Effective Trajectory Imputation using Simple Probabilistic Language Models

by

Hayat Sultan Mohammed

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Trajectory data analysis refers to the systematic exploration of spatial and temporal movement patterns in trajectory datasets. Missing trajectory points pose a challenge as they affect downstream tasks that rely on these datasets, such as public transportation management, wildlife monitoring, and urban planning. Diverse approaches, from statistical methods to deep learning models, have been proposed to address the issue of missing data points in trajectories. In this thesis, we explore two approaches based on relatively simple probabilistic language models in order to address this problem, also known as trajectory imputation. Using a grid-based representation, we assign tokens to each point in a trajectory, representing each trajectory as a sequence of tokens akin to a *sentence* in natural language. This allows the application of language models for predicting missing points (tokens). Our experiments using a real dataset of over 200,000 taxi trips show that we can fill gaps of up to 2km with 85% precision. Furthermore, compared to large language models, probabilistic language models for imputing trajectory points offer a much simpler technique and enhance the interpretability of the results.

Acknowledgements

I am deeply grateful to my amazing supervisors Prof. Mario A. Nascimento and Prof. Denilson Barbosa for their unparalleled support, guidance, and always insightful input that never fails to bring a touch of humor. Thank you both for believing in me, for showing me the way, and for making this journey far more enjoyable than I thought possible.

Thank you to my wonderful parents Sultan M. Alya and Remla H. Alketa. Through you both, I always had the perfect balance of motivation to reach higher and support when things fell through. My siblings Nebiyu, Semir and especially my sister Muna, thank you for always being there, and going through all the highs and lows of the past few years with me.

Last but not least, I would like to thank Dr. Ildar Akhmetov for pushing me to believe in myself when I could not find it in me to do so.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Trajectory data, consisting of ordered and often time-stamped positional records such as GPS coordinates [31], plays a vital role in various domains, including traffic analysis [25], public transportation management [27], wildlife monitoring [1], and urban planning [10]. The effectiveness of these applications heavily relies on the accuracy and completeness of the trajectories. However, trajectories can sometimes be incomplete.

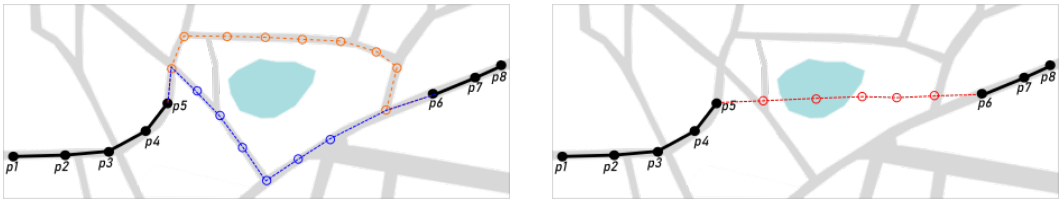
Incomplete trajectories may come in various forms. One of these is instances where a significant gap appears within the trajectory (Figure 1.1a). This gap might result from issues such as weak GPS signals or data being sampled at a low rate, where the points in the trajectory are not close enough to provide a complete and accurate information of the path (Figure 1.1b). Such gaps may obscure critical portions of the journey. Another scenario involves situations where only the starting and ending points of a trajectory are available such as in instances where individuals seek route suggestions based on their origin point and desired destination (Figure 1.1c).

(a) (b) (c)

Figure 1.1: Examples of incomplete trajectories ("missing" information shown with a dahsed line) (a) a trajectory with a significant gap between point *p5* and *p6* (b) a trajectory sampled at a low rate, and (c) a trajectory with only origin and destination point.

Technically, only the first scenario, i.e., gaps caused by weak GPS signals encountered during the trajectory (Figure 1.1a), can be strictly referred to as an incomplete trajectory. However, we extend the definition to encompass all the instances shown in Figure 1.1. We do so because the approach we use can be effectively applied to address all these instances.

While these scenarios differ slightly, they all either require or stand to benefit from the completion of the trajectories by inserting artificial (predicted) points – a process often referred to as *trajectory imputation*. Of course, it is crucial to ensure these imputed points are as close to "natural" points (i.e., points that were observed in previous trajectories) as possible (Figure 1.2a), and that they are not placed in inaccessible areas, such as water bodies, parks and residential areas in urban applications (Figure 1.2b), or obstacles like steep rocks or holes in wildlife applications.



(a) (b)

Figure 1.2: (a) imputed points (shown in *blue* and *orange*) on (or very close to) the actual road network (b) imputed points (shown in *red*) in inaccessible areas

One can approach this trajectory completion problem from two angles: using the existing road network map of the area, or not. While the former simplifies the task, it brings its own set of challenges. It can be prone to errors caused by factors such as inaccurate or outdated map data, or even be infeasible due to the complete absence of map data. Hence, it is important to be able to complete trajectories independently of road network information.

