Papers)*. 464–468.

[26] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *ICML*. 10347–10357.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[28] 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*, Vol. 32.

[29] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).

[30] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling Trajectories with Recurrent Neural Networks. In *IJCAI*. 3083–3090.

[31] Hao Xue, Flora Salim, Yongli Ren, and Nuria Oliver. 2021. MobTCast: Leveraging Auxiliary Trajectory Forecasting for Human Mobility Prediction. *Advances in Neural Information Processing Systems* 34 (2021).

[32] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. 2013. Time-aware point-of-interest recommendation. In *SIGIR*. 363–372.

[33] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han, Junsuk Choe, and Sanghyuk Chun. 2021. Re-labeling imagenet: from single to multi-labels, from global to localized labels. In *CVPR*. 2340–2350.

[34] Pengpeng Zhao, Haifeng Zhu, Yanchi Liu, Jiajie Xu, Zhixu Li, Fuzhen Zhuang, Victor S Sheng, and Xiaofang Zhou. 2019. Where to Go Next: A Spatio-Temporal Gated Network for Next POI Recommendation. In *AAAI*, Vol. 33. 5877–5884.

[35] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing*. 312–321.

[36] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. 2008. Learning transportation mode from raw gps data for geographic applications on the web. In *WWW*. 247–256.

[37] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

[38] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to do next: modeling user behaviors by time-LSTM. In *IJCAI*. 3602–3608.