From Text to Locations: Repurposing Language Models for Spatial Trajectory Similarity Assessment

Wilken C. Dantas de Melo[1], Lívia Almada Cruz[1], Francesco Lettich[2], Ticiana L. Coelho da Silva[1], Regis Pires Magalhães[1]

[1]Insight Data Science Lab – Universidade Federal do Ceará (UFC)

[2]ISTI – CNR, Pisa, Italy

Abstract. *The proliferation of electronic devices with geopositioning capabilities has significantly increased trajectory data generation, thus opening up novel opportunities in mobility analysis. Our work considers the problem of assessing spatial similarity between trajectories, and focus on deep learning-based approaches that discretize trajectories using a uniform grid to generate their embeddings. In this context, t2vec is the reference approach. Large Language Models (LLMs) show promise in capturing patterns in mobility data. In this paper, we investigate whether an LLM can be repurposed to generate high-quality trajectory embeddings for the considered task. Using two real-world trajectory datasets, we consider repurposing three language models: Word2Vec, Doc2Vec, and BERT. Our results show that BERT, trained on dense trajectory datasets, can generate high-quality embeddings, thus highlighting the potential of LLMs.*

1. Introduction

The surge in electronic devices with geopositioning capabilities has led to a notable increase in trajectory data. This data plays a crucial role in various tasks where machine learning is applied to mobility data, such as trajectory similarity, behavioral pattern recognition [Cao et al. 2020], trajectory clustering [Hung et al. 2015], and next location prediction [Cruz et al. 2022, Cruz et al. 2019]. In this work, we focus on the task of assessing spatial similarity between trajectories, which involves assessing the similarity of two trajectories based on their spatial features. Classic methods for this task include Dynamic Time Warping (DTW), Longest Common Sub-Sequence (LCSS), and Edit Distance on Real Sequences (EDR). These methods aim to align points between trajectories optimally using dynamic programming, but they suffer from high computational costs, scaling quadratically with the length of the longer trajectory.

Recent research [Wang et al. 2020, Li et al. 2018, Fang et al. 2022, Fu and Lee 2020, Zhang et al. 2019] has delved into representation learning as a solution to computational challenges. Representation learning involves extracting valuable and low-dimensional features from high-dimensional data, using deep learning models. These models transform trajectories into embeddings, i.e., small dense vectors, which are then used to assess similarity: similar trajectories should ideally have embeddings closely located in the latent space. Methods employing representation learning often use sophisticated models like RNNs, transformers, or attention mechanisms. They require initial trajectory discretization for model training, typically via grid or road-network segment partitioning. We adopt uniform grid-based discretization due to its independence from road networks, applicability to non-road entities like pedestrians, and avoidance of

time-consuming preprocessing steps like map matching. Existing approaches focus on various trajectory similarity types, necessitating specific loss functions during training. Our work focuses on the task of spatial similarity assessment, with t2vec [Li et al. 2018] as our reference approach due to its relevance, as discussed in the related work section.

Natural Language Processing (NLP) has made significant strides, particularly with Large Language Models (LLMs) like BERT and GPT, showcasing human-like reasoning abilities. Drawing parallels between language sequences and trajectories, we can leverage LLMs to generate trajectory embeddings, treating trajectory sequences as sentences or words. Recent research explores this concept, with examples like Habit2vec [Cao et al. 2020] modeling living habits from trajectories. Among the challenges one faces when deciding to use LLMs for trajectory data is how to discretize trajectories into tokens for LLM vocabulary and training large models like BERT from scratch. Grid-based discretization simplifies vocabulary and learning complexity. Despite computational demands, leveraging NLP models, especially LLMs, holds promise for learning trajectory spatial features and generating high-quality embeddings. Accordingly, our research questions are:

- **RQ1**: Can language models generate high-quality embeddings that accurately capture the spatial similarity between trajectories, whether they are dense or sparse?
- **RQ2**: Do language models scale well as we increase the search space in which trajectory embeddings are compared?

Our work aims to address the two research questions with the following contributions: (1) we repurposed selected language models, i.e., Word2Vec, Doc2Vec, and BERT, originally intended for textual data, for the computation of trajectory embeddings; (2) we use their embeddings to tackle the problem of assessing spatial similarity between trajectories. All our experiments are conducted on two publicly accessible real-world datasets, i.e., Porto[1] and T-drive [Jing et al. 2018]. The code behind this work has been made available on a GitHub repository[2].

The rest of the paper is structured as follows: in Section 2, we provide the fundamental concepts used throughout the paper and the problem definition. In Section 3, we present the related works and highlight the differences to our work. In Section 4, we present the methodology used to repurpose NLP models for spatial trajectory similarity assessment. In Section 5 we introduce the experimental setting. Section 6 presents the experimental results. Finally, Section 7 draws the final conclusions.

2. Preliminaries and problem definition

In this section, we introduce some fundamental notions and introduce the problem addressed in this work. We begin by the notion of trajectory.

