UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Oliver Bollverk

CDR-Based Trajectory Reconstruction Using Transformers

Master's Thesis (15 ECTS)

Supervisor: Amnir Hadachi, PhD

Tartu 2022

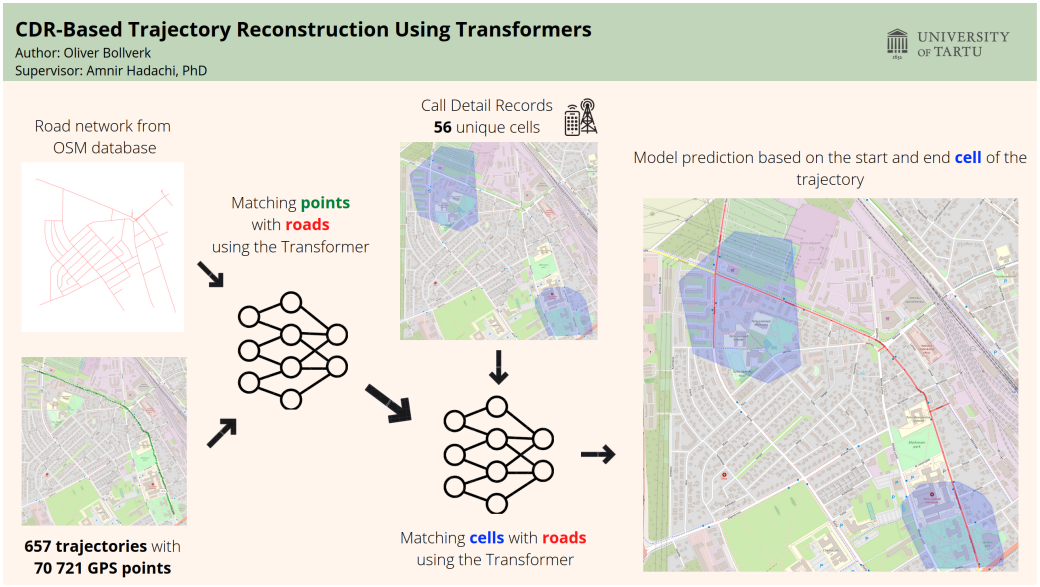
CDR-Based Trajectory Reconstruction Using Transformers

Abstract:
With the development of telecommunication technologies, mobile devices, and data collected via mobile services, it has become of great interest to predict the paths that individuals take in cities. With sparse mobility data, the goal of researchers is to build models that are able to fill the gaps, or in other words, to reconstruct the trajectory of an individual. Recent models proposed for this task utilize Call Detail Records (CDRs) produced when a mobile phone connects to the cellular network, using Monte Carlo or Hidden Markov Model (HMM) based approaches. In this thesis, a novel deep learning method for trajectory reconstruction from CDR data is introduced. GPS points are linked to roads on a road network constructed from the OpenStreetMap (OSM) database, and the resulting labels are used in training as ground truth. Drawing inspiration from prior work in matching GPS points to a network of roads using Transformer neural networks, we present a framework that involves using two Transformers sequentially with partially modified architectures. The final result is a trained Transformer, able to predict the road level path, knowing only the cell, in the area in which movement started. The accuracy of estimating the taken path was compared with that of prior approaches which use probabilistic modeling to predict the next location from CDR data.

Keywords: deep learning, trajectory reconstruction, mobile data, Transformer

CERCS: P170 - Computer science, numerical analysis, systems, control; P176 - Artificial Intelligence

Graphical abstract:



CDR-põhise trajektoori rekonstrueerimine Transformereid kasutades

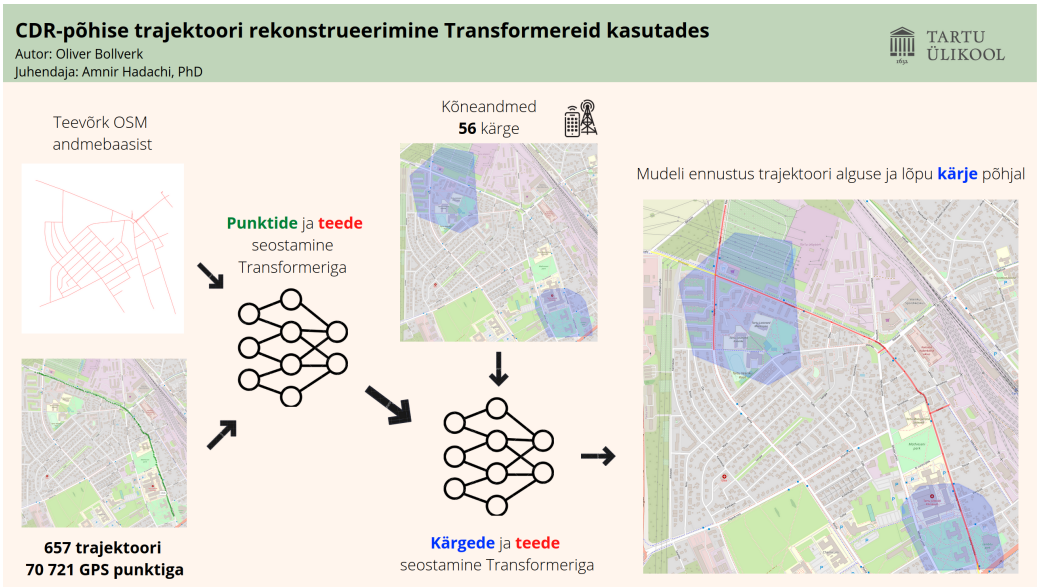
Lühikokkuvõte:

Telekommunikatsiooni tehnoloogiate, mobiiliseadmete ja mobiiliteenuste areng on toonud kaasa huvi inimeste liikumistrajektooride ennustamise vastu linnades. Ajas ja ruumis hajusate liikumisandmete korral on sageli teadlaste eesmärgiks luua mudeleid, mis võimaldaksid täita andmetes olevaid auke, või teisisõnu rekonstrueerida üksikisikute trajektoore. Hiljutised algoritmid selles vallas põhinevad Monte Carlo meetoditel või peidetud Markovi ahelatel ning ennustavad liikumist kõnelogide ehk CDR andmete põhjal. Käesolevas töös tutvustatakse ainulaadset sügavõppel rajanevat raamistikku. GPS andmed seotakse OpenStreetMap (OSM) andmebaasi põhjal loodud teevõrguga ning saadud märgised võetakse mudeli treenimisel aluseks. Rajanedes eelnevale tööle GPS trajektooride sobitamisel kaardile, kasutades Transformer tüüpi tehisnärvivõrke, esitletakse lahendust, milles kaks erinevat Transformerit on seadistatud jadamisi. Lõpptulemuseks on tehisnärvivõrk, mis võimaldab ennustada tee täpsusega teekonda linnapiirkonnas, omades infot vaid selle kohta, millise masti katvusalast liikuma hakati. Meetodit võrreldi teekonna ennustamisel varasemate lahendustega, mis rakendavad tõenäosuslikke mudeleid järgmise asukoha ennustamisel CDR andmete põhjal.

Võtmesõnad: sügavõpe, trajektoori rekonstrueerimine, mobiilsideandmed, Transformer

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria); P176 - Tehisintellekt

Graafiline kokkuvõte:



Contents

1 Introduction

The development of telecommunication technologies has rapidly increased research in urban mobility, which centers on predicting and making sense of movement patterns of individuals equipped with mobile devices. Two prevalent sources of information are Call Detail Records (CDRs), which result from network logs obtained by Mobile Network Operators, and data from the Global Positioning System (GPS). Having access to both types of data, we are interested in determining which roads did the individuals take while moving in the area covered by the mobile network in Tartu, by foot or by vehicle. As the GPS dataset contains exact longitude and latitude coordinates in a dense time series, we start by using that to establish the road level trajectory. The first step in this process involves mapping points of a GPS trajectory to a network of roads, known as map matching. In that regard, we explore using Transformers, an architecture that originates from the field of natural language processing [42], and has been demonstrated to improve performance in image recognition [10], object detection [46] as well as map matching [20].

Only after training a model to solve the map matching problem do we proceed with reconstructing trajectories based on CDR logs, which are sparse in time, and only give an indication of the coverage area of a cell tower where the individual's device most recently connected. To reconstruct a trajectory means to estimate the road level path taken, having information on the coverage areas for only some moments, such as for the starting second and the ending second.

Our goal is to use a neural network that can learn the relationship between sparse coverage area information and the precise movement of individuals on the road network of Tartu. To do so, we introduce a framework for solving two tasks: map matching and trajectory reconstruction. We use the Transformer architecture for both tasks with a modification in the former case. As we have obtained GPS data for the movement patterns of two users, we use this to establish the ground truth for training the map matching model and evaluating the final Transformer. The latter model can estimate the next road knowing only previous information from the mobile network, or rather more precisely, the cell tower from the coverage area of which movement started.

A number of data processing steps precede the training of the model. First, GPS trajectories are extracted, then the closest road to each point is found, and finally trajectory start and end points are associated with CDR records using only the provided coverage area information. While the coverage area information and GPS trajectories are real-life datasets, the events in the mobile network are synthetically created based on the details about cell towers in the CDR records for the two individuals.

1.1 Problem statement

In this section, the problem statement and definitions required to understand the concepts in this thesis are presented.

Definition 1. (GPS trajectory): A GPS trajectory T is a sequence of GPS points $(p_1, p_2, ..., p_n)$, where each point is chronologically ordered. $p_i = (longitude_i, latitude_i, t_i)$ where $longitude_i$ and $latitude_i$ are GPS coordinates and t_i is the timestamp associated with point p_i.

Definition 2. (Road network): A road network or a map is represented by a direct graph $G = (V, E)$. Vertex $v = (x, y) \in V$ distinguishes a road endpoint or an intersection and edge $e = (ID, start, end, l) \in E$ is a directed road starting from vertex $start$ to vertex end along polyline l with a unique id ID. We will refer to this id as *road ID*.

Definition 3. (Point-level route): A point-level route R_p is a sequence of road segments $(e_1, e_2, ..., e_n)$ where $e_i \in E, 1 \leq i \leq n$. A point-level route and a GPS trajectory are equal in length $len(T) = len(R_p)$.

Definition 4. (Segment-level route): A segment-level route R_s is a sequence of connected road segments $(e_1, e_2, ..., e_k)$ where $e_i \in E, 1 \leq i \leq k, k \leq n$, $e_i.end = e_{i+1}.start$, $e_i \neq e_{i-1} \forall i \in (2, 3, ..., k)$ and n is the length of the point-level route $n = len(R_p)$.

Definition 5. (Map matching): Map matching $M_G : T \longrightarrow R_{P/S}$ is a process, whereby the point or segment-level route $R_{P/S}$ is found based on the GPS trajectory T in a given road network G.

Definition 6. (Cell enriched GPS trajectory): A cell enriched GPS trajectory T_{enr} is a sequence of cell enriched GPS points $(p_1, p_2, ..., p_n)$, where each point is chronologically ordered. $p_i = (longitude_i, latitude_i, CellID_i)$ where $longitude_i$ and $latitude_i$ are GPS coordinates and $CellID_i$ is the *cell ID* associated with p_i. $CellID_i = unknown\_cell\_token$ $\forall i \in (2, 3, ..., n-1)$. $CellID_i$ and $CellID_n$ are the geomatched (Algorithm 1) start and end cell ID respectively.

Definition 7. (Cell-level trajectory): A cell-level trajectory T_c is a sequence of cell IDs $(CellID_1, CellID_2, ..., CellID_n)$ where
$CellID_i \in (CellID_1, unknown\_cell\_token, CellID_n) \forall i \in (1, 2, ..., n)$ and each cell ID is chronologically ordered. A cell-level trajectory has the same length as a GPS trajectory and a point-level route $len(T_c) = len(T) = len(R_p)$.

Definition 8. (Trajectory reconstruction): Trajectory reconstruction $T_R : T_c \longrightarrow M_G(T)_{P/S}$ is a process, whereby the result of map matching, the predicted segment or point-level route $M_G(T)_{P/S}$, is found based on the cell-level trajectory T_c in a given road network G. In other words, it is the process of converting cell IDs into road segments, that have been obtained as a result of map matching.

The presented problem in our thesis is to solve trajectory reconstruction, which is defined as including the process of map matching. Our objective is first to obtain a point-level route from a cell-enriched GPS trajectory, without using cell ID in the process. Then we obtain a cell-level trajectory and the point- or segment-level route from a cell-level trajectory, which includes as unique elements only the start and end cell ID and the *unkown_cell_token*.

1.2 Contribution

This thesis contributes to the existing literature by:

- Exploring the performance and applicability of Transformer-based map matching [20] in a more limited real-life data scenario and in a larger and more varied road network.
- Approaching trajectory reconstruction from CDR data as a sequence problem using Transformers.
- Demonstrating the performance and applicability of using two Transformers, with slightly different architectures, in a chained manner, to train first, then predict and then train again from the previous prediction.
- Introducing a novel padding technique for trajectory data, which is labeled with road indices.

1.3 Roadmap

- Background: an overview of deep learning, trajectory reconstruction studies, the Transformer architecture, and deep learning use cases in mobility studies is provided.
- Methodology: the structure of the model along with its use configuration is introduced, methods used in data preprocessing are explained and the dataset and its preprocessing are explained in detail
- Results and Analysis: the experimental setup is introduced along with performance metrics, results are presented and a comparison is made with previous works.
- Discussion: challenges in conducting experiments are presented and an overview is provided of possible aspects of future research along with a reflection of the results in the context of previous research.

2 Literature review and background knowledge

The literature review section introduces the concepts of neural networks, Transformers, CDR data. and trajectory reconstruction. An overview is provided on prior work in deep learning applications on spatiotemporal data along with the use of Transformers in map matching.

2.1 Deep learning

The definitions and concepts introduced in the following paragraph are based on the book "Deep Learning" by Goodfellow, *et al.* [16]. One way to define deep learning is through the concept of representation learning. Representation learning is to use machine learning to discover the mapping from a representation of the data to output as well as the representation itself. For example, if an algorithm aims to detect that a picture of a globe and a picture of a map are different representations of the same concept, it may begin by detecting more exact representations, such as the color of pixels, their positioning in the photo, and do so in a step-wise manner. Deep learning involves incorporating representations that are expressed through other, simpler representations. The central deep learning model is a feedforward neural network (FFN) or multilayer perceptron (MLP). It aims to approximate some function f^*, by defining a mapping between an input x and a category y as $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and then learning the value of parameters $\boldsymbol{\theta}$, which result in the closest approximation of f^*. The name "feedforward" characterizes the flow of information starting from the input $\boldsymbol{x}$, through computations that define f, finally reaching the output $\boldsymbol{y}$. FFNs do not include feedback connections. In other words, the model's output is not sent back to itself. In contrast, recurrent neural networks (RNNs) are cyclic.

FFN is a network because it composes of several different functions connected in a chain. For example, a two-layer network with a depth of n = 2 would consist of $f(\boldsymbol{x}) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$. In the process of training the network, $f(\boldsymbol{x})$ is set to match $f^*(\boldsymbol{x})$. A learning algorithm has the function of deciding how to use the layers in the network to approximate f^*. These layers are referred to as hidden layers, the dimensionality of which is the width of the model.

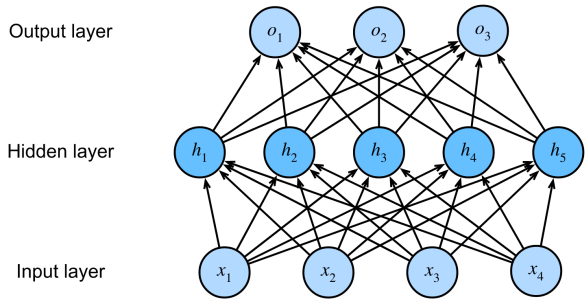


Figure 1. An FFN with a hidden layer of 5 hidden units [44].

An example of a fully connected FFN is provided in Figure 1. In this example, the network contains 4 inputs, 3 outputs, and a hidden layer with 5 units. As the input layer does not involve calculations, this network requires computation for the hidden as well as the output layers. Because both hidden and output layers are fully connected, each of them has a separate set of weights and biases. For each hidden layer unit $h1$, $h2$, etc., a weighted sum of the input is calculated, and a bias is added. This calculation is known as a linear transformation:

$$f(x) = xW^T + b \tag{1}$$

Where W is the weight matrix and b is a bias value. The result of this computation is then passed on to an *activation function*, which determines if the unit is to be activated or not. The resulting values are then sent to the next layer. The most common activation function the Rectified Linear Unit (ReLU) [35], defined as

$$f(x) = max(x, 0) \tag{2}$$

We also present the softmax activation function, which is a part of the Transformer architecture introduced in Section 2.2. The softmax function is often used prior to the final output layer or output probability prediction of a neural network. It is a differentiable function, which transforms the input between 0 and 1, so that a sum of the elements is equal to 1, making it possible to interpret the resulting values as probabilities [44]. The softmax is defined as

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{3}$$

Weights and biases are parameters of the neural network that the model learns in training and need to be initialized. Because training algorithms are usually iterative, it is required to specify an initial start point of the iteration. Initialization methods affect the training process significantly, determining if the algorithm converges, encounters

numerical difficulties, or converges to a point of low or high-cost [16]. Different strategies for initialization exist, with the main ones being sampling from a Gaussian or uniform distribution [16].