1.1 Trajectory Data and Natural Language

Trajectory data and natural language text share a similar attribute: an inherent order and structure that can be leveraged to extract meaningful insights. While trajectories are ordered sequences of points in space, text sentences are ordered sequences of words. This similarity opens up opportunities for the application of language modeling techniques to address challenges related to trajectories.

However, before applying natural language techniques to trajectory data, it is important to assess the challenges that prompted the advancements in language modeling techniques, particularly when it comes to long-term dependency, and determine if these challenges also hold in trajectory data. This assessment should serve as the basis for selecting the appropriate language modeling technique, rather than simply adopting the latest trend.

1.2 Problem Definition and Thesis Statement

A trajectory is an ordered sequence of points in space. We say that a trajectory has a *gap* of length d if there are two consecutive points in the trajectory such that their Euclidean distance is at least d. The trajectory imputation problem is to augment the trajectory with artificial (predicted) points in a way that no gaps are left.

In this thesis, we demonstrate the effectiveness of using relatively simple language modeling techniques to trajectory data to address the trajectory imputation problem.

1.3 Existing Solutions

The need to accurately impute points in incomplete trajectories has led to the exploration of a wide range of solutions. Most of these approaches require access to road network information [2][7][21][22].

Elsherif et al. [5] claim to be the first to perform imputation without relying on road network information, using a geometry-based approach that

completes trajectories by considering factors such as angles and speed. While their method, which adopts a "crowd wisdom" approach to avoid following paths no one is taking, greatly outperforms linear interpolation, it often fails to place imputed points accurately on the actual road network. In contrast, our approach performs well in this regard, with most imputed points closely aligning with the real road networks.

In recent years, the remarkable success of large language models, with their widespread application across diverse domains, has naturally extended to trajectory data, offering new paths for research. Musleh et al. [14], envision a unified deep model for trajectory analysis, and in their subsequent work introduce KAMEL [15], wherein they adopt BERT [3], a prominent example of a large language model, for trajectory imputation. Their work has served as a primary motivation for us to explore the use of language models for trajectory imputation. However, in this thesis, we demonstrate that our simpler and more explainable language model is just as effective for trajectory completion.

1.4 Our Approach

Our approach named Probabilistic Language Models for Trajectory Imputation (PaLMTo) begins by converting trajectories into *sentences* using a grid-based approach. Following this, we generate N-grams from these sentences, breaking the text into contiguous sequences of N words. We use these N-grams to predict the most probable tokens for trajectory completion. In other words, we use the N-grams to determine the most probable next token, given the preceding N-1 tokens. We experiment with different N values to see how it affects the quality of our predictions. To handle situations where we have not seen certain N-grams before, we use a common language modeling smoothing technique – backoff smoothing.

We explore two different approaches for choosing the *next* token. The first approach, PaLMTo-Generative, follows the traditional use of N-grams, where the sequence of N words is used to predict the next word. We also investigate three alternative methods under this approach: PaLMTo-G-1-1 – making a

choice solely based on probability, PaLMTo-G-3-1 – integrating distance as a factor along with probability, and PaLMTo-G-3-3 – using perplexity, which is a language model evaluation technique, to pick the most likely path to complete a gap.

The second approach, PaLMTo-Lookup, adopts N-grams as essentially storage systems, where the missing data can be retrieved based on the information preceding and following the gap. In this approach, instead of generating points individually, we search our N-gram corpus to identify instances where the first and last tokens of the N-gram correspond to the points preceding and following the gap.

1.5 Experiments

We evaluated the effectiveness of our proposed method using two real-world datasets from the cities of Porto and Beijing, and a third synthetic dataset from San Francisco. To assess the performance of our approach, we created gaps of up to 2 kilometers by removing points from complete trajectories. We then compared our findings with TrImpute [5], the latest work that does not rely on road network information for trajectory imputation.

Our results show that 85% of the imputed points fall within 50m of the original trajectory when imputing a gap of 2km, as shown in Figure 1.3, which was a noticeable improvement to the 69% accuracy achieved by TrImpute.

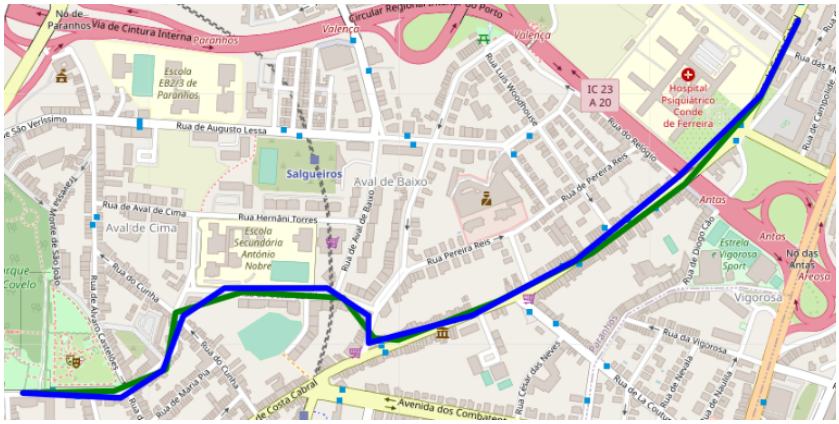


Figure 1.3: Original path (*green*) and the path generated by PaLMTo (*blue*)