Definition 1 (Trajectory)**.** *We define a trajectory generated by a moving object to be* $T = \langle p_1, p_2, \ldots, p_n \rangle$*, i.e., a sequence of time-stamped geographical locations (or samples). Each sample* $p_i = (x_i, y_i, t_i)$ *represents the* i*-th object's position in space (represented by the coordinates* x_i *and* y_i*) and time (represented by the timestamp* t_i*).*

[1]https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i

[2]https://github.com/InsightLab/LMTrajectorySimilarity

This work focuses on transforming trajectories into embeddings using *uniform grid-based discretization* of the area of interest. Importantly, one of the consequences of such discretization is to transform a trajectory into a sequence of tokens processable by NLP models. Moreover, the discretized trajectory might be shorter than the original one. Indeed, in this work, consecutive samples falling within the same cell are collapsed into a single cell identifier in T'. Following, we formally introduce the grid-based discretization, the notion of trajectory embedding, and finally, the problem being addressed.

Definition 2 (Uniform grid-based trajectory discretization)**.** *Given a trajectory $T = \langle p_1, \ldots, p_j \rangle$ and a uniform grid $\mathcal{C}$, its discretization involves converting each sample $p \in T$ into the identifier of the cell $c \in \mathcal{C}$ containing p's location. The result is a discretized trajectory $T' = \langle c_1, \ldots, c_m \rangle$, where $m \leq j$.*

Definition 3 (Trajectory Embedding Generation)**.** *The process of trajectory embedding generation involves transforming a discretized trajectory $T' = \langle c_1, \ldots, c_m \rangle$ into a vector representation (embedding) of the trajectory $v^T \in \mathbb{R}^d$, where $\mathbb{R}^d$ denotes the d-dimensional vector (embedding) space. This transformation is achieved through a suitable function $f : \mathcal{C}^* \rightarrow \mathbb{R}^d$, where $\mathcal{C}^*$ denotes the set of all possible sequences of cells obtainable from the uniform grid $\mathcal{C}$. The embedding v^T should encapsulate the spatial characteristics of the original trajectory T.*

Definition 4 (Problem Definition). *Let $\mathcal{T}$ be a set of trajectories, $\mathcal{T}'$ be the corresponding set of discretized trajectories, and f a function generating embeddings for trajectories in $\mathcal{T}'$. Then, for any trajectory $T \in \mathcal{T}$, let T' be its corresponding discretized trajectory in $\mathcal{T}'$ and $v^T = f(T')$ be its embedding generated by f. Ideally, the ranking produced by measuring distances between v^T and all the other embeddings in $\mathcal{T}'$ should be equivalent or as close as possible to the ranking produced by measuring the spatial similarity between T and all other trajectories in $\mathcal{T}$ via some ideal spatial trajectory similarity function d.*

In other words, we aim to validate that embeddings generated by f preserve the spatial similarity relations inherent in the original trajectories, such that similarity assessments based on embeddings align with those derived directly from the spatial properties of the trajectories.

3. Related work

We explore the generation of trajectory embeddings for spatial similarity assessment using language models, both on dense and sparse trajectory datasets, and assuming that trajectories are discretized by using some grid. This section therefore focuses on research works limited to this scope. **t2vec**, introduced by [Li et al. 2018], is an autoencoder model with GRU layers, encoding discretized trajectories into embeddings. It reconstructs trajectories from downsampled and perturbed versions, with the encoder generating embeddings and the decoder reconstructing trajectories. Despite newer works reportedly surpassing t2vec in embedding quality, we consider it a benchmark due to its reproducibility and well-assessed quality in independent research [Taghizadeh et al. 2021].

Newer deep learning models, such as **NeuTraj** [Yao et al. 2019], instead of focusing on trajectory reconstruction, aim to approximate handcrafted trajectory similarity functions while maintaining linear time complexity for similarity computations. **T3S** [Yang et al. 2021] integrates structural and spatial information to generate trajectory embeddings. It employs a self-attention mechanism for structural insights and an LSTM-based module for spatial details. T3S utilizes a loss function to minimize the difference

Table 1. Overview of the main related works

Paper	Approaches	Sampling rate	Discretization unit	Task
[Li et al. 2018]	GRU-based autoencoder + custom loss	Sparse and dense	Grid cell	Spatial trajectory sim.
[Yao et al. 2019]	Attention + LSTM + custom loss	Sparse and dense	Grid cell	Spatial trajectory sim.
[Yang et al. 2021]	Attention + LSTM + custom loss	Sparse and dense	Grid cell	Spatial trajectory sim.
[Zhang et al. 2021]	LSTM + custom loss	Dense	GPS locations	Spatial trajectory sim.
[Cao et al. 2020]	Word2Vec	Sparse	POI	Semantic sim.
[Damiani et al. 2020]	Word2Vec and Paragraph Vector	Sparse	CDR	Semantic sim.
[Zhang et al. 2019]	Word2Vec, GloVe, RNN	Sparse	Grid cell	Semantic sim.
[Cruz et al. 2022]	BERT, LSTM	Sparse	Traffic sensor	Location prediction
[Crivellari et al. 2022]	BERT	Sparse	CDR	Missing data reconstruct
Our work	**BERT, Word2Vec, Doc2Vec**	**Sparse and dense**	**Grid cell**	**Spatial trajectory sim.**