The training algorithm for an FNN is almost always based on gradient descent of a cost function [16]. A cost function measures how well the model we are training corresponds to reality and "often decomposes as a sum over training examples of some per example-loss function" [16, p. 152]. For multiclass classification problems, the most prevalent loss function is cross-entropy loss, which measures how accurate is the probability value for a class label outputted by the model to that which we obtained from the ground truth data. To calculate it, we define the ground truth probabilities of the input $x_1, x_2, ..., x_n$ as $p_1, p_2, ..., p_n$. Cross-entropy loss is defined as:

$$J = -\sum_{i=1} p_i \log x_i \tag{4}$$

In addition to FNNs, we briefly explain the concepts of an RNN, attention mechanism, residual connection, and embedding, as these are required to understand the Transformer architecture. A widespread concept in deep learning applications on sequence data is that of an encoder and a decoder. An encoder is a function that changes input data into a different representation [16]. A decoder does the opposite and returns the data in its original format from the new representation. Several architectures in sequence modeling involve stacking layers of components into a single decoder or an encoder block. One such category is RNNs, which are used in applications that involve sequential data processing. One of the first implementations of RNN architectures that could solve the task of mapping a variable-length sequence to another variable-length sequence was described as a *encoder-decoder* architecture by Cho, *et al.* [6]. They proposed two RNNs, one which processes the input sequence and emits an output that is sent to the decoder. In translation applications, the output of an encoder may be a fixed-size variable that represents a semantic summary of the input sequence [16]. Advancements in RNN applications on machine translation included the introduction of an attention mechanism, capable of associating elements of an input sequence to elements of an output sequence [3]. The use of the word "attention" relates to the ability of the decoder module to "pay attention to" certain sections of the input sequence which are more relevant in outputting the next element of a sequence. In cases of large sentences, capturing semantic details requires a large RNN, which must be trained for a long time [6]. The attention mechanism fixes the problem on the encoder side, which no longer has to encode all the information in the input sequence to a fixed-length vector [3]. With it, the decoder can select information from a sequence of annotations. To improve back-propagation algorithms, *residual connections* have been used, which enable the inputs to skip certain layers in the network, thereby making forward propagation faster. The comparison between a residual block and a standard FFN block is shown in Figure 2. The solid line on the right figure illustrates a residual connection.

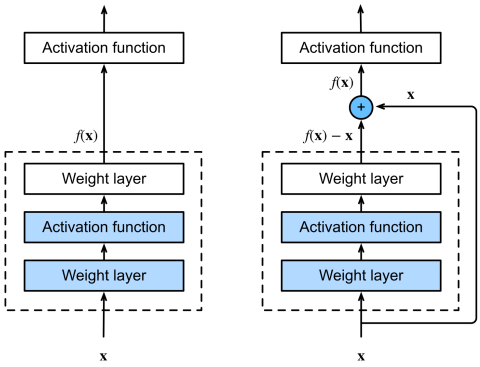


Figure 2. A regular block (left) and a residual block (right) [44].

Another concept that requires introduction is embedding or embeddings. In machine translation, word embeddings are representations of words as points in a dimension equal to the vocabulary size [16]. Therefore the process of mapping words to real vectors is also called *word embedding* [44]. An embedding is different from a one-hot vector representation of words, as it contains information on the similarity of words. In the context of this paper, an embedding is a feature vector representation of a single edge in a road network that contains numerical information about its closeness to other edges. For clarification, this closeness is not geographical proximity or distance but reflects the proximity between edges as they appear in the sequences in the data. These feature vectors are typically obtained in an *embedding layer* for each token or label in the input sequence. For example, in a machine translation RNN encoder, the weight of an embedding layer is a matrix, with a number of rows equal to the size of the input vocabulary and a number of columns equal to the feature vector dimension. In the context of this thesis, vocabulary size is better thought of as the number of edges in a road network.

2.2 Transformer

The Transformer architecture was introduced in 2017 by Vaswani, *et al.* [42] for natural language processing, and has since been applied to a wide range of problems, such as image recognition [40], object tracking [39], and radio map generation [31]. The Transformer has an encoder-decoder structure. The encoder maps an input sequence of symbol representations to continuous representations which are then passed on to the decoder. The decoder generates the output sequence one element at a time and does so auto-regressively, that is, taking in the previously generated symbols as additional input when generating the next. The architecture of the Transformer is shown in Figure 3.

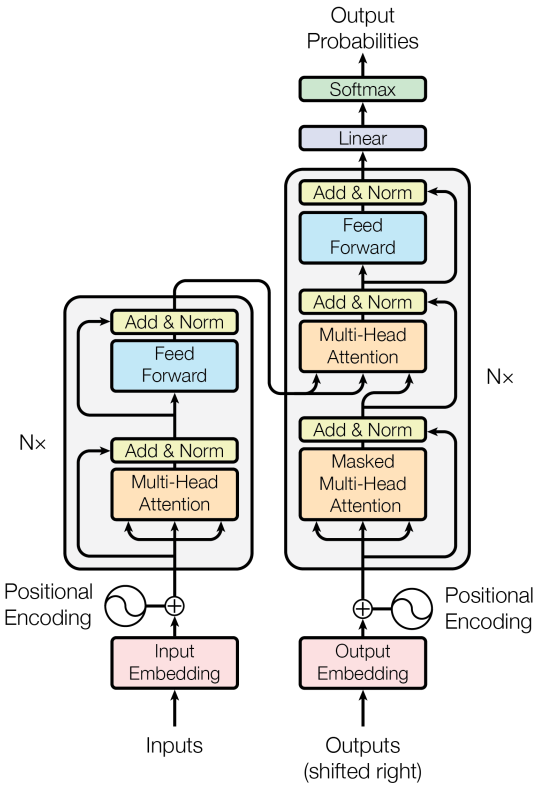


Figure 3. The Transformer - model architecture [42].

The encoder and decoder are illustrated on the left and right sides of the image respectively. In the original paper [42], the number of stacked layers for both the encoder and the decoder is $N = 6$. The encoder consists of two layers, distinguished by the box color on the graph: a multi-head self-attention layer, and an FNN. A residual connection can be observed around each of these layers connecting the input coming from the previous layer, or from the positional encoding, and sending it to the normalization layer. These are visualized as the lines inside the boxes that are the closest to the edges of the boxes (Figure 3). Embedding layers and all sub-layers produce an output of dimension $d_{model} = 512$. The decoder is distinct from the encoder in that it has another layer in between, which implements multi-head attention.

The key advancement in the Transformer architecture compared to RNNs lies in its reliance on the attention mechanisms, rather than on a recurrent layer. This means that there is no cyclical element in the model, and sentences are not processed word by word in a large number of iterations. Rather, the model learns to associate portions in one sequence with portions in another sequence, by processing input on a sentence or batch level, solely using the attention mechanism. The paper [42] introduces an implementation of an attention mechanism referred to as "Scaled Dot-Product Attention", which is a component of Multi-Head Attention that we can observe in Figure 3. The attention

mechanisms are shown in Figure 4.

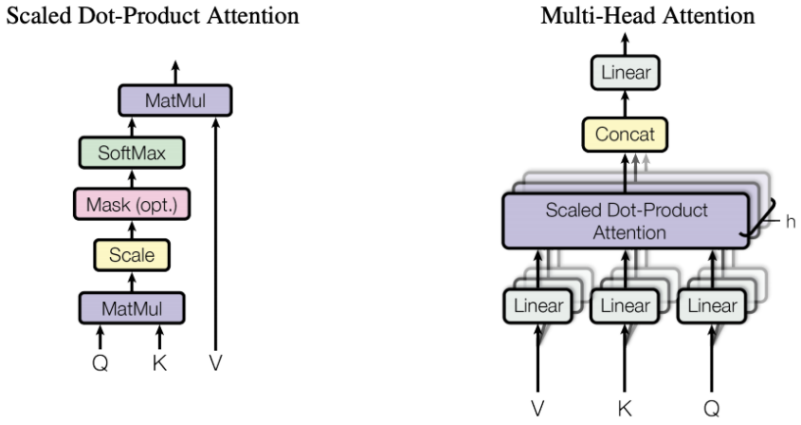


Figure 4. Scaled Dot Product Attention (left) and Multi-Head Attention (right) [42].

The inputs to scaled dot-product attention are three matrices: queries, keys, and values. The naming of these three is analogous to retrieval systems. For example, an image search engine maps a search term, referred to as a query, to a set of keys in a database, each of which is uniquely associated with an image, to obtain the search result. Each of these matrices is the result of a calculation of the input sequence to a weight matrix. For example, given an input z, we find Q, K and V as $Q = z * W_q$, $K = z * W_k$ and $V = z * W_v$, where W_q, W_k and W_v are learnable matrices of weights, the values of which are obtained in backpropagation. In the Transformer architecture, z is different depending on which of the three attention layers we are looking at. For example, in the encoder, z will be the output of the positional encoding layer and thus will not be a matrix of the same shape as the input sequence. Scaled Dot-Product Attention is defined as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{5}$$

where d_k is the dimension of queries and keys. The dot product of queries and keys signifies a search process. A softmax is used for obtaining weights that we can associate with the values. When the dimensions of keys and queries get very large, the softmax function will output values in regions with small gradients. Division by $\sqrt{d_k}$ helps to avoid the common problem of vanishing gradients. When it occurs, it is difficult to determine in which direction should the parameters of the neural network change to improve the cost function [16].

Multi-Head Attention (Figure 4) is a linear projection of scaled dot-product attention. In it, queries, keys, and values are projected h times with different, learned projections.

h is referred to as the number of heads. The projected versions enable to parallelize calculation of the attention function. After projection, the weight matrices of each head are concatenated and projected again, as depicted in Figure 4. As a result, the model can pay attention to different positions for different sequence representations simultaneously.

In addition to previously explained components, the Transformer includes layer normalization layers and masking. Layer normalization is used to scale all hidden units in a layer to a zero mean and unit variance so that they share the same normalization terms [2]. Masking is used to prevent the model from predicting a value in a sequence at position i from outputs in later positions. This is implemented in the decoder and ensures that the model remains auto-regressive, being able to take in prior output in its training phase. Since self-attention layers allow the decoder to attend to the current position, the outputs need to be shifted right (Figure 3). Masking involves setting all connections with future outputs as $-\infty$ inside scaled dot-product attention. Since the input is zero-padded, masking is also used in the encoder to guarantee that the attention mechanism does not pay attention to padding tokens. The input and output padding tokens are referred to as the source and target padding index respectively.

2.3 Transformer-based map matching

The Transformer-based map matching model was introduced by Jin, *et al.* [20] and can be distinguished from the initial Transformer architecture [42], by modifications to the Positional Encoding and Input Embedding.

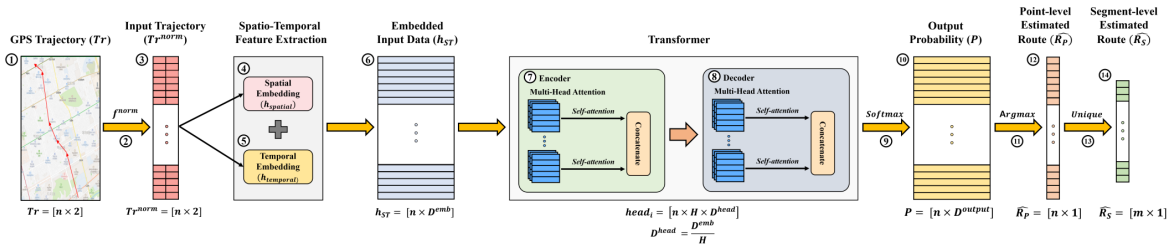


Figure 5. Transformer-based map matching process [20].

In the following paragraphs, we provide an overview of the features and definitions that are shown in Figure 5 and are not explained in the previous subsection. The first step depicted in Figure 5 involves normalizing the GPS coordinates. As a data processing step, it is further explained in Section 3.5. The formula used for normalization is [20]:

$$f^{norm}(X) = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{6}$$

While the original Transformer architecture inputs a one-dimensional array of tokens, then in the case of map matching, the dimension of the input is $[n$ x $2]$, where n is the

length of a GPS trajectory T. In addition, the original positional encoding component is substituted with Spatio-Temporal Feature Extraction, which combines two features: Spatial Feature Extraction and Temporal Feature Extraction [20]. Spatial Feature Extraction extracts the spatial features from the input GPS trajectory required for map matching in a feedforward layer. It is defined as:

$$h_{\text{spatial}} = FC\left(Tr^{norm}\right) \tag{7}$$

where Tr^{norm} is the normalized GPS trajectory and FC is a fully connected FFN. This component of the Transformer architecture extracts features from two-dimensional coordinate data. An alternative method in this step is using one-hot encoding of spatial features [45], however, this does not enable the extraction of the noise component of the GPS trajectory. GPS coordinates naturally contain an element of noise and can be more inaccurate in certain areas with more buildings [25]. Directly passing the normalized coordinates through the FFN layer enables the model to learn to account for the inherent noise effect. In addition, the relationships between GPS points in a given area are received by the model in contrast to if the points were divided into discretely separated zones [20].

Temporal Feature Extraction is the same component referred to as positional encoding in the original Transformer architecture, whereas for the map matching Transformer, the sequential positioning of GPS points in a trajectory is a temporal attribute of the data, not positional, as a GPS trajectory is a chronologically ordered set of points. The positional embedding is defined as:

$$h_{\text{temporal}} = Embedding\left(\ Arrange\left(\ len\ (\mathrm{Tr}^{norm})\right)\right. \tag{8}$$

where $Embedding$ is the dense representation that converts discrete position features to a continuous vector form, $Arrange(X)$ generates a sequence of integers $(1, 2, ..., X-1)$, and $len(Tr^{\text{norm}})$ is the length of the GPS trajectory. Experiments in [42] showed identical results for positional embeddings and cosine or sine function based positional encodings, which are marked as a component on the Transformer architecture graph in Figure 3.

After spatial and temporal feature extraction, the two features are combined in a spatio-temporal feature representation as:

$$h_{ST} = h_{\,spatial} + h_{\,temporal} \tag{9}$$

Because the target sequence that the map matching Transformer receives is of the same shape, as in the design of the original Transformer, modifications are made only to the encoder that receives a sequence of normalized coordinates. If we substitute the Input Embedding in Figure 3 with Spatial Feature Extraction and the Positional Encoding with Temporal Feature Extraction, we are left with the map matching Transformer architecture.

From the spatial and temporal feature extraction layers onward, the Transformer-based map matching process follows the architecture of the original Transformer, with

multi-head attention, normalization, and FFN layers (Figure 3). After obtaining output probability P in the final step, the point-level estimated route $\hat{R}_p$ is calculated as:

$$\hat{R}_p = \textit{argmax}(P) = [\hat{e}_1, ..., \hat{e}_n] \tag{10}$$

where $\hat{e}_1$, ..., $\hat{e}_n$ correspond to the matched road IDs for the GPS points p_1, ..., p_n respectively.

The segment-level route $\hat{R}_s$ is calculated from the point-level route by extracting the unique elements of the sequence without sorting, and keeping the order of the original sequence:

$$\hat{R}_S = \textit{Unique}(\hat{R}_p) = [\hat{e}_1, ..., \hat{e}_m] \tag{11}$$

In the following, we explain some key points from the only prior work on Transformer-based map matching [20]. Applying the Transformer architecture in map matching helps to capture internal correlations of a GPS trajectory and the relationship between the trajectory and the output route. The goal of the authors was to train a high-performance map matching model from generated data, in a limited ground truth data scenario. In so doing, they noticed an increase in map matching performance as the generated trajectories were added noise from a normal distribution with greater variance. The Transformer was able to outperform other RNN-based models in the map matching task when compared by accuracy on both the point and segment level. The authors analyzed the attention weights in the encoder and decoder module and determined that for mapping a single GPS point, the model utilizes GPS information on the current segment as well as neighboring segments of the road network. They discovered that the number of matching errors is greater when matching points at start and end segments of a trajectory, because there is only a single adjacent segment, and so the attention mechanism has less information to work with. The class output probabilities had lower values at intersections than in the middle of the road, and in some cases, the model failed to correctly classify points at intersections.

2.4 Trajectory reconstruction

The spatial and temporal sparseness of CDR data has been the basis for the introduction of several algorithms, which aim to predict the location of the individual for moments when the location is unknown. Chen, *et al.* provide a general definition of trajectory reconstruction as the solution to mobile phone data sparsity, which "aims at completing individual movement information by recovering the unspecified positions of each user" [5, p. 2]. Several algorithms have been proposed to recover missing consecutive events in the mobile network data that utilize only CDR records. We can list the following:

- Particle filter [12, 13]. As a Monte-Carlo algorithm, it provides approximate solutions. The resulting prediction is a sequence of edges that make up the path the user took

to travel between cells. The edges are part of a road network built from OSM data. Minimally, the input mobile data events need to have timestamp, cell ID, and coverage area information.

