DTU Compute
Department of Applied Mathematics and Computer Science

Masked Location Prediction using Transformers

Master Thesis

Masked Location Prediction using Transformers

Master Thesis
February, 2024

By
David Immanuel Hartel
MSc in Enginnering; Human-Centered Arificial Intelligence,
Technical University of Denmark

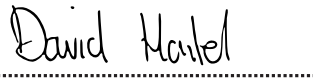
Cover photo: Privacy-aware hexbin-histogram visualisation of the raw data, 2024
Published by: DTU, Department of Applied Mathematics and Computer Science, Asmussens Alle, Build. 322, 2800 Kgs. Lyngby Denmark
www.compute.dtu.dk

Approval

This thesis has been prepared over six months, from August 28th 2023 to February 28th 2024, at the Section for Cognitive Systems, Department of Applied Mathematics and Computer Science, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc Eng.

The supervisor of this project was Laura Alessandretti, Assistant Professor at the Department of Applied Mathematics and Computer Science DTU.

David Immanuel Hartel - s212588



Signature

February 28, 2024

Date

Abstract

Understanding, modeling, and predicting human mobility in urban areas is an essential task for various domains such as transportation modelling, disaster risk management, and infectious disease spreading. In this thesis, we introduce a custom BERT-based model, MOBERT, designed for human mobility modelling. Trained and tested on a processed dataset derived from the Copenhagen Network Study, which captures location data from smartphones of 840 individuals, MOBERT effectively predicts masked locations, outperforming three baseline models and achieving superior results compared to existing-next location prediction approaches. Our analysis of the impact of additional features, including user ID, time, and location rank, on prediction accuracy showed no significant improvements. Furthermore, we evaluated the model's performance across different grid sizes and location types, emphasizing its proficiency in learning individual-specific regular patterns while highlighting challenges with explorative mobility patterns. Further limitations include the difficulty in comparing our results with other studies due to the different datasets used across studies and our bidirectional approach, which differs from the one-directional next location prediction task. We suggest future research focus on further parameter optimization, employing more comprehensive datasets, and fine-tuning to a next location prediction task. Furthermore, enhancing privacy while maintaining utility remains a critical area for future exploration.

Acknowledgements

Laura Alessandretti, Assistant Professor, DTU
First and foremost I would like to thank my supervisor Laura Alessandretti for the opportunity to work with her on this thesis. I am very grateful for the weekly meetings that nearly always ended in overtime. They were a great help and support for my thesis, both motivational and professionally.

Marco De Nadai, Data Scientist, Research Scientist
Special thanks also go to Marco De Nadai for his guidance and help with the BERT model.

Friends & Family
My heartfelt thanks to my friends at Regensen and DTU, as well as to my family, for their emotional support and care.

Contents

Glossary

attention mechanism a technique in neural networks that allows the model to focus on different parts of the input sequence when producing an output, improving the ability to remember long-range dependencies. 7, 9, 38–41, 44, 61

bidirectional encoder representations from transformers a transformer-based machine learning technique for natural language processing pre-training, designed to understand the context of words by looking at the words that come before and after in a text. 35, 43

convolutional neural network a class of neural networks, inspired by the human visual cortex. They are characterized by their use of convolutional layers neurons that are responsive to specific areas of the visual field. 6, 8, 40

decoder a neural network component that converts the abstract representation produced by the encoder back into a more detailed or specific format. 8–10, 37, 41–43

deep learning an area within machine learning that employs neural networks to learn representations of data with multiple levels of abstraction. 4, 6, 8, 11, 12, 36, 72

embeddings dense representations of words, phrases, or other elements in a continuous vector space, facilitating the capture of context and relationships among terms. 36–38, 44, 48

encoder a component of a neural network that transforms input data into a more abstract, high-level representation. 8–10, 37, 41–46, 48, 61

features individual measurable properties or characteristics of a phenomenon being observed, used as input variables for machine learning models. 2, 6, 9, 11, 12, 21, 22, 24, 27–29, 40, 49, 50, 52–55, 59, 61, 62, 64, 67, 69–72

fully connected network neural network where each neuron in one layer is connected to all neurons in the next layer, allowing for complex transformations of the input data. 6, 8

gated recurrent unit a type of RNN with gating mechanisms to control the flow of information, designed to solve the vanishing gradient problem of standard RNNs. 6, 7, 9

long short-term memory an extension of RNNs designed to make them capable of learning long-term dependencies. 6

machine learning a subset of artificial intelligence focusing on the development of algorithms that can draw inferences from patterns in data make without following explicit instructions. 4–7, 11, 12, 27, 35

mobility optimised bidirectional encoder representations from transformers our adaptation of BERT designed to enhance the prediction of mobility patterns by leveraging bidirectional context in mobility data. 49, 53

multi-head attention an extension of the attention mechanism that allows the model to jointly attend to information from different representation subspaces at different positions. 40, 41, 43, 48

natural language processing a field of artificial intelligence that focuses on the interaction between computers and humans through natural language, aiming to enable computers to understand, interpret, and generate human language. 5, 6, 47

next location prediction the task of predicting the future location of an individual based on their current and past locations. 1, 2, 4, 5, 8–12, 72

positional embedding vectors that represent the position of tokens within a sequence, used in models like transformers to add information about the order of tokens. 37, 38, 44, 48, 52, 53

recurrent neural network a class of neural networks where connections between nodes form a directed graph along a temporal sequence, enabling them to exhibit temporal dynamic behavior. 6, 8

samples individual instances or observations from the data set used for training, testing, or validation of the model. Each sample consists of one or more features. 21, 32, 33, 62, 63

special tokens tokens with specific meanings used in natural language processing models, such as tokens indicating the start or end of a sentence, or separating segments. 46, 47, 49, 52

transformer a deep learning model architecture designed for handling sequential data, notable for its use of self-attention mechanisms, allowing it to weigh the importance of different parts of the input data differently. 7, 9, 42, 60, 64

1 Introduction

The urban population is rapidly growing, leading to increasingly complex and extensive human mobility. This has significant effects on critical aspects of life, such as the spread of infectious diseases, people's responses to natural disasters and disaster management, the dynamics of public and private transport leading to traffic, citizen mental health and well-being, and environmental impacts like air pollution, energy, and water use. Additionally, movements of crowds between cities are often prompted by rural-to-urban migrations driven by factors like natural disasters, climate change, and conflicts. Hence understanding human mobility is essential for addressing societal challenges like disease spread urban development, mental health, and pollution.

The rise of computing technologies, including mobile phones, the Internet of Things, and social media, offers urban planners and policy makers an accurate and constantly updated means to monitor human movements across different times and places. Mobility data comes nowaday in various forms, for example as GPS device tracks in smartphones [1, 2] or vehicles [3, 4], phone network communications [5], and geotagged social media posts [6].

Modelling human mobility is import in a variety of areas. Predicting the mobility trajectories of vehicles can greatly improve traffic flow and reduce congestion. By anticipating where traffic build-up will occur, city planners and traffic management systems can redirect vehicles in real-time, optimizing routes and reducing travel times [7].

The character of natural disasters has significantly changed over the past decades since its frequency and intensity increased. In emergency situations, predicting the movement of people and potential hazards can save lives. For instance, during natural disasters, predictive analytics can inform evacuation strategies, guiding people away from danger zones [8]. Human mobility modelling can assist patient care through better resource allocation and emergency services. For example, anticipating the areas with higher demands for medical services can help in strategically positioning ambulances and medical personnel, reducing response times in critical situations [9].

Mobility trajectory prediction can further significantly assist in monitoring public health. By understanding and patterns of social interaction and movement, interventions can be designed to enhance mental health and well-being [10, 11]. By predicting the movements of individuals within and between communities, public health officials can better monitor and control the spread of infectious diseases [12, 13]. Beyond commercial applications, next location prediction can enhance personalization in services ranging from entertainment to education, delivering content and experiences tailored to an individual's context and preferences [14].

The recent advancemens of Artificial Intelligence coupled with the availability of extensive mobility data has opened new doors for tackling complex challenges such as next location prediction. Deep Leanring techniques surpass classical machine learning techniques in several key aspects. Deep learning models are able to learn from vast amounts of data and are able to recognize patterns and nuances in the data that are often too subtle or complex for traditional algorithms.

1.1 Scope

In this thesis, introduce MOBERT, an adapted BERT model for the purpose of human mobility prediction. We utilize a dataset consisting of 245 million data points from 840 individual user trajectories collected over a two-year period from September 2013 to August 2014. Our objective is to enable the model to learn human mobility patterns in a manner analogous to its proven capability to learn human language.

First, we explore and analyze the dataset to identify the data necessary for mobility modeling. We preprocess and clean it to achieve condensed, high-quality data suitable for training the model. This involves simplifying the location coordinates by using a custom-sized coordinate plane with square-shaped grid cells.

Second, we adapt the original BERT model architecture to process human mobility trajectories and incorporate additional features such as time, user ID, and location popularity rank.

Third, we create a framework for testing the model's performance with respect to various combinations of input features and different grid granularities.

Finally, we analyze the results to determine how the features and grid granularities influence the model's performance and assess how effectively the model learns locations.

1.2 Outline

The thesis is structured into the following chapters, along with an appendix.

- **Chapter 1 - Introduction**: This chapter introduces the topic and outlines the scope of the thesis.
- **Chapter 2 - Background**: Provides a discussion of the relevant literature, offering an overview of the different existing approaches to generating and predicting human mobility.
- **Chapter 3 - Predictability in Human Mobility**: Discusses the level of accuracy we can expect from a model when predicting locations.
- **Chapter 4 - Data Set**: Details about the dataset utilized in this thesis are provided here. It also describes the necessary data preprocessing and cleaning steps prior to model training.

- **Chapter 5 - Methods**: Reviews the principles and architecture of Transformers and BERT models.
- **Chapter 6 - MOBERT**: Describes the practical part of the thesis, including the implementation of the model.
- **Chapter 7 - Experimental Setup**: Explains the setup used to evaluate the model's performance.
- **Chapter 8 - Results**: Describes the results achieved.
- **Chapter 9 - Discussion and Conclusion**: Summarizes the main findings of this thesis.
- **Appendix**: Contains the implementation, figures, and additional information.

2 Background

In this section, we provide an overview of the existing research and deep learning methodologies in the field of human mobility. In section 2.1, we briefly introduce the notation used throughout this thesis. Then, in section 2.2, we explore both traditional machine learning techniques and the more recent advancements in deep learning related to location predicion. Finally, in section 2.2.6, we will examine the level of accuracy that can be expected based on perfomance of existing approaches for next location prediction.

2.1 Notation

We follow *A Survey on Deep Learning for Human Mobility* [15] when stating the following definitions:

Location [15]: Let x, y be spatial coordinates in a given reference system, e.g. latitude and longitude. Then we call $l = (x, y)$ a location.

Spatio-temporal points [15]: A spatio-temporal point is a pair $p = (t, l)$, where t indicates the time when point $l = (x, y)$ is visited.

Semantic spatio-tempopal point [15]: A semantic spatio-temporal point p is a tuple $p = (o, t, l)$, where t indicates the time when point $l = (x, y)$ is visited by u, I is a pair of coordinates (x, y), and o is a parameter that brings some context to the point (e.g. home workplace, or some other categories), if any.

Trajectory [15]: Let u be anindividual,a trajectory $T_u = (l_1, l_2, ..., l_{n_u})$ is a time-ordered sequence composed by n_u locations visited by u.

Next location prediction [15]: Next location prediction consists of forecasting the next location (stay point) that an individual will visit based on their previous mobility data. Let u be a user, T_u their trajectory, and $p_t \in T_u$ be u's present position, next location prediction seeks to forecast u's next destination p_{t+1}. This problem may be treated in two ways:

1. As a multi-class classification task, in which we have as many classes as locations and we aim at predicting teh next visited location p_{t+1}
2. As a regression task, predicting $p_{t+1} = (x_{t+1}, y_{t+1})$ where x_{t+1} and y_{t+1} are the next location's geographic coordinates.

Masked location prediction Let u be a user and T_u their trajectory, where p_t is an unknown, masked location and $(p_1, \ldots, p_{t-1})$ and $(p_{t+1}, \ldots, p_n)$ are known locations. Masked location prediction seeks to forecast the user u's masked position p_t, based on the previous and subsequent locations. Similar to next location prediction, this task can be treated either as regression or classification task.

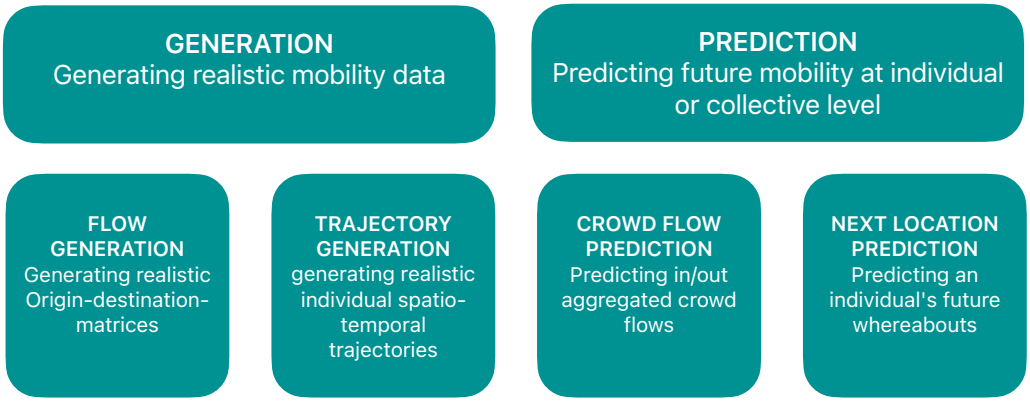


Figure 2.1: A taxonomy of mobility tasks as described in [15]

In this thesis, we treat the problem as multi-class classification task.

Generative Model [15]: "A generative model for human mobility M is any algorithm capable of generating a collection of n synthetic trajectories $T_M = \{T_{a_1}, \ldots, T_{a_n}\}$, which describe the motions of n independent agents $a_1, \ldots, a_n$ over a given time period."

2.2 Literature Review

In this section we discuss existing DL solutions to next location prediction.

To locate next location prediction in the research landscape, we again refer to *A Survey on Deep Learning for Human Mobility* [15]. There, Luca et al. differentiate between two kinds of mobility tasks related to machine learning in human mobility: **prediction**, which aims at forecasting future mobility trajectories at an individual or collective level, and **generation**, which aims at creating realistic mobility flows (see figure 2.1). The predictive tasks include two categories: Next location prediction and crowd flow prediction. On the generative side, trajectory generation aims to create realistic synthetic human mobility patterns, and flow generation involves creating realistic flows between locations without prior knowledge of actual flows. Although classical machine learning methods have been successful in these areas [16, 17], the adoption of DL techniques has been driven by their ability to extract patterns from unstructured data and their proven success in other domains like computer vision and natural language processing.

In the following we discuss how DL improves classical machine learning approaches.

2.2.1 Comparing Traditional Machine Learning Approaches and Deep Learning

Earlier methods for next location prediction often relied on probabilistic models that incorporate various factors influencing human mobility. For example, Calabrese et al. [18]

proposed a model that uses people's trajectories combined with geographical features like land use, points of interest (POIs), and trip distances. These models aim to calculate the likelihood of moving to a certain location based on observed patterns and contextual information. Ashbrook et al. [19] utilized GPS data clustered into meaningful locations, using these clusters as states in a Markov model to predict future movements. Similarly, Gambs et al. [20] introduced a Mobility Markov Chain (MMC), where states represent POIs, and transitions indicate movements between them. These approaches are grounded in the assumption that future locations depend on recent history, essentially treating mobility prediction as a state-transition problem.

However, a significant challenge with traditional methods is, according to Luca et al. [15], the need for extensive feature engineering, which is both labor-intensive and requires deep domain knowledge. This process involves manually selecting and transforming input data into a format suitable for modeling, which can overlook potentially valuable features. Further, traditional models often struggle to capture long-range dependencies in time and space, as well as the complexity of sequential patterns and additional influences on mobility (like weather conditions, individual preferences, or social factors).

DL methodologies, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, gated recurrent units (GRUs), fully connected networks (FCs), attention mechanisms, and convolutional neural networks (CNNs) have the advantage of learning complex and non-linear representations from data without explicit feature engineering. Those DL models are better suited for learning the multifaceted nature of human mobility and are addressing the limitations of traditional approaches. They can learn from data patterns that represent both routine behaviors and deviations from those routines, as well as the influence of external factors and individual preferences.

In the following section, we discuss the development of the most important deep learning architectures that originally were developed for applications within NLP and computer vision.

2.2.2 Deep Learning and Transformers

Transformers have revolutionized the field of machine learning, particularly in natural language processing (NLP) since their introduction in the paper "Attention is All You Need" [21]. Prior to transformers, recurrent neural networks [22] and convolutional neural networks dominated the landscape of sequence modelling tasks.

Recurrent Neural Networks (RNNs)

RNNs are a type of neural network designed for handling seqences, using past outputs as inputs while keeping hidden states. They are capable of processing inputs of any lenght without altering the model's size. Furthermore, they incorporate both historical data and weights that have remained constant over time in their calculations. RNNs have shown proficency in tasks such as text generation, machine translation and time-series analysis.

However, RNNs posses inherent limitations, particularly when it comes to their sequential architecture. The process of back-propagation, which is essential for training many machine learning algoithms, can become less effective in RNNs due to vanishing or exploding gradients. Furthermore, the backprogation can be computationally expensive and time-consume due to the inherernt sequential computation process that hinders parralelization. This makes it especially difficult for RNNs to process long sequences and learn long-term dependencies.

Long Short-Term Memory (LSTMs), and Gated Recurrent Units (GRUs)

To address these challenges, various solutions have been proposed. One solution is the implementation of LSTMs, which are structured with multiple units, each equipped with input, output, and forget gates. This design allows each unit to genereate a state that influences the subsequent input [23].

Bidirectional LSTMs were developed to enhance the network's understanding in both forward and reverse directions, although their capability to manage long sequence dependencies comes at the cost of increased training time due to numerous parameters. As a more efficient alternative, gated recurrent units (GRUs) were introduced [24]. GRUs simplify the structure to just two gates, leading to faster performance. In some cases, GRUs outperform LSTMs, especially in identifying grammatical aspects of texts [**g**].

Overall, the architectures of RNNs, including LSTMs and GRUs, face several issues: slow computation due to a lack of parallelization, a lack of capacity to consider future inputs for current state determinations, and difficulties in accessing long-term information.

The introduction of the transformer model architecture marked a paradigm shift and significantly advanced the quality of the research on DL and NLP. Unlike traditional networks, transformer models allow for the parallel processing of entire sequences. this is made possible by the self-attention mechanism [25], which assesses the importance of different parts of the input data relative to each other, irrespecitve of their opsition in the sequence. Further, the attention mechanism solves the inability of handling long sequences by computing the relationships between all parts of the input sequence, regardless of their distance from each other. The information about the relationship between two elements are stored as key-value pairs in long-time memory and can be retrieved by creating a query. Hence the model is able to look up information that it has already seen. This ensures that even elements far apart within the sequence can influece each other's representation.

Another innovation is the use of positional encoding to inject the model information about the position of each element in the sequence, since the transformer architecture does not inherently understand the order of the sequence. This ensures that the model maintain an awareness of sequence order, which is crucial for tasks like language understanding.

The impact of transformer models on machine learning has been profound. Applied on a variety of NLP tasks, models with attention mechanism outperfomed traditional RNN or CNN models [26, 27]. But they have not only set new standards of performance on a

wide range of NLP tasks as for example machine translation [28], text summarization [29] and question-answering [30], but have also been adapted for applications outside of NLP, such as computer vision [31] and reinforcement learning [32].

2.2.3 Deep Learning Approaches for Next Location Prediction

There are various deep learning (DL) strategies for next location prediction that have continuously evolved alongside state-of-the-art technologies. In the following, we provide an overview of existing DL approaches, structured by their main architectural components. Table 2.1 provides finally an overview over the described approaches.

Fully Connected Networks

De Brébisson et al. [33] employ a fully connected network (FC) to predict taxi passenger drop-off locations, utilizing a mix of GPS trajectory data and metadata such as departure time and driver identity. The model's performance is evaluated using the equirectangular distance metric on the Porto taxi dataset [34].

CNNs

T-Conv treats trajectories as images, using convolutional neural networks (CNNs) to capture spatial patterns. Its predictions are also evaluated on the Porto taxi dataset [34] but with the Haversine distance.

GRUs

DeepJMT [35] predicts both the next POI and its visitation time by integrating multiple context extractors (spatial, periodicity, and social-temporal) with hierarchical GRUs. It is evaluated on Foursquare check-ins across various cities and showcases the importance of a multi-faceted approach to understanding mobility.

RNNs

ST-RNN [36] incorporates time- and space-specific transition matrices into recurrent neural networks (RNNs), allowing for the consideration of both spatial and temporal dimensions in mobility prediction. Yang et al. [37] use an RNN and a unique flashback technique to predict the next POI based on sparse semantic trajectories. It is evaluated on a data set from Gowalla [38] and the Global Terrorism Dataset [39].

LSTMs

HST-LSTM [40] combines LSTM with an encoder-decoder module to predict short-term locations, focusing on periodic patterns and spatial-temporal characteristics. It's evaluated on Baidu's private data and demonstrates effectiveness in capturing short-term mobility behaviors. [41] leverage in 2019 an LSTM network with a self-attention module for taxi drop-off location prediction. It is tested on datasets from Porto [34], New York City [42], and San Francisco [2] using the Haversine distance. SERM [43] focuses on predicting the next Point of Interest (POI) using semantic trajectories, embedding social media data

into an LSTM. It is evaluated on Foursquare and Twitter data from New York City [44] and Los Angeles [45]. BiLSTM-CNN [46] combines bi-directional LSTMs and CNNs to predict the next area of interest from POIs. It is evaluated on a priveate Weibo check-in dataset from Wuhan, China. CLNN [47] integrates an LSTM with two FC networks to predict taxi destinations, merging coordinate, date, time, and POI information. It's evaluated on the Porto taxi dataset [34] using the mean Haversine distance.

GANs

The LSTM-TrajGAN model [48] combines LSTM and GANs to create synthetic trajectory data that protect privacy without sacrificing the data's spatial, temporal, and thematic integrity. It introduces a TrajLoss metric to ensure user anonymity. Evaluated on a real semantic trajectory dataset, it aims to balance privacy and utility.

Attention Mechanisms

VANext [49] employs variational attention and causal encoders to model semantic relationships among POIs, combining CNN and GRU outputs through an attention mechanism. DeepMove [50] introduces an attentional recurrent network to predict mobility from sparse trajectories. It utilizes a multi-modal embedding to integrate spatio-temporal and individual-specific features, employing an attention mechanism for historical trajectories and a gated recurrent unit (GRU) for current trajectories. Abideen et al. [51] adopt with DWSTTN a transformer-like architecture for predicting taxi destinations, using encoder-decoder structures to process historical and current taxi data. It is evaluated on datasets from Porto [34] and New York [42] with the Haversine distance and highlights the power of attention mechanisms in capturing spatial-temporal dependencies. Corrias et al. [52] investigate GPT- and GCN-based models for next location prediction, adapting them from general time series forecasting frameworks, and testing them on two sparse (check-in-based) and one dense (continuous GPS-based) datasets.

2.2.4 Evaluation Metrics

The performance of trajectory predictions is quantitatively measured by comparing different metrics between the real dataset and the generated dataset. Luca et al. [15] provide an overview over the most important measures used for next location-prediction:

If the prediction is modelled as **regression task**, the divergence between these distributions is often calculated using statistical measures such as the Kullback-Leibler (KL) divergence or the Jensen-Shannon (JS) divergence. More detailed information about those metrics can be found in the appendix B.

If the model is designed to solve a **multi-class classification task**, the predctions are often evaluated with accuracy, recall, ROC-curve, F1-score or the Mean Average Percentage Error and Area Under the Curve (AUC). In the follwing we define evaluation metrics for regression and multi-class classification.