between computed and known trajectory similarities. **Traj2SimVec** [Zhang et al. 2021] efficiently generates triplet training samples from large datasets and uses an LSTM-based Trajectory Encoder to produce trajectory embeddings. The encoder calculates distances between sub-trajectories to capture overall and sub-trajectory similarities. It employs supervised loss functions that leverage sub-trajectory distances and optimal point-matching relationships under various metrics.

Other research explores semantic trajectory similarity using trajectory embeddings. **Habit2vec** [Cao et al. 2020] creates embeddings based on user paths to detect behavioral patterns, integrating temporal features and spatial information from Points of Interest (POI). **mob2vec** [Damiani et al. 2020] assesses behavioral similarity using NLP techniques (Word2Vec and Paragraph Vector) on sparse trajectories from Call Detail Records. **At2vec** [Zhang et al. 2019] generates embeddings from sparse check-in trajectories, considering spatial, temporal, and semantic information, employing Word2Vec and GloVe on a gridded space. Further studies employ LLMs for trajectory embeddings. **EST Prediction** [Cruz et al. 2022] uses BERT to encode road network sensor data, predicting the next traffic sensor in a trajectory. **TraceBERT** [Crivellari et al. 2022] uses trajectory embeddings to address missing data reconstruction.

Table 1 summarizes the related works, highlighting the methods employed to create embeddings (**Approaches**), the average sampling rate of the trajectory datasets (**Sampling rate**), the trajectory discretization unit (**Discretization unit**), and the task being addressed (**Task**). Overall, none of the surveyed works consider language models (including large ones) to generate embeddings, both for dense and sparse trajectory data, for the task of assessing spatial trajectory similarity between trajectories.

4. Repurposing language models for spatial trajectory similarity assessment

In this section, we outline our methodology for repurposing language models for the task of assessing spatial trajectory similarity. First, we discuss the data preparation pipeline (Section 4.1), and then describe how we repurposed the training phase of selected language models to use the augmented training data (Section 4.2).

4.1. Data preparation pipeline for language models

The data preparation pipeline operates on a preprocessed trajectory dataset $\mathcal{D}$ already divided into training set $\mathcal{D}_{train}$ and test set $\mathcal{D}_{test}$. This pipeline generates augmented

versions of these sets that are suitable for training and evaluating language models for the task of spatial trajectory similarity assessment. Note that this pipeline is not in charge of trajectory discretization. We leverage part of the data preparation pipeline from t2vec's paper [Li et al. 2018], which offers several benefits: (1) it facilitates the adaptation of language models in our study, (2) it addresses sampling variability and noise common in GPS data by simulating varied sampling rates and controlled Gaussian noise, and (3) it enhances training data diversity to prevent overfitting and improve generalization.

4.1.1. Generating augmented training data

The approach involves augmenting the training set $\mathcal{D}_{train}$ to create $\mathcal{D}^{aug}_{train}$. For each trajectory $T \in \mathcal{D}_{train}$, the pipeline generates downsampled trajectories (while preserving start and end samples) and adds Gaussian noise to some of these downsampled trajectories' locations. This process results in $\mathcal{D}^{aug}_{train}$. To ensure compatibility with RNN-based architectures like t2vec, cells in $\mathcal{D}^{aug}_{train}$ that are not considered *hot cells* in the uniform grid $\mathcal{C}$, i.e., cells that occur less than a given threshold, are removed. This ensures that only trajectories made up entirely of hot cells remain in $\mathcal{D}^{aug}_{train}$.

4.1.2. Generating augmented test data

Evaluating spatial trajectory similarity approaches is challenging without a ground truth. Inspired by self-similarity and cross-similarity comparisons, t2vec introduced a method called *most similar search* to assess embedding quality, which we adopt in our experimental evaluation. This requires to select trajectories from the test set $\mathcal{D}_{test}$ that are guaranteed to be entirely made of hot cells. We randomly choose a fixed number of trajectories, forming the set Q from $\mathcal{D}_{test}$. Additionally, we create a larger set B from $\mathcal{D}_{test}$, with $Q \cap B = \emptyset$. The size of B varies during evaluation to test scalability and resilience to noise and trajectory diversity. For each T_Q in Q, we generate two sub-trajectories, $T_{Q'}$ and $T_{Q''}$, by alternating points between odd and even indices. This divides Q into two new subsets: the set of odd sub-trajectories Q', and the set of even sub-trajectories Q''. The same transformation is applied to B, yielding B' and B''. Ultimately, Q' is used as the *query trajectory set*, while $S = Q'' \cup B''$ is used as the *search space*: ideally, for every $T_{Q'} \in Q'$ an approach should recognize $T_{Q''} \in S$ as the most spatially similar trajectory. We denote the augmented test data consisting of Q' and S by $\mathcal{D}^{aug}_{test}$. Further details on how Q' and S are used to quantitatively assess the quality of embeddings generated by the language models are provided in the experimental setting (see Section 5.5).