- Context-enhanced Trajectory Reconstruction (CTR) [5]. The trajectory data is transformed into a three-dimensional tensor format, enabling the use of Tensor Factorization to complete the missing elements in the data. The prerequisite for this approach is long-term and large-scale CDR data.

- Origin-Destination matrix and Markov chain based method [18]. The estimated trajectory is a cell-level trajectory.

- Switching Kalman Filter [17, 41]. It is a dynamic linear system that inputs the radius of the coverage area and outputs a probability for a move, stay or jump episode [41]. Even though it does not predict a segment-level route, it can be used to do so by first estimating GPS points from the sequence of CDR events, then mapping the point to road segments, and finally connecting the estimated segments as the shortest path [12].

The Particle filter outputs a most probable path on the road network and can be considered most similar to what we aim to achieve - segment-level reconstruction using CDR data [12]. The CDR data is preprocessed to account for ping-pong effects. For road network creation OpenStreetMap (OSM) data is used and a graph structure is created where every building represents a node. The single hyperparameter of the algorithm is the number of particles S, used to model the distribution of the user location. The Particle filter starts generating a path by first selecting S number of edges in the road network from a uniform distribution. It then continues to narrow the selection down to road segments, the endpoints of which are both in the given cell. Another random selection of S edges follows from the subset, to calculate the shortest paths between edge endpoints. Here, the algorithm takes into account the travel time on a path, using the timestamp column values in CDR records, to get a time difference between the current and previous timestamp. The model is tested on a simulated CDR dataset as well as a real-life dataset. For the real-life dataset, the model is evaluated based on GPS trajectories associated with the CDR data. Evaluation indicates a clear correlation between the coverage area diameter of a cell and the error of the model.

2.5 Deep learning applications in mobility studies

In this section, ideas from various mobility studies that apply deep learning are briefly mentioned. To the author's knowledge, there is no prior work on trajectory reconstruction from CDR data using neural networks. A common theme in trajectory estimation studies is to encode low-level trajectory data, such as GPS trajectories, in some categorical

spatial representation, such as spatial bins [24], or clusters [1, 8]. An exception to this is presented in [36], where latitude and longitude coordinates are standardized to the centroid of all trajectories in the dataset, before sending them to the RNN encoder. However, because of the low performance of RNNs on long sequences and the lack of multi-head attention, temporal attributes need to be separately sent to the encoder with such an approach and are done so using one-hot encoding [36].

In the study presented by Feng, *et al*. [14], an attention-based RNN is trained on CDR data to predict the next *location identification*. This means that the algorithm is trained to estimate the sequence of cell IDs, or the cell-level trajectory. The RNN includes an attention mechanism to capture the spatiotemporal relationship in the input sequence. The location identification, user ID, and time of a single trajectory point are translated into a one-hot vector. In addition to the current trajectory, the model inputs trajectory history, that is, all trajectories of the users to the current point in time. The similarities with the Transformer are feature extraction, the use of embedding layers, FFNs prior to prediction, and a softmax layer that outputs a class probability.

One of the few mobility studies that utilize the Transformer architecture for location prediction, does so, by using a grid representation of a city and GPS data [1]. Spatial features are not extracted directly from the GPS trajectory, but rather points of interest, spatial zones, and clustering results of GPS coordinates are used in training. The authors of [24] took the strategy of dividing spatial and temporal values into discrete bins. Time-specific and distance-specific transition matrices are calculated based on GPS data. As in [14], the RNN receives the historical data of all users.

Another study involved applying long-short term memory (LSTM) neural networks with self-attention mechanisms to predict the location and travel time from Bluetooth data [38]. The output of the model is a sequence of intersections, each intersection representing a Bluetooth receiver at that intersection, and the input includes a pair of longitude and latitude coordinates. The input trajectories include special token values to indicate the start and end of a trip. It resembles the use case of the Transformer in translating from one language to another, where input sequences include tokens to distinguish between a start and end of a sentence. Our use case is a symmetric opposite of this type of configuration - the start and end of a sequence are never fixated, rather the values in between them have a fixed token.

The study by Choi, *et al*. included predicting a cell-level trajectory using RNNs [8]. A vehicle trajectory from Bluetooth data is represented as a sequence of cells, where the cells are artificially created over an urban area, to form a hexagon-like structure. The model receives the current network traffic state and the cell sequence representation of vehicle trajectory data. The cells in that study are artificially created and are similar in shape and area, and in that aspect, the solved problem is very different from CDR data scenarios.

3 Methodology

3.1 Introduction

Prior work on trajectory reconstruction from CDR records is characterized by some application of probabilistic modeling, such as HMMs. The Kalman Filter, for example, presupposes a Gaussian distribution of noise in the data [41]. An alternative way to approach the sparsity of CDR data is to apply methods in trajectory studies, which focus on sequential problems - tasks where the goal is to predict the next label in an output sequence from the entire input sequence up to the current moment. This type of research involves using neural networks, primarily RNN based architectures, in trajectory generation [9] and path estimation [8]. The closest application of these ideas to the CDR context involved a synthetically created cell-like structure for an urban area [8]. Using neural networks to reconstruct urban vehicle or pedestrian trajectories from CDR data is a novel approach in mobility studies.

Our goal is to introduce a framework that involves using GPS data to provide the ground truth trajectory and training a model that is capable of predicting a point as well as a segment-level route from a cell-level trajectory, obtained by associating CDR data with GPS data. This model aims to learn the relationship between the mobility of individuals on the road network and the cell coverage areas in the mobile network. As a sequence model, it is used to estimate both the point and segment-level route from a sequence of chronologically ordered cell IDs. The latter process is defined as trajectory reconstruction in Section 1.1. By exploring how well the model estimates the location on the point level as well as the segment level, we leave ourselves the option to use the model in more precise location estimation in future studies.

3.2 Designed model

In the following section, we provide a basic overview of the data preprocessing steps and explain the design of our framework in greater detail. Data preprocessing steps are shown in Figure 6 and our framework is outlined in Figure 7 with a detailed example provided in Figure 8. Section 3.5 goes in greater depth into data preprocessing steps.

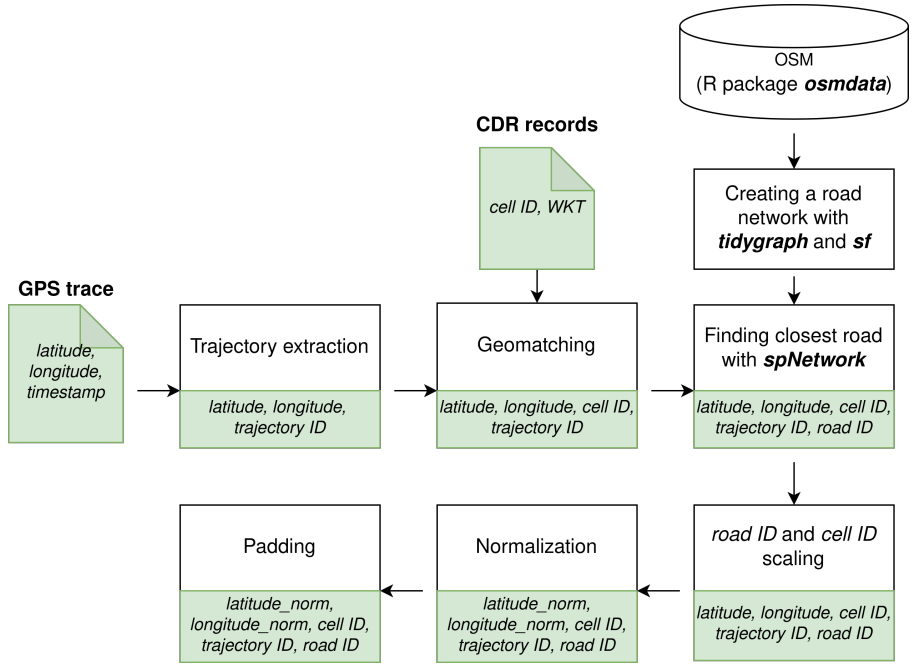


Figure 6. Data processing steps to generate input for dual Transformer framework. Green boxes list the columns that are present as a result of each step.

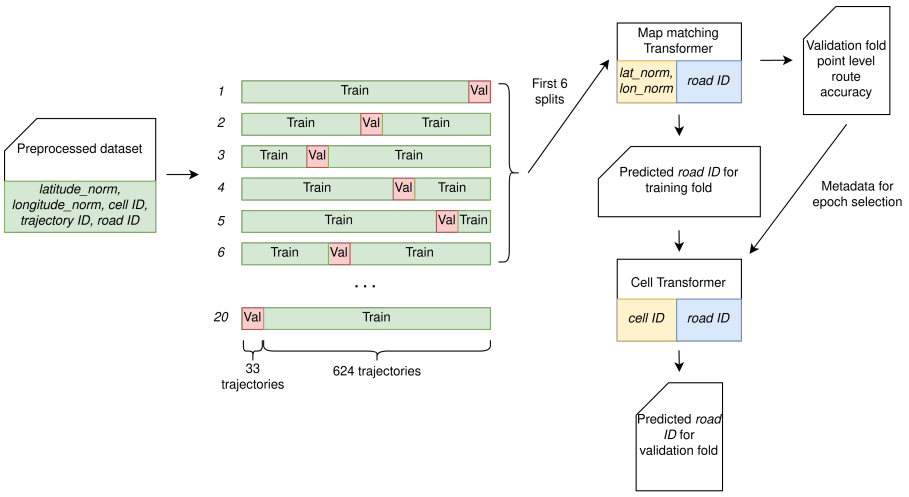


Figure 7. Configuration of dual Transformer framework. Yellow and blue are used to distinguish the input (source) and output (target) features for both Transformers.

Figure 8. An example of a single trajectory in the dual Transformer framework, with $unkown\_cell\_token = 1$. Yellow and blue are used to denote the source and target sequences for both Transformers.

Since we use GPS data to provide the ground truth, we begin by explaining how we use it in our framework. Firstly, we obtain the ground truth point-level route from GPS data. For this, we require a map matching model. The goal of map matching is to match an estimated location with a geographical shape in a road network to determine the street, which a person is on [4]. A map matching algorithm is necessary for several reasons. GPS data contains positioning errors due to issues such as changes in satellite visibility or receiver malfunctions. Network representations of roads may include misrepresentations of intersections and missing segments. Because our dataset contains pedestrian trajectories in addition to vehicle ones, we need an algorithm that is able to associate the GPS patterns on pedestrian walkways in proximity to the road to the roads themselves. Having chosen one, we can establish proper paths or routes, even if the user is moving on the sidewalk, for example. Map matching can be done solely based on the geometrical

distances between points and roads [43, 4], using applications of probability theory or machine learning, such as HMMs [27], or neural networks [20, 45]. The first option lacks modeling the inherent noise present in GPS data and is therefore considered the least accurate. Here, the Transformer architecture is chosen for its high performance in an urban context [20]. This component of the framework is referred to as the map matching Transformer, described in greater detail in Section 2.3, the use case of which is outlined in Figures 7 and 8.

Once GPS trajectories are coupled with CDR data and a cell-enriched GPS trajectory is obtained, we may proceed to train a model for point-level route estimation. By solving the map matching problem first, we expect to transfer the knowledge of the relationship between GPS trajectories and the road network to the neural network that reconstructs pathways from CDR data. Our framework involves two Transformers. Having learned the problem of map matching, the first Transformer can predict the point-level route. We expect this prediction to approximate the ground truth point-level route and to be a suitable input to the next Transformer, which we refer to as the cell Transformer. As we train this neural network to estimate the predicted point-level route from the output of the map matching Transformer, we expect its output to approximate the ground truth point-level route. Therefore, trajectory reconstruction depends on the performance of the map matching Transformer, as it produces the input for the cell Transformer. An overview of how both Transformers are stacked is provided in Figure 7.

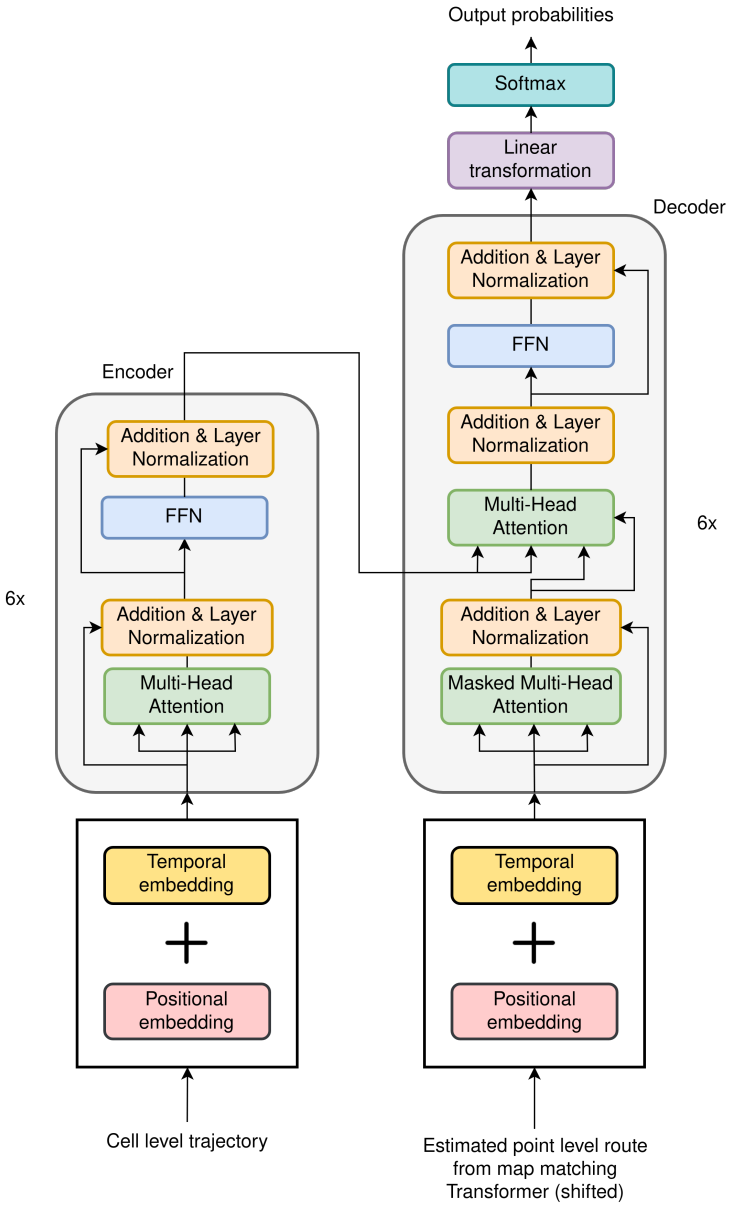


Figure 9. Cell Transformer architecture

The cell Transformer, the architecture of which is shown in Figure 9, is tasked with associating the sparse sequence of a cell-level trajectory with the corresponding dense, point-level route. An example of its application is shown in Figure 8. It learns the relationship between the source and target sequence. The former is a sequence made up of cell IDs and the latter of road IDs. Any cell ID may only be the trajectory start cell ID, end cell ID, or $unknown\_cell\_token$ representing no knowledge of which cell tower

the individual is connected to. In this step, we estimate as many road IDs for a road as the number of original GPS points on it. In training the cell Transformer, each sequence starts with as many identical cell ID values as GPS points were associated with the first road, and ends with as many identical cell ID values, as GPS points were associated with the last road. The example illustrated in Figure 8 serves to illustrate this pattern. The final output is a point-level estimated route, obtained after the $argmax$ function, in the same way as a point-level route is estimated in the map matching step. The estimated segment-level route is calculated from the estimated point-level route using Equation 11. For a more transparent and step-by-step evaluation, we also calculate the segment-level route from the point-level route estimated by the map matching Transformer.

3.3 Geomatching

To obtain the cell-enriched GPS trajectory, we need to associate GPS trajectories with CDR records. In this aspect, we choose to generate events based on cell metadata in the records. The use and rationale of geomatching (Algorithm 1) is further explained in Section 3.5.

Algorithm 1: Geomatching

Input: lat_1, lon_1, lat_n, lon_n, where n is length of GPS trajectory and lat and lon are the sequence of *latitude* and *longitude* coordinates respectively of a single GPS trajectory T,
Cellplan - cell plan for the user with the columns "WKT", describing the geographical shape of the cell coverage area, and cell ID with length N

Result: Start and end cell ID for GPS trajectory: $CellID_1$ and $CellID_n$

for $i = 1;\ i \Leftarrow N;\ i = i + 1$ **do**
  if $contains(Cellplan[i].WKT, Point(lon_1, lat_1))$ **then**
    $Cellplan[i].start\_area = area(Cellplan[i].WKT)$;
    if $Cellplan[i].area = min(Cellplan[i].start\_area)$ **then**
      $CellID_1 = Cellplan[i].CellID$;
  if $contains(Cellplan[i].WKT, Point(lon_n, lat_n))$ **then**
    $Cellplan[i].end\_area = area(Cellplan[i].WKT)$;
    if $Cellplan[i].area = min(Cellplan[i].end\_area)$ ***and*** $Cellplan[i].CellID \neq$ ***CellID***$_1$ **then**
      $CellID_n = Cellplan[i].CellID$;

3.4 Dataset

Call Detail Records are collected by telecommunication companies using cellular network systems primarily for billing purposes. They describe events triggered by receiving or

sending SMS messages, using internet services, or making calls. Recorded event logs describe the behavior of individuals in the network, and their generation is dependent on a number of factors, such as the user's location and signal strength. While no events are being produced in the network, a user's location is unknown. Therefore, the time differences between events in these records can extend over several hours. Because of this reason, and others listed in Section 3.5, we use only the coverage area information and generate synthetic CDR events rather than utilizing those in the records. The list of coverage areas corresponds to cells to which the users have recorded events in the network during a two-month period that includes the one-month period in the GPS data.