Reference	Name	Year	DL Modules	Evaluation
Corrias et al.	-	2023	Encoder, Decoder, Attention	ACC@k
Abideen et al.[1]	DWSTTN	2021	Encoder, Decoder, Attention, FC	Distance
Tang et al.[182]	CLNN	2021	LSTM, Embedding, FC	Distance
Bao et al.[8]	BiLSTM-CNN	2020	Embedding, BiLSTM, CNN	ACC@k
Chen et al.[33]	DeepJMT	2020	GRU, FC, Encoder	ACC@k
Yang et al.[212]	Flashback	2020	Attention, RNN	ACC@k
Ebel et al.[49]	-	2020	RNN, FC, Embedding	Distance
Rossi et al.[153]	-	2019	Attention, LSTM	Distance
Gao et al.[64]	VANext	2019	CNN, GRU, Attention	ACC@k
Kong et al.[99]	HST-LSTM	2018	LSTM	ACC
Lv et al.[118]	T-CONV	2018	CNN, FC	Distance
Feng et al.[54]	DeepMove	2018	Attention, GRU, FC	ACC
Yao et al.[214]	SERM	2017	LSTM	ACC@k
Liu et al.[114]	ST-RNN	2016	RNN	Rec@k, F1@k, MAPE, AUC
De Brébisson et al.[44]	-	2015	FC	Distance

Table 2.1: Following Luca et al. [15], the table gives an overview over DL apporaches for next location prediction. Earlier models utilized RNN, CNN, and FC approaches, while more recent research focuses on encoder-decoder architectures. There are not only different DL modules and evaluation methods but also various datasets and experiment setups, as we will see in section 2.2.6.

Since our model is about to solve a multi-class classification task, we will follow Luca et al. [15] to provide a more comprehensive overview over the according metrics:

Evaluation Metrics for Multi-Class Classification

The accuracy (**ACC**) measures the proportion of locations an individual will visit that are correctly identified by the model. Instead of using ACC alone, k-accuracy (**ACC@k**) is often preferred: this involves the model generating a list of k potential next locations for an individual, ordered by likelihood. ACC@k then represents how often the actual location appears within the top k predictions made by the model, essentially showing the percentage of times the top k list includes the true location.

Precision assesses the model's accuracy specifically for the positive class, while Recall calculates the True Positive Rate (**TPR**), which is the ratio of positive instances that were accurately identified. The **F1-score** provides a single metric that describes a model's performance by taking the harmonic mean of Precision and Recall. Precision, Recall, and F1-Score are derived from the counts of True Positives (TPs), False Positives (FPs), and False Negatives (FNs).

These metrics offer a way to quantify the similarity between the empirical data and the model's output, essentially evaluating how well the generated trajectories mirror the complexity and variability of real movements.

2.2.5 Overview of Datasets

Most datasets are publicly available, originating either from location-based online social networks such as Foursquare or Gowalla, or consisting of GPS locations collected through smartphones or vehicles, such as taxis. Although GPS data from cell phones or taxis offer a more consistent temporal resolution, check-in data from location-based social networks provide more precise location details. This precision encompasses information about the purpose of the visit, yielding insights more specific than the broader location information obtained from cell phone tower data. According to Luca et al. [15], the most commonly used datasets for next location prediction include those from Cho et al. [38] and Feng et al. [50], which consist of check-ins from social networks, and taxi traces collected in Porto, Portugal [2], and San Francisco, California [34].

2.2.6 Comparative Analysis of Prediction Results

The existing approaches to predicting locations vary fundamentally, as they utilize different spatial and temporal features. Similarly, the evaluation metrics also vary. Furthermore, the datasets used in these studies can differ significantly, as described in Section 2.2.5. This variability makes it challenging to compare the results directly and to define based on other studies an expected accuracy for our model. However, to provide the reader with an idea of possible prediction outcomes, we will present some results from approaches that all use the ACC@k measure:

Reference	Name	Year	Evaluation	Dataset	ACC@10	ACC@5	ACC@1
Corrias et al. [52]	-	2023	ACC@k	[38]	0.3572	0.2710	0.1076
Bao et al. [46]	BiLSTM-CNN	2020	ACC@k	private	-	0.4370	0.2840
Chen et al. [35]	DeepJMT	2020	ACC@k	[53]	0.4924	0.4122	-
Yang et al. [37]	Flashback	2020	ACC@k	[38]	0.3479	0.2754	0.1158
Gao et al. [49]	VANext	2019	ACC@k	[38]	0.4745	0.4103	0.2321
Kong et al. [40]	HST-LSTM	2018	ACC@k	private	0.4677	0.3711	0.1574

However, these results should be interpreted cautiously, as factors like spatial resolution and temporal density may significantly diverge from our methodology. Moreover, our approach involves predicting randomly masked locations within trajectories, whereas the bidirectional nature of our model allows it to account for both preceding and subsequent locations.

2.3 Summary

In this chapter, we provided an overview of the existing research and methodologies in deep learning in human mobility. We began by introducing the notation used throughout the thesis, including definitions related to locations, trajectories, and next location prediction. Then, we explored traditional machine learning techniques and recent advancements in deep learning as discussed in the existing literature. We highlighted the

evolution from traditional methods to deep learning, emphasizing the advantages of deep learning in capturing complex patterns and dependencies in human mobility data without the need for extensive feature engineering. Deep learning models leveraged architectures like RNNs, LSTMs, CNNs, and attention mechanisms to extract meaningful representations from data.

We discussed evaluation metrics commonly used for next location prediction, such as k-accuracy, precision, recall and F1-score. Further, we mentioned datasets used in mobility science, including publicly available datasets from location-based online social networks and GPS traces from smartphones or vehicles.

Further, we compared prediction results from various approaches, highlighting the challenges in directly comparing results due to differences in spatial and temporal features, evaluation metrics, and datasets.

Overall, we provided an overview of the research landscape in next location prediction, highlighting the transition from traditional machine learning to deep learning approaches and discussing the key methodologies, evaluation metrics, and datasets used in the field.

3 Predictability of Human Mobility

In this chapter, we aim to discuss the level of accuracy we can expect from a model when predicting human mobility trajectories. Although each individual may have very unique movement patterns, there are common patterns shared by all humans. Typically, a person has no more than one location where they live and spend the night, and likely only a few locations that are regularly visited in their daily life, such as their workplace, school, and the local grocery store. These recurring and targeted movement patterns, that we will call patterns of exploitation, are evidently more predictable than the explorative movement patterns that emerge when visiting new places for the first time.

3.1 Background

3.1.1 Defining an Upper Bound for Predictability

In [54], Song et al. investigate the degree to which human mobility behavior is predictable. They explore the limits of predictability by studying the mobility patterns of anonymized phone users. By measuring the temporal entropy of each individual's trajectory and using Fano's inequality, they define an upper bound of predictability. Surprisingly, for a majority of users, this upper bound is very high, at 93%.

In *A Refined Limit on the Predictability of Human Mobility* [55], Lin et al. study the effects of spatial and temporal resolution on the predictability limit. They arrive at a similarly high upper bound as in [54] for hourly temporal resolution and a spatial resolution where the size of each location is roughly that of a large building. However, they also conclude that temporal and spatial scales are correlated with the predictability limit. This means that predictability can decrease as the spatial resolution becomes finer and the temporal resolution wider. Furthermore, they conclude that this is independent of the overall mobility area covered, suggesting that mobility trajectories are highly regular. Perhaps even more importantly, the invariance in predictability observed by varying the scales suggests "that certain trade-offs between predicting accuracy and spatial-temporal resolution are unavoidable." [55, p.381]

Smith et al. [56] consider spatial reachability constraints and show that the upper bound might be between 11-24% less than claimed in [54]. They conclude similarly, that the predictability is highly correlated to the spatial quantisation and temporal resolution.

3.1.2 Entropy Measures for Mobility Patterns

Entropy, in the context of information theory, is a measure of the average level of unpredictability or randomness of a time series. The intuition behind the concept is to measure how much surprise there is in an event: A high entropy corresponds to low predictabil-

ity and a low entropy to high predictability. When a user's entropy S equals 0, it means their movements are entirely systematic, making their location entirely predictable. On the other hand, if a user's entropy equals the random entropy $S_{rand} = \log_2 N$, their movements are completely random and only predictable with a probability of $\frac{1}{N}$. The entropy of most users falls somewhere between these two extremes, suggesting their movements are influenced by both randomness and some degree of predictability.

Entropy

We follow [57, p. 51] when defining entropy in information theory:

Let X be a discrete random variable, which takes values in the alphabet $\mathcal{X}$ and is distributed according to $p : \mathcal{X} \to [0, 1]$. For a single event $x \in \mathcal{X}$ the quantity of information inherent in the event is determined using the likelihood that the event is occuring. This concept is referred to as "Shannon information", "self-information", or more generally as "information" and is defined by:

$$\textit{information}(x) = -log(p(x))$$

Depending on the application, different bases for the *log* can be chosen. Here, $log()$ denotes the logarithm to base 2, and $p(x)$ represents the probability of the occurrence of event x. Using a base-2 logarithm implies that the information is quantified in binary digits (bits). This can be understood as it corresponds to the number of binary digits needed to encode the event. The inclusion of a negative sign in the formula ensures that the information value is greater or equal than zero. It equals zero, when the probability of an event is 1. For example, an event x with probability $p(x) = 0.5$ will result in $\textit{information}(x) = -log(p(x)) = 1$ bits.

Now the intuitve description for entropy is the average number of bits required to represent an event drawn from random variable. More formally, entropy can be defined as:

$$H(X) := -\sum_{x \in \mathcal{X}} p(x) log(p(x)),$$

In the analysis of mobility patterns, entropy can be used to quantify the degree of predictability in an individual's movements. Three specific measures of entropy were defined in [54]. They range from the simplistic Random Entropy to the more complex Actual Entropy, offering different levels of insight into the structure and randomness of movement patterns.

Random Entropy (S_{rand})[54]:

Random entropy represents the maximum entropy state, assuming each location is visited with equal probability. It is defined as:

$$S_{\text{rand}} \equiv \log_2 N_i \tag{3.1}$$

where N_i is the number of distinct locations visited by the user i,

Temporal-Uncorrelated Entropy (S_{unc})[54]:

Temporal-Uncorrelated Entropy accounts for the historical probability of visiting each location but does not consider the order of visits. It is given by:

$$S_{\text{unc}} \equiv -\sum_{j=1}^{N_i} p_i(j) \log_2 p_i(j) \tag{3.2}$$

where $p_i(j)$ is the historical probability of user i visiting location j.

Actual Entropy (S_i)[54]:

Actual Entropy is the most comprehensive measure, accounting for both the frequency and the order of location visits. It is calculated as:

$$S_i = -\sum_{T_i' \subset T_i} P(T_i') \log_2[P(T_i')] \tag{3.3}$$

where $P(T_i')$ is the probability of observing a specific time-ordered subsequence T_i' in the trajectory $T_i = \{l_1, l_2, \ldots, l_L\}$.

For each user, it holds naturally that $0 \leq S_i \leq S_{\text{unc}} \leq S_{\text{rand}} < 1$ [54].

The computation of the actual entropy S_i requires the analysis of all possible time-ordered subsequences within a user's trajectory. For a trajectory of length n, there are $2^n - 1$ non-empty subsequence. This number grows exponentially with the length of the trajectory, leading to a combinatorial explosion even for moderately sized trajectory (a trajectory of length 30 has one billion subsequences). For each subsequence the probability $P(T_i')$ of its occurence within the overal trajectory T_i must be calculated. This involves counting the occurences of each subsequence within the larger sequence. Further, any of these subsequences will overlap with each other, requiring the algorithm to repeatedly calculate probabilities for similar patterns. Because of the exponential runtime, the calculation of the Actual Entropy is computationally extremly costly.

Lempel-Ziv Estimator

The Lempel-Ziv compression [58] offers a more computationally efficient estimator:

For a trajectory of length n, the actual entropy is estimated by

$$S_i \approx S^{est} = (\frac{1}{n} \sum_i \Lambda_i)^{-1} \ln(n)$$

where Λ_i is the length of the shortest substring starting at position i which doesn't previously appear from position 1 to $i-1$. This estimation is known to rapidly converge to the actual entropy S_i [54].

The key reasons for this efficiency is, besides the nature of the approximation, also its algorithmic design. The Lempel-Ziv estimator focuses on identifying new, previously unseen subsequences as it iterates through the series instead of calculating the probability

of each subsequence. Further, it iterates through the trajectory only once and is identifying the shortest new subsequence that hasn't appeared before at each step. This iterative process is much more efficient, as it inherently avoids redundant calculations for subsequences that have already been encountered.

3.1.3 Entropy as an Estimate for Predictability

This section aims to determine how well we can predict a user's next location based on the history of their movements. Specifically, we want to understand the relationship between a user's entropy and their next location's predictability. To do this, we'll follow [54] and apply a form of Fano's inequality to derive the maximum predictability based on the user's mobility history.

Notation: Let $h_{n-1} = \{l_{n-1}, l_{n-2}, \ldots, l_1\}$ denote a user's past history from time interval 1 to $n-1$, where l_i corresponds to the user's location at time step i. Let $\mathsf{Pr}[l_n = \hat{l}_n | h_{n-1}]$ be the probability that our guess $\hat{l}_n$ for a user's next location agrees with his actual next location l_n given his location history h_{n-1}. Let $\pi(h_{n-1})$ be the probability the user will be in his most likely next location l_{ML} given his history h_{n-1}. Thus

$$\pi(h_{n-1}) = \sup_x \mathsf{Pr}[l_n = x | h_{n-1}],$$

where $\mathsf{Pr}[l_n = x | h_{n-1}]$ is the probability that the next location l_n is x given the history h_{n-1}. That is, $\pi(h_{n-1})$ contains the full predictive power including the potential long-range correlations present in the data.

Predictability Definition [54]:

For a given trajectory h_{n-1} of length $n-1$, let $P(h_{n-1})$ be the probability of observing the trajectory. Then, the predictability is given by

$$\Pi(n) \equiv \sum_{h_{n-1}} P(h_{n-1}) \pi(h_{n-1}) \tag{3.4}$$

where the sum is taken over all possible histories of lenth $n-1$. The overall predictability Π is then given by the limit

$$\Pi \equiv lim_{n \to \infty} \frac{1}{n} \sum_{i}^{n} \Pi(n)). \tag{3.5}$$

Using Jensen's inequality, we can relate the actual entropy S defined in Equation 3.3 to the predictability Π [54]. Furthermore, by employing Fano's inequality, we can establish an upper bound Π_{max} of Π, given by the solution to the equation:

$$S_i = -\Pi_{\mathsf{max}} \log_2(\Pi_{\mathsf{max}}) - (1 - \Pi_{\mathsf{max}}) \log_2(1 - \Pi_{\mathsf{max}}) + (1 - \Pi_{\mathsf{max}}) \log_2(N-1) \tag{3.6}$$

where N the is number of unique locations.

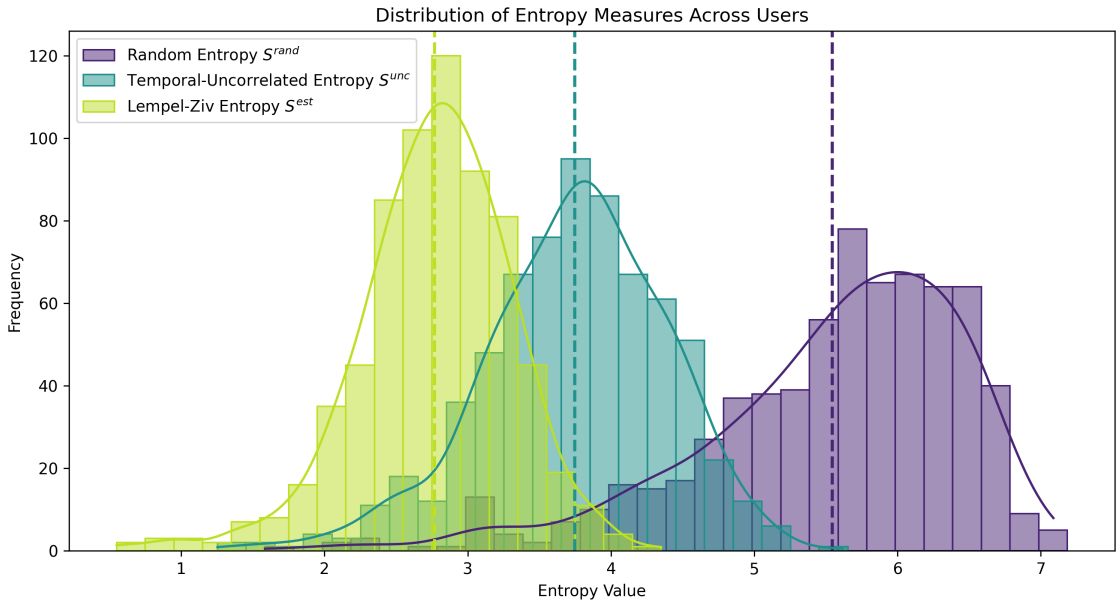


Figure 3.1: This plot confirms that the entropy distribution follows the expected pattern, $0 \leq S_{\text{est}} \leq S_{\text{unc}} \leq S_{\text{rand}} < 1$. It further demonstrates that the Random Entropy exhibits the widest distribution, suggesting significant variability across users in the number of unique locations visited, more so than in the locations most likely to be visited.

3.2 Computing Entropy and Predictability in the Dataset

To characterize the inherent uncertainty and randomness for our processed data (see chapter 4), we will use the equations presented above to compute the various entropies and derive the maximum predictability for each user in our preprocessed dataset.

Entropy Measures

In contrast to the actual entropy, the random entropy and temporal-uncorrelated antropy are computationally effortless. The code can be found in the appendix A. The obtained probability distributions of the three entropies are shown in figure 3.1. Similar to the results in the orginal article [54], the shifts of the distributions follow the expected pattern $0 \leq S_{est} \leq S_{\text{unc}} \leq S_{\text{rand}} < 1$. Notable is, similarly, the shift between the distribution of S_{rand} and S_{est}. The distribution of S_{rand} peaks at $S_{\text{rand}} \approx 6$. That suggests that, on average, each update to a user's location introduces six bits of new information; in other words, a user making random choices for their next location could be predicted to be in one of approximately $2^{S_{\text{rand}}} \approx 64$ possible locations. However, the peak of the distribution of S_{est} at $S_{\text{est}} = 2.8$ reveals that the actual uncertainty in a typical user's location is not 64 but closer to $2^{2.8} \approx 7$, meaning less around seven locations are likely. However, the results from the original article are even more surprising. There, the distribution peaks at $S = 0.8$, indicating that "the real uncertainty in a typical user's whereabouts is not 64 but $2^{0.8} \approx 1.74$, i.e. fewer than two locations" [54].

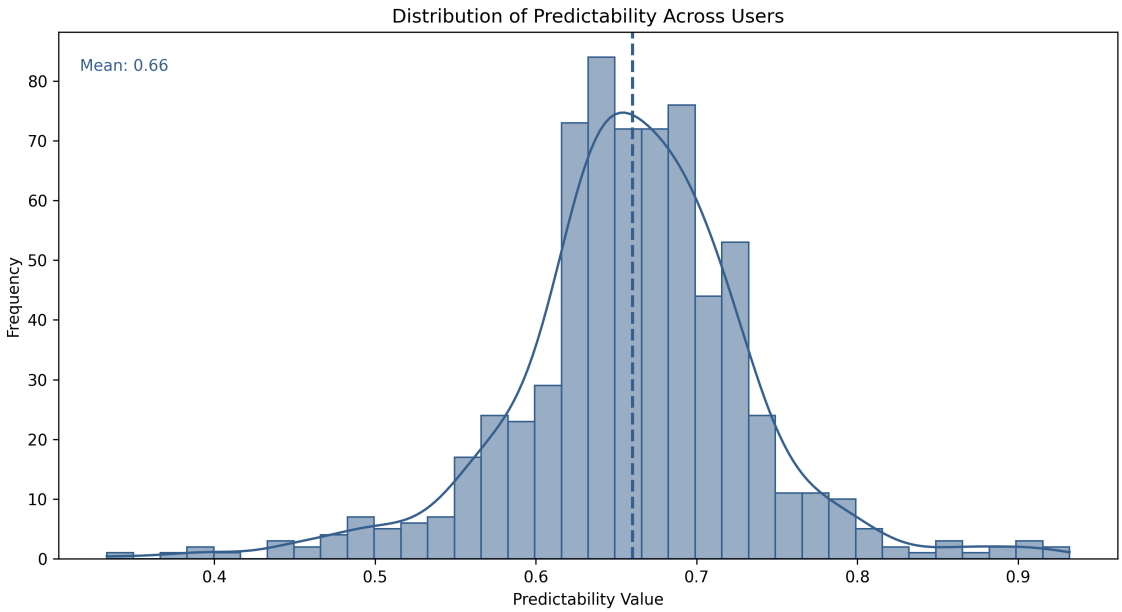


Figure 3.2: The figure shows that predictability appears to be normally distributed across all users, ranging from around 30% to over 90%, with an average of 66%.

An Upper Bound for Predictability

We calculated Π_{max} individually by solving equation 3.1.3 numerically using the Lempel-Ziv estimator $S_i \approx S^{\text{est}}$. We found, as shown in Figure 3.2, that the distribution of $\Pi_{\text{max}})$ exhibits a fat-tailed pattern, where most users have predictability between 55% and 75%, with a mean of 66%.

This implies that for an ”average” user with $\Pi_{\text{max}} = 0.66$, in at least 34% of instances, the user selected their location in a manner that seemed random, while in the remaining 66% of instances, their location could be predicted from previous movement patterns. In other words, regardless of our model's performance, we cannot achieve more than 66% accuracy in predicting the future locations of a user with $\Pi_{\text{max}} = 0.66$. Thus, Π_{max} establishes the upper bound on predictability for the user.

It is somewhat surprising that we achieve such a high value, despite the seemingly random nature of individuals' movements. However, the mean value is still significantly lower than the 93% potential predictability that Song et al. achieved [54].

Lastly, we must also note that this predictability has been computed based on past trajectories and refers to the prediction of the next location. Our BERT model, however, processes the trajectory bidirectionally, having access to both the future and the past. Hence, the upper bound should be taken with a grain of salt and cannot serve as a strict upper boundary.

3.3 Summary

In this chapter, we discussed the predictability of human mobility. We highlighted common movement patterns shared by all humans, such as regular visits to specific locations like home, work, and grocery stores. These recurring patterns, termed ”patterns of exploitation,” are typically more predictable than explorative movements to new places.

We reviewed studies that investigated the predictability of human mobility. Researchers like Song et al. [54] and Lin et al. [55] explored the limits of predictability by analyzing the mobility patterns of phone users, finding surprisingly high upper bounds for predictability. However, factors like spatial resolution and temporal scale were shown to affect predictability, with predictability decreasing as spatial resolution increased and temporal resolution decreased.

Entropy measures were discussed as a way to quantify the predictability of mobility patterns. Random entropy, temporal-uncorrelated entropy, and actual entropy were defined, with actual entropy being the most comprehensive but computationally costly to compute. We introduced the Lempel-Ziv estimator as a more efficient method for estimating actual entropy.

Finally, we computed entropy and predictability for our dataset, finding distributions that aligned with expectations based on previous research. The upper bound for predictability, derived using Fano’s inequality, suggested a mean predictability of 66%, indicating that in 66% of instances, future locations could be predicted from past movement patterns. However, we noted that this upper bound should be interpreted cautiously, especially considering MOBERTs bidirectional way of processing of trajectories.

4 Data Set

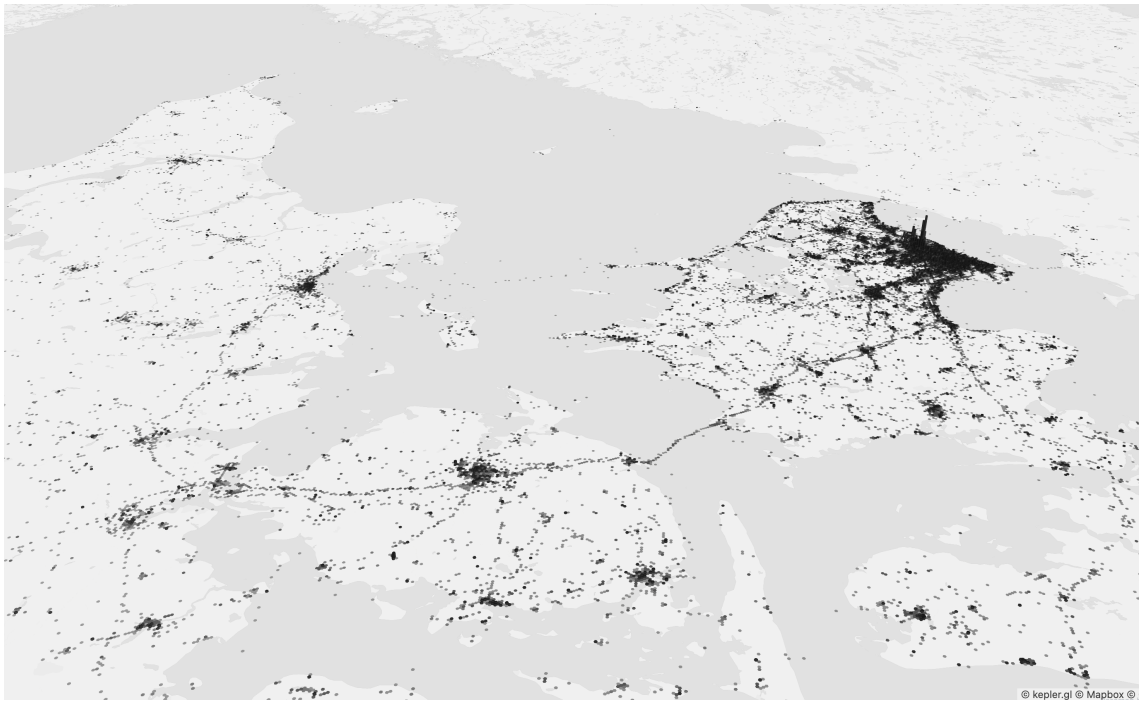


Figure 4.1: This histogram map illustrates the distribution of raw data across Denmark, with locations randomly yet uniformly selected and aggregated into hexagon-shaped bins. The height of each bin along the z-axis reflects the frequency of visits to the area. The highest concentration of data points occurs at the DTU campus, as indicated by the spike north of Copenhagen. Additionally, the distribution of locations highlights major Danish cities such as Roskilde, Odense, and Aarhus, as well as mobility corridors, including highways and train routes. It should be noted that the density decreases with distance from the Copenhagen area.