4.2. Training language models on augmented training data

Repurposing language models for trajectory data involves transforming trajectories into sequences that these models can process. To address this, we implement the grid-based discretization technique employed in the t2vec approach and other works. First, we partition the geographical area where the trajectories from $\mathcal{D}^{aug}_{train}$ move, considering the uniform grid $\mathcal{C}$ previously used to identify the hot cells (see also Section 4.1.2). Subsequently, every trajectory in $\mathcal{D}^{aug}_{train}$ and $\mathcal{D}^{aug}_{test}$ are converted from a sequence of samples into a sequence of identifiers corresponding to the cells within this grid. This process ultimately transforms the augmented trajectory data into discretized augmented trajectory data, yielding the augmented and discretized training data $\mathcal{D}'^{aug}_{train}$ and the augmented and discretized $\mathcal{D}'^{aug}_{test}$ test data. Next, we explain how we trained selected language models.

Word2Vec. Word2Vec [Mikolov et al. 2013] is a model capable of projecting words to an $\mathbb{R}^n$ space. We train the Word2Vec model from scratch using the augmented and discretized training data $\mathcal{D}'^{aug}_{train}$ – as such, the training data acts as the corpus for Word2Vec, while the grid cells traversed by the trajectories represent the vocabulary. As the model trains, it employs the Continuous Bag of Words (CBOW) architecture to learn embeddings for each cell $c \in \mathcal{C}$ appearing in at least a discretized trajectory. This process involves predicting a target cell identifier based on the context of surrounding cell identifiers within a discretized trajectory. With the trained model, the embedding of a trajectory T' can be computed as the mean of the embeddings of the cells that T' traverses, i.e., $v^T = \frac{1}{|W_{T'}|} \sum_{w \in W_{T'}} w$.

Doc2Vec. We also consider a model capable of learning from sentence-like structures, i.e., Doc2Vec using PV-DBOW [Le and Mikolov 2014]. Here, we detail how we repurposed this model's training. When using PV-DBOW, each discretized trajectory $T' \in \mathcal{D}'^{aug}_{train}$ is assigned a unique identifier or tag. The training objective under PV-DBOW is to predict the cells of a trajectory from this unique identifier without the context tokens (i.e., cells). The key output of PV-DBOW training is a set of trajectory embeddings, one per trajectory in $\mathcal{D}'^{aug}_{train}$, that attempt to encapsulate the spatial patterns characterizing each trajectory in the corpus. With a trained Doc2Vec model at hand, to generate embeddings for unseen discretized trajectories, these must be first processed akin to those of the training corpus, followed by an inference phase where a trajectory's initial random vector is refined across epochs to predict cells accurately, utilizing the model's learned cell embeddings.

BERT. In our work, we repurposed the training of BERT [Devlin et al. 2019], an LLM, as follows. We used the Masked Language Model (MLM) task to train BERT on cell embeddings, wherein 15% of the cell identifiers in each discretized trajectory $T' \in \mathcal{D}'^{aug}_{train}$ were randomly masked. The model was then trained to predict these masked identifiers based on the surrounding context within the trajectory. Upon completing the training, trajectory embeddings are computed by feeding a discretized trajectory as input to the trained BERT model. The model's last layer outputs a sequence of cell embeddings corresponding to each cell identifier in the trajectory: similarly to Word2Vec, we generate the final trajectory embedding by computing the average of the above-mentioned embeddings.

5. Experimental Setting

This section describes the experimental setup for the evaluation in Section 6, covering the trajectory datasets and their preprocessing (Section 5.1), the evaluated approaches (Section 5.2) and their training (Sections 5.3 and 5.4), and the evaluation process for the task of spatial trajectory similarity assessment (Section 5.5).

5.1. Datasets and their preprocessing

We consider two publicly accessible real-world trajectory datasets: Porto and T-drive. The Porto dataset, whose data have been collected in Porto, Portugal, from 2013 to 2014, spans 19 months and includes 1.7 million trajectories from 442 taxis equipped with GPS devices. The taxis reported their locations every 15 seconds on average, leading to an average distance of 130.4 meters between consecutive samples. Consequently, this dataset prevalently contains dense trajectories. In contrast, the T-drive dataset, whose data was gathered in Beijing, China, in February 2008, comprises 10,357 trajectories with a longer

average interval of 187 seconds between samples. This results in a larger average distance of 4.2 kilometers between samples, yielding considerably sparser trajectories compared to Porto's. Considering the diverse collection methodologies and characteristics of the Porto and T-drive datasets, we customized the preprocessing steps for each. These steps were essential to refine each dataset, ensuring the trajectories were representative and suitable for accurate model training and evaluation. After preprocessing, both datasets were divided into training (70%) and test sets (30%).