The dataset obtained for this thesis consists of CDR logs of users in Estonia and a record of their GPS locations. It is a subset of the data used by Dyrmishi and Hadachi [12] when evaluating the Particle Filter on a real-life dataset. GPS movement traces exist for around a one-month period. Since our analysis is solely interested in the urban context, a spatial bounding box filter is applied to include GPS points in the urban areas of Tartu, where we expect the users to travel more often. The bounding box coordinates are: $longitude\_min$ = 26.6825294. $longitude\_max$ = 26.7408943, $latitude\_min$ = 58.3459925, $latitude\_max$ = 58.3786785.

The GPS data includes features *longitude*, *latitude* and *timestamp*. The CDR records have columns for timestamp, event type, and heading. Since our approach involves synthetically creating CDR events, we only require the identifiers of the cell towers and the coverage area information of each cell tower. The former is recorded in the column *CGI*, which stands for Cell Global Identity, consisting of the Mobile Country Code, Mobile Network Code, Location Area Code, and Cell Identity. Only the Cell Identity is extracted from CGI, which we refer to as the cell ID. Each cell ID corresponds to an area from which one can connect to the transceiver of a base station, known as the coverage area. Information on the coverage area for a given cell ID is recorded as Well-known text (WKT), containing the longitude and latitude coordinates of points. These coordinates form a polygon shape, a geometric subtype that enables computing spatial relationships between geometries, as defined in the Dimensionally Extended Nine-Intersection Model (DE-9IM) [37].

3.5 Dataset preparation

In this section, we explain data processing in further detail. The preparation of the dataset is illustrated in Figure 6 and includes the following stages: trajectory extraction, trajectory splitting, geographical matching of CDR and GPS data, obtaining ground truth road IDs, trajectory normalization, and padding.

In the trajectory extraction phase, the GPS traces are split into trajectories or batches, each representing a short trip in the area of Tartu. An example of trajectory extraction is provided in Figure 10. First, the time difference from the previously registered timestamp is calculated for each point in the dataset, separately for both users. The data is split

when the difference exceeds 20 seconds. If a trajectory that has been split at 20 seconds contains more than 200 GPS points, it is further split until no trajectory is more than 370 points. Further splitting is required because, in later stages of padding, unusually long trajectories would considerably increase the memory consumption of the model.

In splitting the GPS point sequences, a time gap value of 30 seconds and 1 minute was tested, before opting for 20 seconds. For a 1 minute gap, we could observe hopping between non-intersecting roads in the GPS trace. If the user is in a vehicle, a gap of 1 minute is expected to show clearly on a map as a sparse sequence of points, because the GPS transmitter signal is dense in time. Since the extracted trajectories will be used in map matching, they are expected to be dense, so that points can be mapped to consecutive roads in a path.

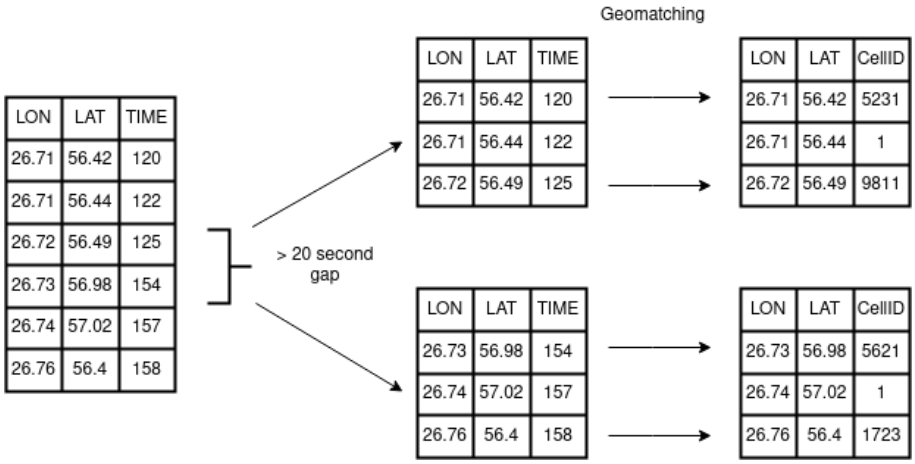


Figure 10. Example of extraction of two trajectories and the result after geomatching (a cell-enriched GPS trajectory) with $unkown\_cell\_token = 1$

As a next step, the cell ID column is added to the dataset using geomatching (Algorithm 1), the reasoning of which is explained in this paragraph. Its use helps to replicate the type of trajectories usually extracted from CDR logs. Based on determining if a geographical point is in a given area, it uses the spatial relationship *contains*, which is defined in the DE-9IM [37]. GPS coordinates and details of the cell coverage area are received as input. Only start and end GPS points in a trajectory obtain a cell ID value, as illustrated in Figure 10. Starting by searching which cells cover the area of the trajectory starting point, the cell with the smallest total coverage area is selected. This step is repeated for the endpoint. If the smallest cell containing the endpoint is the same as the one found in the previous step, the second smallest cell is selected for the endpoint. The search starts from the smallest cells, so we can obtain a wider distribution of cell IDs and avoid unusually large coverage areas. The result of geomatching is used after obtaining the ground truth road ID, to extend cell ID values and create the cell-level trajectory.

Extension here means that the beginning and end of a cell-level trajectory correspond to the number of points, that according to ground truth, are associated with the starting and ending roads. In Figure 8, we can see the repetition of the cell IDs in the cell-level trajectory - this is the result after geomatching and extension. The cell-level trajectory always contains less or an equal amount of $unkown_cell_token$s than a cell-enriched GPS trajectory.

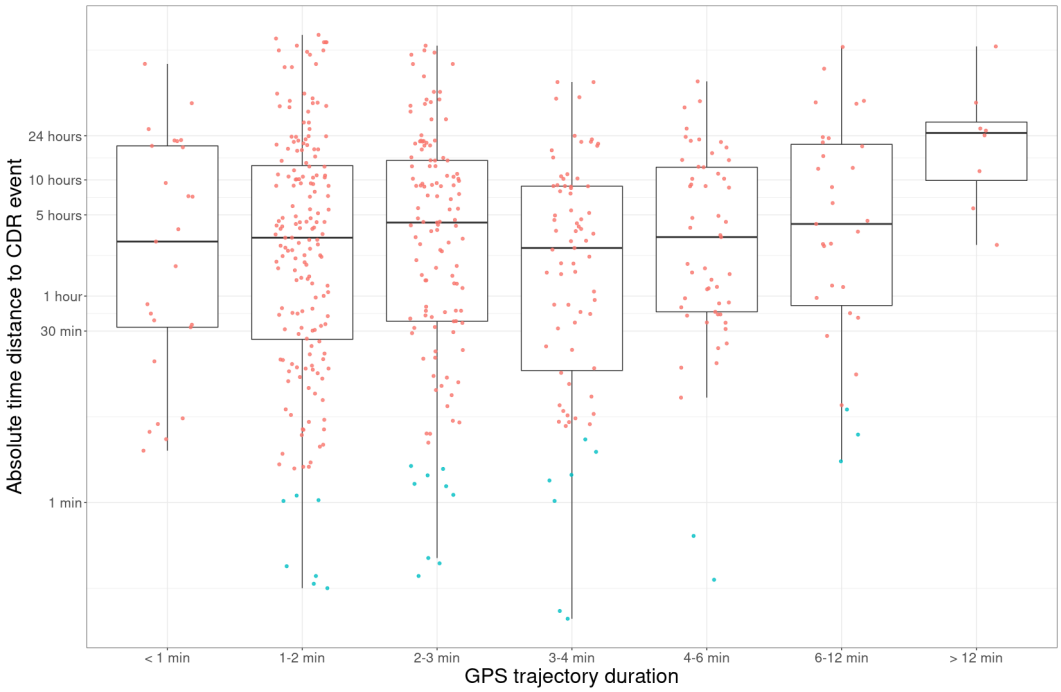


Figure 11. Boxplot of absolute time distance to closest CDR event for starting points of extracted GPS trajectories by GPS trajectory duration. Blue points represent trajectories where the closest CDR event is closer in time than the duration of the GPS trajectory.

When using GPS data with CDR data as ground truth, the standard way to link the two is using timestamp values [13]. Geographically joining GPS points to Cell IDs based on coverage areas is uncommon. For one, the coverage area does not reflect the real coverage zone [23]. A user may be outside the reported coverage area and nonetheless connected to the transceiver. One prior study included applying coverage area optimization [23]. Nevertheless, we present several reasons for ignoring events in CDR logs and synthetically generating them based on geographical information.

Firstly, the spatial integrity and coherency of the data are maintained. Trajectory starting roads that are closer to each other are more likely to be associated with the same cell ID than would be the case if joining were to be done based on timestamps. This is required if we expect our model to inherently learn the spatial relationship between cell

areas and the network of roads by learning to associate cell IDs with road IDs.

Secondly, we can extract more GPS trajectories with cell IDs. Network coverage in Tartu is dense, enabling us to associate a cell for every beginning and starting point. In contrast, cell events in the mobile network are sparse in time for the studied one-month period when the users were in Tartu.

Thirdly, timestamp values of CDR data and the GPS transmitter have discrepancies that we do not have to consider with geographical matching. These are illustrated in Figure 11. Only 28 trajectories are such that the GPS timestamp of their starting point is closer in absolute time to a mobile network event than is the duration of the GPS trajectory (marked in blue in Figure 11). The time density of CDR logs and GPS data is vastly different. We cannot extract CDR trajectories based on consecutive CDR events at a temporal density on par with GPS. The author of [13] associated GPS points to CDR data using events in CDR records from a dataset of which ours is a subset. However, the work in that thesis included GPS trajectories and CDR data outside of Tartu, for a larger number of users and a longer period of time [13].

Fourthly, the size of our CDR dataset is small compared to prior studies. Many CDR data based positioning algorithms have been tested on large-scale data and for macroscopic tasks, such as urban mobility research of 2.2 million users in the city of Riyadh [7] or for 300 000 users in Estonia [18]. Our CDR data contains 651 events for the two users, while the number of extracted GPS trajectories is 657.

Fifthly, the problem of ping-pong or oscillation effects is avoided. This occurs when a mobile device bounces between nearby antennas, triggering multiple events related to different cells, which could be located from a considerable distance apart. This effect occurs in timescales of seconds in CDR data [28, 21] and therefore would considerably affect the possibility of obtaining a start and end cell ID for every extracted trajectory. The result would be a dataset characterized by a degree of spatial incoherency, meaning that trajectories with a similar start or end point can have cells with varying coverage area and location.

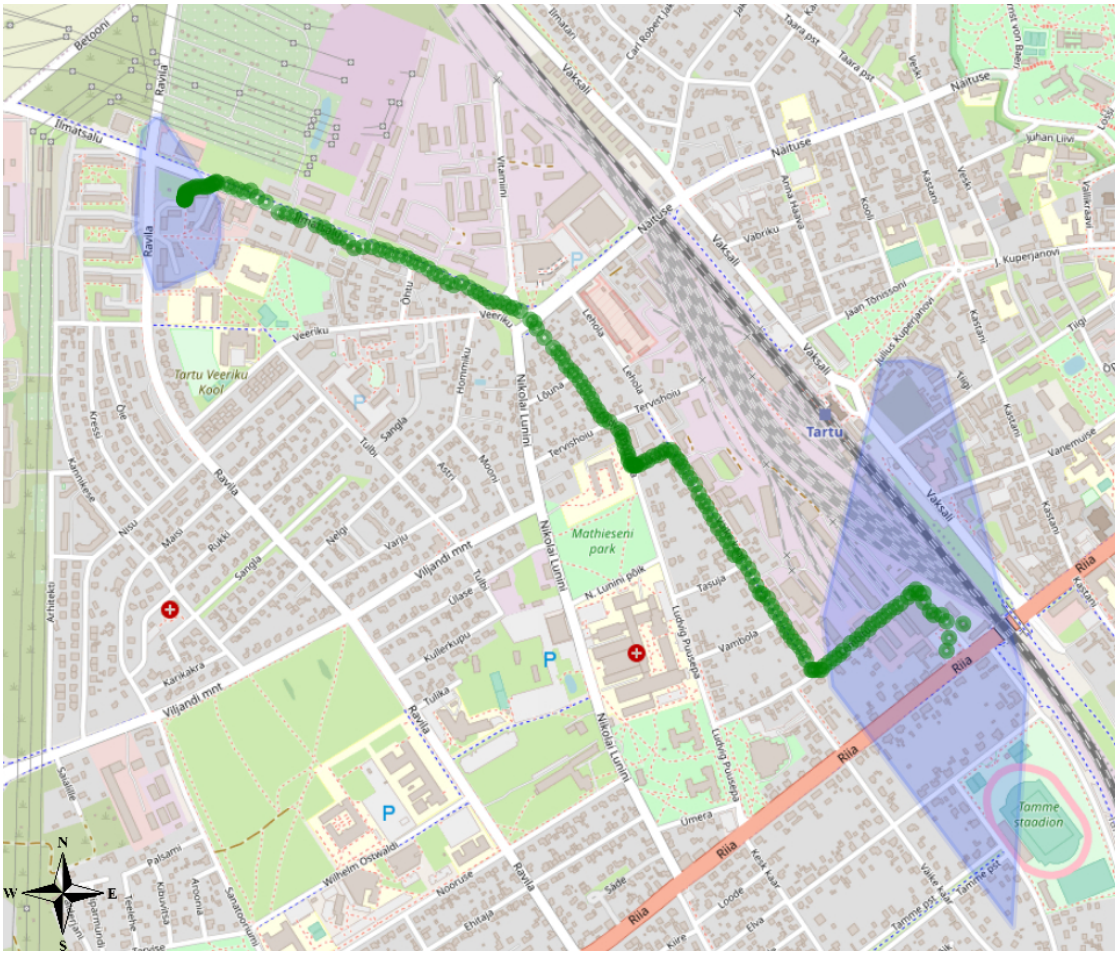


Figure 12. Example of an extracted and geomatched GPS trajectory. Blue polygons represent coverage areas corresponding to cell IDs obtained from geomatching. Green circles are GPS trajectory points.

The final feature we require for training is road ID, which is the basis for calculating evaluation metrics. For this purpose, a road network is built from the OpenStreetMap database [29]. OpenStreetMap is publicly available and maintained by volunteers [29]. To query road information on Tartu area, the *osmdata* package [30] for R is used. A set of unique vector geometry objects are downloaded with unique OSM IDs. This label does not represent a unique edge in a road network. To build a network, first, intersections between the geometry objects are found using the *tidygraph* [34] package. Then, the resulting edges are collected. We assign road IDs to GPS points by a simple distance calculation between each GPS point and vector geometry objects representing roads. The choice here is to use the R-tree indexing technique, which helps to improve query performance in cases of multi-dimensional or spatial data [22]. The exact function used is

`find_nearest_object_in_line_rtree` from the *spNetwork* package [15], with a snap distance of 5 and maximum number of iterations set to 10. Alternatively, GPS points may be manually labeled, which was the approach taken in [20] when implementing map matching on real data. Given the complexity and extent of the Tartu area road network, manual labeling would be too time-consuming. Because of the high density of GPS data, points on intersections may be labeled with intersecting road indices. As the line geometry objects at intersections are close, the resulting sequence of road IDs is charactarized by alternations at intersection points. We notice a significant unbalance in the distribution of resulting road IDs with the chosen geometric matching method. For 76 edges in the road network, only a single GPS point is mapped (Figure 13). These are largely intersecting roads on popular paths, resulting in a spatially uneven distribution of GPS points across the road network (Figure 15).

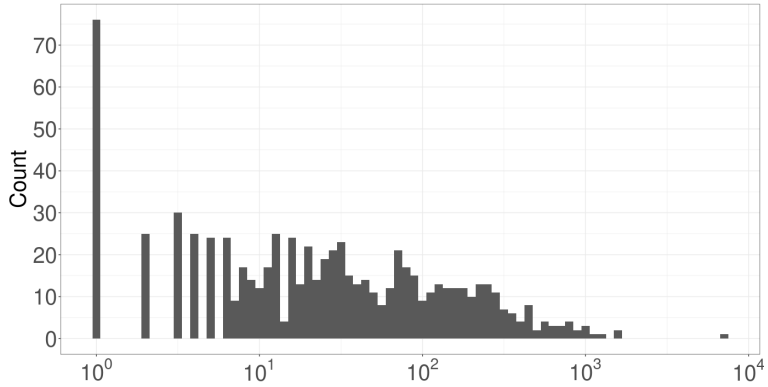


Figure 13. Number of GPS points on road in the ground truth dataset.