4.1 Understanding the Raw Data

4.1.1 Data Collection

The data used in this thesis stems from the *Copenhagen network study*, a large scale study designed to measure human interactions across a variety of communicaiton channels [59]. With high temporal resolution and spanning multiple years, data was collected using state-of-the-art smartphones. Although the data included also face-to-face interactions, telecommunication, social networks and background information such as personality, demographics, health and politics was collected, we only use a single channel - the location data - from the study. According to Sapiezynski et al., the location data was obtained by ”periodically collecting the best position estimate from the location sensor on

each phone, as well as recording location updates triggered by other applications running on the phone (opportunistic behavior)" [59, p.15]. Almost 90% of the samples have a reported accuracy better than 40 meters.

Figure 4.2: Visualisation of the trips that are included in the raw data set: The participants were not only traveling within Sealand, Denmark or Europe but also across the whole world.

4.1.2 Data Features

The location data comprises trajectories from 840 individuals, measured over two years between September 2013 and August 2015. In total, the dataset contains 245,304,797 data points. Hereafter, we refer to the individuals as users. Five rows of the raw data might look like the synthetic data example in the table below:

	user	timestamp	latitude	longitude	accuracy
0	12	1.386241e+09	55.783641	12.518414	30.689
1	12	1.386242e+09	55.783691	12.518467	12.11
2	12	1.386245e+09	55.782622	12.521572	6.59
3	12	1.386246e+09	55.782607	12.521314	3.51
4	12	1.386247e+09	55.782632	12.521542	25.67

In the following, we want to take a closer look at the different features.

User

This feature represents a unique user ID that distinguishes each individual within the dataset. A total of 840 distinct users are identified, each with their unique records. However, as shown in Figure 4.3, not all 840 users continuously shared their location throughout the entire duration of the original study. A common characteristic among these users is their enrollment as students at the Technical University of Denmark (DTU), making it highly likely for them to visit the DTU campus in Lyngby. On average, each user has 292,029 location records.

Timestamp

This feature records the moment a location is logged, utilizing Unix timestamp format. A Unix timestamp denotes the total count of seconds that have passed since the start of the

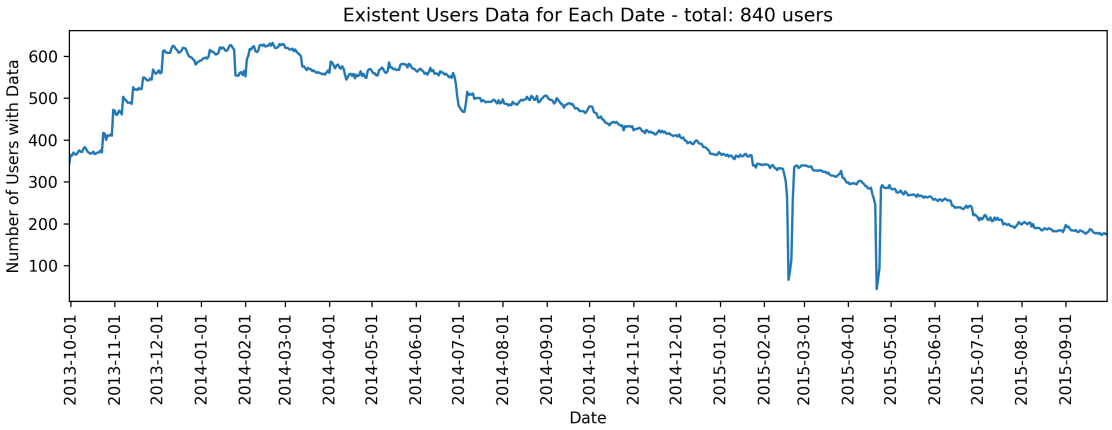


Figure 4.3: Number of users per date: The figure displays the count of users with at least one positional record on the corresponding date. It reveals that the number of users peaks between January and Summer 2014, then declines thereafter. Several holidays and two server outages contribute to valleys in the number of active users.

Unix epoch at 00:00:00 UTC on January 1, 1970 (excluding leap seconds) [60]. While the raw timestamps do not reveal the frequency of location logging, the interval between two consecutive records can vary significantly. Notably, every fifth record is observed to have a timestamp identical to one previously recorded. This could result from users turning off their phones, disabling GPS functionality, or losing GPS signal. Conversely, intervals between records can also be very large. On average, the time difference between two records is approximately $1e^{-7}$ seconds.

Latitude and Longitude

These features specify the geographical coordinates of a location. Latitude indicates the position north or south of the equator, measured in degrees, while longitude represents the position east or west of the Prime Meridian, also in degrees. Together, these coordinates provide a precise global positioning for each recorded location. In the context of our dataset, these coordinates are the essence that allows for learning the spatial patterns. Figure 4.1 shows that the location records are spread across the whole world.

The granularity of the location data is illustrated in figure 4.4, particularly through records captured on the highway on the right side of the image, where locations are logged at intervals of 5 to 10 meters apart. This demonstrates the precision of timestamp record in capturing detailed movement patterns.

Accuracy

The accuracy feature, measured in meters, represents the confidence radius around the recorded latitude and longitude coordinates, indicating where the user's true position is likely located. Despite the average accuracy radius being approximately 32 meters, this feature will not be an important part of our data. In subsequent preprocessing steps,

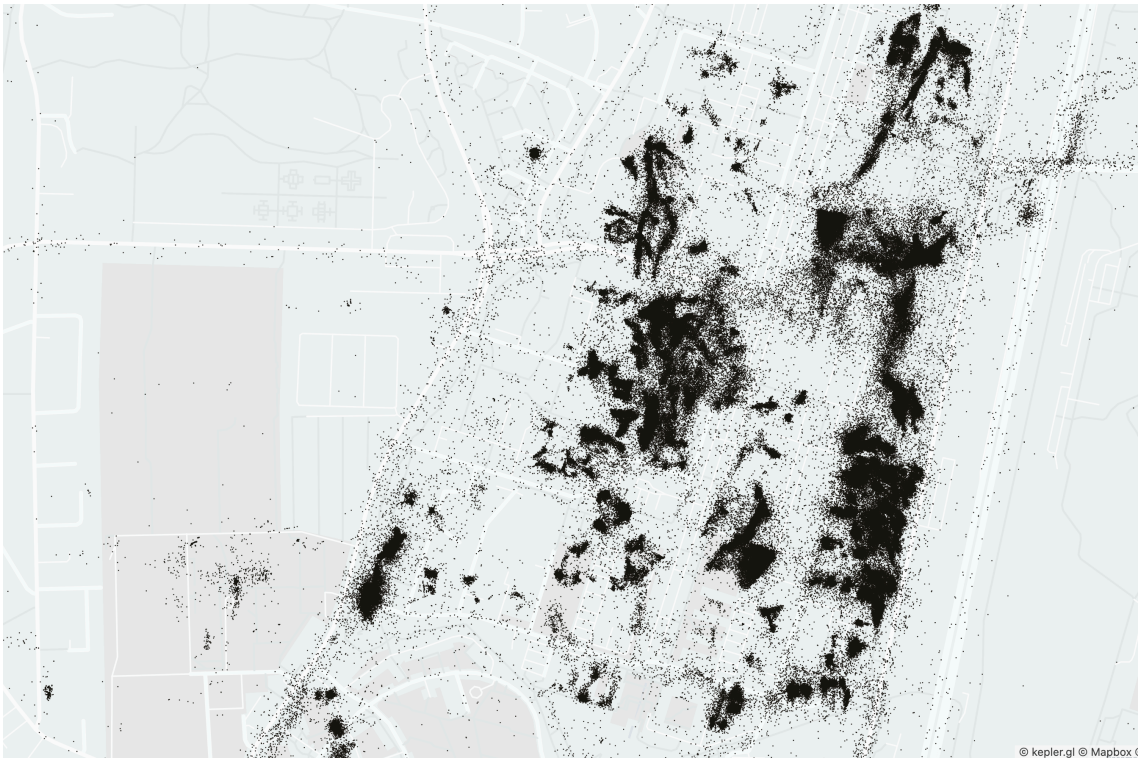


Figure 4.4: This map displays randomly selected location records from all users on the DTU campus in Lyngby. The density of these records varies significantly across the campus, with aggregations indicating different buildings. However, the accuracy is not sufficient to determine the exact room or venue where the user was located. Note, that streets do not show significantly higher density than adjacent areas, confirming that the accuracy is not precise to the meter but rather has a confidence radius radius of multiple meters.

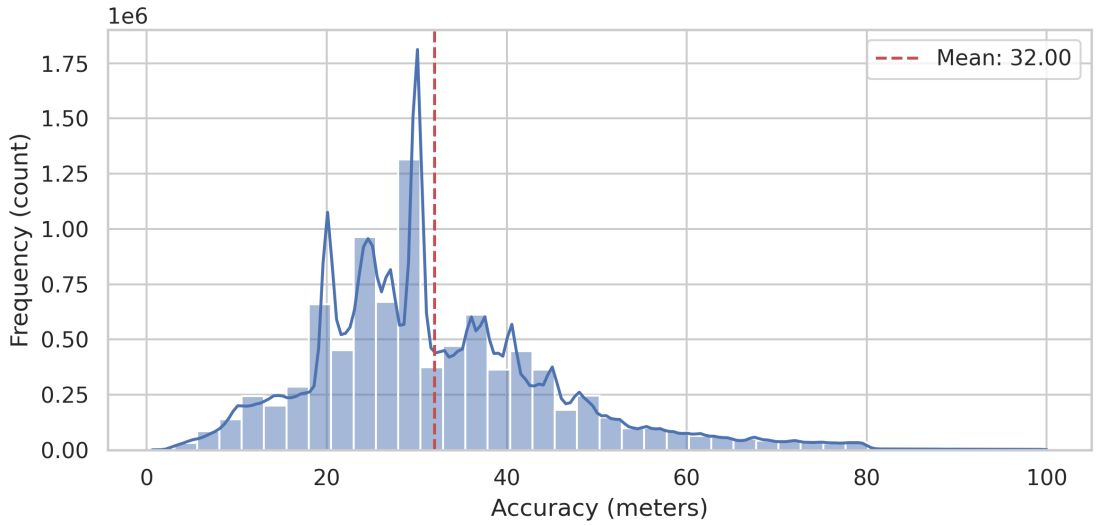


Figure 4.5: Distribution of values in the accuracy feature. The histogram indicates that the accuracy values are somewhat normally distributed, with an average of 32 meters.

we will aggregate location records into larger spatial units, effectively averaging over the precise locations.

4.1.3 Challenges with the Raw Location Data

The raw location data presents several challenges that we need to adress to effectively model and predict human mobility trajectories in urban areas:

High Density of Temporal and Spatial Resolution

The data features highly detailed timestamps, capturing movements with precision down to seconds or milliseconds. Similarly, the spatial resolution of the data is extremely detailed, providing latitude and longitude coordinates to centimeter accuracy. This level of detail, although valuable for certain analyses, can complicate the task of modeling broader mobility patterns, as it necessitates managing an enormous volume of spatial information and may obscure larger trends in human movement.

Irrelevant Location Records

Data captured while individuals are in transit, such as during car, bus, or train rides, presents another layer of complexity. These records often exhibit rapid changes in location over short periods, making it challenging to distinguish between meaningful stops and the trajectory path itself. The inclusion of location records outside the Copenhagen area, introduces further irrelevant data. Since our focus is on human mobility within an urban context, records of movements outside the greater copenhagen area can detract from the accuracy and relevance of the predictive modeling. These records could add noise into the model and potentially skew the predictions.

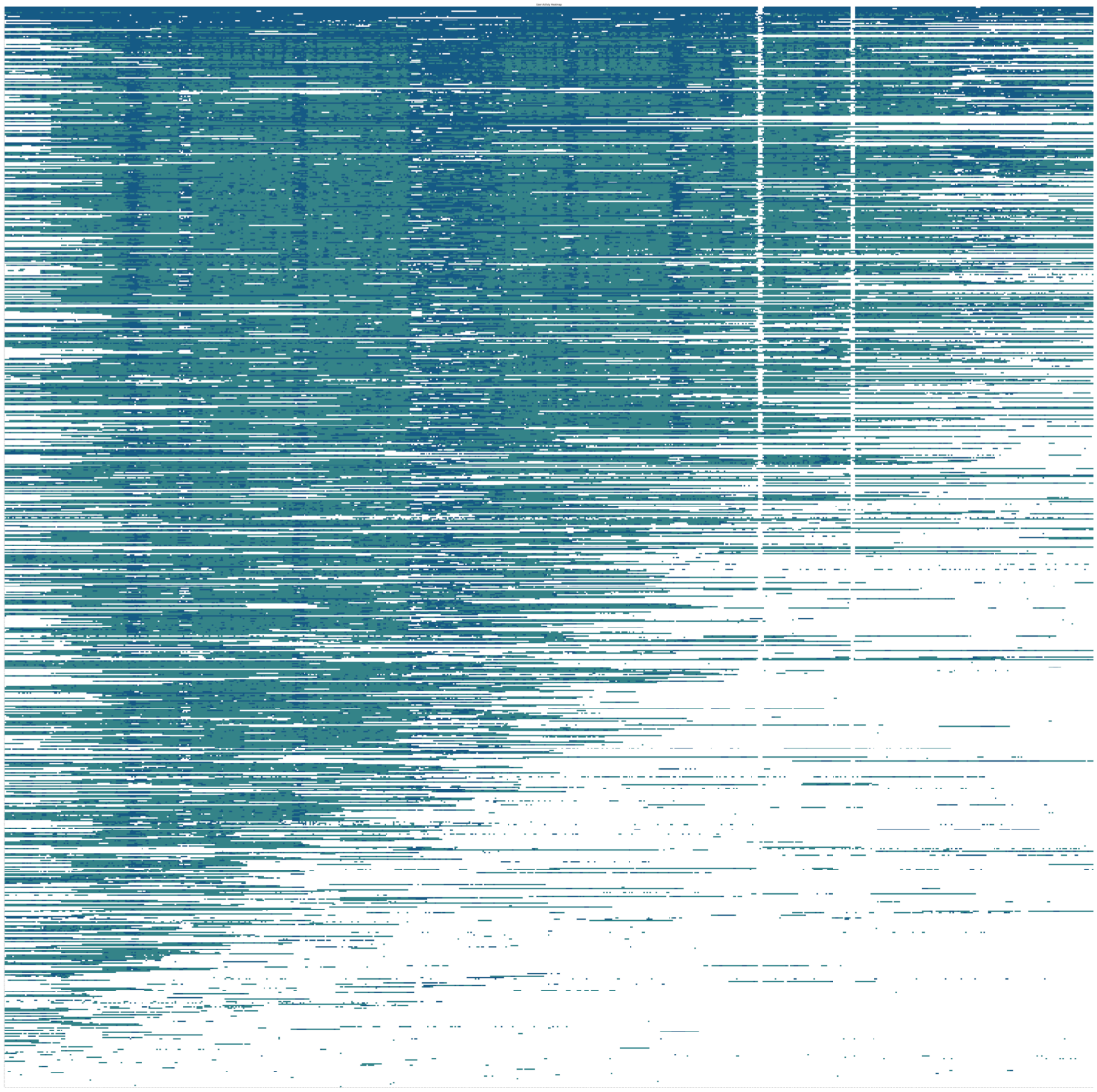


Figure 4.6: **Distribution of Data Over Time**: Each row represents one of the 840 users, sorted by the amount of active days, and each column represents one day between September 30, 2013, and September 30, 2015, totaling 730 days. A plot cell appears white if there are no location records from the user on that day, dark green if the majority of location records fall within the defined grid encompassing the Greater Copenhagen area (see subsection 4.2.2), and blue if the majority of location records are outside the grid. The figure reveals a decrease in the number of active participants over the years. Additionally, it highlights clear temporal patterns, as marked by horizontal stripes, which indicate periods when users moved outside of the Copenhagen area.

Missing Data

Our dataset comprises real-world data, which is often unstructured, messy, and susceptible to containing errors from multiple sources. Figure 4.6 provides a realistic view of the sparsity and incompleteness inherent in the raw, unprocessed dataset. Apart from two server outages, all observed data gaps can be attributed to the participants. This

illustrates that a significant majority of the 840 study participants did not share their location data consistently over the two-year study period. Some participants stopped sharing their data after a few months, while others only began sharing one or two months into the study. Additionally, gaps in the data may result from participants not using their phones for several days or weeks, turning them off during weekends or holidays. There are observable patterns in the times when users were outside the predefined grid. On average, weekends were more commonly used for leaving the city area than weekdays. Notably, common Danish school holidays manifest as vertical stripes in Figure 4.6. For instance, on the left side of the figure, we can readily identify periods such as the time between Christmas and New Year in 2013, the Danish winter holidays from 22 January to 4 February 2014, the Easter week from 10 April to 23 April 2014, and the summer months of June and July 2014.

We analyzed every 30-minute interval for the entire timespan across 840 users, labeling each interval based on the presence and nature of the data points it contained. Intervals without a location record were labeled as ”missing.” Intervals where all recorded points were outside the Copenhagen area were labeled as ”outside,” and intervals where the stop-location algorithm (see section 4.2.1) identified records solely as locations during a trip were labeled as ”moving.”

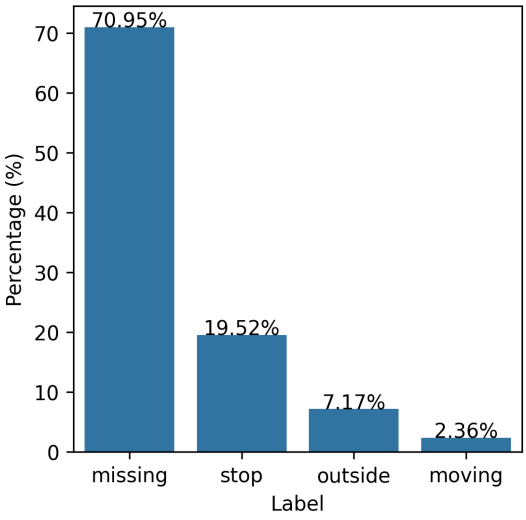


Figure 4.7: Distribution of data quality across all users and 30-minute intervals over the entire timespan. The only important data records for our project are the one labeled as ”stop”.

Figure 4.7 shows that approximately 71% of the intervals are missing. Furthermore, the ”interesting” stop location data which accounts for around ”20%” of the total is hidden by a large volume of qualitatively uninteresting location records—either outside of Copenhagen or recorded during travel. However, it is crucial to contextualize these figures; a high number of missing intervals does not necessarily detract from the data quality for our predictive task. For instance, phones turned off during the night may result in many missing 30-minute intervals but do not bias the trajectory analysis in any significant way. Similarly, users who stopped contributing data early in the study significantly impact the volume of missing data, yet the data they did contribute is of high quality.

Even though data gaps are clearly visible on an individual level, the distribution of the data across days and hours is very balanced. Figure 4.8 shows that there are clearly patterns that indicate a decreased amount of records during the nighttime and weekends. Since

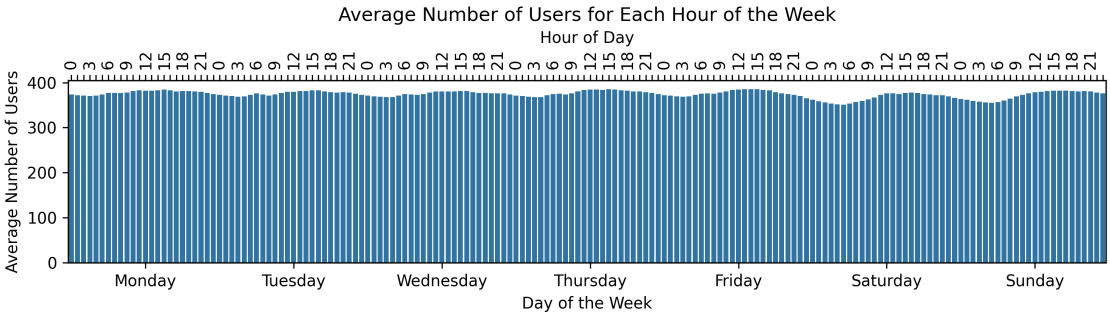


Figure 4.8: This figure depicts the average number of users that logged at least one location record per weekday and hour.

the deviations is comparably very small, we can assume that we have consistent data throughout the days and hours

In the following section, we describe the preprocessing techniques that we used to solve the abovementioned problems.

4.2 Data Preprocessing

Data preprocessing is a crucial step in the machine learning process. It may involvevarious processes like cleaning, transforming or augmenting the data in order to prepare it for the traning. Our overal goal of the data preprocessing is to adress the three problems of our raw data:

1. Very dense temporal and spatial resolution
2. Location records that are not relevant for location predition since they were recorded during a trip or far away from the city area
3. Missing data across all users

In terms of the training process, we want to improve the quality of the data and the perfomance of the model, accelerate the training time, and make the predictions more understandable.

The preprocessing of our data involves the following steps:

1. Data transformation: Based on the location features we use an algorithm to detect stop locations and add them to the data.
2. Data Reduction: We simplify the data by translating the positions into a discrete grid and the collect the data in 30-minute intervalls.
3. Data cleaning: We remove part trajectories of low quality and consecutive duplicate locations.

All changes decrease the size of the dataset while preserving the important information

and even increasing the quality. In the following we will discuss the single preprocessing steps that were necessary to prepare the data for the model training.

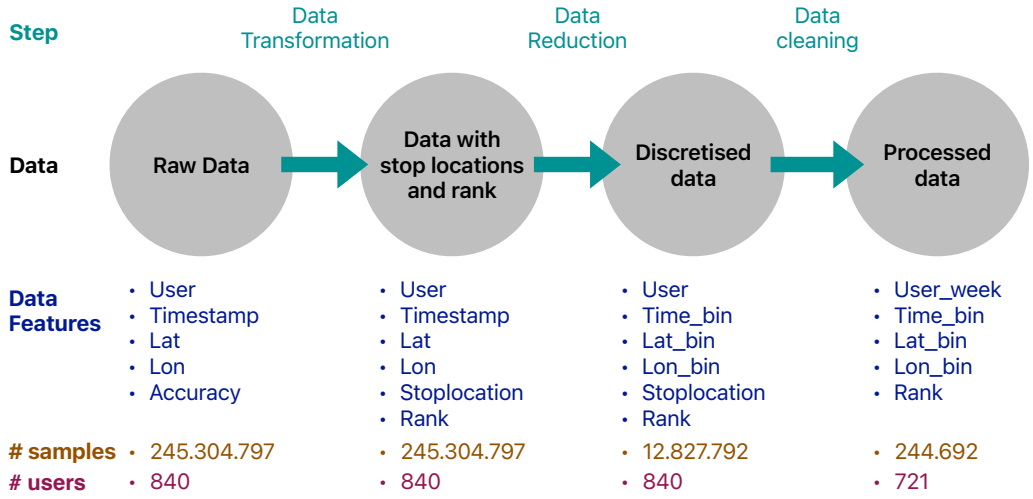


Figure 4.9: The diagram describes visually the steps involved in the preprocessing pipeline. The raw data is first enriched by calculating the stop location and rank for every user. Second it is reduced and discretized based on the spatial and temporal features. Third it is cleaned to ensure a high data quality.