Porto preprocessing: the initial step involved removing trajectories with fewer than 30 samples to eliminate shorter, less informative trajectories. This action reduced the dataset size to 1.2 million trajectories. We then remove noisy data, often caused by GPS signal gaps in areas with challenging geography (e.g., urban canyons).

T-drive preprocessing: in the T-drive dataset, each vehicle produced a unique TAX_ID trajectory spanning an entire week, resulting in lengthy routes averaging 1,708 samples. Our preprocessing segmented these long trajectories into daily paths, eliminating noise and duplicates. To distinguish between consecutive rides by the same taxi, we split trajectories where the vehicle was stationary for over 5 minutes. We also discarded trajectories with fewer than 5 samples and limited the maximum length to 50 samples per trajectory $(5 < |T| \leq 50)$. These steps expanded the dataset from 10,357 to 532,511 trajectories, enriching its representation of real driving patterns.

5.2. Competitors

We compared a variety of methods spanning three distinct categories: classic approaches, a reference deep learning-based architecture tailored for spatial trajectory similarity assessment, and selected language models.

Classic approaches: we included three classic methods for assessing spatial trajectory similarity, all based on dynamic programming. These methods, serving as our baselines, include Dynamic Time Warping (DTW) [Kruskal 1983], Edit Distance on Real Sequences (EDR) [Levenshtein 1966], and Longest Common Subsequence (LCSS) [Shuncheng et al. 2019].

Reference deep learning-based architecture for spatial trajectory similarity assessment: in the experimental evaluation, we consider t2vec [Li et al. 2018].

Language models: the models considered in the experimental evaluation are BERT, Doc2Vec, and Word2Vec.

5.3. Preparation of the augmented training data

The classic methods do not need a training phase, hence what is explained below does not apply to them. For the selected *language models*, recall that the initial training set $\mathcal{D}_{train}$ is first augmented, yielding $\mathcal{D}^{aug}_{train}$ (see also Section 4.1.1). The trajectories in $\mathcal{D}^{aug}_{train}$ are then discretized according to a uniform grid $\mathcal{C}$ (see also 5.4), and only those trajectories made entirely of sequences of hot cells are kept: this yields the augmented and discretized training data $\mathcal{D}'^{aug}_{train}$ used to train the language models. Regarding *t2vec*, as detailed in [Li et al. 2018] this GRU-based autoencoder aims to reconstruct an original trajectory $T \in \mathcal{D}_{train}$ from its downsampled and distorted counterpart $T^{aug}_{train} \in \mathcal{D}^{aug}_{train}$. Similarly to the language models, t2vec operates on discretized trajectories (i.e., sequences of tokens). As such, $\mathcal{D}'^{aug}_{train}$ must undergo an additional transformation: each trajectory $T'^{aug}_{train} \in \mathcal{D}'^{aug}_{train}$ is paired with the discretized version of the trajectory $T \in \mathcal{D}_{train}$ from

Table 2. Model training – hyperparameter settings

model	hyperparameters
t2vec	hidden_size = 256; embedding_size = 256; window_size = 10; learning_rate = 0.001; k-nearest cells = 20
Word2Vec	embedding_size = $\{16, 32, \mathbf{64}, 128, 256, 512\}$; window_size = $\{4, \mathbf{5}, 6, 8, 10\}$; sg = CBOW; min_count = 3; epochs = $\{1, 2, \mathbf{3}, \ldots, 1000\}$
Doc2Vec	embedding_size = $\{16, 32, \mathbf{64}, 128, 256, 512\}$; min_count = 3; epochs = $\{1, \ldots, \mathbf{30}, \ldots, 1000\}$; n_grams = $\{4, \mathbf{5}, 6, 8, 10\}$; dm = {**0 (PV-DBOW)**}
BERT	hidden_size = {32, 64, 128, 256, 512, 1024, 2048}; epochs = $\{1, \ldots, 10\}$; num_hidden_layers = $\{\mathbf{6}, 12\}$; num_attention_heads = $\{8, \mathbf{16}\}$; max_position_embeddings = $\{\mathbf{512}, 1024\}$

which has been derived. This pairing forms a new version of the augmented and discretized training data, where each example consists of a pair (T'^{aug}_{train}, T'). t2vec can be then trained on this data. Finally, for both t2vec and language models, $\mathcal{D}^{aug}_{train}$ is generated using several downsampling rates, i.e., $\{0, 0.2, 0.4, 0.6\}$. Subsequently, various fractions, i.e., $\{0, 0.2, 0.4, 0.6\}$, of surviving samples undergo perturbation with Gaussian noise. In the end, for each $T \in \mathcal{D}_{train}$, 16 different downsampled and perturbed versions are created in $\mathcal{D}^{aug}_{train}$. Finally, we report that we used uniform grids whose cell size was 100 meters, and a grid cell is deemed a hot one if at least 50 samples of the trajectories in $\mathcal{D}^{aug}_{train}$ fall in it.