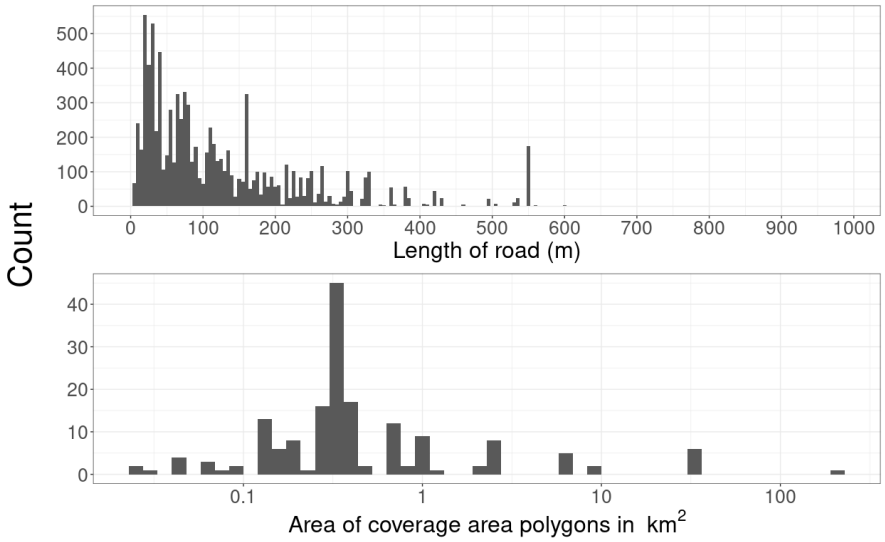


Figure 14. Length of road IDs (in meters) and area of cell polygons (in square kilometers) for the entire dataset.

The histogram of road IDs and cell IDs (Figure 6) illustrates the sparsity of the data and the dimensional difference between CDR data and road network data. The coverage areas may reach several square kilometers, whereas the sequence of roads we want to reconstruct may be composed of roads shorter than 100 meters. Some road IDs and cell IDs are over-represented in the distribution. Despite a geographical matching approach to the cell with the smallest coverage area, a few cells with coverage surface area above 10 square kilometers remain in the final processed dataset.

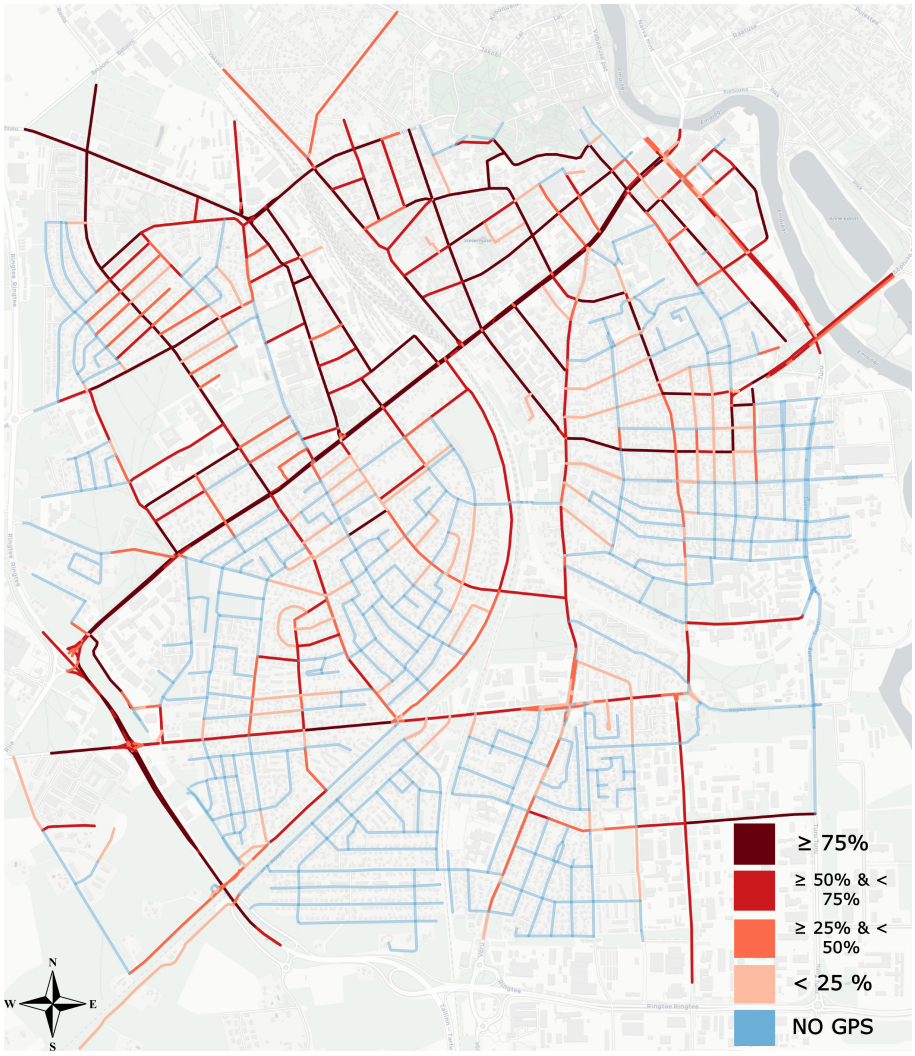


Figure 15. Road network and GPS point density. Colors indicate ranges in the empirical cumulative distribution of GPS points. For example, $< 25\%$ indicates that 75% of roads have more GPS points as ground truth.

When compared to the road network in the only prior work using Transformer-based map matching [20], our network is significantly more complex with varying shapes and lengths of roads (Figure 15).

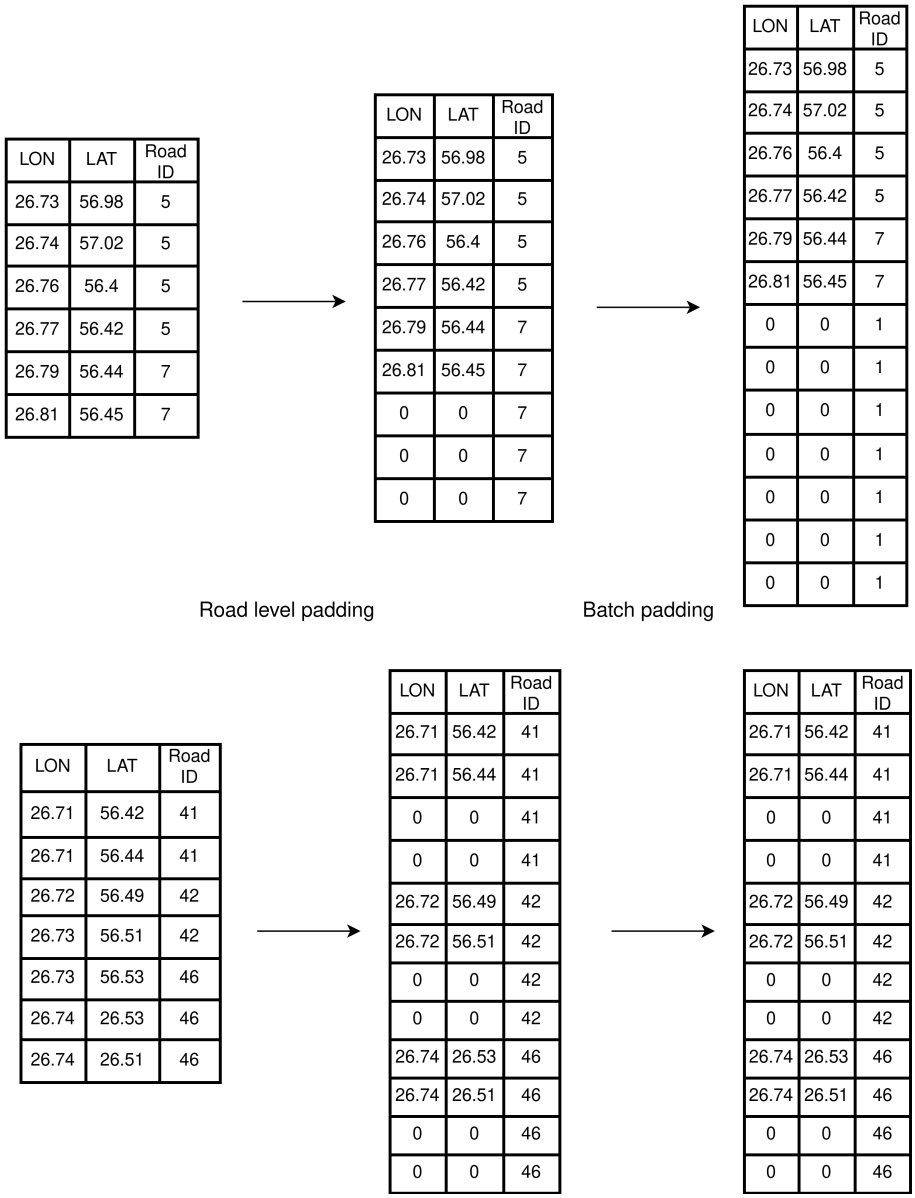


Figure 16. Example of padding for a dataset of 2 batches with the longest road level sequence of 4 points, source padding index of 0 and target padding index of 1.

Before normalizing GPS trajectories, road ID and cell ID are scaled so that the maximum value is equal to the unique number of values. Then, both latitude and longitude coordinates are normalized between 0 and 1. This is done using Equation 6. Ranges are set by the northmost, southmost, eastmost, and westmost GPS points of the whole dataset. In the padding step, every input trajectory is zero-padded so that the number of points on each road segment is equal in every batch, noted as *road level*

padding in Figure 16. As a result, unusually dense GPS traces in the dataset significantly increase the memory consumption of the whole model. The number of rows in our dataset increases with road level padding from 70 721 to 1 129 464. We choose this type of extensive padding because of later experiments with two strategies: only batch level post-padding and both road and batch level post-padding. The results are presented in Section 4.3. Padding is maintained throughout the training processes of both the map matching as well as cell Transformer. There are only a few studies that deal with the topic of the effect of different padding strategies to model performance in deep learning. One explored the differences between post and pre-padding for LSTMs for Sentiment Analysis [11]. The authors reported a 30% increase with pre-padding over post-padding, on both the training and test sets [11]. Another study focused on the effect that padding has on detecting images with Convolutional Neural Networks (CNNs) [19]. It was concluded that without padding, the neural network does not receive position information concerning the border areas of an image [19]. If we expand this idea to Transformers, we may experiment if padding in between the sequence, where roads begin and end, helps the model to be more accurate in predicting labels at intersections or borders of roads. Therefore, the selection of a padding technique is a component we need to consider from the perspective of training the model.

4 Results and Analysis

4.1 Experimental setup

Both Transformers were trained with a learning rate of 0.0006, a dropout of 0.1, an embedding size of 512, 8 heads, and 6 layers. The optimizer used was Adam with $\epsilon = 10^{-9}$. Both Transformers were implemented in PyTorch [33] and trained on the HPC cluster of the University of Tartu on 3 Nvidia A100-80GB GPUs.

The dataset is split 6 times, each time creating a separate train and validation set with a ratio of 95% and 5%. The splitting strategy is random, and the 6 validation datasets have no common trajectory IDs. This ratio is chosen due to the sparse distribution of the GPS data across the road network.

An overview of the dual Transformer setup and splitting is provided in Figure 7. The second Transformer receives the same split of the data as the first one and uses the first's output on the training split in its training. Point-level route accuracy is evaluated for each epoch in the first stage. The training data corresponding to the epoch with the highest evaluation split point-level accuracy is sent to the cell Transformer. This sequence of predicted road IDs is used in training by the cell Transformer, as is illustrated by the example in Figure 8. Both Transformers were trained for 9 epochs for each fold of the training data.

4.2 Performance metrics

Two performance metrics are introduced for our model: Bilingual Evaluation Understudy (BLEU) and accuracy. This selection follows that of Jin, *et al.* [20], who emphasize adding a performance metric for route integrity, in addition to popular metrics, such as accuracy, Average Hamming Distance (AHD), or F-score. Route integrity is a measure of the completeness of the map matching process to road segments at the trajectory level [20]. For example, if the ground truth is (2, 2, 5, 5, 5, 5, 1, 1) and predicted path (2, 2, 5, 5, 3, 5, 5, 1, 1), accuracy score is 0.875, but the predicted segment-level route (2, 5, 3, 5, 1) does not match the actual sequence of roads (2, 5, 1). In the context of map matching, such a predicted sequence involves a road path along the network that is less direct. BLEU reflects the matching of the order of the resulting sequence and its integrity [20]. It originates from language modeling, where it is used to compare sentences or corpora of text [32]. It is widely applied in sequential problems, including trajectory prediction [38], trajectory generation [9], cell-based trajectory estimation [8] and map matching [20].

BLEU uses a clipping method to calculate a modified form of precision. The modified

n-gram precision is defined as:

$$P_n = \frac{\sum_{g \in C_n} \min\left[m_{pred}(g), m_{obs}(g)\right]}{\sum_{g \in C_n} m_{pred}(g)} \tag{12}$$

where P_n represents the fraction of the n consecutive road label blocks in the predicted route that are also present in the ground truth route. C is a set of unique blocks in the output route, $m_{\text{pred}}(g)$ is the number of times a given n-consecutive road label block, g, occurs in the predicted route and $m_{obs}(g)$ is the number of times n-consecutive road label block g occurs in the ground truth route. BLEU score of order N is denoted as $BLEU_N$, which is made up of the geometric mean of N individual n-gram precisions P_n and a brevity penalty, as:

$$BLEU_N = \min\left[1, e^{1-\frac{l_{obs}}{l_{pred}}}\right] \exp\left(\sum_{n=1}^{N} w_n \log P_n\right) \tag{13}$$

where w_n is the weight for each n-gram precision (set to $w_n = 1/N$ for all n; i_{pred} and l_{obs} are the lengths of the predicted and ground truth routes respectively; and $\min\left[1, e^{1-\frac{l_{obs}}{l_{pred}}}\right]$ is the brevity penalty, used to account for unusually high precision scores that short trajectories may produce. The intuition behind $BLEU_N$ for point-level route estimation is that it measures how accurate is the model in predicting n consecutive road labels in a given trajectory from $n = 1$ up to $n = N$. For segment-level route accuracy evaluation we set $n = 1$ and for point-level route, we report $n = 1$ and $n = 3$, which is also the selected value for n in the original map matching paper [20].

The accuracy score of a point-level route is defined as the ratio between the common elements of a list of ground truth road labels and a list of predicted road labels. An accuracy score of 1 indicates a perfect matching to the ground truth road label. Point-level accuracy score is defined as follows:

$$Acc = \frac{\text{len}\left(\hat{R}_P \cap \hat{E}_P\right)}{\text{len}\left(\hat{R}_P\right)} \tag{14}$$

where $\hat{R}_P$ is the ground truth point-level route and $\hat{E}_P$ is the estimated point-level route.

Segment-level accuracy score is defined as:

$$Acc = \frac{\text{len}\left(\hat{R}_S \cap \hat{E}_S\right)}{\text{len}\left(\hat{R}_S\right)} \tag{15}$$

where $\hat{R}_S$ is the ground truth segment-level route and $\hat{E}_S$ is the estimated segment-level route. Sequence alignment is used prior to segment-level metrics calculation, which is discussed in Section 4.3.

4.3 Evaluation

In the following section, we present the results of our experiments and explain nuances concerning the calculation of evaluation metrics. Metrics are calculated separately for point and segment-level predictions, as in [20]. To provide a transparent and step-by-step overview of the performance of the framework, the output of the map matching and cell matching Transformers are presented separately, with the latter representing the final output of our framework. BLEU scores are calculated individually on trajectory level and the reported values reflect a mean across all trajectories, rather than a corpus level BLEU.

We need to use a sequence alignment algorithm to calculate segment-level route metrics, as the predicted and the actual segment-level route may have different lengths. We use the same idea as in Jin, *et al.* [20], and apply the Needleman-Wunsch algorithm [26] to align sequences in segment-level metrics calculation. This algorithm originates from bioinformatics, where it is used for aligning protein or nucleotide sequences [26]. It is applied locally for each trajectory, to align the ground truth and the estimated segment-level route. We use a Python implementation of the algorithm available at `https://github.com/zaneveld/full_spectrum_bioinformatics`. As the algorithm is used only in segment-level evaluation, a detailed explanation of it is out of the scope of this thesis.

We report our results for 6 non-overlapping evaluation or test folds of the data with a training split size of 624 and evaluation split size of 33 trajectories, corresponding to a 95% and 5% split. Both BLEU and accuracy scores are calculated for each trajectory batch and then averaged to get the fold average for a selected epoch.