4.2.1 Step 1: Data Transformation

The GPS signals were captured approximately every 5 minutes. This implies that a user trajectory contains data points that do not necessarily represent a stationary location where the user spent time. Consequently, the dataset also contains data points captured during travel from location A to location B, for example, during a train or bike ride. Data points that were captured during travel are not relevant for the model. Hence, the first step involved determining the locations where the user stopped and spent a certain minimum amount of time, as well as the locations where the user was in transit. We refer to such a location as a stop location in the subsequent sections.
We used the algorithm implemented in [61] (Infostop: Scalable Stop-Location Detection in Multi-User Mobility Data) which detects stop locations from raw sequences of spatio-temporal data. The algorithm operates in two steps: First, it assigns a label of either trip, stay to each data point, and then it clusters a collection of spatial points using the clustering algorithm for networks, Infomap [62].

We ran the stop location detection algorithm with the default parameters, notable among which are:

- `r1=10` - This parameter represents the maximum distance (in meters) between time-consecutive data points required to label them as stationary. If the distance between

two consecutive points is 10 units or less, they are considered to be stationary

- `r2=10` - Similarly, this parameter denotes the maximum distance between stationary points to form an edge between them
- `label_singleton=True` - This flag indicates that even if a stationary location was visited only once, it should be assigned its own unique label
- `min_staying_time=300` - This parameter sets the minimum duration, in seconds, that a stop at a location must last to be considered a stop. In our case, a stop must last at least 300 seconds (or 5 minutes)
- `max_time_between=86400` - This parameter sets the maximum duration, in seconds, that can pass for a stop at a location to be considered a single stop. Here, 86400 seconds (or 24 hours) is the maximum duration for a stop
- `min_size=2` - The minimum size of a group to consider it stationary
- `min_spatial_resolution=0` - The minimal difference allowed between points before they are considered the same points
- `distance_metric="haversine"` - Either 'haversine' (for geo data) or 'euclidean'

The algorithm yields a dataset that additionally contains a column assigning each data point a stop location as a positive number, and "-1" if it is a trip.

4.2.2 Step 2: Data Reduction

To address the challenges of high temporal and spatial density data and to improve the model's performance, we reduce the complexity of the data in two steps: First, we transform the continuous temporal feature, timestamp, into discrete intervals with a fixed length of 30 minutes. Second, we limit the continuous spatial features, latitude and longitude, by selecting only a smaller area and discretizing them by dividing the area into grid cells. Limiting the area has the advantage of allowing us to exclude mobility patterns characterized by exploration, such as holiday destinations and random walks through foreign cities, and focus on those characterized by exploitation, such as the known places individuals consistently choose in their everyday life. Discretizing the spatial data simplifies the representation of spatial information, reduces noise and makes making it easier to store and process the data.

Timestamp: We discretize the timestamp by splitting it into year, month, day, hour, and 30-minute intervals. For each 30-minute interval, the stop location is determined as the most frequently occurring stop location in the corresponding interval. The location itself is calculated by averaging the coordinates that belong to the most frequent stop location (see drawing in figure 4.10. That means, we group spatio-temporal points by 30-minute intervals based on their time t. For each group, we calculate the centroid $(\bar{x}, \bar{y})$ by aver-

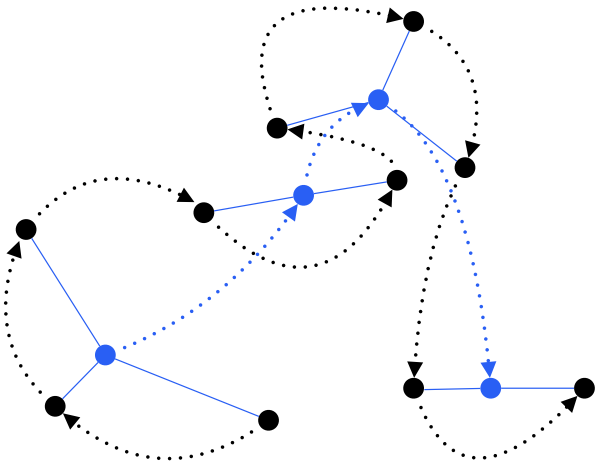


Figure 4.10: This illustration depicts how new locations are computed when merging multiple locations into one. We compute the centroid of the grouped locations (black points) by averaging over its latitude and longitude coordinates. The blue points depict the new locations and the dotted blue line the new trajectory.

aging the spatial coordinates of the points within the group:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$
$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

where (x_i, y_i) are the coordinates of the i-th point in the group, and n is the total number of points in the group.

Location: To simplify the data and to translate the location prediction from a regression task into a classification task we descretize also spatial information. To define an area, we calculate the percentiles of the latitude and longitude distributions and plot them for different values into the map (see figure 4.11). As anticipated, the distribution center is located in Copenhagen, specifically between the inner city and the DTU campus in Lyngby.

Nonetheless, a skew towards the south and east is observed for latitude and longitude, respectively. This skewness arises probably from more frequent travels within Denmark, which is mostly situated west of Copenhagen, and trips to countries in the western or southern directions rather than to the east. Given our focus on urban mobility patterns, we choose to define the area using the 20th and 80th percentiles, expanded by 0.1 degrees in latitude and 0.2 degrees in longitude, respectively. Hence we end

We subsequently divide the area into a $N \times N$- grid with equal-sized and square-shaped grid cells, where $N \in \{50, 100, 200, 500, 1000\}$. In chapter 8, we will discuss how the grid cell size will influence the performance of the model. Let A be the above defined area within a coordinate system bounded by (x_{min}, y_{min}) and (x_{max}, y_{max}). The division results in N cells along the x-axis and N cells along the y-axis. Hence the width w and height h

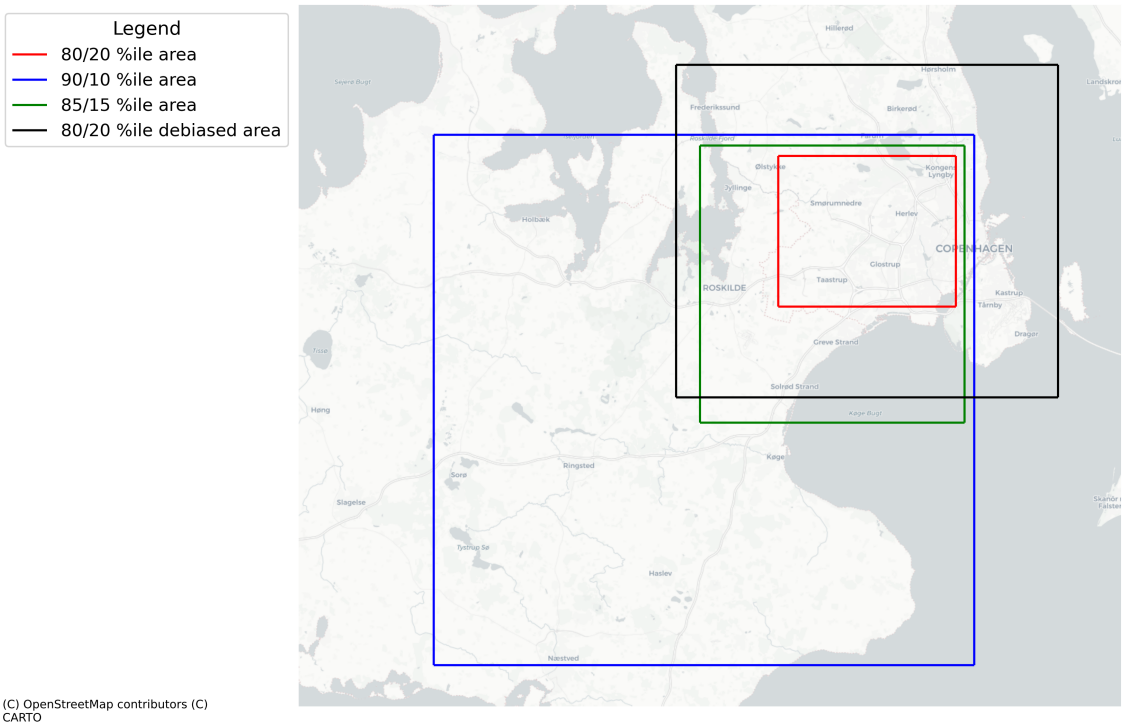


Figure 4.11: Areas corresponding to different percentiles of the coordinate values. Red: 20th/80th percentile, Green: 10th/85th percentile, Blue: 10th/90th percentile, and Black: 20th/80th percentile with padding.

of each grid cell are calculated as follows:

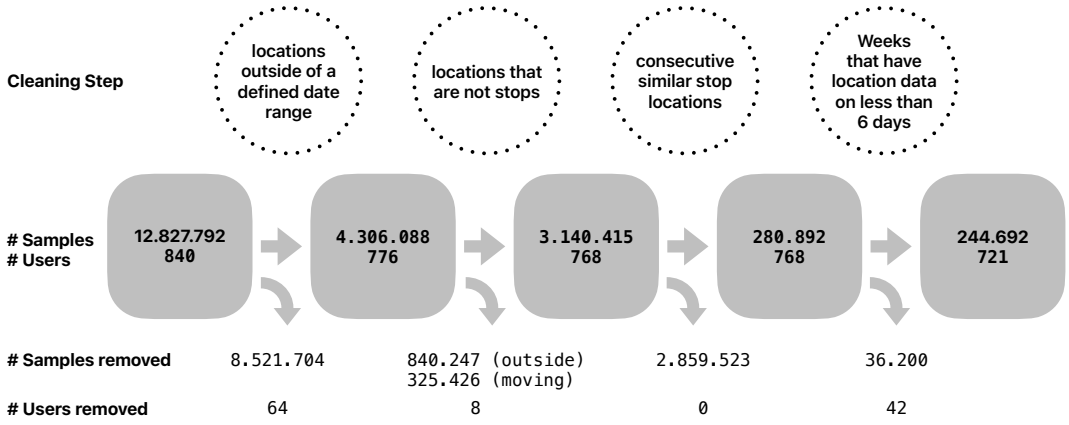
$$w = \frac{x_{max} - x_{min}}{N},$$
$$h = \frac{y_{max} - y_{min}}{N}.$$

To categorize a point $p = (x, y)$ into one of the grid cells, we follow these steps: First, we identify the grid cell coordinates (i, j) where the point p belongs, calculated as:

$$i = \left\lfloor \frac{x - x_{min}}{w} \right\rfloor,$$
$$j = \left\lfloor \frac{y - y_{min}}{h} \right\rfloor.$$

Here, $\lfloor \cdot \rfloor$ denotes the floor function, which maps a real number to the largest previous integer.The point p is then categorized into the grid cell (i, j).

4.2.3 Step 3: Data Cleaning



In the final preprocessing step, we address the issue of missing data as described in Section 4.1.3 and remove data that exhibit poor quality or significant data gaps. The specific steps are as follows:

Removal of samples lying outside a defined data range

Initially, we opt to exclude data points that fall outside the range from the 5th of January to the 25th of June 2014. Figures 4.6 and 4.3 clearly indicate that this range contains the majority of users. Restricting the range offers additionally the advantage that movement patterns are less likely to vary within this timeframe. Over a longer two-year period, students are particularly prone to changes in their movement patterns due to frequent relocations, different semester schedules, internships, and work.

Removal of locations that are not stops

We remove all data points that could not be assigned to any stop by the algorithm (labelled as -1), which likely mark locations during travel, and locations that lie outside of the defined grid. The data consists of around 73% stop locations, 20% locations outside of the grid and non-labelled locations (moving), and 7% of locations that cannot be assigned to any location.

Removal of all rows with consecutive similar stop locations

To streamline the data, we remove rows with consecutive duplicate locations from the list. This means that only changes in location are captured by the data. However, the duration of a stay is still encoded in the time feature.

Group into weeks and remove weeks that have location data on less than 6 days

We aim for using the weekly trajectories as sequences for the model to learn. In preparation for the data loader for training, we add the feature ”user_id,” which combines the

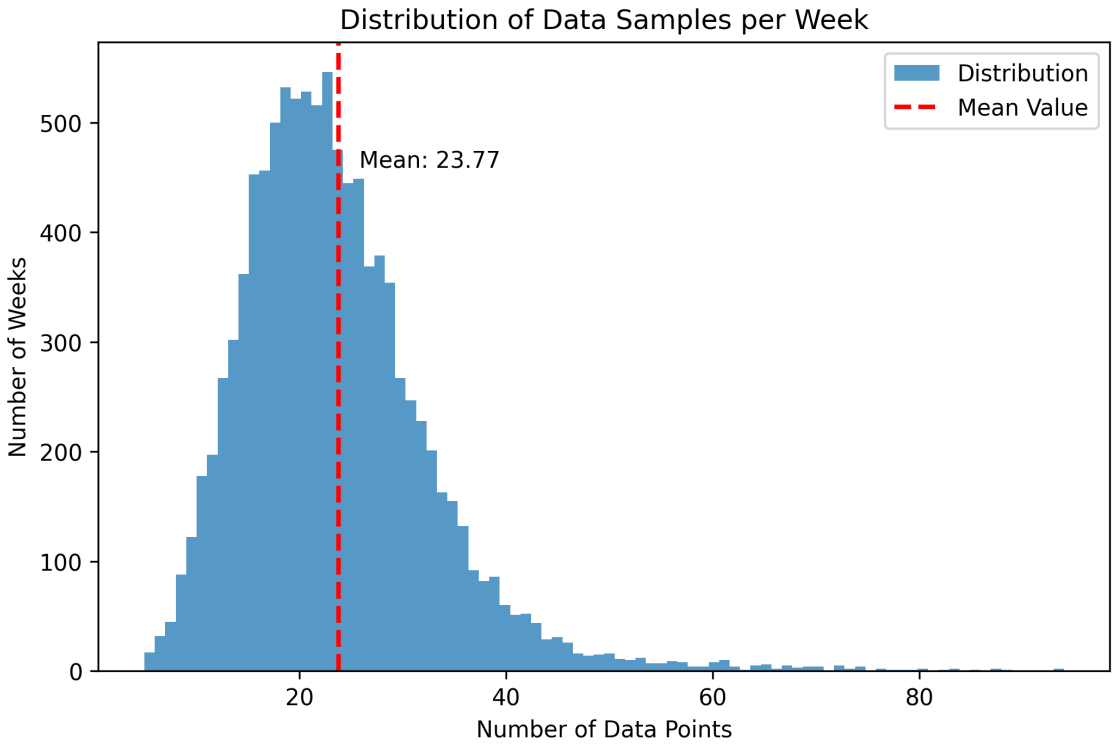


Figure 4.12: This figure illustrates the or the sequence length distribution. The distribution is flat-tailed, particularly on the right side, indicating the presence of a few sequences with up to 100 locations. However, most weeks typically have between 15 to 30 locations, suggesting an average of 2 to 4 locations visited per day.

user ID and week number into one string. This gives each week's data for a user a unique identifier. For each user, we retain only the weeks that contain at least one record for at least 6 days. This approach results in an average of around 24 locations per week (see figure 4.12).

4.3 Summary

In this chapter, we provided an overview of the dataset used for this thesis, which stems from the Copenhagen network study that captured location data from smartphones. It includes trajectories from 840 individuals over two years, totaling 245,304,797 samples.

However, the raw data posed several challenges. High temporal and spatial resolution made it difficult to model broader mobility patterns effectively. Additionally, the dataset contained irrelevant records from travel and locations outside the Copenhagen area, along with significant missing data, particularly during weekends and holidays and due to participants stopping their study participation, which compromised the quality of the data

To address these challenges, we processed the data thourougly. This involved steps such as data transformation, where we identified stop locations, and data reduction, where we discretized timestamps into 30-minute intervals and spatial coordinates into a grid. Fi-

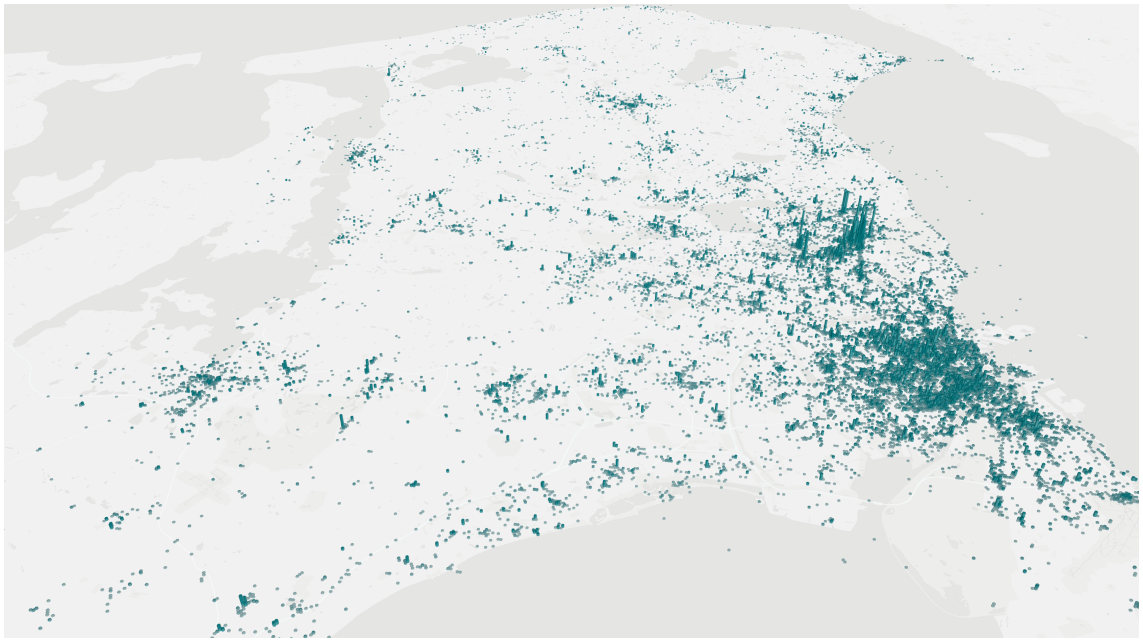


Figure 4.13: This figure shows the distribution of the processed data in the Copenhagen area, privacy-aware filtered. All locations are binned into hexagon-shaped areas, with the height defining the number of samples.

nally, we cleaned the data by removing data points outside a defined date range, those not identified as stop locations, and consecutive duplicate locations.These preprocessing steps aimed to improve data quality, reduce complexity, and prepare the data for subsequent modeling tasks.

5 Methods

In this chapter, we discuss the underlying mechanisms of MOBERT, a modification of the original BERT model. BERT, an abbreviation for bidirectional encoder representations from transformers, is an open-source machine learning framework tailored for natural language processing (NLP) tasks. It was developed by researchers from Google AI Language in 2018 [63] who modified the original Transformers architecture [21]. Before we discuss the BERT model architecture, we will first discuss fundamental concepts in NLP and the mechanics of the original Transformer in section 5.1. We delve into the BERT model in section 5.2 before we, finally, discuss the changes that we made to adapt the BERT model to our location data.

5.1 NLP and Transformers

Tokenizers

Word tokenization plays a fundamental role in processing natural language text for machine learning models. Since machines can't understand raw text, tokenization helps by breaking down text into smaller, manageable units. Essentially, tokenization is the process of breaking down text into its constituent parts, such as words, phrases, symbols, or other meaningful elements, known as tokens.

The first step is the tokenization itself, where a sentence or a piece of text is split into its constituent words. Consider the sentence: ”Natural language processing enables computers to understand human language.” Through word tokenization, this sentence is divided into tokens: ”Natural”, ”language”, ”processing”, ”enables”, ”computers”, ”to”, ”understand”, ”human”, ”language”.

Once we have the tokens, the next step is to convert these tokens into numbers so that machine learning models can understand it. This conversion involves building a vocabulary, which is a unique list of all the tokens that appear in our dataset. Each word in the vocabulary is then assigned a unique integer. For example, if our vocabulary consists of all the unique words in the above sentence, it might look something like this: {Natural: 1, language: 2, processing: 3 , enables: 4, computers: 5, to: 6, understand: 7, human: 8 }

In this vocabulary, each word is associated with a unique number. The repetition of ”language” in the sentence is intentional to illustrate that, despite its multiple occurrences, each unique word has a single entry in the vocabulary.

The tokenized sentence can now be represented numerically by substituting each word with its corresponding integer from the vocabulary: [1, 2, 3, 4, 5, 6, 7, 8, 2]. This numerical representation is what machine learning models can process. It's important to note that

the sequence of numbers maintains the original order of words in the sentence, which is crucial for models to understand the context and semantics of the text.

However, this method has its limitations, primarily because it treats each word as an independent entity without considering the relationship between words; the numbers are arbitrary assignments without inherent meaning.

5.1.1 Word Embeddings

Word embeddings come into play to overcome these limitations. Instead of mapping each word to a unique integer, embeddings map each word to a vector in a high-dimensional space (typically hundreds of dimensions). These vectors are learned from the text data in a way that positions vectors for words with similar meanings closer together in the vector space. For instance, ”computers” and ”language” might be positioned in parts of the space related to technology and communication, respectively.

Embeddings are learned through models such as Word2Vec [64], GloVe [65], or through the internal layers of deep learning models like BERT. These models are trained on large corpora of text and learn to predict a word from its context (or vice versa), effectively capturing semantic relationships. For example, in our sentence, the model learns the use of ”Natural language processing” in contexts related to ”enables”, ”computers”, and ”understand”, which helps it understand that ”Natural language processing” is a technology enabling computers to interpret human language.

Using the sentence ”Natural language processing enables computers to understand human language,” each word is converted into a dense vector. While we can’t display actual vectors here due to their usually high dimensionality, imagine that each word is transformed into a vector like:

”Natural” -> [0.21, -0.84, ..., 0.58]
”language” -> [0.19, -0.82, ..., 0.62]

Notice that ”language” would have the same vector representation each time it appears because the basic form of word embeddings does not account for polysemy (words having multiple meanings based on context). However, advanced embedding techniques like contextual embeddings (e.g., those produced by BERT) generate different vectors for the same word in different contexts, capturing even more nuanced semantic information.

The primary advantage of embeddings is their ability to capture semantic relationships, such as synonyms, antonyms, and more abstract associations, within the learned vector space. For instance, in the embedding space, vectors for words like ”understand” and ”comprehend” would be closer together, reflecting their similar meanings.

Embeddings can be applied to various types of sequential data beyond text, including location trajectories. The underlying principle remains the same: transforming discrete sequences into continuous, high-dimensional representations that capture underlying patterns and relationships. Each ”token” in the context of human mobility trajectories could

be a specific location or a more abstract point of interest like ”home”, ”work”, or ”park”. In this context, embeddings can capture the relationships and patterns in the sequences of locations. Embeddings can then be learned for each category in a way that reflects the transition probabilities or similarities in the types of activities they represent. For example, embeddings might capture that transitions from ”home” to ”work” are common in the morning hours, or that ”restaurant” visits often occur after ”cinema” visits.

5.1.2 Positional Embeddings

Positional embeddings are a crucial component in Transformer-based models, such as BERT, to incorporate the notion of word order or sequence position into the model's understanding of language. Unlike traditional sequence processing models like RNNs or LSTMs, which inherently process input data sequentially and thus have an innate sense of order, Transformer models process input data in parallel. This parallel processing greatly improves efficiency but does not naturally account for the order of tokens in a sequence. Positional embeddings address this limitation by providing a mechanism to encode the position of each token within a sequence

Positional embeddings are added to the token embeddings before feeding them into the Transformer's encoder or decoder layers. This addition ensures that the model can discern not just which tokens are present in the input but also their positions in the sequence, which is essential for understanding language structure and meaning.

There are several ways to implement positional embeddings, but the original Transformer model uses a specific formula based on sine and cosine functions of different frequencies:

$$PE(pos, 2i) = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE(pos, 2i+1) = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Where:

- $PE(pos, 2i)$ and $PE(pos, 2i+1)$ represent the positional embedding for position pos at even and odd dimensions $2i$ and $2i+1$, respectively.
- pos is the position of the token within the sequence.
- i represents the dimension index.
- d_{model} is the dimensionality of the token embeddings, indicating the size of the embedding vector.

Consider the sentence ”Natural language processing enables computers to understand human language.” After tokenization, each token is assigned a token embedding based on its meaning. However, to capture the sequence information, positional embeddings

corresponding to each token's position in the sequence are added to these token embeddings.

For instance, the word "Natural" at position 1 in the sequence will have a positional embedding based on its position, which is then added to the token embedding for "Natural." Similarly, "language" at positions 2 and 9 will have their positional embeddings added to the token embedding for "language." Despite the token embedding for "language" being the same for both instances, their positional embeddings will be different, reflecting their different positions in the sentence. This process is repeated for each token in the sequence, resulting in a set of embeddings that encapsulate both the semantic information of each token and its position within the sequence. These combined embeddings are then fed into the Transformer model.