5.4. Deep learning-based models training

In the following, we provide details on how the models have been trained. All the models were trained on the augmented and discretized dataset $\mathcal{D}'^{aug}_{train}$ according to the procedure outlined in Section 4.2. Table 2 shows all the hyperparameters tried out, with the values in bold representing the best hyperparameter configuration.

t2vec. We trained this model according to the authors' guidelines in [Li et al. 2018] on a slightly modified version of the augmented and discretized train data shown in Section 5.3. The model internally uses two three-layered GRUs, one in the encoder and the other in the decoder part. The layers of the two GRUs have a number of internal hidden units (i.e., *hidden_size*) equal to 256. *embedding_size* indicates the dimensionality of the trajectory embeddings, and it is set to 256. Each cell's representation learning uses a context window *window_size* of 10 cells. The *learning_rate* parameter controls weight updates during optimization.

Word2Vec. We experimented with various *embedding_size* and *window_size* values to learn word vectors. We used the CBOW architecture (see the *sg* hyperparameter) for its efficiency in predicting the target word from context words. The model excludes words appearing less frequently than *min_count*, thus improving training efficiency. Finally, we trained the model using a number of *epochs* ranging from 1 to 1,000.

Doc2Vec. Doc2Vec is trained with a range of *embedding_sizes* values. Moreover, we trained the model with various *n_grams* values: this hyperparameter determines the size of the word sequences the model will consider as it learns the embeddings. We selected the PV-DBOW architecture for its effectiveness in learning document context. The *min_count* hyperparameter sets a threshold for the minimum frequency of words in the corpus to be included in the vocabulary. Finally, we trained the model using a number of epochs ranging from 1 to 1,000.

BERT. We explore a variety of *hidden_size* values and transformer configurations to learn token embeddings. More precisely, *num_hidden_layers* determines how many of these layers are stacked on top of each other, *num_attention_heads* determines how many

distinct attention mechanisms operate in parallel within each transformer layer, and *max_position_embeddings* determines the maximum length of input sequences that the model can process. Finally, the training is conducted using a number of *epochs* ranging from 1 to 10.

5.5. Evaluating the effectiveness of approaches for the task of spatial trajectory similarity assessment

To evaluate the effectiveness of the various competitors, we proceed as follows: for each query trajectory $T_{Q'} \in Q'$, we calculate its most spatially similar trajectories within $S = Q'' \cup B''$: ideally, $T_{Q''} \in S$ should be identified as the most similar one, and as such ranked first in the query results. The well-known mean rank measure is employed to quantify this, i.e., $mr(Q', S) = \frac{1}{|Q'|} \sum_{T_{Q'} \in Q'} rank(T_{Q'}, S)$, where *rank* determines the position of the correct response for a query $T_{Q'} \in Q'$ in the ranked search space S. Implementing the *rank* function depends on the considered approach. The classic methods represent distance functions applied directly to trajectory pairs from Q' and S: consequently, *rank* will sort these computed distances. For approaches that generate trajectory embeddings, trajectories from the augmented test set must first undergo discretization and then conversion into embeddings. The *rank* function then sorts based on the Euclidean distances between embeddings of trajectories in Q' and the embeddings of those in S.

5.6. Code implementation and experimental setup

The code behind our work has been written in Python and has been made available and documented on a GitHub repository[3]. The machine used to conduct the experimental evaluation is a Dell XPS 8940 server, equipped with an Intel CPU Core i7-10700, 128GB of RAM, and a NVIDIA GeForce RTX 3060 GPU (version with 12GB GDDR6 memory).

6. Experimental Evaluation

6.1. Addressing RQ1: most similar trajectory search in a dense dataset

Using the Porto dataset, we set the trajectory query set Q' size to 1,000 trajectories. We gradually expand the search space S from 20,000 to 100,000 trajectories. The effectiveness of the embeddings produced by different methods in spatial trajectory similarity assessment is evaluated using the mean rank metric. The evaluation follows the methodology outlined in Section 5.5. Table 3 presents the results – note that BERT_<SIZE> indicates a trained BERT model generating embeddings of a specific size.

From the results, we see that the performance of the competitors decreases as the size of the search space S increases. However, we also observe that all the language models outperform the other competitors, including t2vec. This possibly indicates that language models, especially the large ones such as BERT, have the potential to consistently produce high-quality trajectory embeddings for the task of spatial trajectory similarity assessment when trained on dense trajectories. Finally, observe that even when gradually reducing the embedding size to 64, BERT still consistently outperforms t2vec.