In Table 1 and 2, performance of both map matching and cell matching steps is reported. Significant variability can be observed in both the map matching and cell matching phases by training and test split. Because of the uneven distribution of road lengths and GPS data as well as the random selection of evaluation splits, we expect it to be the case. By comparing Figures 15 and 26 we can better understand the spatial differences between different splits, which explain the variability in both point and segment-level routes accuracy. The first test split includes trajectories in areas, which are less represented in the data, and vary in shape more than those in split 6. For some streets in the Karlova area, for example, there is no underlying GPS data (Figure 15). Therefore, the low performance in test split 1, which includes trajectories in that area (Figure 26), is best explained by the lack of GPS points in that region, in the absence of which the map matching Transformer struggles to map the GPS signal in that area with the road network. In addition, some long roads are represented only in a single trajectory in the whole dataset, meaning that the model will not learn to associate GPS coordinates and road IDs in these areas when they are split into the test set. Given that our goal is to learn the historical movement patterns of the users and their relation to the mobile network, lower accuracy in sparser areas is expected.

We observe that the output from the cell matching Transformer approximates the ground truth more accurately than the one from the map matching Transformer, for nearly all folds (Table 1, 2). Therefore, despite errors in the predicted sequences in the map matching step, the final Transformer can approximate the ground truth. We are closest to fulfilling our primary goal, reconstructing trajectories on the segment level, with data split 6, obtaining a path accuracy score of 0.56 (Table 1).

Table 1. Map matching and cell matching model performance on point level. Epoch signifies the epoch with the highest accuracy score for a particular train-test split - for segment-level results this is the segment-level accuracy score, and point-level results, the point-level accuracy score.

	Map matching transformer			**Cell matching transformer**				
Data	Epoch	BLEU	Accuracy	Epoch	BLEU	Accuracy	Train size	Test size
Test split 1	5	0.522	0.5217	7	0.5216	0.5203	624	33
Test split 2	9	0.7622	0.7539	7	0.783	0.7782	624	33
Test split 3	7	0.7488	0.748	9	0.754	0.7513	624	33
Test split 4	5	0.8487	0.8432	3	0.7831	0.7784	624	33
Test split 5	9	0.8087	0.8062	8	0.8145	0.8076	624	33
Test split 6	2	0.8087	0.8045	9	0.8349	0.8302	624	33

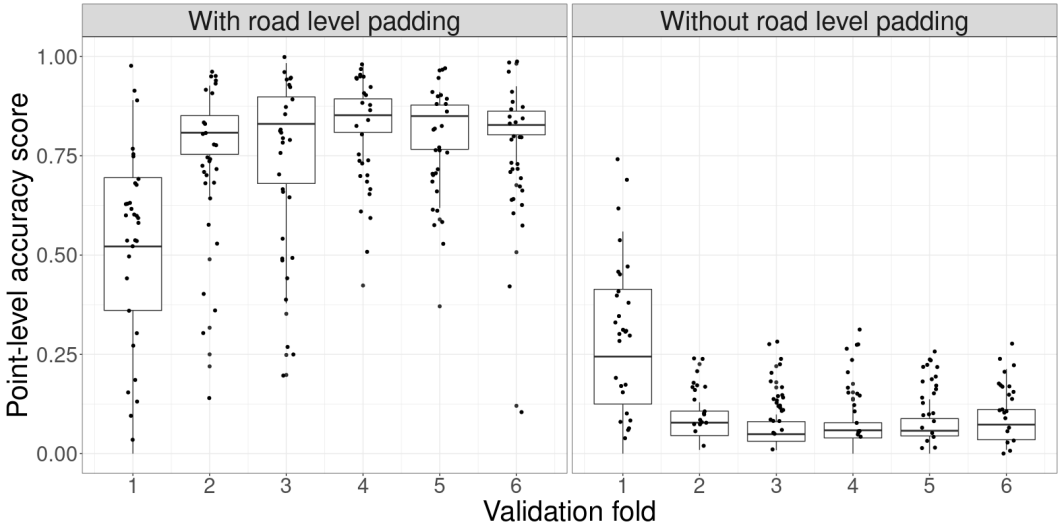


Figure 17. Map matching Transformer point-level accuracy score for all validation batches without and with road level padding by validation fold.

To illustrate the effect of the padding method, we present results on identical validation folds, but without road level padding, in Figure 17. The drastic difference between

the distributions supports our choice of padding technique in the map matching step, and from thereon onward in the cell Transformer step.

Table 2. Map matching and cell matching model performance on segment level.

	Map matching transformer				**Cell matching transformer**					
Data	Epoch	BLEU-3	BLEU	Accuracy	Epoch	BLEU-3	BLEU	Accuracy	Train size	Test size
Test split 1	5	0.0	0.1468	0.187	7	0.0	0.1489	0.1816	624	33
Test split 2	9	0.0118	0.3289	0.4415	7	0.0	0.3406	0.4484	624	33
Test split 3	7	0.0	0.3426	0.4622	9	0.0025	0.3368	0.4784	624	33
Test split 4	5	0.0	0.395	0.5323	6	0.0059	0.2885	0.4006	624	33
Test split 5	8	0.0059	0.3444	0.4614	8	0.0031	0.3707	0.4978	624	33
Test split 6	2	0.0	0.3555	0.4825	9	0.0106	0.3963	0.5612	624	33

Further investigation of the results shows that the map matching Transformer is systematically mismatching the first point on a road after an intersection. This mismatching transfers to the cell matching Transformer. We can observe this phenomenon by comparing the ground truth road ID and the predicted road ID at positions in the output sequence, where the previous ground truth road ID is different from the current one. Therefore, for illustration, segment-level accuracy is recalculated for each batch in each fold for all epochs separately on a dataset, where the first road ID on a road is removed. For example, for a ground truth point-level route of (2, 2, 3, 3, 3, 4) and a predicted point-level route of (5, 2, 4, 3, 3, 6), the new ground truth would be (2, 3, 3, 4) and prediction (2, 3, 3, 6). The systematic error begins to have a more substantial effect on the resulting accuracy calculation in later epochs as the model converges and errors remain at predictions in the predicted point-level route, which correspond to intersections (Figure 18). The mislabeled road IDs in the segment-level route can be distributed over various areas over the road network and do not make up a connected route. From Figure 25 we can see that the model is outputting seemingly random road labels, which we cannot interpret as taking another road at an intersection in our given road network.

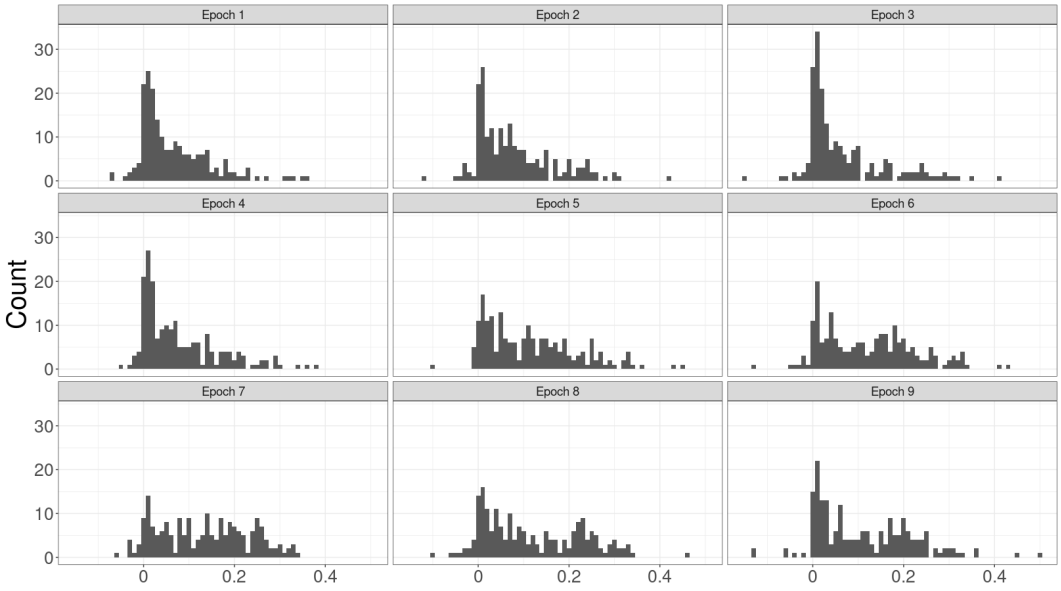


Figure 18. Change in segment-level accuracy of cell Transformer after ground truth based truncation for all 198 test trajectories from 6 different splits.

To account for the noise in the estimated point-level route, we may apply a consecutive filtering rule to obtain a more precise segment-level route. The idea is the following: the previous outputted road ID is rejected if it differs from the current one and the length of the chunk of the previous one is 1. For example, if the predicted point-level route is (2, 5, 5, 9, 11, 11, 3), we obtain a new point-level route of (5, 5, 11, 11), and from it a segment-level route of (5, 11). Another example is also provided in Figure 8. This method results in removing all chunks that contain only one element. We then evaluate the obtained segment-level route to the ground truth one. The results are presented in Table 3. After processing the output sequence by this rule, both accuracy and BLEU scores increase considerably, reflecting that the result contains significantly fewer random discrepancies or misalignment concerning the ground truth than the raw output of the model. A comparison can also be seen in Figure 23.

	Map matching transformer				Cell matching transformer					
Data	Epoch	BLEU-3	BLEU	Accuracy	Epoch	BLEU-3	BLEU	Accuracy	Train size	Test size
Test split 1	5	0.4003	0.5515	0.5095	6	0.3997	0.601	0.5477	624	33
Test split 2	9	0.5393	0.7061	0.6875	6	0.5772	0.7468	0.7258	624	33
Test split 3	1	0.5803	0.7236	0.7148	8	0.5738	0.7324	0.7127	624	33
Test split 4	5	0.5819	0.7852	0.7777	2	0.5306	0.765	0.7332	624	33
Test split 5	9	0.6609	0.8101	0.7876	7	0.6623	0.8147	0.7964	624	33
Test split 6	8	0.6539	0.792	0.784	8	0.6796	0.8014	0.7974	624	33

Table 3. Map matching and cell matching model performance on segment level after applying the consecutive filtering rule. Epoch is selected according to the highest segment-level accuracy after consecutive filtering.

An example of the output of our framework on an evaluation set of 33 trajectories after consecutive rule filtering is provided in Figure 19. The red line represents the estimated route and the yellow roads are the road IDs for which the cell Transformer received the cell ID. For each red-colored road, the corresponding cell ID value in the input sequence was the $unkown\_cell\_token$. A case with one large cell, where cells fully overlap, is provided in Figure 24. In this example, the segment-level trajectory is perfectly reconstructed after consecutive rule filtering. Similar performance of the model with large and overlapping examples, as with non-overlapping and smaller cells, is discussed a few paragraphs below.

The example in Figure 25 illustrates the difficulties of estimating a segment-level route from a prior point-level route, when a single point may be associated with an intersecting road at an intersection. When visualized, it reflects a wrong turn on an intersection. The model learns to estimate these turns, as the ground truth is a result of solely geometrical matching.

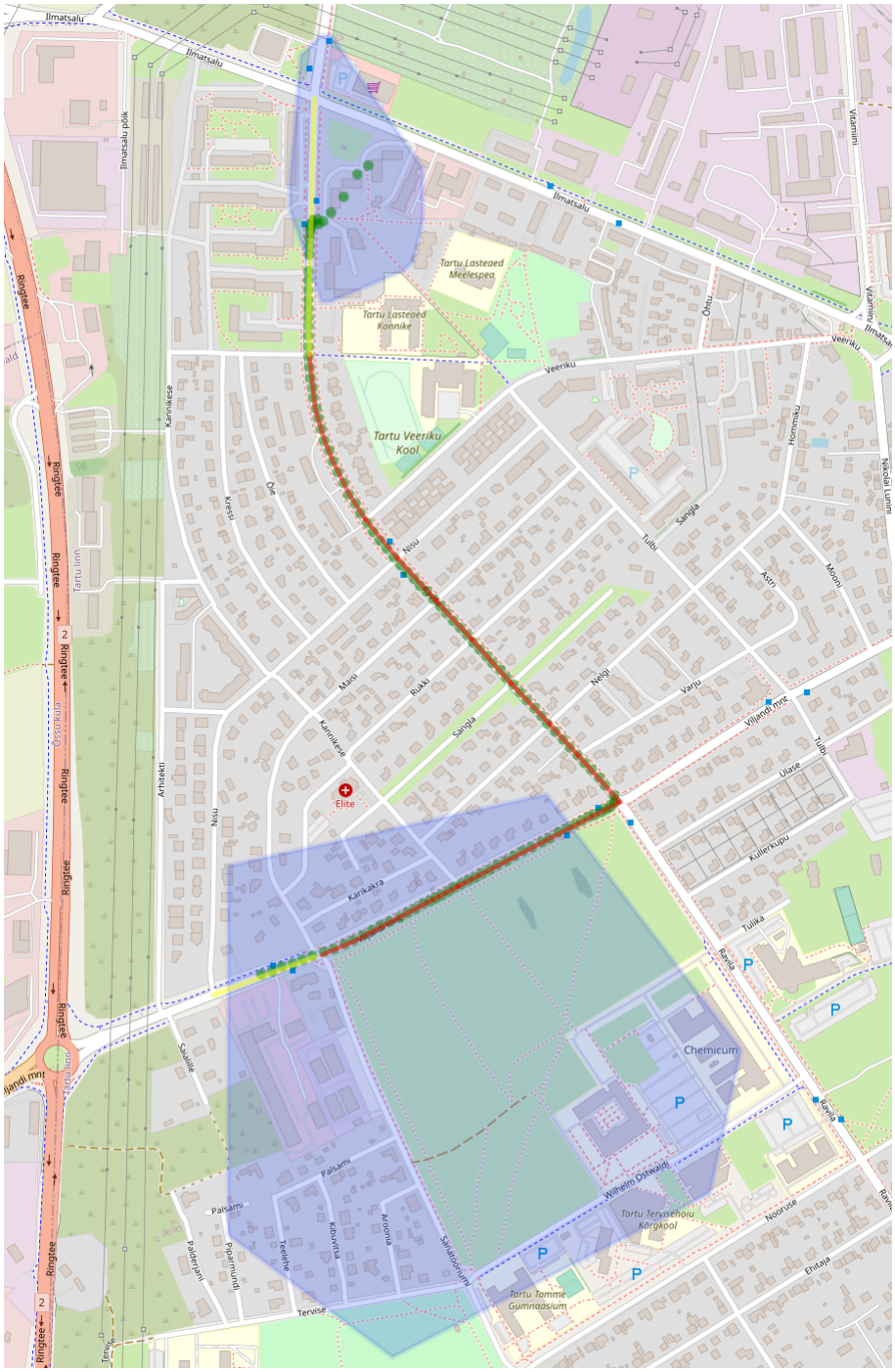


Figure 19. Example of trajectory reconstruction result after consecutive rule filtering. Yellow lines are the start and end roads, red line is the trajectory in between, green points are the underlying GPS trajectory and blue areas represent coverage areas for the start and end cell.

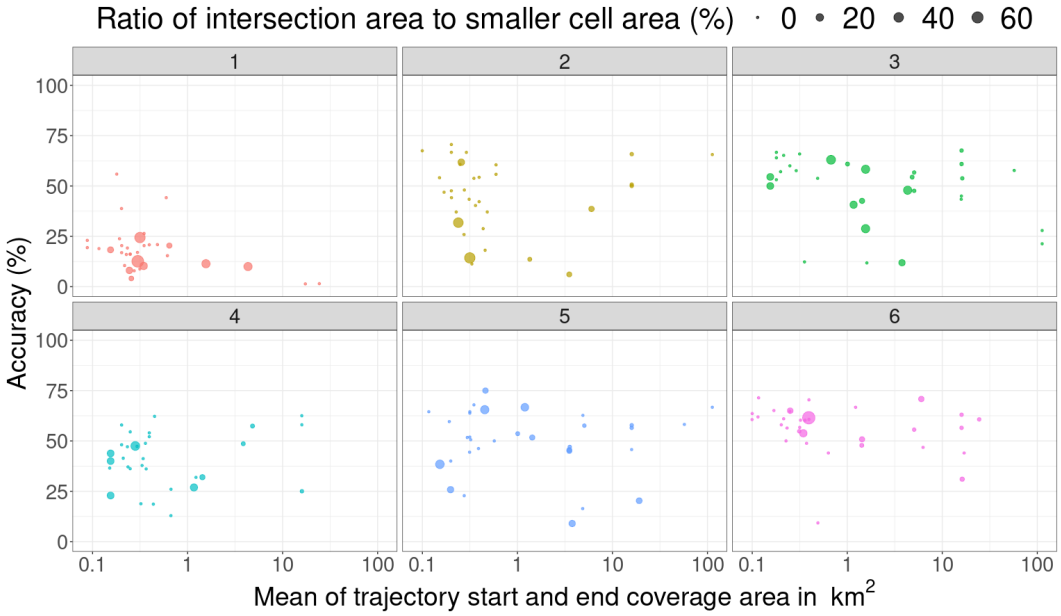


Figure 20. Segment-level accuracy by mean coverage area of trajectory and ratio of smaller cell coverage area to the intersection of coverage areas and fold.