5.1.3 Attention Mechanism

Transformers rely heavily on the attention mechanism to capture dependencies between input and output without regard to their positions in the sequence. The core idea behind the attention mechanism is to weigh the significance of different parts of the input data differently when processing a particular element of the data. We will use the notation from Vaswani et al. [21] when discussing the attention mechanism:

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \tag{5.1}$$

The attention mechanism in Transformers is often described as "Scaled Dot-Product Attention "[21], following the principles of the general attention mechanism. This attention mechanism starts by computing a dot product for each query q with all keys k. Then, it divides each result by a scaling factor $\frac{1}{\sqrt{d_k}}$ and applies the softmax function. Through this process, it obtains weights that are used to scale the values. Practically, these computations can be efficiently applied to the entire set of queries simultaneously. For this purpose, the matrices containing the queries (Q), keys (K), and values (V) are provided as inputs to the attention function. We can write express the individual rows in V and K as following:

$$Q = \begin{pmatrix} - & \vec{q}_0 & - \\ - & \vec{q}_1 & - \\ - & \vdots & - \\ - & \vec{q}_m & - \end{pmatrix}; K^T = \begin{pmatrix} | & | & & | \\ \vec{k}_0 & \vec{k}_1 & \cdots & \vec{k}_n \\ | & | & & | \end{pmatrix}$$

As a first step, alignment values are calculated by multiplying the set of queries stored in the matrix Q with the keys in the matrix K. If Q is a matrix of dimension $m \times d_k$ and K is a matrix of dimention $n \times d_k$, then the resulting matrix QK^T is a matrix of dimension $m \times n$:

$$QK^T = \begin{pmatrix} \vec{q}_0 \cdot \vec{k}_0 & \vec{q}_0 \cdot \vec{k}_1 & \cdots & \vec{q}_0 \cdot \vec{k}_n \\ \vec{q}_1 \cdot \vec{k}_0 & \vec{q}_1 \cdot \vec{k}_1 & \cdots & \vec{q}_1 \cdot \vec{k}_n \\ \vdots & \vdots & \ddots & \vdots \\ \vec{q}_m \cdot \vec{k}_0 & \vec{q}_m \cdot \vec{k}_1 & \cdots & \vec{q}_m \cdot \vec{k}_n \end{pmatrix}$$

Each element is then scaled with $\frac{1}{\sqrt{d_k}}$ and softmax function is applied row-wise to obtain a set of weights:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_0 \cdot \vec{k}_0, \vec{q}_0 \cdot \vec{k}_1, ..., \vec{q}_0 \cdot \vec{k}_n \rangle) \\ \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_1 \cdot \vec{k}_0, \vec{q}_1 \cdot \vec{k}_1, ..., \vec{q}_1 \cdot \vec{k}_n \rangle) \\ \vdots \\ \text{softmax}(\frac{1}{\sqrt{d_k}}\langle \vec{q}_m \cdot \vec{k}_0, \vec{q}_m \cdot \vec{k}_1, ..., \vec{q}_m \cdot \vec{k}_n \rangle) \end{pmatrix} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,n} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,0} & s_{m,1} & \cdots & s_{m,n} \end{pmatrix}$$

As a result of the softmax operation, for each row i it holds that $\sum_{j=0}^{n} s_{i,j} = 1$.

The resulting weights are then applied to the elements of matrix V of dimension $n \times d_v$ through matrix multiplication:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,n} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,0} & s_{m,1} & \cdots & s_{m,n} \end{pmatrix} \begin{pmatrix} - & \vec{v}_0 & - \\ - & \vec{v}_1 & - \\ - & \vdots & - \\ - & \vec{v}_n & - \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} s_{0,i}\vec{v}_i \\ \sum_{i=0}^{n} s_{1,i}\vec{v}_i \\ \vdots \\ \sum_{i=0}^{n} s_{m,i}\vec{v}_i \end{pmatrix}$$

Important to note is that the process has no learnable parameters but it only consists of matrix and vector calculations, which results in a series of weighted averages of the rows of V. The weighting depends on the input queries and keys. Each of the queries in Q results in a specific weighted sum of the value vectors. It has the effect that the Transformer model is able to dynamically focus on different parts of the input sequence, assigning more weight to the relevant elements and effectively capturing the context and dependencies within the sequence.

Understanding how the weights are made helps explain the terms ”queries,” ”keys,” and ”values.” This process uses specific keys to select the matching values. The keys we’re looking for are determined by the queries - the relationship between a key and a query can be described by the angle x between them.

To reintroduce some context from machine translation, in this framework, each row of the keys, queries, and values represents a vector form of sequence elements. A limitation identified in RNN-based models is their struggle to incorporate information from elements that appeared a long time ago. More broadly, they find it hard to connect parts of a sequence that are distant from each other. Using the attention mechanism and creating bidirectional models were efforts to overcome this problem.

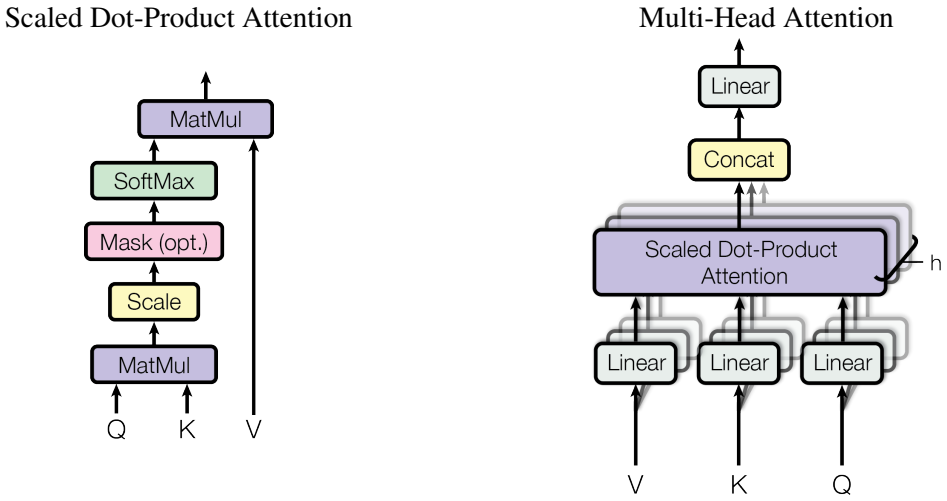


Figure 5.1: An illustriation of the Scaled Dot-Product Attention and multi-head attention from *Attention is all you need* [21].

Essentially, Scaled Dot-Product Attention allows us to quickly and effectively link each element in one sequence to all elements in another sequence, and also to every other element in the same sequence.

5.1.4 Multi-Head Attention

Multi-Head Attention builds upon the concept of Scaled Dot-Product Attention by applying linear transformations to the queries, keys, and values, creating multiple sets of inputs for the attention mechanism. These sets are then processed in parallel by the attention function, and their outputs are combined into a single matrix. This combined matrix is then transformed linearly to produce a final output matrix of the required size. Formally, multi-head attention is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{5.2}$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{5.3}$$

and W_i^Q, W_i^K, W_i^V, and W^O are parameter matrices. Each W_i^Q, W_i^K, W_i^V is used to project the queries, keys, and values in each head into a distinct representation subspace.

By using multi-head attention, the Transformer can capture various aspects of the input data and process the same input through multiple ”lenses” or perspectives, each defined by a different linear transformation. This is somewhat analogous to how convolutional neural network use different filters to capture various features from the same input.

5.1.5 Encoder and Decoder in Transformer Models

Central to the design of the Transformer are the encoder and decoder components, which leverage multi-head attention mechanisms to process and generate sequences. To illustrate the mechanics of the architecture, we us assume that we have an input sequence $\vec{F} = (f_0, ..., f_n)$ that we want to translate to a target sequence $\vec{E} = (e_0, ..., e_m)$. Note that both sequences can have different lengths.

The encoder's primary function is to process the input sequence $\vec{F}$ and map it into a high-dimensional space — a representation that encodes both the semantic and syntactic properties of the input. The encoder is composed of a stack of identical layers, each containing two sub-layers: Multi-Head Self-Attention layer, followed by a fully connected layer with ReLU activations (see figure 5.2). Both the multi-head attention layer and the fully connected layer are followed by an "Add & Norm" step - the "add" refers to a residual connection that adds the input of each layer to the output, and the "norm" refers to Layer Normalization. During training, the input words $\vec{F} = (f_0, ..., f_n)$ are passed into the first encoder block all at once, and the output of that block is then passed through its successor. The process is repeated until all N encoding blocks have processed the input. Each block has two components: a Multi-Head Self-Attention layer, followed by a fully connected layer with ReLU activations that processes each element of the input sequence in parallel. When the input has passed through all the encoding blocks, we are left with the encoded representation of $\vec{F}$.

Moving onto the decoder, it consists of three steps: a Masked Multi-Head self-attention layer, a multi-head attention layer connecting the encoded source representation to the decoder, and a fully-connected layer with ReLU activations. Just like in the encoder, each layer is followed by an "Add & Norm" layer. The decoder accepts all of the target words $\vec{E} = (e_0, ..., e_m)$ as input. There are several key differences from the encoder to note here - one is that inputs to the first attention operation in the decoder blocks are masked, hence the name of the layer. We explore this in detail in the multi-head attention section, but in a nutshell, this means that any word in the target output can only attend to words that came before it.

The decoder's role is to generate the output sequence from the encoded representation. Similar to the encoder, the decoder is composed of a stack of identical layers, each with three sub-layers: a multi-head self-attention mechanism, a multi-head attention mechanism over the encoder's output, and a position-wise fully connected feed-forward network. The decoder's first sub-layer involves multi-head self-attention, but with a modification to prevent positions from attending to subsequent positions. This masking ensures that the predictions for position i can depend only on the known outputs at positions less than i. The reason for this is simple: during inference, we generate the predicted translation $\vec{E}$ word-by-word using the source sentence $\vec{F}$. In the process of predicting a word e_i, the decoder has access to previously generated words. It cannot, however, have access to words that follow e_i, as those have yet to be generated and depend on the model's

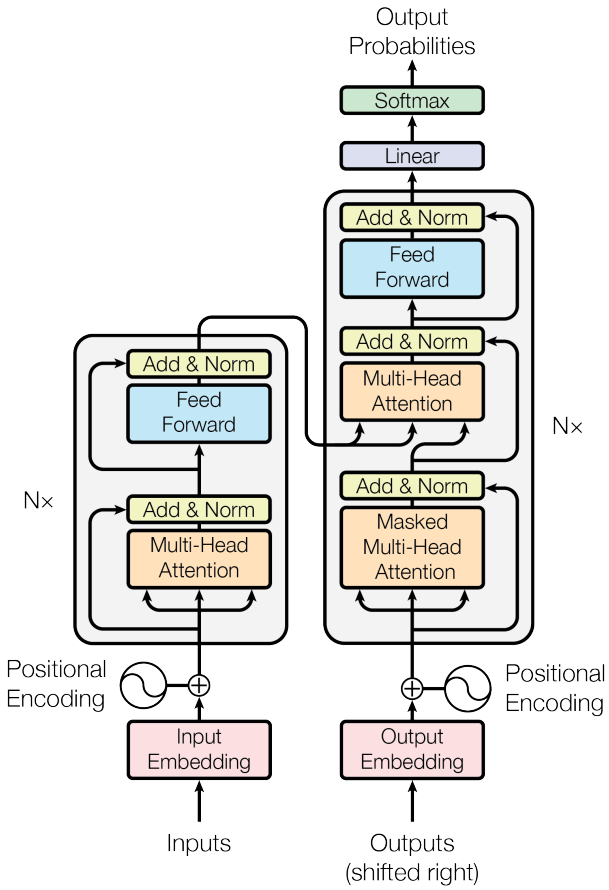


Figure 5.2: An illustriation of the transformer architecture as described in the article *Attention is all you need* [21]. The left part depicts the encoder, the right part the decoder.

choice of e_i. Masking during training allows us to emulate the conditions that the model will face during inference. The second multi-head attention layer is also referred to as the Encoder-Decoder Attention and performs multi-head attention over the encoder's output. Another difference is here, that unlike the attention layers at the start of the encoder and decoder blocks, this layer is not a self-attention layer which means that it ties together the encoded representation with the target words in $\vec{E}$. Here, the queries (Q) come from the previous decoder layer, while the keys (K) and values (V) are the outputs of the encoder. This allows each position in the decoder to attend to all positions in the input sequence $\vec{E}$,integrating information from the entire sequence for generating the next element of the output sequence.

Traditional language models or other transfomer models like, for example, OpenAI's GPT process text linearly, either from left to right or right to left, limiting their contextual understanding. In contrast, BERT adopts a **bidirectional** approach, considering both the left and right context of words simultaneously within a sentence. This bidirectional perspective enhances the model's comprehension of context, enabling it to learn nuanced meanings more effectively.

5.2 The BERT Model

5.2.1 Background

BERT (bidirectional encoder representations from transformers) is a groundbreaking model introduced by Devlin et al. at Google AI in 2018 [63] and builts directly upon the Transformer model. In its research stage, the open source framework achieved state-of-the-art results in 11 natural language understanding tasks, including sentiment analysis, text classification, semantic role labeling,and the disambiguation of words with multiple meanings. The capacity to address ambiguity was one of the greatest challenges in natural language understanding, that distinguished BERT from previous language models such as word2vec [64] and GloVe [65]. Nowadays BERT models are used in various artificual intelligence systems that rely on the understanding of natural language. Being trained on more than 70 different languges, the BERT model had a large impact on Google search and other text-based or voice based search engines.

5.2.2 Architecture

Unlike the original Transformer architecture, which includes both encoder and decoder modules, BERT employs an encoder-only architecture. The original Transformer model, designed for sequence-to-sequence tasks, comprises both an encoder and a decoder. BERT's architecture, focusing solely on the encoder. At its core, the architecture can be divided into three primary components: the embedding, a stack of encoders, and unembedding. This structure facilitates the model's ability to understand and generate language by converting input sequences to and from a rich, vectorized representation.

Embedding Layer

The embedding layer is the entry point for input data into the Transformer model. Its primary function is to convert an array of tokens into an array of dense vectors. This conversion is crucial because it transforms sparse, high-dimensional representations into a form that captures semantic and syntactic nuances of the input tokens in a lower-dimensional space.

$$E = \text{Embedding}(X) \tag{5.4}$$

Where X represents the one-hot encoded input tokens, and E is the resulting array of embedding vectors. This embedding process incorporates not just token embeddings but also positional embeddings, ensuring that the model retains information about the sequence order of the tokens.

A Stack of Encoders

Following the embedding layer, the input vectors are processed by a stack of encoders. Each encoder in the stack is a Transformer encoder unit that performs a series of transformations on the input vectors, refining and re-encoding the information at each level.

$$H_i = \text{Encoder}_i(H_{i-1}) \tag{5.5}$$

For $i = 1 \dots N$, where $H_0 = E$, and N is the number of encoders in the stack. Each encoder consists of two main sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Layer normalization and residual connections are applied around each of these sub-layers to enhance training stability and efficiency. Originally, BERT was implemented at two model sizes: $\text{BERT}_{\text{BASE}}$ with 12 encoders with 12 bidirectional self-attention heads (in total 110 million parameters) and $\text{BERT}_{\text{LARGE}}$ with 24 encoders with 16 bidirectional self-attention heads (in total 340 million parameters).

Un-embedding Layer

The final encoder's output is transformed back into a sequence of tokens. This process, which we refer to as un-embedding, involves converting the dense vector representations output by the last encoder into probabilities over the possible output tokens, typically followed by selecting the token with the highest probability at each position in the sequence. The un-embedding step is realized through a linear transformation followed by a softmax function:

$$P = \text{softmax}(H_N W + b) \tag{5.6}$$

Where P represents the probabilities of each token in the vocabulary being the next token in the sequence, W is a weight matrix, and b is a bias vector. The softmax ensures that

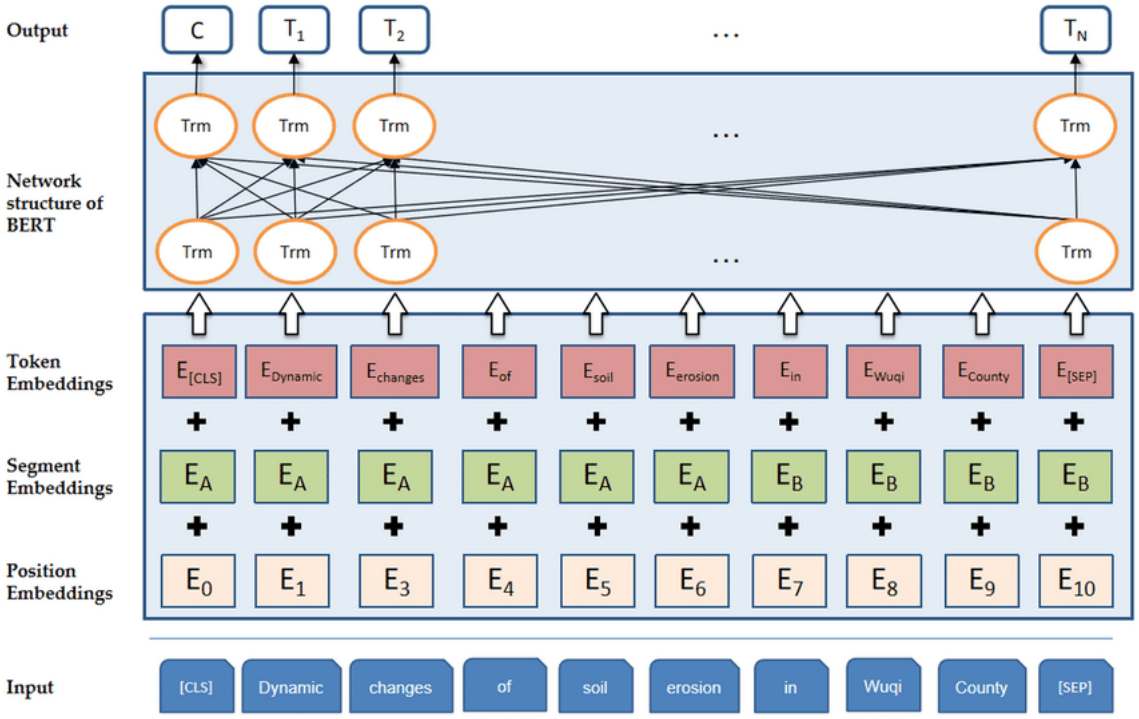


Figure 5.3: The original BERT embedding structure visualised by an exemplary sentence [63].

the output is a valid probability distribution for each position in the sequence.

5.2.3 Bidirectional Context

A key innovation of BERT is its ability to pre-train deep bidirectional representations. Unlike the original Transformer, which processes sequences in a left-to-right or right-to-left manner, BERT's encoder captures context from both directions simultaneously. This bidirectionality is achieved through the use of MLM and next sentence prediction (NSP) during pre-training.

5.2.4 Masked Language Modeling

In the MLM pre-training task, BERT randomly masks a percentage of the tokens from the input, and the model must predict the original token of the masked positions. The strategy for masking involves the 80-10-10 rule:

- **80% of the time**, the selected tokens are replaced with the '[MASK]' token.

  For example: `Natural language processing enables [MASK] to understand human language.`

- **10% of the time**, the selected tokens are replaced with a random token from the vocabulary.

  For example: `Natural language processing enables` *`elephants`* `to understand human language.`

- **10% of the time**, the selected tokens are left unchanged.

  For example: texttttNatural language [MASK] enables *computers* to understand human language.

Although it is a convention to mask 15% of the tokens with the above technique other ratios have also been explored and proven to be successful [66].

The mixed strategy of masking, random replacement, and unchanged tokens ensures that the Transformer encoder develops a comprehensive contextual understanding of each input token and has various advantages:

When a model is exclusively trained to predict the original word in the presence of a [MASK] token, there's a risk that the model will learn to rely solely on this token to make predictions. Such a model might perform well in the training environment but fail to generalize to real-world scenarios where no [MASK] token indicates which word needs to be predicted. By sometimes replacing the masked token with a random word, BERT is encouraged to understand the entire context provided by all words in the sentence, not just the masked ones. This approach helps in preventing the model from overfitting to the mask token itself.

Introducing random replacements also serves to increase the model's tolerance to noise. In real-world applications, input data may be imperfect or contain errors. By training the model to deal with randomly replaced tokens, BERT learns to better handle such noisy conditions. This aspect is particularly useful for processing unstructured text from sources like social media, where typographical errors and unconventional uses of language are common.

In summary, this approach ensures that the model does not merely learn to predict the presence of the '[MASK]' token but instead learns a deeper understanding of the context surrounding each word.

5.2.5 Next Sentence Prediction

The NSP task further trains BERT to understand relationships between sentences, which is crucial for tasks that involve understanding the connection between separate sentences, such as question answering and natural language inference.

BERT is trained to predict whether a second sentence logically follows a first. During training, the model is fed pairs of sentences, where half the time, the second sentence is the actual next sentence, and half the time, it is a random sentence from the corpus. The model learns to predict whether sentences are related or not, enhancing its ability to understand narrative flow and context.

5.2.6 Special Tokens

BERT introduces special tokens to facilitate various NLP tasks and to structure the input data:

- **[PAD]**: Used to fill in blank spaces to ensure that input sequences are of the same length when batched together for processing.
- **[CLS]**: Stands for ”classification” and is added to the beginning of each input instance. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
- **[SEP]**: Used to separate sentences in tasks that involve multiple input sequences (e.g., question answering, where the model needs to understand the relationship between a question and an answer passage).
- **[MASK]**: Used in the MLM pre-training task to denote tokens that have been masked and need to be predicted by the model.

During pre-training, BERT utilizes these special tokens to structure the input data. For example, in the NSP task, sentences are concatenated with the ‘[SEP]‘ token in between, and the ‘[CLS]‘ token is used at the very beginning. The model then predicts whether the second sentence logically follows the first, leveraging the representations associated with the ‘[CLS]‘ token.

Example:

Given the pair of sentences:

1. ”Natural language processing enables computers to understand human language.”
2. ”This capability has revolutionized how we interact with technology.”

An application of MLM and NSP might result in: ”[CLS] Natural language [MASK] enables [MASK] to understand human language [PAD] [PAD] [PAD] [PAD] [SEP] This capability has revolutionized how we interact with technology [PAD] [PAD] [PAD] [PAD]”

Through the MLM and NSP tasks, BERT learns to deeply understand both the context surrounding individual words and the relationships between sentences. This bidirectional understanding of language enabled BERT to achieve state-of-the-art performance on a wide range of natural language processing tasks.

BERT represents a significant evolution of the Transformer model’s architecture, emphasizing language understanding over sequence generation. By adopting a bidirectional approach and introducing with MLM and NSP new pre-training tasks, BERT set a new standard for NLP tasks, demonstrating the versatility and potential of Transformer-based models for advancing the field of artificial intelligence.

5.3 Summary

In this chapter we discussed the underlying mechanisms of MOBERT. It builds up on BERT, which was released in 2018 by Google AI Language researchers. BERT, a modification of the original Transformers architecture, revolutionized NLP by considering both

left and right context within sentences simultaneously, enhancing comprehension of nuanced meanings.

We discussed the key concepts that are fundamental to NLP and BERT's architecture, such as tokenization, word embeddings, positional embeddings, and attention mechanisms. Tokenization splits text into smaller units, while word embeddings represent these units in high-dimensional space and thereby capture semantic meanings. Positional embeddings are crucial for incorporating the word order, and attention mechanisms, particularly Scaled Dot-Product Attention, and multi-head attention, allow the model to focus on different parts of the input sequence to understand the context and relationships between words.

BERT's architecture is unique because it relies solely on the encoder part of the Transformer model which comprises an embedding layer, multiple encoders, and an un-embedding layer. The embedding layer converts tokens into dense vectors, while the encoder stack refines input vectors through multi-head self-attention mechanisms. The un-embedding layer transforms encoder output back into token sequences.

A significant innovation of BERT is also its pre-training tasks: masked language modeling (MLM) and next sentence prediction (NSP). MLM allows BERT to learn contextual relationships by predicting randomly masked tokens in a sentence, while NSP improves its ability to understand the relationship between sentences. Special tokens ([PAD], [CLS], [SEP], [MASK]) are introduced to facilitate these tasks and structure the input data effectively.

Through its bidirectional approach and pre-training tasks, BERT achieved remarkable success in various NLP tasks, demonstrating the efficacy of Transformer-based models in the understanding of language.

6 MOBERT

In this chapter, we discuss our adaptation of the architecture and training techniques of BERT, initially designed for Natural Language Processing (NLP), to the task of predicting masked locations in mobility trajectories. First, we discuss how we adapted the general concepts of NLP and Masked Language Modeling (MLM) for use with location data. Second, we describe the additional input features used for training—user ID, rank, and time—and their implementation. Lastly, we explain our modifications to the BERT architecture to allow for the processing of location data alongside these features.