[3]https://github.com/InsightLab/LMTrajectorySimilarity

Table 3. *Mean rank* versus *search space size*, Porto and T-drive datasets

Dataset	Porto					T-drive				
|S|	20K	40K	60K	80K	100K	10K	20K	30K	40K	50K
DTW	18.28	26.29	35.27	45.53	54.27	454.61	892.04	1327.98	1774.94	2214.63
EDR	23.52	43.04	71.01	91.09	120.04	587.55	1193.97	2089.50	2693.07	3472.31
LCSS	31.19	62.75	92.83	123.37	155.14	660.58	1355.62	2004.37	2677.75	3362.30
t2vec	2.44	3.68	5.08	6.84	8.30	**21.29**	**37.28**	**52.50**	**67.92**	**85.49**
Word2Vec	2.38	3.61	4.91	6.66	7.97	156.79	249.10	420.70	541.68	550.48
Doc2Vec	1.44	1.80	**2.16**	2.62	2.99	111.59	207.70	291.71	370.06	482.87
BERT_64	1.85	2.61	3.36	4.24	5.04	56.85	93.06	146.38	189.27	236.26
BERT_128	1.63	2.23	2.84	3.77	4.19	177.95	348.83	509.21	687.17	867.35
BERT_1024	**1.40**	**1.78**	2.17	**2.62**	**2.96**	94.95	162.23	280.19	310.17	402.55

6.2. Addressing RQ1: most similar trajectory search in a sparse dataset

In these experiments, we aim to assess whether language models can achieve similar results with sparse trajectory datasets. To this end, we consider the T-drive dataset and apply the same evaluation methodology. Due to the smaller number of trajectories contained in T-drive, we set the query set Q' size to 500 trajectories and vary the number of trajectories contained in the search space S from 10,000 to 50,000 (Table 3). From the results, we observe that language models consistently achieve worse results than t2vec, suggesting that these are sensitive to data with low sampling rates. We conjecture that sparse trajectories might be analogous to the notion of sparse text, which refers to documents or text datasets with limited content or sparse information, thus leading to less contextual information. This, in turn, might explain the lower quality of the embeddings generated by these models compared to those from t2vec.

6.3. Addressing RQ2: scalability analysis

Scalability plays a critical role in spatial trajectory similarity assessment. Classic methods, with their quadratic complexity relative to trajectory size, become impractical for large datasets. In contrast, recent deep learning-based approaches such as t2vec exhibit linear complexity with respect to trajectory and embedding size. This set of experiments aims to gauge the scalability of language models against t2vec. In this batch of experiments conducted on the Porto dataset (the largest dataset used), we consider a query trajectory set Q' of size 1,000 and incrementally enlarge the search space S from 20,000 to 100,000 trajectories. For each combination of Q' and S, we compute the embeddings of their trajectories and index them using a kd-tree. These steps can be done offline and are not included in time measurements. We then calculate the k-nearest-neighbors (k-NN, with $k = 50$) for each query trajectory's embedding in Q' against those in S. The results in Figure 1 show the average computation time per query and confirm that the average query time increases with the size of S, as expected. More interestingly, BERT shows that its average query time varies with the embedding size (again, expected) but remains consistently competitive with t2vec, despite BERT's more complex architecture. Taking into consideration also the results shown in Section 6.1, this strongly hints that BERT, at least with dense trajectory datasets, has the potential to produce smaller, higher-quality embeddings than competitors specifically designed for the task of spatial trajectory similarity assessment while displaying strong scalability.

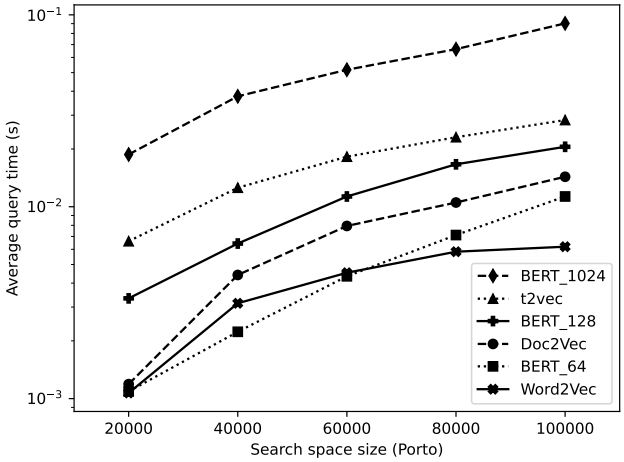

Figure 1. Scalability analysis: computing k-nn queries versus search space size (y-axis in log-scale).

7. Conclusions

In this study, we investigated the use of selected and repurposed language models to generate embeddings for the task of assessing spatial trajectory similarity. We adapted these models to learn trajectory embeddings and found that while a large model like BERT produced high-quality embeddings for dense datasets and scaled linearly with the query search space, it fell short with sparse datasets. Overall, the results reveal new research opportunities. One potential improvement is adapting our data preparation pipeline, which currently removes trajectories with gaps due to non-hot cells. Allowing such gaps might leverage LLMs' resilience to sequence discontinuities, potentially boosting performance on sparse datasets. Further exploration could involve refining BERT-based architectures or adopting different loss functions to better represent spatial features. Additionally, exploring more advanced open source LLMs like Llama, Mistral, Gemma, Phi, and others, as suggested by [Gruver et al. 2023] for encoding time series, may offer insights given the parallels with trajectories.