As we observe great variability in the cell sizes of the original data, we are interested in how well our model performs in cases where the cells are overlapping, or very large. Prior work on the Particle Filter and Switching Kalman Filter showed a clear positive correlation between cell diameters and prediction errors in meters [12]. Looking at all of the folds, we do not see a correlation between predicted path accuracy and the average of the start and end coverage area. This is the case whether we look at the unfiltered (Figure 20) or filtered output sequences (Figure 21). For fold 1, we observe a negative Pearson correlation coefficient of 0.72, however, this fold has the lowest map matching accuracy. In fold 4, there is even a positive Pearson correlation of 0.29. Fold 6 with the highest segment-level accuracy (Table 2) has the least overlap between the start and end cell, yet cells with above 10 square kilometers of area are still represented. We observe no overall relationship between the degree of overlap between the starting and ending cell coverage area and the model results (Figure 21).

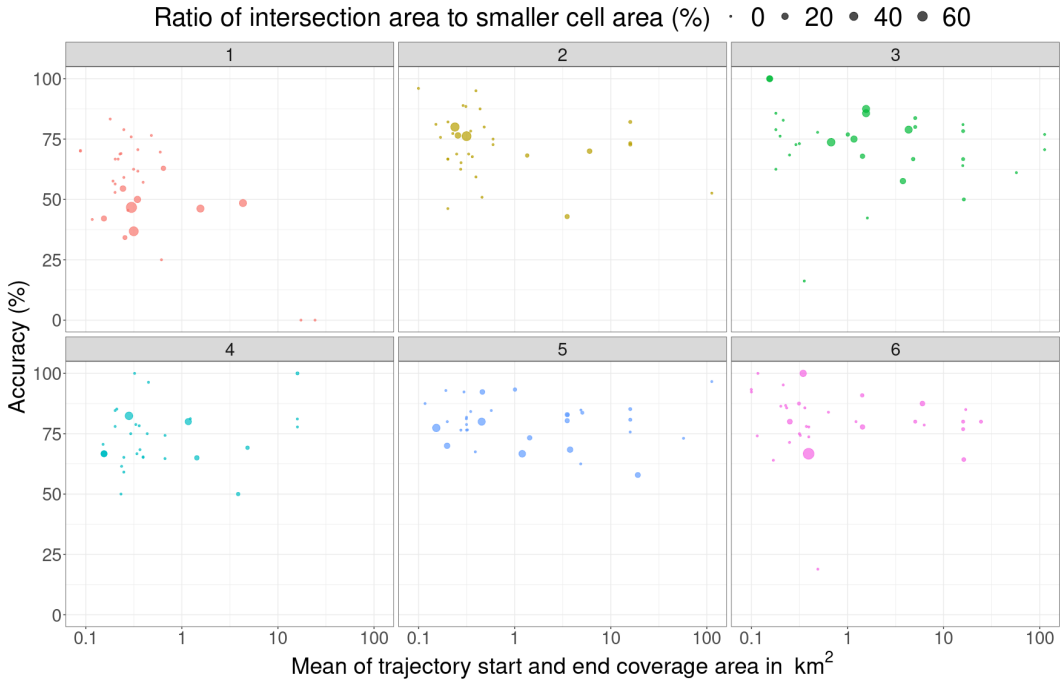


Figure 21. Segment-level accuracy after consecutive filtering by mean coverage area of trajectory and ratio of smaller cell coverage area to the intersection of coverage areas as well as fold.

4.4 Comparison with other approaches

In this section, we focus on comparing our approach to the Switching Kalman Filter [41] and Particle Filter [12]. Firstly, we explain the differences between our work and prior work. The original dataset we used is the same as in [12], however, our analysis is based on data inside the bounding box of Tartu described in Section 3.5. Most importantly, GPS trajectories were associated with CDR records by calculating the closest timestamps in [12]. Therefore, the data used in [12] includes cases where the actual location of the individual is not within the coverage area of a cell tower. As we produced CDR events synthetically, our data does not exhibit this. Secondly, we can list reasons why a comparison is suitable. The GPS data as well as the coverage areas have the same distribution. This is because we use the same CDR coverage areas - those for which we know from network data, that the two individuals were connected for 2 months. Furthermore, the Particle filter takes as input a cell ID and outputs a predicted path along a road network. The difference between the use case of the cell Transformer and the Particle filter lies in the latter using timestamp values as part of the algorithm. Our model, on the other hand, receives all the temporal information it needs in the sequence itself. In our case, the model will start reconstructing a trajectory from a cell ID based on

the set of road IDs that it has learned to associate with it. Because of geomatching, at least one point on the starting road ID is within the coverage area of the starting cell ID. Therefore, we expect our model to learn to start the predicted sequence from a road at least partially within the given cell coverage area. In addition, the authors calculated the ground truth road label based on simple geometrical distance, which is the case with our setup (Section 3.5).

In contrast to the Particle filter, the Kalman Filter does not output a sequence of road labels. Rather, it predicts the user's location as geographical coordinates and uses the radius of coverage area polygons to do so [41]. However, the author in [12] applied it as a comparison to the Particle filter by geographically mapping each predicted point to the closest road in the road network. We compare their results (Figure 22) to segment-level accuracy in each fold (Figure 23). Dyrmishi and Hadachi report a Switching Kalman Filter average accuracy of 14% and a Particle Filter average accuracy of 17% [12]. This is lower than for any of the 6 train-test splits with our method (Table 2). Comparison of accuracy distributions in Figure 22 and Figure 23 indicates a drastic improvement, even without post-processing the output. The lack of examples in the above 90% range without consecutive rule filtering is due to the systematic errors discussed in Section 4.3.

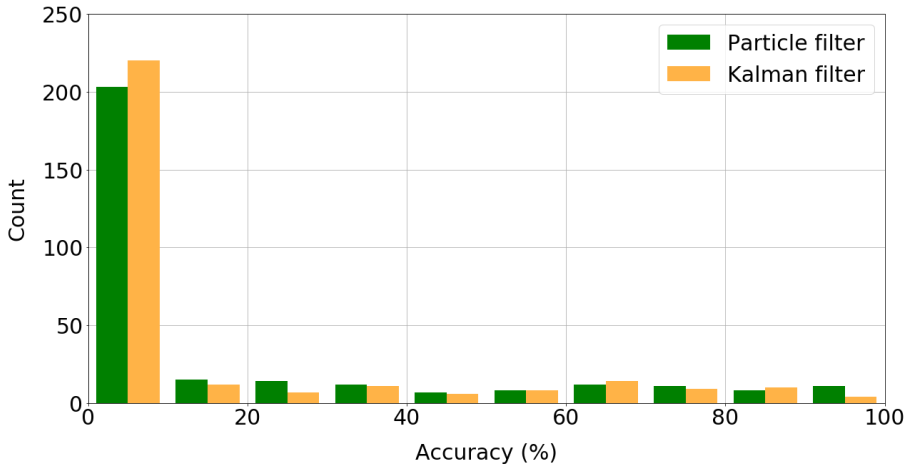


Figure 22. Switching Kalman Filter and Particle Filter path accuracy in real data [12].

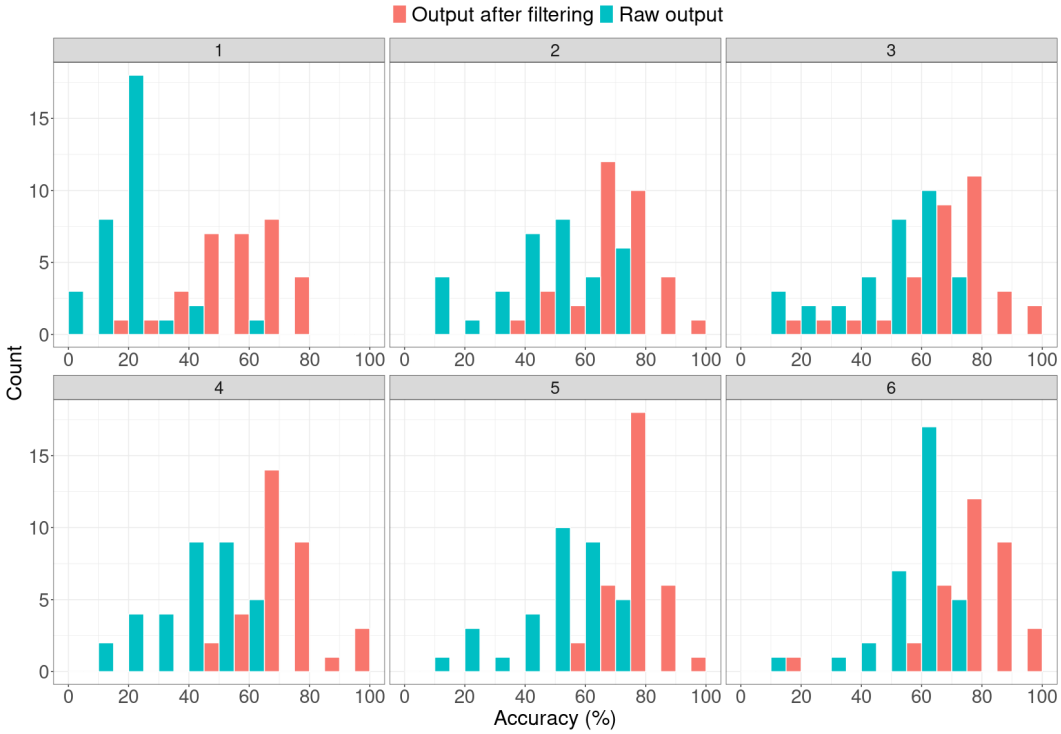


Figure 23. Segment-level accuracy for 6 different splits, before and after consecutive rule filtering.

5 Discussion

5.1 Challenges

Implementing our experimental setup included several conceptual and technical challenges, such as the approach to associating CDR data with GPS data, extracting suitable trajectories in an acceptable size for our computations, padding of input, memory management, and configuring the spatial feature extraction layer unique to the map matching Transformer. Firstly, our initial attempts to use the timestamp values in CDR data to enrich GPS data with cells indicated that the time discrepancies between the GPS time and CDR event times are too great for a micro-level urban analysis. Since keeping to cell activation times in the data would leave us too small and sparse of a dataset for the learning process, we implemented geomatching (Algorithm 1). Secondly, after trajectory extraction, we saw that some of the GPS traces were of movement in a park or in between houses, not on the city roads. We manually removed them. Thirdly, our initial experiments included only batch padding. In these experiments, we observed significantly lower accuracy scores than with road level and batch padding, as shown in Figure 17. The idea to pad the sequences on the road level came solely from a previous experiment conducted on a small synthetic dataset used to test the framework, which was generated with an equal number of GPS points for each road. Fourthly, because of road level padding, we had to set an upper limit to trajectory length, as too long trajectories would cause the input data size to grow unacceptably large. Even with the final setup, the memory consumption of the model due to the padding remains unusually high, in the range of 160-240GB. Since we observed high accuracy on the point level in early epochs for several splits, we opted for a lower number of epochs and 6 different splits of the data. Training time with 6 splits was around 5 hours. Fifthly, the original implementation of Transformer-based map matching is not publicly available at the writing time of this thesis. In our implementation in PyTorch, the spatial feature extraction layer has an input dimension of (2, 512), an output dimension of (512, 512) and uses the ReLU activation function (Equation 2). Finally, parameter optimization in a close range to that chosen in [20] showed no clear effect on performance. Experiments with a learning rate of 0.07 resulted in numerical instability. Point-level accuracy began to decrease in the map matching step with a learning rate of 0.00085. After experiments with learning rates of 0.0003, 0.0005, 0.0006, and 0.0007 resulted in no clear differences, 0.0006 was chosen.

5.2 Discussion of results and future work

A significant point of discussion concerns obtaining the ground truth road labels. Because of the density in the GPS traces, geographically mapping points to roads in the dataset preprocessing step presents complications for training. The map matching Transformer seems to struggle in learning the relationship between GPS point sequences and road

labels when sequences break at intersections, and there are practically no examples of trajectories on many of the intersecting roads. This is the most likely explanation of the systematic mismatching of the first point in point-level route estimation. Even though in the original paper introducing this method, a lower accuracy at intersections was also observed [20], in our case, this error is manifested in the output route, rather than the output probabilities, and remains across training epochs. Knowing the distribution of the data, we may filter the output sequence by a logical rule. Still, some trajectories display pedestrian movement over crossroads, and the issue of how to provide the ground truth label in these cases remains. Future work could benefit from a more detailed approach to road network building. Using intersections between OSM road geometry resulted in some micro-level as well as complex geometry at intersections and roundabouts. A possible future direction would be to replicate the work of Jin, *et al.* [20] further, and synthetically generate GPS data with road labels from the start.

In contrast to probabilistic solutions of the Particle and Kalman Filter, our model results do not indicate a correlation between accuracy and coverage area overlap or coverage area size. This is likely due to the spatial distribution of the data and the geomatching approach. Our cell data is distributed such that large cells often contain other smaller cells, which can be geographically matched to GPS points in these areas. Therefore the actual area, in which a GPS point can be associated with a large cell, depends on the distribution of smaller cells within it. Cases with overlapping coverage areas do not display a lower performance. Geographical matching ensures that GPS points in the same area are always associated with the same cell ID, regardless of overlap. The Transformers learn to associate sequences by ID and therefore no spatial information regarding coverage areas is processed by the model.

The Transformer-based method is more accurate in path estimation than prior Particle and Kalman filter models. The inherent advantage comes from our framework learning from past historical patterns, unlike approaches based on probabilistic modeling. This also means that a high percentage of training data is required, and the distribution between the training and test sets must have some similarities. In addition, the attention mechanisms in the Transformer learn to associate two sequences on a point level and therefore require no processing of the coverage area information, such as coverage optimization. Probability calculations based on CDR records, such as movement probability in the Kalman Filter case [41], or transition probability for the Particle Filter [12] are not required. Working with sequences means that temporal sparsity does not present an obstacle, as time is treated as discrete intervals. Of course, this is the case when synthetically constructing CDR using the given cell tower coverage area coordinates in CDR logs. Future studies could explore the effect that inherent ping-pong and oscillation phenomena have on the performance and training of the model. This work would be needed to apply our framework in practical CDR network scenarios, such as predicting the next road in real-time. Furthermore, our approach includes normalizing GPS coordi-

nates to a selected box in an urban area. When applied to a larger area it must be further studied, how effectively the map matching Transformer can map the significantly sparser GPS signal to the sparser road network, that would be produced by non-urban areas.

Experiments indicated that in the map matching training phase, the spatial distribution of the data plays a key role. A lack of GPS training data in certain areas of the city will result in low performance in GPS point matching to roads, and this will transfer on to the final mapping process. Despite its applicability for urban environments with roundabouts and complex intersections, the current technique for sequence padding necessitates high computing capability. Future research could explore the attention weights of the map matching Transformer for individual points at intersection moments to gain more insight into the mislabeling of GPS points. In addition, it needs to be further explored why padding the data in-between, to zero pad the sequences on the road level to an equal length, helps in achieving higher performance. A possible explanation is that padding helps the model in detecting intersections or border areas between roads. Even though our goal was to solve reconstruction on a segment level, the current framework enables us to explore more exact positioning on the point level as well. Further studies could compare the cell Transformer with deep learning models in trajectory generation, and aim to approximate the more precise, point-level estimations, which is the goal of many trajectory studies utilizing neural networks for sequential problems.

6 Conclusion

The challenges presented by sparse mobility data in user location estimation are especially evident in an urban context. Using GPS data as ground truth and approaching trajectory reconstruction from CDR data as a sequential problem, synthetic CDR events are created for the cells for which we know data events existed. Such generated CDR events prove to be enough for predicting pathways on a road network using a Transformer. We give to it as input only the starting and ending mobile network tower identifier for a given trip, having map-matched GPS points to roads to establish ground truth using another Transformer.

We trained and evaluated our two-step Transformer framework using real-life datasets, and compared the results with prior works that utilize coverage area information. A comparison with the Particle Filter and Switching Kalman Filter from a study utilizing in part the same coverage area information as our work, indicated a performance increase from 1% to 39%, depending on the data split. Further analysis showed that the first stage of our framework produces systematic errors at intersections which we can easily detect in the estimated sequence of road labels. What contributes to this is the size of the dataset and GPS distribution, complex road network and obtaining ground truth road labels by a simple distance calculation. When removed by applying a logical rule sequentially, the model can perfectly reconstruct 3 trajectories out of 33 for the most optimal data split. Additionally, it can predict pathways on a level that is as dense as a GPS trajectory sequence.

For application in practical CDR scenarios, future work should include training on larger real-life datasets with inherent ping-pong or oscillation effects or dealing with these components in the data prepossessing steps. Alternatively, accessible online road network and cell area databases make it possible to train the whole framework using large-scale synthetic data with. Which option works best in real-life applications, is yet to be explored. Our case study combined the mobility patterns of two users and involved learning from randomly selected movement patterns in the same month. Further work is required to see how differences in the spatial distribution of data for a larger number of users over a longer period affect the model's performance. In addition, for practical application from a memory consumption perspective of the model, sequence padding needs to be further investigated, and more memory-efficient solutions explored.