In the following sections, our specialized adaptation of BERT for mobility trajectories will be referred to as 'MOBERT' (mobility optimised bidirectional encoder representations from transformers).

6.1 Applying BERT's NLP Concepts to Next Location Prediction

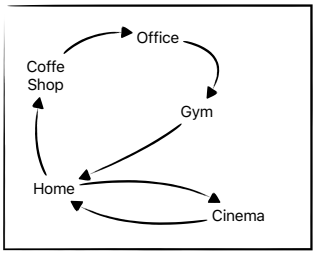
By treating locations analogously to words in text, we hope leverage the deep contextual learning capabilities of Transformer models to predict future locations. This adaptation involves treating sequences of locations similarly to sentences in a text, where each location represents a 'word' and the trajectory represents a 'sentence'.

6.1.1 Adapting MLM for Location Data

MLM can be adapted to location data by randomly masking certain locations in a user's trajectory and training the model to predict these masked locations based on their context (previous and following locations). This adaptation encourages the model to learn contextual relationships between locations and hence improving its accuracy for predicting the next location.

Example:

Consider a sequence of locations: [Home, Coffee Shop, Office, Gym, Home, Cinema, Home]. Applying MLM, we might mask 'Coffee Shop' and 'Cinema', resulting in [Home, [MASK], Office, Gym, Home, [MASK], Home]. The model then learns to predict the masked locations based on the context provided by 'Home' and 'Office'.



The other special tokens used in BERT can also be repurposed for masked location prediction as follows: The token [PAD] facilitates processing batches of location sequences of varying lengths by

padding shorter sequences. The token [CLS] is prepended to each sequence to aggregate its representation, which can be useful for classification tasks such as categorizing the overall nature of a trajectory (e.g., work-oriented and regular, leisure-oriented, explorative). And the token [SEP] separates sequences when modeling interactions between two or more trajectories (e.g., comparing two users' location histories).

The model can be trained on historical location sequences with randomly masked locations, using the context of surrounding locations to predict the masked ones. This process not only teaches the model the typical patterns of movement but also how certain locations are contextually related to others. Once trained, the model could also predict the next location in a sequence by treating the last known location as the context and generating predictions for the immediate next (masked) location.

We will as an initial step, however, implement and evaluate the model based on the MLM pre-training.

6.1.2 Additional Input Features

Predicting locations in a sequence of visited places can be substantially improved by incorporating additional contextual information. Specifically, incorporating features such as user ID, the rank of the place (indicating its popularity), and the time of visit can provide the predictive model with a richer context, improving its accuracy and relevance. This chapter explores how these features can be integrated into Transformer-based models for masked location prediction.

Feature 1 - User ID

Each user's movement pattern is unique and influenced by personal preferences, routines, and constraints. By including the user ID as an input feature, the model can learn user-specific patterns and tailor its predictions accordingly. This personalization allows the model to distinguish between different users' trajectories and can improve the prediction accuracy.

Implementation: We assign for each user a unique number. Since the userID does not change, we don't need to use this feature for every location in a trajectory. Instead, this number will be embedded and instead of the start token ([CLS]) concatenated with location embeddings of the trajectory before being fed into the Transformer model. See section 6.1.3 for a more detailed description of the embedding structure.

Feature 2 - Rank

This feature provides the model with information regarding the relevance of a location for the individual. The intuition is that the most frequently occuring location is the place where the individual lives and that the individual visits daily. The second most occuring location could be the work place or the school that the indivual is attenting. The third location the local supermarket, a close friend etc. Hence the rank of a place can significantly influence its likelihood of being visited next. Incorporating location rank as an input feature allows

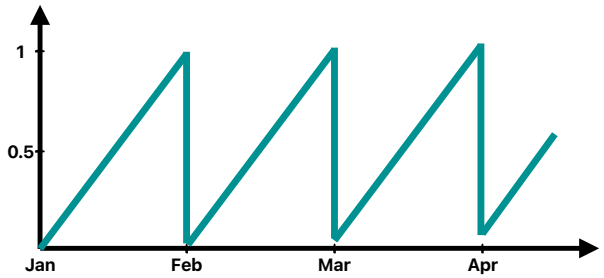


Figure 6.1: A linear periodic time feature transformation, linearly mapping each date within a month to the interval [0,1].

the model to weigh the popularity of locations in its predictions.

Implementation: To compute this feature we count for each user the occurances of all locations that are in the training data and sort them accordingly. This feature will be added to every location in a trajectory and is unique for every user.

Feature 3 - Time

Humans visit different locations depending on the time of day, month, and year. Hence time of a visit provides context that influences the likelihood of visiting certain places. For instance, school and work places might be more likely visited in the morning, while the home location or entertainment venues might be preferred in the evening. Therefore, time can be valuable information for the model.

Implementation: This feature is also unique for every location in a trajectory and hence needs to be added similarly to the rank information. There are, however, various ways to incorporate time information into a model. The most straightforward method would be to use the date and time as input feature, formatting it as a string, for example, `"YYYYMMDDTHH MMSS.mmmZ"`, `"MM/DD/YY HH:MM:SS"` or `"YYYY-MMDDTHH: MM:SS"`. However, this approach is not ideal for several reasons. Firstly, including the year is unnecessary as it does not provide the model with information about the season (furthermore, our training data was collected solely in 2014). More importantly, the model would need to learn how the Christian Gregorian calendar works; that is, understanding that a year has 12 months, a month has between 28 and 31 days, and a day has 24 hours, etc. This means the model needs to learn the relative time position within a year, a month, and a day based on the date string. Although this could be possible with sufficient training data and a highly complex model, there is a simpler approach.

To simplify the process for the model, we could instead calculate the relative time within a year, month, or week and express the feature through a corresponding value between 0 and 1, as shown in Figure 6.1.

However, this approach has a disadvantage: there is a large jump between the last timepoint of an interval and the first timepoint of the subsequent interval. Ideally, a short proximity between two timepoints should be reflected by a small difference in their transformed values.

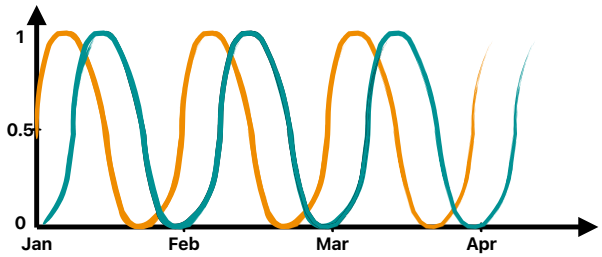


Figure 6.2: A drawing of the periodical time feature transformation. In orange the value $x = \frac{1}{2}(\sin(\alpha) + 1)$ and in green $y = \frac{1}{2}(\cos(\alpha) + 1)$

We choose a better way to represent the time data: Since the data is of cyclical nature we use periodic functions instead of linear ones and express the position within a repeating interval as a position on the unit circle. This means we scale the relative position in a time interval to a point α in $[0, 2\pi]$ and obtain the coordinates by calculating $x = \sin(\alpha)$ and $y = \cos(\alpha)$. This results in a continuous, periodic function that preserves the distances beyond the boundaries of a time interval (see Figure 6.2).

6.1.3 Adapting Embeddings for Location Data

The original BERT utilizes a combination of token, segment, and positional embeddings to create a rich representation of each input. Token embeddings capture the semantic meaning of each word or subword, segment embeddings distinguish between different sentences or segments within the input, and positional embeddings provide the model with information about the order of tokens in the sequence. We adapt BERT for masked location prediction by customizing the embedding process to accommodate the specific types of additional input data.

Embedding Location Data

To embed the location data, we count the distinct locations and create a vocabulary for the locations that translates each location into a distinct token. The first four tokens are reserved for the special tokens [PAD], [CLS], [SEP], and [MASK]. The remaining tokens represent the locations in the dataset sorted by their overall popularity. The token five, for instance, represents a location on the DTU campus in Lyngby, since it is the most visited location in the dataset. This does not change the training process but makes it easier to comprehend results and to debug the model. When training the model, vector representations for the single tokens are learned.

Embedding Additional Features

The translation of additional features into tokens requires little effort. Rank and the user ID are already discrete integer values that do not need translation via an additional vocabulary. The periodic time feature, however, needs to be grouped into bins and hence translated into categorical variables. For all additional features, we keep number 0 as the ”empty” padding token.

It would not be necessary to add the user ID as information to every location since it does

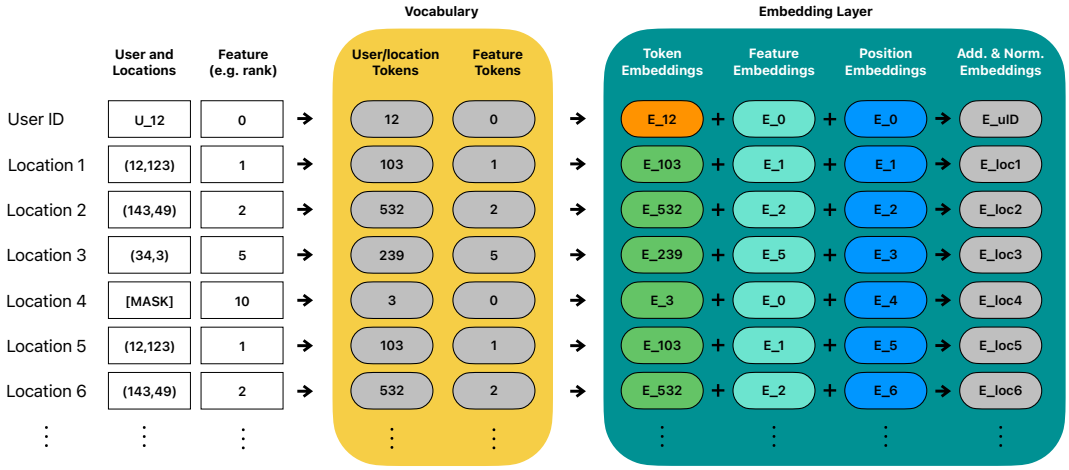


Figure 6.3: Embedding Architecture: The distinct embeddings for the UserID, location, and features are indicated by different colors in the embedding layer. Although the figure illustrates only one feature, it is possible to append more than one unique feature. Each feature is embedded in its own space and subsequently concatenated. After concatenating the token, feature, and positional embeddings, the combined embeddings are normalized (i.e., linearly transformed and mapped to a lower-dimensional space defined by `d_model`, and scaled by the square root of `d_model`).

not change throughout a sequence. Hence, when using the user ID, we add the token as a start token to the sequence instead of the [CLS] token.

Concatenating the Embeddings

After embedding each input type, the embeddings are concatenated along with positional embeddings to form a comprehensive representation of each input instance. These combined embeddings are then normalized. Normalization here involves a linear transformation to a lower-dimensional space defined by `d_model`, and scaling by the square root of `d_model`. This step ensures that the embeddings are in a suitable form for the subsequent layers.

6.2 Summary

In this chapter, we explained how we applied BERT's architecture and training techniques to predict masked locations in mobility trajectories and call our adaptation 'MOBERT' (mobility optimised bidirectional encoder representations from transformers).

We adapted Masked Language Modeling (MLM) for location data by masking certain locations in a trajectory and training the model to predict them based on context. Special tokens like [PAD], [CLS], and [MASK] were repurposed for masked location prediction.

Further, we explained how we incorporated the additional features user ID, rank, and time

to improve prediction accuracy that provide additional contextual information

Finaly we described, how we customized the embedding process to accommodate additional input data types. Through MOBERT, we aimed to leverage contextual learning for location prediction in mobility trajectories.

Overall, our adaptation of BERT for mobility trajectories, MOBERT, aims to provide location predictions by leveraging contextual learning and incorporating additional features relevant for human mobility modelling.

7 Experimental setup

Training a BERT model is a nuanced process that goes beyond just running a script. In this chapter, we outline the experimental framework that we used to assess the performance of MOBERT. We desribe how we trained and tested the model and compared it against three other baseline models that we will introduce in the following. Further, we describe how we compared the impact of the different features described in 6.1.2 on model performance. Lastlym we describe how we tested the performance of prediction tasks across various grid resolutions.

7.1 Baseline Models

To compare and better understand the results from MOBERT, we implement baseline models. In the following, we will describe them and their properties.

7.1.1 Baseline model 1 - Fixed Location Based on Location Rank

The first baseline model ist the most simple one. For each user, the model is given the users trajectory from the training data $T_u = (p_1, p_2, ..., p_{n_u})$. When predicting a new location, the model always chooses the most frequently occuring location in T_u. If there are two most frequently occuring locations, it chooses always one of them (for example, the one that appeared first in the trajectory). Therefore, the model learns unique locations for each user. Since the most frequently occurring location is often where the user resides, we may subsequently refer to this location as the *home location* and to this strategy as ”always staying home.”

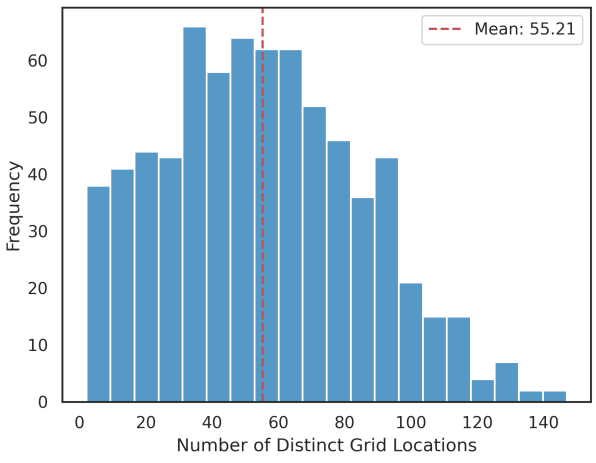


Figure 7.1: This figure illustrates the number of grid locations visited by a user in the processed data on the 200 $\times$ 200 grid (covering the period between the 5th of January and the 25th of June). The distribution is quite broad, with an average of 55.21 locations. Introducing randomness in selecting among these locations would result in a wide range of prediction outcomes.

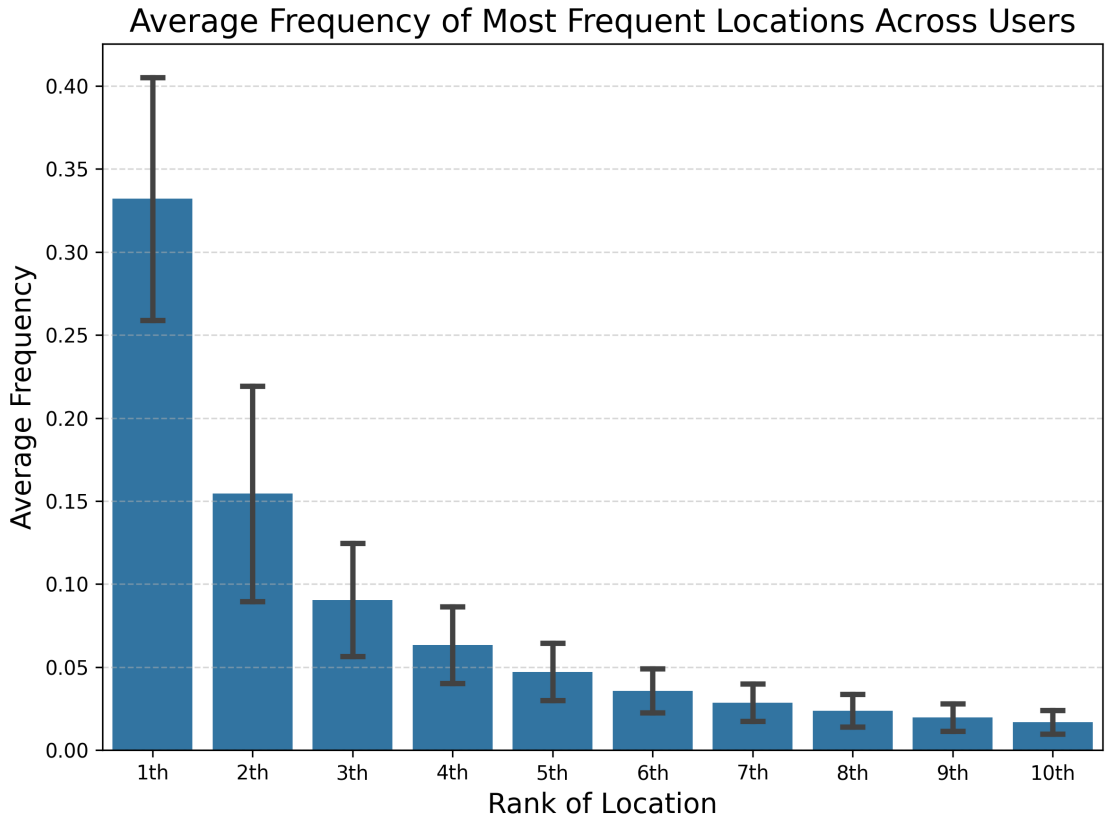


Figure 7.2: The plot shows the average relative frequency of the top 10 most visited positions by users. Each bar represents the mean percentage of visits to a position across all users for that rank, providing a view of how certain positions hold varying levels of importance among the user base. Error bars on each bar indicate the standard deviation and showcase the variance. Not surprisingly, the variance is larger for the higher-ranked positions.

7.1.2 Baseline model 2 - Sampled Location Based on Location Rank

The second baseline model calculates, similarly to the first baseline model, the frequency of the locations in the training data for each user. The only difference is that the location is selected with probability according to their frequency. That means a location l_i that appears m times in the trajectory $T_u = (p_1, p_2, \ldots, p_{n_u})$ will be selected with probability $p(l_i) = \frac{m}{n_u}$. This approach adds more randomness to the prediction and allows for wide range of possible prediction labels, compared to the first baseline model (see figure 7.2). However, Figure 7.2 shows that the average frequency of the top 10 most visited positions already accounts for over 50%.

7.1.3 Baseline Model 3 - First-Order Markov Model

In this third baseline, we introduce a more flexible approach by using a first-order Markov model. The underlying intuition, akin to Markov chains, is to model the transition from one location to another with a certain probability. Depending on the order of the model, a

varying number of previous steps are taken into account when predicting the next step. For a model of order 1, the prediction is based solely on the previous position. For a model of order 2, the prediction considers the previous two positions, and so on.

Let d be the order of the model. The model is given the training trajectory $T_u = (p_1, p_2, \ldots, p_{n_u})$ of a user u. For a first-order Markov model ($d = 1$), the model learns a conditional probability distribution with probabilities $P(p_i|p_{i-1})$ for a position p_i given the immediately preceding position p_{i-1}. This approach captures the likelihood of transitioning to a given location based on the current location. The probability of transitioning to a position p_i from a subtrajectory S of positions, where S has a length equal to the order of the model, is then given by:

$$P(p_i|S) = \frac{\text{Number of transitions from } S \text{ to } p_i}{\text{Total number of transitions from } S}$$

For the first-order model, S would simply be the last known position. This model assumes that the probability of moving to the next location is dependent only on the current position, reflecting the memoryless property of Markov processes. By learning these transition probabilities from the training data, the model can predict the next location based on the current state.

7.2 Training procedure

In this section we describe the process of training the model. Our overall goal was not to optimize hyperparamters to achieve marginally better accuracy results but rather to find a setting that allows for comparing the various models while being computationally affordable to train and does not overfit on the training data. We discribe how we grouped the location data into subtrajectories and distributed those trajectories in such a way, that all users are represented in all sets. We further describe the model parameters used for training.

7.2.1 Stratified Data Splitting

The primary purpose of data splitting is to ensure that the model generalizes well to new inputs. The conventional split comprises training, validation, and test sets. The training set is used to fit the model, the validation set for tuning hyperparameters and preventing overfitting, and the test set for evaluating the model's performance. Proper data splitting ensures that the model learns the underlying patterns without memorizing the data, thereby enhancing its predictive capabilities on new, unseen data.

We follow the convention and utilize a 70-15-15 data split. Although our goal is not to train and compare for different hyperparameters, we use the validation set to monitor the training process for overfitting. We then use the test set to compare the model's performances with each other.

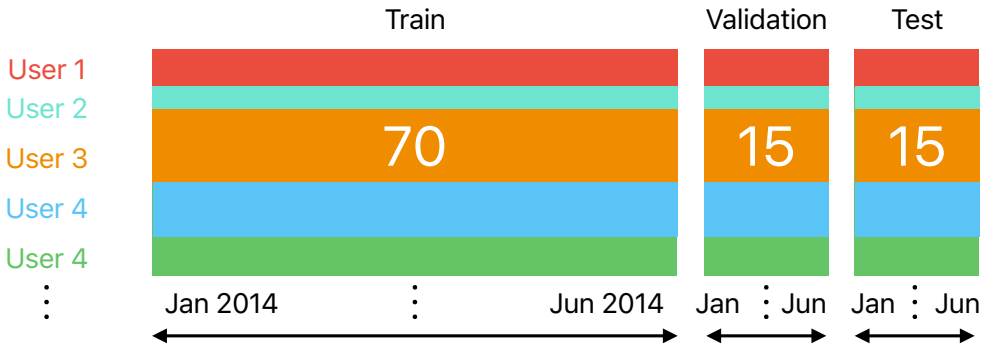


Figure 7.3: Visualisation of the concept of stratified sampling applied to our data set. Each user is represented in each split. Further the distribution of the amount of data from each user is similar in each split.

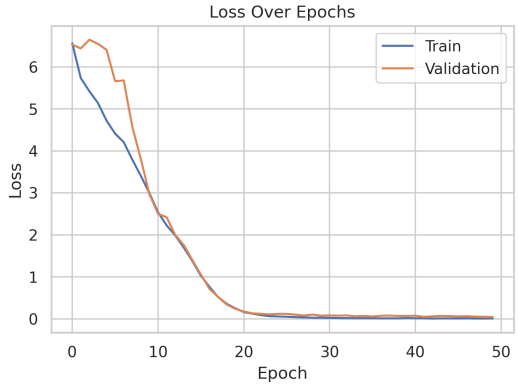
Since we are dealing with sequential data, the splitting strategy must preserve the temporal order of the data points. This preservation is crucial because the order carries significant information that we want the model to learn.

By grouping the users' trajectory into predefined time intervals of one calendar week and sorting them according to their date, we ensure that the order of the locations in a trajectory is preserved. Hence, the sequence length is determined by the user's trajectory length within one calendar week. However, when splitting the data randomly by its weeks and users, it could happen that the proportions of data across users differ from set to set. Having different proportions means the model would learn user behavior from the training set that is not reflected in the other splits and hence worsens the model's performance. In the worst case, there are users in the validation or test split that are not in the training split. Therefore, we need to ensure that the distribution of the data for each user is similar across all splits.

To mitigate this sampling bias, we use stratified sampling, commonly known and used in survey sampling [67]. By implementing a stratified sampling technique, we divide the data into groups based on single user weeks and select randomly from those groups in proportion to each group's size relative to the size of the whole dataset. This approach ensures that the distribution of the user variable among the different splits remains consistent. If there is too little data for a user, specifically less than three weeks that can be split up across train validation and test set, we remove the user entirely from the dataset.

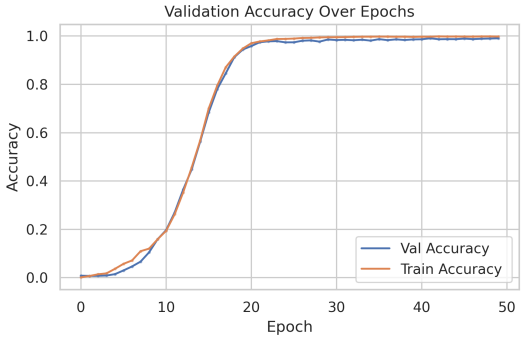
7.2.2 Setting Up the Training Pipeline and Overfitting on Synthetic Data

In this section, we describe the steps we used to debug the model and set up an infrastructure that efficiently trains and tests the model. After some initial trials with our processed data, where the performance could not surpass that of the simple baseline model,



d_model: 32
d_hid: 32
nlayers: 1
nhead: 2
dropout: 0
lr: 0.001

Figure 7.4: Training and validation loss on synthetic data with a 100 $\times$ 100 grid. The model learns quickly and converges after 20 epochs to an accuracy of 1 and a loss of 0. Since the synthetic data is very similar for both training and validation, there is barely any visible generalization gap.



d_model: 32
d_hid: 32
nlayers: 1
nhead: 2
dropout: 0
lr: 0.001

Figure 7.5: Loss and accuracy training on synthetic data with a 100 $\times$ 100 grid. The model learns quickly and converges after 20 epochs to an accuracy of 1 and a loss of 0. Since the synthetic data is very similar for both training and validation, there is barely any visible generalization gap.

we realized the need to debug the model and create a more effective training pipeline. Therefore, we took inspiration from the methodology outlined by Karpathy [68] to ensure that the model learns correctly.