Acknowledgments

Funding for this research has been provided by Spoke 1 "Human-centered AI" of the M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research". However, the views and opinions expressed are those of the authors only and do not necessarily reflect those of the EU or European Commission-EU. Neither the EU nor the granting authority can be held responsible for them.

References

Cao, H., Xu, F., Sankaranarayanan, J., Li, Y., and Samet, H. (2020). Habit2vec: Trajectory semantic embedding for living pattern recognition in population. *IEEE Transactions on Mobile Computing*, 19(5):1096–1108.

Crivellari, A., Resch, B., and Shi, Y. (2022). TraceBERT – a feasibility study on reconstructing spatial-temporal gaps from incomplete motion trajectories via BERT training process on discrete location sequences. *Sensors*, 22(4):1682.

Cruz, L., Coelho da Silva, T., Magalhães, R., Melo, W., Cordeiro, M., de Macedo, J., and Zeitouni, K. (2022). Modeling trajectories obtained from external sensors for location prediction via NLP approaches. *Sensors*, 22(19).

Cruz, L., Zeitouni, K., and Macedo, J. (2019). Trajectory prediction from a mass of sparse and missing external sensor data. In *IEEE MDM*.

Damiani, M. L., Acquaviva, A., Hachem, F., and Rossini, M. (2020). Learning behavioral representations of human mobility. In *ACM SIGSPATIAL*, page 367–376, New York, NY, USA. Association for Computing Machinery.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the ACL: Human Language Technologies, Vol.1*, pages 4171–4186, Minneapolis, Minnesota. ACL.

Fang, Z., Du, Y., Zhu, X., Hu, D., Chen, L., Gao, Y., and Jensen, C. (2022). Spatio-temporal trajectory similarity learning in road networks. In *28th ACM SIGKDD*, KDD '22, page 347–356, New York, NY, USA. Association for Computing Machinery.

Fu, T.-Y. and Lee, W.-C. (2020). Trembr: Exploring road networks for trajectory representation learning. *ACM TIST*, 11(1):1–25.

Gruver, N., Finzi, M. A., Qiu, S., and Wilson, A. G. (2023). Large language models are zero-shot time series forecasters. In *NeurIPS*.

Hung, C.-C., Peng, W.-C., and Lee, W.-C. (2015). Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *The VLDB Journal*, 24(2):169–192.

Jing, Y., Yu, Z., Chengyang, Z., Wenlei, X., Xing, X., Guangzhong, S., and Yan, H. (2018). Tdrive: driving directions based on taxi trajectories. In *18th ACM SIGSPATIAL*, GIS '10, pages 99–108, New York, NY, USA. Association for Computing Machinery.

Kruskal, J. (1983). An overview of sequence comparison: time warps, string edits, and macromolecules. *SIAM*, 2(25):201–237.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st ICML*, ICML'14, page II–1188–II–1196. JMLR.org.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710.

Li, X., Zhao, K., Cong, G., Jensen, C. S., and Wei, W. (2018). Deep representation learning for trajectory similarity computation. In *34th IEEE ICDE*, pages 617–628.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *1st ICLR Workshop Track Proceedings*.

Shuncheng, L., Su, H., Zheng, B., Zhou, X., and Zheng., K. (2019). A survey of trajectory distance measures and performance evaluation. *VLDB*, 408:3–32.

Taghizadeh, S., Elekes, A., Schaler, M., and Bohn, K. (2021). How meaningful are similarities in deep trajectory representations? In *Information Systems*, volume 98, page 101452. Elsevier.

Wang, S., Cao, J., and Philip, S. Y. (2020). Deep learning for spatio-temporal data mining: A survey. *IEEE TKDE*, 34(8):3681–3700.

Yang, P., Wang, H., Zhang, Y., Qin, L., Zhang, W., and Lin, X. (2021). T3S: Effective representation learning for trajectory similarity computation. In *37th IEEE ICDE*, pages 2183–2188.

Yao, D., Cong, G., Zhang, C., and Bi, J. (2019). Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *35th IEEE ICDE 2019*, pages 1358–1369.

Zhang, H., Zhang, X., Jiang, Q., Zheng, B., Sun, Z., Sun, W., and Wang, C. (2021). Trajectory similarity learning with auxiliary supervision and optimal matching. In *29th IJCAI*, IJCAI'20.

Zhang, Y., Liu, A., Liu, G., Li, Z., and Li, Q. (2019). Deep representation learning of activity trajectory similarity computation. In *2019 IEEE ICWS*, pages 312–319. IEEE.