7 Acknowledgement

I would like to express my gratitude to my supervisor, Amnir Hadachi, for his guidance, support and encouragement.

References

[1] Zain Ul Abideen et al. "Deep Wide Spatial-Temporal Based Transformer Networks Modeling for the Next Destination According to the Taxi Driver Behavior Prediction". In: *Applied Sciences* 11.1 (2021). ISSN: 2076-3417. DOI: 10.3390/app11010017. URL: https://www.mdpi.com/2076-3417/11/1/17.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. DOI: 10.48550/ARXIV.1607.06450. URL: https://arxiv.org/abs/1607.06450.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[4] Dave Bernstein and Alain L. Kornhauser. "An Introduction to Map Matching for Personal Navigation Assistants". In: 1998.

[5] Guangshuo Chen et al. "Complete Trajectory Reconstruction from Sparse Mobile Phone Data". In: *EPJ Data Science* 8 (Dec. 2019). DOI: 10.1140/epjds/s13688-019-0206-8.

[6] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179.

[7] Philip Chodrow et al. "Demand and Congestion in Multiplex Transportation Networks". In: *PLoS ONE* 11 (Sept. 2016). DOI: 10.1371/journal.pone.0161738.

[8] Seongjin Choi, Jiwon Kim, and Hwasoo Yeo. "Attention-based Recurrent Neural Network for Urban Vehicle Trajectory Prediction". In: *Procedia Computer Science* 151 (Jan. 2019), pp. 327–334. DOI: 10.1016/j.procs.2019.04.046.

[9] Seongjin Choi, Jiwon Kim, and Hwasoo Yeo. "TrajGAIL: Generating urban vehicle trajectories using generative adversarial imitation learning". In: *Transportation Research Part C: Emerging Technologies* 128 (July 2021), p. 103091. DOI: 10.1016/j.trc.2021.103091. URL: https://doi.org/10.1016%2Fj.trc.2021.103091.

[10] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929.

[11] Mahidhar Dwarampudi and N. V. Subba Reddy. "Effects of padding on LSTMs and CNNs". In: *ArXiv* abs/1903.07288 (2019).

[12] Salijona Dyrmishi and Amnir Hadachi. "Mobile Positioning and Trajectory Reconstruction Based on Mobile Phone Network Data: A Tentative Using Particle Filter". In: *2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. 2021, pp. 1–7. DOI: 10.1109/MT-ITS49943.2021.9529277.

[13] Saljona Dyrmishi. "Creating a novel approach for mobile positioning based on CDR data". MA thesis. Tartu, Estonia: University of Tartu, 2020. URL: https://comserv.cs.ut.ee/home/files/dyrmishi_computer_science_2020.pdf?study=ATILoputoo&reference=4C4089DB2C7FD8F46460F4DEFE8DD4CFF10FD5E1.

[14] Jie Feng et al. "DeepMove: Predicting Human Mobility with Attentional Recurrent Networks". In: Apr. 2018, pp. 1459–1468. DOI: 10.1145/3178876.3186058.

[15] Jeremy Gelb. "spNetwork: A Package for Network Kernel Density Estimation". In: *The R Journal* 13.2 (2021), pp. 561–577. DOI: 10.32614/RJ-2021-102. URL: https://doi.org/10.32614/RJ-2021-102.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[17] Amnir Hadachi and Artjom Lind. "Exploring a New Model for Mobile Positioning Based on CDR Data of The Cellular Networks". In: *CoRR* abs/1902.09399 (2019). arXiv: 1902.09399. URL: http://arxiv.org/abs/1902.09399.

[18] Amnir Hadachi, Mozhgan Pourmoradnasseri, and Kaveh Khoshkhah. "Unveiling large-scale commuting patterns based on mobile phone cellular network data". In: *Journal of Transport Geography* 89 (2020), p. 102871. ISSN: 0966-6923. DOI: https://doi.org/10.1016/j.jtrangeo.2020.102871. URL: https://www.sciencedirect.com/science/article/pii/S0966692320309480.

[19] Md Amirul Islam et al. *Position, Padding and Predictions: A Deeper Look at Position Information in CNNs*. 2021. DOI: 10.48550/ARXIV.2101.12322. URL: https://arxiv.org/abs/2101.12322.

[20] Zhixiong Jin et al. *Transformer-based Map Matching Model with Limited Ground-Truth Data using Transfer-Learning Approach*. 2021. DOI: 10.48550/ARXIV.2108.00439. URL: https://arxiv.org/abs/2108.00439.

[21] Panagiota Katsikouli et al. "Characterizing and Removing Oscillations in Mobile Phone Location Data". In: *WoWMoM 2019 - 20th IEEE International symposium on a World of Wireless, Mobile and Multimedia Networks*. Washington, United States: IEEE, June 2019, pp. 1–10. URL: https://hal.archives-ouvertes.fr/hal-03173080.

[22] Taewon Lee and Sukho Lee. "OMT: Overlap Minimizing Top-down Bulk Loading Algorithm for R-tree". In: *CAiSE Short Paper Proceedings*. 2003.

[23] Artjom Lind, Amnir Hadachi, and Oleg Batrashev. “A new approach for mobile positioning using the CDR data of cellular networks”. In: *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. 2017, pp. 315–320. DOI: 10.1109/MTITS.2017.8005687.

[24] Q. Liu et al. “Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts”. In: *AAAI*. 2016.

[25] Krista Merry and Pete Bettinger. “Smartphone GPS accuracy study in an urban environment”. In: *PLOS ONE* 14.7 (July 2019), pp. 1–19. DOI: 10.1371/journal.pone.0219890. URL: https://doi.org/10.1371/journal.pone.0219890.

[26] Saul B. Needleman and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins.” In: *Journal of molecular biology* 48 3 (1970), pp. 443–53.

[27] Paul Newson and John Krumm. “Hidden Markov map matching through noise and sparseness”. In: Jan. 2009, pp. 336–343. DOI: 10.1145/1653771.1653818.

[28] Ana-Maria Olteanu Raimond et al. “Moving and Calling: Mobile Phone Data Quality Measurements and Spatiotemporal Uncertainty in Human Mobility Studies”. In: vol. 2013. May 2013. ISBN: 978-3-319-00614-7. DOI: 10.1007/978-3-319-00615-4_14,.

[29] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017.

[30] Mark Padgham et al. “osmdata”. In: *The Journal of Open Source Software* 2.14 (June 2017). DOI: 10.21105/joss.00305. URL: https://doi.org/10.21105/joss.00305.

[31] Ankur Pandey, Ryan Sequeira, and Sudhir Kumar. “Joint Localization and Radio Map Generation using Transformer Networks with Limited RSS Samples”. In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2021, pp. 1–6. DOI: 10.1109/ICCWorkshops50388.2021.9473553.

[32] Kishore Papineni et al. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

[33] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[34] Thomas Lin Pedersen. *tidygraph: A Tidy API for Graph Manipulation*. R package version 1.2.1. 2022. URL: https://CRAN.R-project.org/package=tidygraph.

[35] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. DOI: 10.48550/ARXIV.1710.05941. URL: https://arxiv.org/abs/1710.05941.

[36] Jinmeng Rao et al. *LSTM-TrajGAN: A Deep Learning Approach to Trajectory Privacy Protection*. 2020. DOI: 10.48550/ARXIV.2006.10521. URL: https://arxiv.org/abs/2006.10521.

[37] Christian Strobl. "Dimensionally Extended Nine-Intersection Model (DE-9IM)". In: Jan. 2008, pp. 240–245. ISBN: 978-0-387-30858-6. DOI: 10.1007/978-0-387-35973-1_298.

[38] Jie Sun and Jiwon Kim. "Joint prediction of next location and travel time from urban vehicle trajectories using long short-term memory neural networks". In: *Transportation Research Part C: Emerging Technologies* 128 (2021), p. 103114. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103114. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21001339.

[39] Peize Sun et al. *TransTrack: Multiple-Object Tracking with Transformer*. Dec. 2020.

[40] Hugo Touvron et al. *Training data-efficient image transformers amp; distillation through attention*. 2020. DOI: 10.48550/ARXIV.2012.12877. URL: https://arxiv.org/abs/2012.12877.

[41] Toivo Vajakas et al. "Mobility Episode Discovery in the Mobile Networks Based on Enhanced Switching Kalman Filter". In: Nov. 2018. DOI: 10.1109/ICUMT.2018.8631264.

[42] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[43] Christopher White, David Bernstein, and Alain Kornhauser. "Some Map Matching Algorithms for Personal Navigation Assistants". In: *Transportation Research Part C: Emerging Technologies* 8 (Feb. 2000), pp. 91–108. DOI: 10.1016/S0968-090X(00)00026-7.

[44] Aston Zhang et al. *Dive into Deep Learning*. 2021. DOI: 10.48550/ARXIV.2106.11342. URL: https://arxiv.org/abs/2106.11342.

[45] Kai Zhao et al. "DeepMM: Deep Learning Based Map Matching with Data Augmentation". In: Nov. 2019, pp. 452–455. ISBN: 978-1-4503-6909-1. DOI: 10.1145/3347146.3359090.

[46] Xizhou Zhu et al. “Deformable DETR: Deformable Transformers for End-to-End Object Detection”. In: *ArXiv* abs/2010.04159 (2021).

Appendix

I. Reconstruction example with overlapping cells

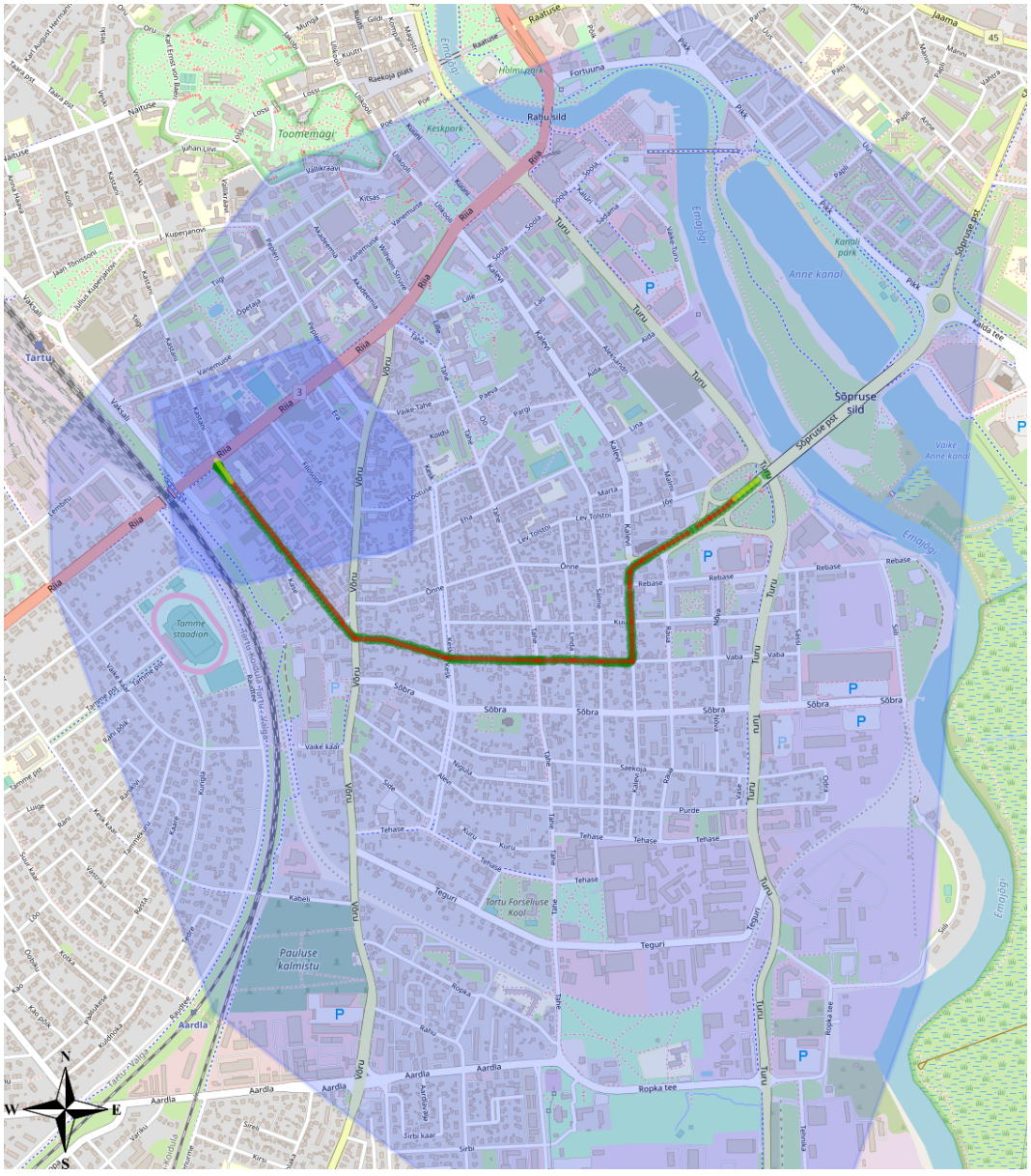


Figure 24. Example of trajectory reconstruction after consecutive rule filtering. Red lines are predicted roads, yellow lines are predicted start and end roads, green points are the underlying GPS trace and two blue areas are the starting and ending cell coverage area.

II. Reconstruction example with error

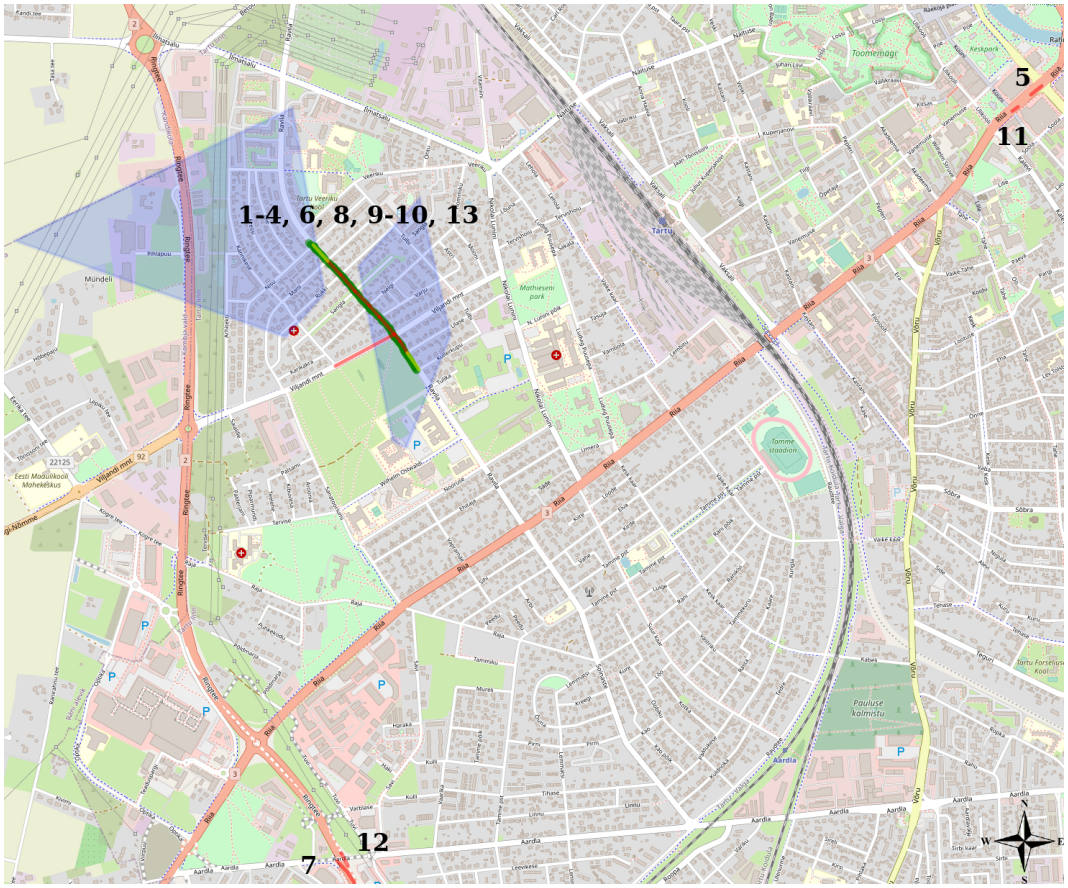


Figure 25. Example of trajectory reconstruction result with errors and with a segment-level route length of 13. Numbers represent the order of the road ID values in the output sequence. 1 is the starting road, 13 is the final road. Red lines are predicted roads, yellow lines are predicted start and end roads, green points are the underlying GPS trace and two the blue polygons represent coverage area for starting and ending cell.

III. GPS point density on road network for two selected validation folds

Figure 26. Road network and distribution of GPS points for test split 1 (left) and 6 (right).

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Oliver Bollverk**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   CDR-Based Trajectory Reconstruction Using Transformers,
   *(*title of thesis)

   supervised by Amnir Hadachi, PhD.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Oliver Bollverk
08/08/2022