Instead of debugging and experimenting with the complex real data, we established a training and evaluation pipeline using synthetic data. The synthetic data consisted of 100 users who visited 7 randomly selected locations from a 100 $\times$ 100 grid weekly (one location per day) for 100 days.

After multiple debugging steps and some updates in the code, we achieved that the model completely learns the repeating, but distinct trajectories for the 100 users (see Figure 7.5). This allowed us to return to the real training data. And as shown in Figure 7.6, the model now correctly trained on the data, as the training loss continuously decreased, while the validation loss stagnated at some point. This indicated that the model started to overfit on the training data and that we were ready to explore the different features and model parameters.

7.2.3 Model Parameters

This section outlines the crucial parameters selected for our model's training process and contrasts them with those employed in the training of the original BERT model [63]. Our approach involved opting for significantly smaller parameters. This decision was guided by the objective to achieve an optimal balance between training efficiency and

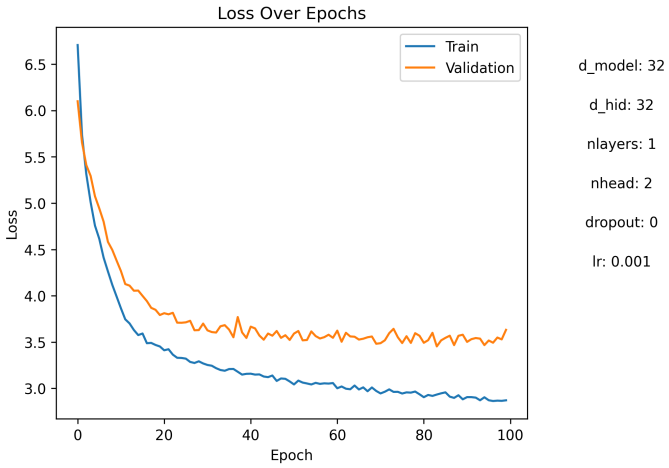


Figure 7.6: Exemplary training loss and accuracy on our processed data: By plotting the train and validation loss, we can ensure that the generalization gap, e.g., the difference between the performance on the training and the unseen validation data, does not become too big and that the model does not overfit. The fact that the training loss seems to decrease while the validation loss is stable suggests that further training would only lead to further overfitting.

model accuracy, allowing us to significantly reduce the training duration to approximately 1-2 hours. We aim to provide insights into the considerations and trade-offs involved in choosing the model architecture and parameters for training.

Parameter	Value chosen for training
`epochs`	100
`d_model`	32
`d_hid`	32
`nlayers`	1
`nhead`	2
`dropout`	0
`lr`	0.001

Epochs (`epochs = 100`): An epoch represents a single cycle through the full training dataset. Setting `epochs` to 100 instructs the model to use the training data 100 times to adjust its weights. This seemed to be a good balance between adequate learning (achieving convergence in the validation loss and accuracy curve) and preventing overfitting.

Model Dimension (`d_model = 32`): This parameter specifies the size of the input and output vectors in the transformer model, thereby influencing the dimensions of the embeddings and the sizes of the hidden layers. Finding the optimal embedding size is discussed in research [69] but is often determined by empirical studies, taking into account various factors. One significant factor is the vocabulary size, as a larger embedding size can enhance the model's capacity to capture information. *Machine Learning Design Patterns* recommends a heuristic approach: use either the fourth root of the number of unique categorical elements or an embedding dimension approximately 1.6 times the square root of

the number of unique category elements, ensuring a minimum dimension of 600 [70]. Our decision to set `d_model = 32` is based on the comparison to the original BERT model dimensions of 768 and 1024, which are significantly larger [63]. However, given our smaller training data size, a `d_model` of 32 seemed sufficient.

Hidden Dimension (`d_hid = 32`): This parameter determines the size of the hidden layers within each feedforward network in the model's architecture and influences the model's ability to learn complex abstractions. A `d_hid` value of 32 means that the intermediate representations in these networks will also have a dimensionality of 32. In contrast, the original BERT model employs a hidden dimension that is four times the model dimension, equating to either 3072 (4×768) or 4096 (4×1024) [63]. For our purposes, a hidden dimension of 32 was deemed sufficient.

Number of Layers (`nlayers = 1`): The `nlayers` parameter specifies how many encoder layers the model contains. The BERT-Base model utilizes 12 layers, while BERT-Large uses 24 [63]. Setting `nlayers` to 1 greatly simplifies the model, making it faster to train and more interpretable.

Number of Attention Heads (`nhead = 2`): Multiple attention heads allow the model to attend to different aspects of the input sequence simultaneously, as desrcibed in 5.1.3. With `nhead = 2`, the model splits its attention mechanism into two paths and might, for instance, be able to differentiate between routine and non-routine movements or distinguish between weekdays and weekends.

Dropout Rate (`dropout = 0`): Applying dropout can help in regularizing the model, encouraging it to learn more robust and generalizable patterns from the mobility data. Using a dropout rate of 0 suggests that all features and neurons are used in every training step. This leads to faster convergence since the model is simple and the dataset is not prone to overfitting. However, it might also risk the model learning noise or overfitting to the training data.

Learning Rate (`lr = 0.001`): The learning rate determines the size of the steps the model takes during optimization. A conservative learning rate of 0.001 ensurs gradual convergence to a solution.

7.3 Testing Procedure

We evaluate the model using three distinct comparison strategies:

1. We compare the accuracy of the baseline models against each other and against MOBERT on data organized into a 200×200 grid.
2. Furthermore, we assess the impact of three additional input features—time, rank,

and users—by comparing all possible combinations of these features on the 200 $\times$ 200 grid.

3. Lastly, we evaluate the accuracy of MOBERT without any additional features across different $n \times n$ grids, where $n = [50, 100, 200, 500, 1000]$.

Given that our test dataset comprises approximately 45,000 data samples over 1650 weeks, which is relatively sparse, and considering that token masking introduces some level of randomness, we iterate over the test loop ten times to accumulate more stable training results.

7.4 Summary

In this chapter, we described our experimental setup for assessing the performance of MOBERT. We outlined the process of training and testing the model, including comparisons with three baseline models. These baselines based on the rank popularity and transition probabilites, provided essential benchmarks for evaluating MOBERT's effectiveness.

Our training procedure involved stratified data splitting to ensure consistent representation of users across training, validation, and test sets while preserving the temporal order of the data. We also described the setup of the training pipeline and the steps taken to prevent overfitting using synthetic data for debugging purposes.

We selected the model parameters carefully to balance computational efficiency with model accuracy. These parameters included epoch count, model dimension, hidden dimension, number of layers, number of attention heads, dropout rate, and learning rate. Through iterative adjustments and experimentation, we established a setting conducive to effective model training.

The testing procedure involved three comparison strategies: assessing the accuracy of baseline models against MOBERT on a 200 $\times$ 200 grid, evaluating the impact of additional input features (time, rank, and users) on MOBERTs performance, and testing MOBERT across various grid sizes (ranging from 50 $\times$ 50 to 1000 $\times$ 1000).

8 Results

This section presents the performance outcomes of the MOBERT model variants tested for masked location prediction tasks. We utilized the data in the test split to compare the model variants with baseline models and across different grid sizes. The evaluation metrics include test loss and top-1 (ACC@1), top-5 (ACC@5), and top-10 (ACC@10) accuracies. These metrics represent the model's accuracy in predicting the exact next location (ACC@1), any of the top 5 predicted locations (ACC@5), and any of the top 10 predicted locations (ACC@10), respectively.

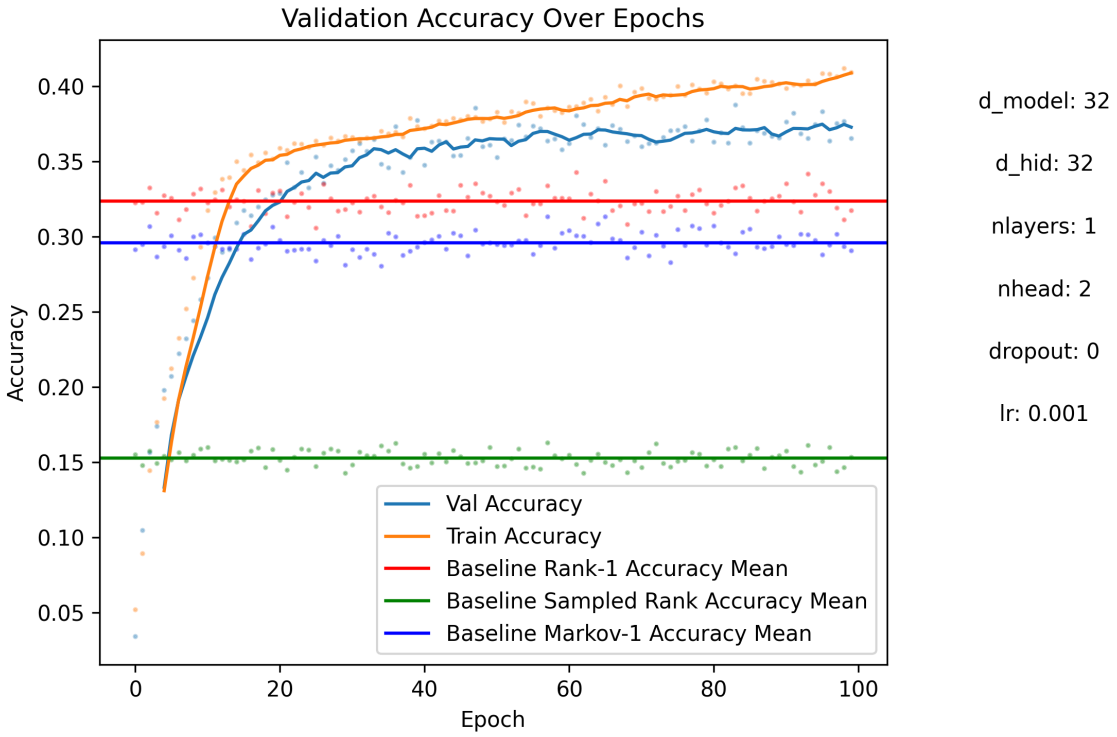


Figure 8.1: This plot demonstrates that MOBERT's accuracy surpasses that of the three baseline models when trained on the dataset with a 200 $\times$ 200 grid. The small size of the evaluation set, combined with the random masking, leads to noisy results. We mitigated this noise by plotting the rolling average mean.

8.0.1 Baseline Performance

We evaluated the baseline models on the 200 $\times$ 200 grid. Among these models, the Rank-1 model, being the simplest, achieves the highest top-1 accuracy (ACC@1). With an accuracy of 32.36%, it correctly predicts nearly a third of the locations. It is followed by the first-order Markov model with an accuracy of 29.57%, and the model that samples based on the rank with an accuracy of only 15.25%.

Model variant	Test loss	ACC@1	ACC@5	ACC@10
Simple	3.0054	0.3994	0.6931	0.7811
User	3.4832	0.3807	0.6555	0.7496
Rank	3.4976	0.3785	0.6788	0.7692
Time	3.0471	0.4064	0.6909	0.7793
Rank + Time	3.6757	0.3644	0.6291	0.7218
User + Rank	3.5456	0.3699	0.6521	0.7487
User + Time	3.5286	0.3665	0.6581	0.7490
User + Rank + Time	3.5354	0.3634	0.6513	0.7450

Table 8.1: Comparison of the input feature variants across different top-k accuracies.

The results indicate that always guessing the home location is a more effective strategy than adding randomness and sampling locations based on their past probability. This is unsurprising, as randomly chosen locations do not account for previously visited locations. For example, the model could predict a rank-1 location immediately following another rank-1 location, which, due to preprocessing, is not a viable pattern in the data. It is more surprising that the simple baseline model outperforms the more complex first-order Markov accuracy model, which considers the previously visited location. However, the inherent randomness in the data could explain why considering just the previous location does not give significant information about the next location.

Overall, we can conclude that the simplest baseline model performs the best and provides a robust benchmark for the transformer model. Nevertheless, all models are outperformed by MOBERT, as we will see in the following section.

8.0.2 Feature Embeddings

We compared all possible combinations of the three features: User, Time, and Rank, as well as the pure location data without any additional features (Simple). Figure 8.2 presents a comparison of ACC@1 and its distribution, whereas Table 8.1 details the test loss and the accuracies ACC@1, ACC@5, and ACC@10.

MOBERT's performance varies depending on the information provided to the model. Overall, all combinations of input features outperform the baseline models. The results suggest that simpler models tend to have better prediction accuracy. However, the model incorporating the Time feature as an additional input achieved the highest accuracy, at 40.64%. The other single additional features - Rank and User — however, reduce the accuracy by about 2%. Combinations of two or three additional features result in a test accuracy of 36 to 37 percent.

This trend is consistent across broader accuracy measures, ACC@5 and ACC@10, as detailed in Table 8.1: The version utilizing the Time feature and the version without any additional features achieve the best results, followed by the versions with single and lastly multiple additional features.

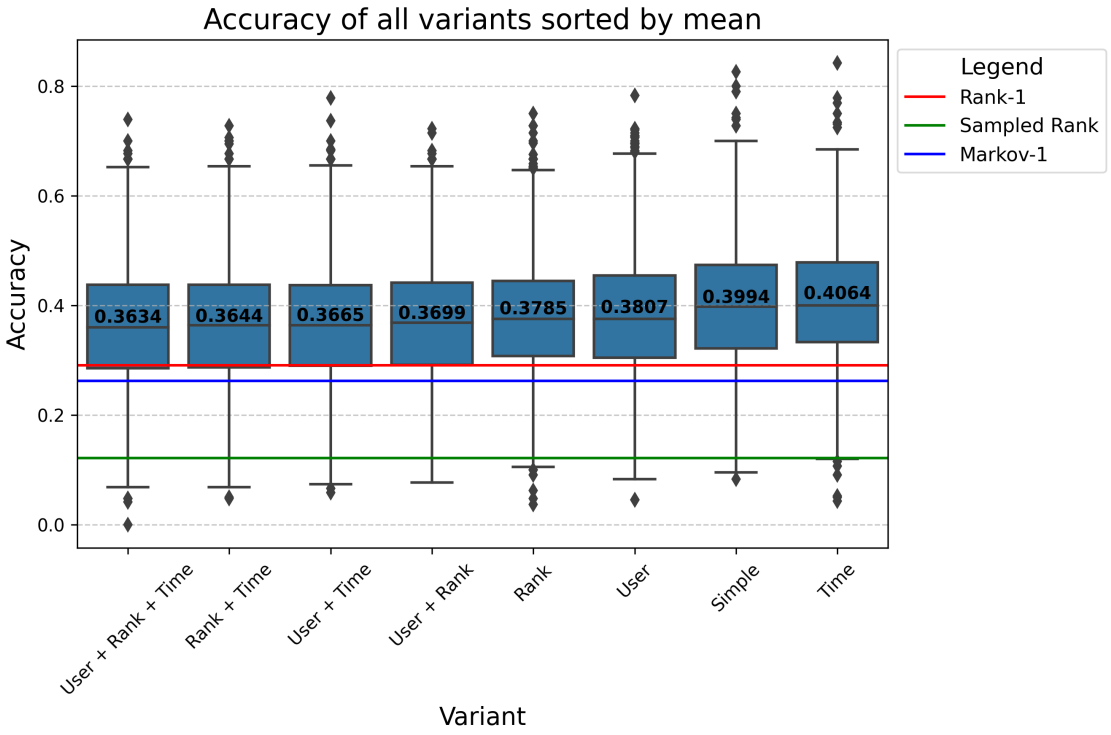


Figure 8.2: This plot illustrates the top-1 accuracy of MOBERT with various input feature variants. The boxplot displays the distribution of batchwise calculated accuracy values, where each value is derived from the prediction accuracy of 8 masked trajectories. Since the trajectories have different lengths and characteristics, the accuracy can vary significantly, leading to high variance.

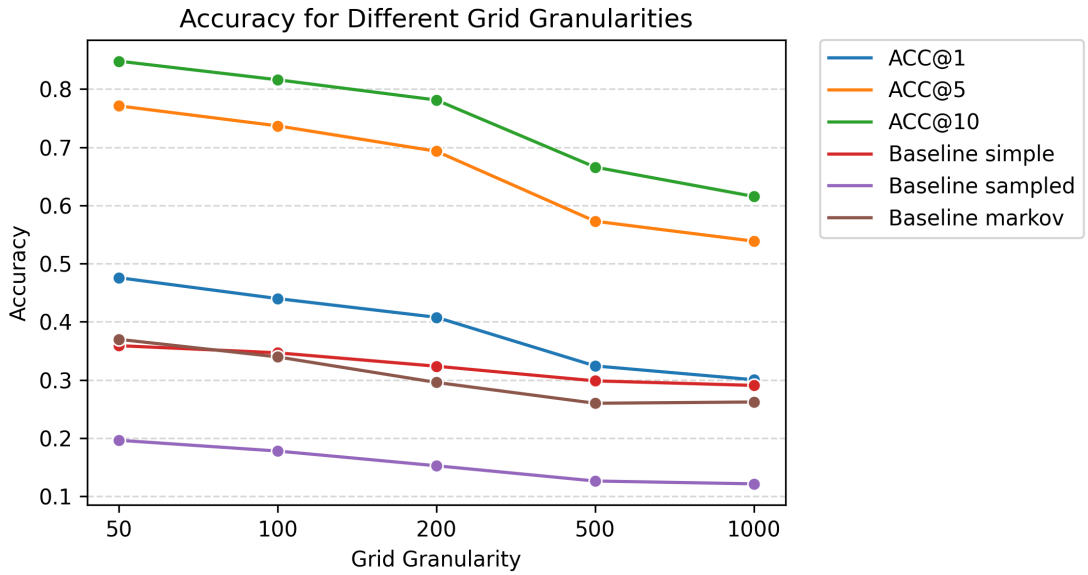


Figure 8.3: Accuracy Values for Varying Grid Granularities

8.0.3 Grid Sizes

In the final step, we evaluate the impact of grid size on prediction accuracy, measuring the top-1, top-5, and top-10 accuracies as well as the top-1 accuracy for the three baseline models:

Grid granularity	ACC@1	ACC@5	ACC@10	Rank-1	Sampled-Rank	Markov-1
50×50	0.4757	0.7710	0.8479	0.3589	0.1963	0.3698
100×100	0.4398	0.7367	0.8159	0.3466	0.1778	0.3396
200×200	0.4077	0.6931	0.7811	0.3236	0.1525	0.2957
500×500	0.3241	0.5728	0.6658	0.2984	0.1263	0.2599
1000×1000	0.3003	0.5385	0.6154	0.2906	0.1214	0.2621

The number of grid cells significantly affects accuracy, with a visible decrease in accuracy across all models and accuracy measures as grid granularity increases. However, the relationship between grid size and accuracy does not appear to be linear. For instance, the top-1 accuracy decreases from 40.77% for the 200 $\times$ 200 grid to 32.41% for the 500 $\times$ 500 grid, a reduction of approximately 20%, whereas the decrease from the 500 $\times$ 500 grid at 32.41% to the 1000 $\times$ 1000 grid at 30.03% is only about 8%. Furthermore, the baseline model that always predicts the rank-1 location appears to be more stable with respect to changes in grid size. On the 1000 $\times$ 1000 grid, it performs nearly as well as the MOBERT model, whereas on the 50 $\times$ 50 grid, it even underperforms compared to the first-order Markov model (see Figure 8.3).

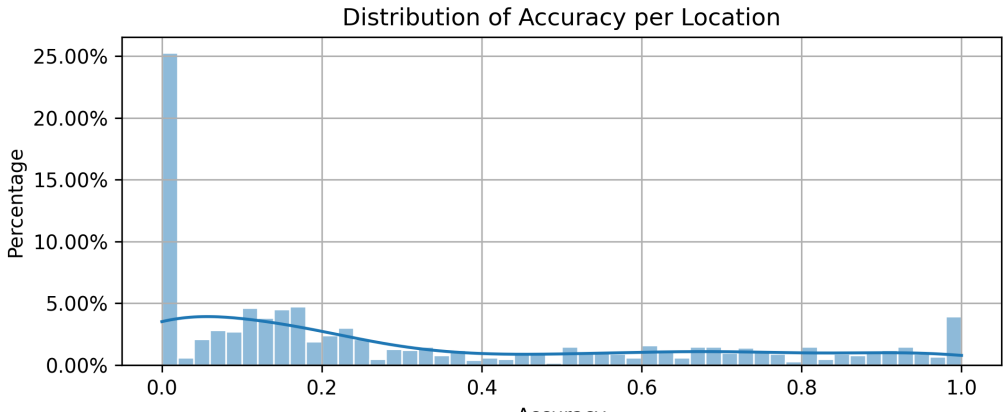


Figure 8.4: Distribution of the prediciton accuracy per location

8.0.4 Data Analysis: Accuracy for Users and Locations

We measured the top-1 accuracy for the MOBERT version without additional features with respect to the predicted locations. We considered only locations that were masked at least 5 times during the test to gain a more precise impression of the model's performance in predicting different locations. Figure 8.4 shows that the model fails to correctly predict a large number of locations at all. However, except for an increased number of locations with accuracy between 10 and 30 percent, as well as locations that are predicted correctly all the time, the accuracies appear uniformly distributed across locations between 30% and 92%. Figure 8.5 provides quantitative information on how often a grid cell location has been predicted, as well as qualitative information on how often the location has been predicted correctly for each grid cell. Areas that have been most often predicted include the DTU campus and places around it. It is visible that prediction accuracy is low in the inner city areas of Copenhagen and Lyngby, as well as on the DTU campus in Lyngby (violet points). Better accuracy results are achieved outside of the center in mixed residential areas, such as in Amager, Nørrebro, Vesterbro, Østerbro, or in places around the DTU campus in Lyngby (green points). The best results, however, are achieved in residential areas like Nordvest, Søborg, or Vanløse (yellow points).

This result might prompt the question of whether there is a correlation between the accuracy for certain grid cells and the number of unique users who have visited the grid cell. Figure 8.6 indicates that for cells visited by 1 to around 50 users, there is no observable correlation, as the accuracy values appear uniformly distributed. Besides a few locations with accuracy 1, there are also many locations with accuracy 0. However, for the cells that have been visited by approximately 50 or more users, there seems to be a negative correlation, suggesting that the more users visit a location, the lower the prediction accuracy becomes. On the other hand, those locations with many users also have accuracy values of 0 or 1. The number of occurrences of locations does not seem to follow a pattern, as places with many occurrences are evenly distributed across the plot.

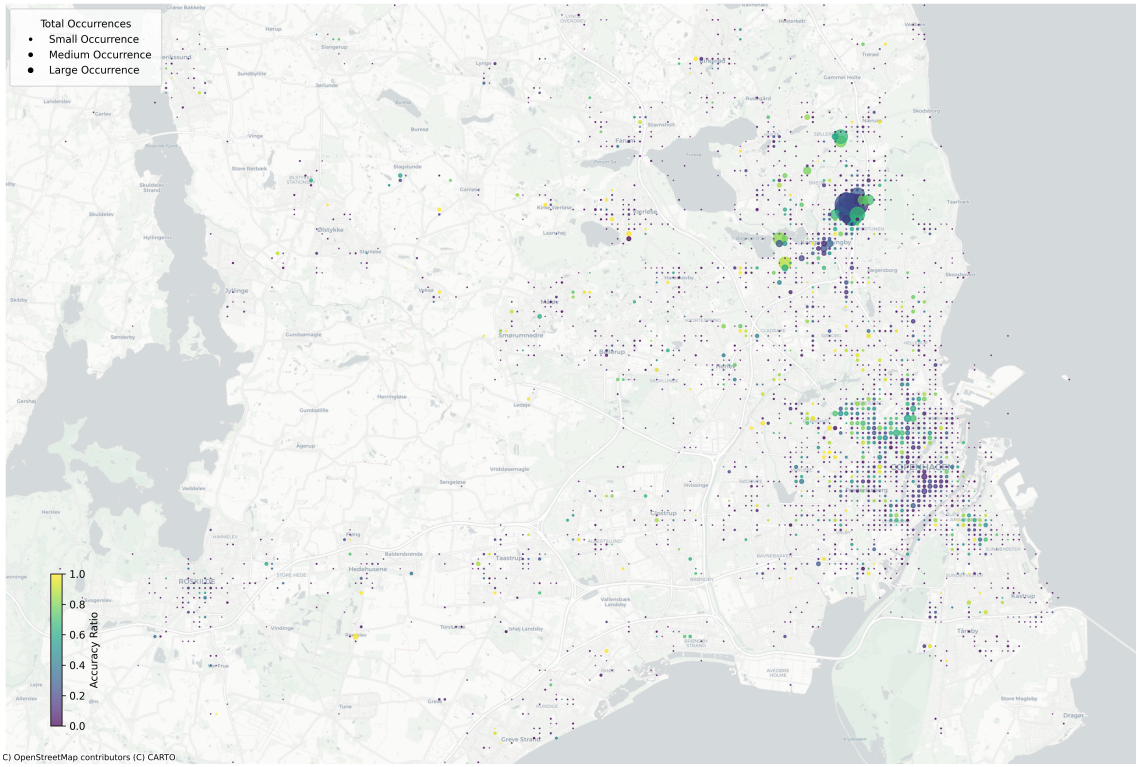


Figure 8.5: Prediction accuracy per location: This map displays the grid cells in the test set that have been predicted at least 5 times. The colour indicates the accuracy, the size the relative number of occurences (the smallest cells were predicted 5 times, the largest on the DTU campus 1253 times.

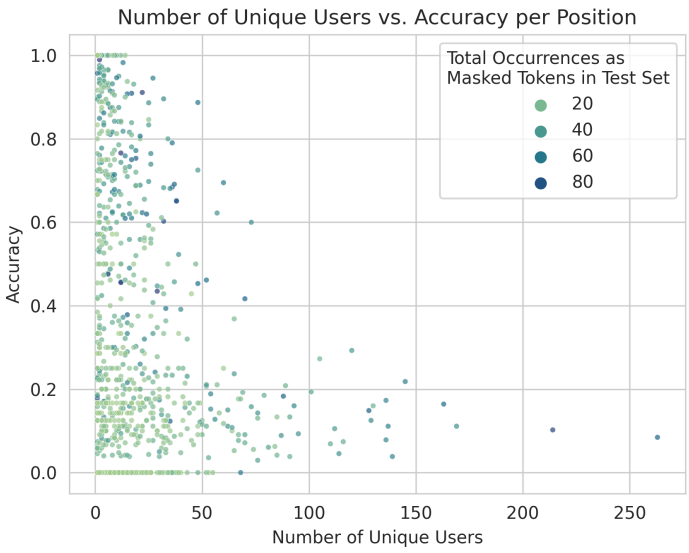


Figure 8.6: This figure illustrates the correlation between the number of distinct users visiting a grid cell and the accuracy achieved by the model in predicting it. The color represents the frequency of predictions for each grid cell. It is evident that locations visited by a smaller number of distinct users tend to have higher prediction accuracy. Conversely, places visited by a large number of distinct users are often incorrectly predicted.

8.1 Summary

In this section, we presented the results of our evaluation of MOBERT for predicting the masked location. We compared MOBERT's ability to predict the next location against several baseline models, focusing on accuracy across different grid sizes.

We first assessed the performance of baseline models on a 200 x 200 grid. The simplest model, Rank-1, achieved the highest top-1 accuracy, correctly predicting nearly a third of the locations. This finding suggested that a straightforward approach of always guessing the most frequent location could be more effective than models adding randomness or relying on past locations.

We then explored MOBERT's performance with the different input features user, time and rank. We found that adding the time feature improved accuracy slightly, while combinations of multiple features did not enhance performance as expected which suggests a potential overcomplication of the data.

We also observed how grid size affected accuracy and discovered that accuracy declined with increasing grid granularity, yet the decrease was not linear across all models.

Lastly, our investigation into MOBERT's accuracy across different locations showed showed that locations that are regularly visited by only few users are generally easier to predict than locations that are shared by many users or only sporadically visited.

9 Discussion and Conclusion

We explored the application of a custom BERT model for masked location prediction. We trained and evaluated the model on our custom dataset, which we preprocessed and condensed into an $N \times N$-grid. Thus, the regression task to determine a specific location became a classification task with $N \times N$ possible classes.

With a top-1 prediction accuracy ranging from 30% to 47%, depending on the grid size, the model clearly outperforms all three baseline models as well as the existing next-location prediction approaches discussed in section 2.2.6. However, due to the bidirectional approach and the use of different datasets, a direct comparison seems not feasible.

We studied the effect of different additional features like information about time, rank, and the user, and found that the features had barely any positive impact on the prediction accuracy. On the contrary, more added features led to worse prediction performance. However, we did not fine-tune the hyperparameters to see if different parameters or a longer training time could change the results to better performance for the feature-heavy model variants. Due to the increased amount of information, we cannot exclude with certainty the possibility that a more complex model architecture or a longer training time with more epochs could lead to better results. Interestingly, the `time` feature was the exception, as it did not reduce performance and even appeared to improve it. This suggests that the model may have implicitly learned distinctions between users and recognized popular locations for each user based on the extensive training data. Thus, adding explicit information about user ID and location rank may not provide additional insights. However, `time`, which the model could not learn implicitly, likely offered new, valuable information not otherwise inferable from the data.As expected, the results improved when increasing the number of accepted locations and measuring top-5 or top-10 accuracy. Still, relatively to the variants, the performance barely changed. This suggests that the additional features do not significantly alter the model's internal heuristics for selecting the most probable locations.

We also examined the effect of grid cell size on location grouping. As expected, a finer grid resulting in more prediction classes decreased accuracy, while a coarser grid with fewer classes increased it. Somewhat unexpectedly, the reduction in prediction accuracy appeared to diminish with an increasing number of grid cells, suggesting that accuracy may eventually converge to a base value. This base value would represent the model's accuracy if each location were treated as a distinct class.

If we take a closer look at the model's performance on a finer scale, we observe that the model excels at predicting locations shared by only a few people, such as those in residential areas, but often fails at accurately predicting locations frequented by many people,

such as inner city districts or the DTU campus in Lyngby. However, there are also many places with few visitors that have been predicted with poor accuracy. This observation leads to two conclusive points: First, the model is adept at learning locations that are specific to a user but struggles with learning locations that are common in the patterns of many other users. When translating this into language modeling, it becomes apparent that words used in a wide variety of contexts are difficult to learn and to understand. Second, the model is proficient at learning repetitive and exploitative mobility patterns, such as those associated with a home or school location (note that the locations on the DTU campus had much better accuracy compared to locations in the inner city, even though they were more frequently visited), but is less effective at dealing with explorative patterns, such as wandering through the inner city or visiting specific locations that haven't been visited before.

9.1 Limitations and Future Work

The theoretical predictability of 66% derived in Section 3.2 can hardly be compared to the significantly lower top-1 accuracy results achieved with our model, since the theoretical predictability is based solely on past mobility patterns, whereas our model also had access to future data, which we expected to result in higher accuracy values. Still, several limitations in our approach may have hindered achieving higher accuracy.

First, our final dataset, with 170,340 data points in the training set (equivalent to 7,131 weeks), was relatively sparse. This limitation saved training time as well as computational ressources and enabled the use of a sparse model architecture. However, it also raises two questions:

- What improvements can be achieved by tuning the parameters and altering the model architecture to capture more complex information?

- What results could be attained by providing more data? This could be achieved either by less rigorous filtering of the same data and, most importantly, allowing for a longer time span than we did, or by utilizing a different dataset.

Second, our results are hardly comparable with those outlined in section 2.2.6 as different datasets were used. Datasets that were used for comparable approaches might have been collected in very different ways and might not contain the features of the dataset we used. MOBERT was tailored to our specific data preprocessing and sequence processing methods. Furthermore, its prediction results were evaluated on a bidirectional masked prediction task. Translating this into a next-location prediction task would require an additional iteration of fine-tuning the model on a downstream task or only masking the last token in the input sequence. However, this would also open the door to trajectory generation, enabling more comprehensive performance assessment and further utilities.

9.1.1 Privacy

Another aspect we scarcely addressed is privacy. It is possible to trace back individuals based on their trajectory even after anonymizing the data [71]. Simply removing private features from the data does not ensure privacy, as advancements in trajectory user linking (TUJ) make it possible to efficiently identify users by their trajectories [72].

Therefore, future work needs to focus on making privacy-aware predictions that do not violate individual privacy. One of the main challenges will be balancing the trade-off between the utility and privacy of the model [48]. Fontana et al. present a framework that employs deep learning methods for user identification, safeguarding the privacy of synthetically generated or predicted locations without lowering the quality of the GAN responsible for next location prediction [73]. Such privacy-aware approaches should be further explored for modeling with transformers.

9.2 Conclusion

In this thesis, we evaluated a BERT-based model for next location prediction, focusing on its ability to process mobility data bidirectionally. Our findings indicate that the model effectively predicts locations, outperforming all three baseline models and achieving superior results compared to existing next location prediction approaches. Notably, the inclusion of additional features, such as time and location rank, did not significantly enhance the model's performance, suggesting its inherent capability to discern user patterns and identify popular locations.

We want to highlight two main insights: the model excels in learning patterns for locations unique to individual users but faces challenges with locations common to many users. It performs well with repetitive mobility patterns but less so with unpredictable explorative patterns.

Notable limitations of this thesis stems from the data processing approach we adopted, which resulted in a relatively sparse dataset. Additional challenges include the difficulty in comparing our results with other studies due to the different datasets used across studies and our bidirectional approach, which differs from the one-directional next location prediction task. We suggest future research focus on parameter optimization, employing more comprehensive datasets, and fine-tuning to a next location prediction task. Furthermore, enhancing privacy while maintaining utility remains a critical area for future exploration.

Bibliography

[1] Juha K Laurila et al. *The mobile data challenge: Big data for mobile computing research*. Tech. rep. 2012.

[2] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. *CRAWDAD data set epfl/mobility (v. 2009-02-24)*. 2009.

[3] Luca Pappalardo et al. "Understanding the patterns of car travel". In: *The European Physical Journal Special Topics* 215 (2013), pp. 61–73.

[4] Riccardo Gallotti et al. "Entropic measures of individual mobility patterns". In: *Journal of Statistical Mechanics: Theory and Experiment* 2013.10 (2013), P10022.

[5] Hugo Barbosa et al. "Human mobility: Models and applications". In: *Physics Reports* 734 (2018), pp. 1–74.

[6] Justine I Blanford et al. "Geo-located tweets. Enhancing mobility maps and capturing cross-border movement". In: *PloS one* 10.6 (2015), e0129202.

[7] Yan Shi et al. "A survey of hybrid deep learning methods for traffic flow prediction". In: *Proceedings of the 2019 3rd international conference on advances in image processing*. 2019, pp. 133–138.

[8] Xuan Song et al. "Prediction of human emergency behavior and their mobility following large-scale disaster". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 5–14. ISBN: 9781450329569. DOI: 10.1145/2623330.2623628. URL: https://doi.org/10.1145/2623330.2623628.

[9] George Grekousis and Ye Liu. "Where will the next emergency event occur? Predicting ambulance demand in emergency medical services using artificial intelligence". In: *Computers, Environment and Urban Systems* 76 (2019), pp. 110–122.

[10] Luca Pappalardo et al. "An analytical framework to nowcast well-being using mobile phone data". In: *International Journal of Data Science and Analytics* 2 (2016), pp. 75–92.

[11] Luca Canzian and Mirco Musolesi. "Trajectories of depression: unobtrusive monitoring of depressive states by means of smartphone mobility traces analysis". In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 2015, pp. 1293–1304.

[12] Moritz U. G. Kraemer et al. "The effect of human mobility and control measures on the COVID-19 epidemic in China". In: *Science* 368.6490 (2020), pp. 493–497. DOI: 10.1126/science.abb4218. eprint: https://www.science.org/doi/pdf/10.1126/science.abb4218. URL: https://www.science.org/doi/abs/10.1126/science.abb4218.

[13] Nuria Oliver et al. "Mobile phone data for informing public health actions across the COVID-19 pandemic life cycle". In: *Science Advances* 6.23 (2020), eabc0764. DOI:

10.1126/sciadv.abc0764. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.abc0764. URL: https://www.science.org/doi/abs/10.1126/sciadv.abc0764.

[14] Wen-Yuan Zhu et al. “Modeling user mobility for location promotion in location-based social networks”. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1573–1582.

[15] Massimiliano Luca et al. *A Survey on Deep Learning for Human Mobility*. 2021. arXiv: 2012.02825 [cs.LG].

[16] Ingrid Burbey and Thomas L Martin. “A survey on predicting personal mobility”. In: *International Journal of Pervasive Computing and Communications* 8.1 (2012), pp. 5–22.

[17] Zeinab Ebrahimpour et al. “Comparison of main approaches for extracting behavior features from crowd flow analysis”. In: *ISPRS International Journal of Geo-Information* 8.10 (2019), p. 440.

[18] Francesco Calabrese, Giusy Di Lorenzo, and Carlo Ratti. “Human mobility prediction based on individual and collective geographical preferences”. In: *13th international IEEE conference on intelligent transportation systems*. IEEE. 2010, pp. 312–317.

[19] Daniel Ashbrook and Thad Starner. “Learning significant locations and predicting user movement with GPS”. In: *Proceedings. Sixth International Symposium on Wearable Computers,* IEEE. 2002, pp. 101–108.

[20] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. “Show me how you move and I will tell you who you are”. In: *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*. SPRINGL ’10. San Jose, California: Association for Computing Machinery, 2010, pp. 34–41. ISBN: 9781450304351. DOI: 10.1145/1868470.1868479. URL: https://doi.org/10.1145/1868470.1868479.

[21] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[22] David E. Rumelhart and James L. McClelland. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[23] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ArXiv* 1409 (Sept. 2014).

[25] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-attention with relative position representations”. In: *arXiv preprint arXiv:1803.02155* (2018).

[26] Jan K Chorowski et al. “Attention-based models for speech recognition”. In: *Advances in neural information processing systems* 28 (2015).

[27] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. “Multi-way, multilingual neural machine translation with a shared attention mechanism”. In: *arXiv preprint arXiv:1601.01073* (2016).

[28] Qiang Wang et al. *Learning Deep Transformer Models for Machine Translation*. 2019. arXiv: 1906.01787 [cs.CL].

[29] Dmitrii Aksenov et al. *Abstractive Text Summarization based on Language Model Conditioning and Locality Modeling*. 2020. arXiv: 2003.13027 [cs.CL].

[30] Imen Akermi, Johannes Heinecke, and Frédéric Herledan. "Transformer based Natural Language Generation for Question-Answering". In: *Proceedings of the 13th International Conference on Natural Language Generation*. Ed. by Brian Davis et al. Dublin, Ireland: Association for Computational Linguistics, Dec. 2020, pp. 349–359. DOI: 10.18653/v1/2020.inlg-1.41. URL: https://aclanthology.org/2020.inlg-1.41.

[31] Salman Khan et al. "Transformers in Vision: A Survey". In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3505244. URL: https://doi.org/10.1145/3505244.

[32] Lili Chen et al. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 15084–15097. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

[33] Alexandre De Brébisson et al. "Artificial neural networks applied to taxi destination prediction". In: *arXiv preprint arXiv:1508.00021* (2015).

[34] Luis Moreira-Matias et al. "Predicting taxi–passenger demand using streaming data". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1393–1402.

[35] Yile Chen et al. "Context-aware deep model for joint mobility and time prediction". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 106–114.

[36] Qiang Liu et al. "Predicting the next location: A recurrent model with spatial and temporal contexts". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[37] Dingqi Yang et al. "Location prediction over sparse user mobility traces using rnns". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. 2020, pp. 2184–2190.

[38] Eunjoon Cho, Seth A Myers, and Jure Leskovec. "Friendship and mobility: user movement in location-based social networks". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pp. 1082–1090.

[39] GTD START. *Global terrorism database*. 2017.

[40] Dejiang Kong and Fei Wu. "HST-LSTM: A hierarchical spatial-temporal long-short term memory network for location prediction." In: *IJCAI*. Vol. 18. 7. 2018, pp. 2341–2347.

[41] Alberto Rossi et al. "Modelling taxi drivers' behaviour for the next destination prediction". In: *IEEE Transactions on Intelligent Transportation Systems* 21.7 (2019), pp. 2980–2989.

[42] *Greater New York Taxi Association v. New York City Taxi and Limousine Commission*. 2015.

[43] Di Yao et al. “Serm: A recurrent model for next location prediction in semantic trajectories”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 2411–2414.

[44] Chao Zhang et al. “Splitter: Mining fine-grained sequential patterns in semantic trajectories”. In: *Proceedings of the VLDB Endowment* 7.9 (2014), pp. 769–780.

[45] Chao Zhang et al. “Gmove: Group-level mobility modeling using geo-tagged social media”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1305–1314.

[46] Yi Bao et al. “A BiLSTM-CNN model for predicting users’ next locations based on geotagged social media”. In: *International Journal of Geographical Information Science* 35.4 (2021), pp. 639–660.

[47] Jinjun Tang et al. “Trip destination prediction based on a deep integration network by fusing multiple features from taxi trajectories”. In: *IET Intelligent Transport Systems* 15.9 (2021), pp. 1131–1141.

[48] Jinmeng Rao et al. *LSTM-TrajGAN: A Deep Learning Approach to Trajectory Privacy Protection*. 2020. arXiv: 2006.10521 [cs.LG].

[49] Qiang Gao et al. “Predicting human mobility via variational attention”. In: *The world wide web conference*. 2019, pp. 2750–2756.

[50] Jie Feng et al. “Deepmove: Predicting human mobility with attentional recurrent networks”. In: *Proceedings of the 2018 world wide web conference*. 2018, pp. 1459–1468.

[51] 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 (2020), p. 17.

[52] Riccardo Corrias, Martin Gjoreski, and Marc Langheinrich. “Exploring Transformer and Graph Convolutional Networks for Human Mobility Modeling”. In: *Sensors* 23.10 (2023). ISSN: 1424-8220. DOI: 10.3390/s23104803. URL: https://www.mdpi.com/1424-8220/23/10/4803.

[53] Dingqi Yang et al. “Revisiting User Mobility and Social Relationships in LBSNs: A Hypergraph Embedding Approach”. In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 2147–2157. ISBN: 9781450366748. DOI: 10.1145/3308558.3313635. URL: https://doi.org/10.1145/3308558.3313635.

[54] Chaoming Song et al. “Limits of Predictability in Human Mobility”. In: *Science* 327.5968 (2010), pp. 1018–1021. DOI: 10.1126/science.1177170. eprint: https://www.science.org/doi/pdf/10.1126/science.1177170. URL: https://www.science.org/doi/abs/10.1126/science.1177170.

[55] Miao Lin, Wen-Jing Hsu, and Zhuo Qi Lee. “Predictability of Individuals’ Mobility with High-Resolution Positioning Data”. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp ’12. Pittsburgh, Pennsylvania: Association for

Computing Machinery, 2012, pp. 381–390. ISBN: 9781450312240. DOI: 10.1145/2370216.2370274. URL: https://doi.org/10.1145/2370216.2370274.

[56] Gavin Smith et al. “A refined limit on the predictability of human mobility”. In: *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2014, pp. 88–94. DOI: 10.1109/PerCom.2014.6813948.

[57] R.K. Pathria and Paul D. Beale. “3 - The Canonical Ensemble”. In: *Statistical Mechanics (Third Edition)*. Ed. by R.K. Pathria and Paul D. Beale. Third Edition. Boston: Academic Press, 2011, pp. 39–90. ISBN: 978-0-12-382188-1. DOI: https://doi.org/10.1016/B978-0-12-382188-1.00003-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123821881000037.

[58] I. Kontoyiannis et al. “Nonparametric entropy estimation for stationary processes and random fields, with applications to English text”. In: *IEEE Transactions on Information Theory* 44.3 (1998), pp. 1319–1327. DOI: 10.1109/18.669425.

[59] Arkadiusz Stopczynski et al. “Measuring large-scale social networks with high resolution”. In: *PloS one* 9.4 (2014), e95978.

[60] Feb. 2024. URL: https://en.wikipedia.org/wiki/Unix_time.

[61] Ulf Aslak and Laura Alessandretti. *Infostop: Scalable stop-location detection in multi-user mobility data*. Mar. 2020.

[62] Ludvig Bohlin et al. “Community Detection and Visualization of Networks with the Map Equation Framework”. In: Sept. 2014, pp. 3–34. ISBN: 978-3-319-10376-1. DOI: 10.1007/978-3-319-10377-8_1.

[63] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[64] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[65] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global Vectors for Word Representation”. In: vol. 14. Jan. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[66] Alexander Wettig et al. “Should you mask 15% in masked language modeling?” In: *arXiv preprint arXiv:2202.08005* (2022).

[67] Raghunath Arnab. “Chapter 7 - Stratified Sampling”. In: *Survey Sampling Theory and Applications*. Ed. by Raghunath Arnab. Academic Press, 2017, pp. 213–256. ISBN: 978-0-12-811848-1. DOI: https://doi.org/10.1016/B978-0-12-811848-1.00007-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128118481000078.

[68] Andrej Karpathy. *A Recipe for Training Neural Networks — karpathy.github.io*. https://karpathy.github.io/2019/04/25/recipe/. [Accessed 24-02-2024].

[69] Weiwei Gu et al. “Principled approach to the selection of the embedding dimension of networks”. In: *Nature Communications* 12 (June 2021), p. 3772. DOI: 10.1038/s41467-021-23795-5.

[70] V. Lakshmanan, S. Robinson, and M. Munn. *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps*.

O'Reilly, 2020. ISBN: 9781098115784. URL: https://books.google.dk/books?id=Y52uzQEACAAJ.

[71] Qiang Gao et al. "Identifying Human Mobility via Trajectory Embeddings". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 1689–1695. DOI: 10.24963/ijcai.2017/234. URL: https://doi.org/10.24963/ijcai.2017/234.

[72] Fan Zhou et al. "Trajectory-User Linking via Variational AutoEncoder". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 3212–3218. DOI: 10.24963/ijcai.2018/446. URL: https://doi.org/10.24963/ijcai.2018/446.

[73] Ivan Fontana, Marc Langheinrich, and Martin Gjoreski. "GANs for Privacy-Aware Mobility Modeling". In: *IEEE Access* 11 (2023), pp. 29250–29262. DOI: 10.1109/ACCESS.2023.3260981.

A MOBERT - Implementation and Figures

The code that implements the model and produces the plots in the thesis can be found in this repository.

B Evaluation Metrics

Evaluation metrics for regression

Kullback-Leibler (KL) divergence: The KL divergence quantifies how one probability distribution diverges from a reference distribution. It is defined for two discrete probability distributions P and Q over the same probability space X as:

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

This formula calculates the expectation of the logarithmic difference between the probabilities P and Q, where the expectation is taken over the probabilities of P. The KL divergence is always non-negative ($D_{KL}(P \parallel Q) \geq 0$) and asymmetric, indicating $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$. It equals zero ($D_{KL}(P \parallel Q) = 0$) when P and Q are identical.

Jensen-Shannon (JS) divergence: The JS divergence offers an alternative. It provides a symmetric and bounded measure of similarity between two probability distributions. It is defined as:

$$D_{JS}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M)$$

where $M = \frac{1}{2}(P + Q)$ is the average of the distributions P and Q. This measure's symmetry ($D_{JS}(P \parallel Q) = D_{JS}(Q \parallel P)$) and its bounded range $[0, 1]$ make it valuable for assessing the performance of generative mobility models.

These measures provide a rigorous framework for comparing the statistical properties of generated trajectories against those derived from real-world mobility data, enabling quantitative assessment of the models' realism and accuracy in replicating human mobility patterns.

Abstract

Understanding, modeling, and predicting human mobility in urban areas is an essential task for various domains such as transportation modelling, disaster risk management, and infectious disease spreading. In this thesis, we introduce a custom BERT-based model, MOBERT, designed for human mobility modelling. Trained and tested on a processed dataset derived from the Copenhagen Network Study, which captures location data from smartphones of 840 individuals, MOBERT effectively predicts masked locations, outperforming three baseline models and achieving superior results compared to existing-next location prediction approaches. Our analysis of the impact of additional features, including user ID, time, and location rank, on prediction accuracy showed no significant improvements. Furthermore, we evaluated the model's performance across different grid sizes and location types, emphasizing its proficiency in learning individual-specific regular patterns while highlighting challenges with explorative mobility patterns. Further limitations include the difficulty in comparing our results with other studies due to the different datasets used across studies and our bidirectional approach, which differs from the one-directional next location prediction task. We suggest future research focus on further parameter optimization, employing more comprehensive datasets, and fine-tuning to a next location prediction task. Furthermore, enhancing privacy while maintaining utility remains a critical area for future exploration.

Technical
University of
Denmark

Asmussens Alle, Build. 322
2800 Kgs. Lyngby
Tlf. 4525 1700

www.compute.dtu.dk