The experiments in this thesis address the following research questions:

- Can PLMs solve the trajectory imputation problem?
- What are the trade-offs between grid size and quality of predictions?
- How does the choice of N-gram size affect the quality of predictions?
- What are the trade-offs between N-gram size and efficiency (memory/computation time)?
- How do methods that rely solely on probability, those incorporating probability with distance, and those incorporating probability with distance and perplexity compare with one another?
- How does the dataset (size, sparsity) affect the quality of the predictions?
- How does the gap size affect the quality of the predictions?
- Can a database-like "lookup" on previous trajectories be as effective as generating points using PLMs "as usual"?

1.6 Thesis Structure

Chapter 2 provides a background on various trajectory imputation methods, categorized based on their approaches. Additionally, that chapter will provide a brief overview of the evolution of language models over time. This historical context lays the foundation for understanding why simple language models may prove to be just as effective, if not more, for trajectory data compared to the large language models that dominate most recent works.

Chapter 3 provides a detailed explanation of the different approaches we used to address the challenge of missing points in a trajectory. It will explain the PaLMTo–Generative and the PaLMTo–Lookup approach in detail. Additionally, we explain the three different methods used within the PaLMTo–Generative approach.

Chapter 4 starts with a brief analysis of the datasets used for the experiments. The rest of the chapter focuses on the results of a series of experiments

performed to analyze the benefits and drawbacks of the different approaches. Additionally, we will examine various parameters, such as grid size, N-gram size, and data size, to provide an overall understanding of our approach's performance.

Lastly, Chapter 5 serves as the thesis's conclusion, summarizing key insights drawn from the experiments and discussing directions for future research.

Chapter 2

Background

In this chapter, the main concepts behind the terms trajectory imputation and language models are discussed, along with an exploration of previous research conducted in the field of trajectory imputation.

2.1 Trajectory Imputation

A trajectory is a trace generated by a moving object within a certain spatiotemporal context and is generally represented by a series of chronologically ordered points. A single trajectory of n geographical points can be formally described as $T = (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)$, where (x_i, y_i) represents the latitude and longitude of the i^{th} point, and t_i represents the timestamp associated with that specific observation, where $t_i > t_j$ for $i > j$.

Trajectory imputation is the task of filling in missing data points in a trajectory with *predicted* values, often using information from the available observed points. This, in turn, enhances the usability of the dataset for various downstream applications such as route planning, traffic management etc. Trajectory interpolation, trajectory reconstruction, trajectory restoration and trajectory completion are alternative terms that are often used interchangeably or in a similar context to describe the process of trajectory imputation.

The term trajectory imputation encompasses two key dimensions: spatial imputation and temporal imputation. Spatial imputation primarily deals with predicting missing location data within trajectories, while temporal imputation focuses on restoring missing timestamps. For the scope of this thesis, we

focus on spatial imputation, specifically addressing the challenges associated with completing location information within trajectories.

Incomplete trajectory can refer to two different scenarios. (1) It may refer to an unusual gap observed between consecutive points when compared to the rest of the dataset. In this case, the trajectory is considered incomplete due to the missing data points. (2) It can describe a situation where the gap between all points is excessively wide, rendering the trajectory unsuitable for the subsequent task at hand.

Trajectory imputation can be applied to various types of trajectories, including those that follow road networks, such as vehicle or pedestrian data, as well as trajectories in free space, like flying birds, which do not adhere to restricted paths. In Section 2.3, when discussing related works, we solely focus on trajectories that follow road network paths. This encompasses approaches that use road network information during the imputation process, as well as those that do not rely on such information, like ours.

2.2 Language Models

A language model is a distribution of probabilities associated with words or sequences of words. It is trained on large amounts of text like books, articles, and websites to extract statistics from the data and use those statistics as the basis for tasks like answering questions or generating text. It learns by estimating how likely certain word combinations are, with more probable combinations having a higher chance of being considered feasible. This does not mean grammatical or even factual correctness, but rather the model's ability to capture the patterns and style observed in the text used as its learning data.

Based on the underlying techniques and methodologies used to build and train them, we categorize language models into probabilistic language models and neural network language models.

2.2.1 Probabilistic Language Models

Probabilistic language models assign probabilities to words or sequences of words based on statistical methods. These models estimate the likelihood of a particular word or sequence of words occurring in a given context by analyzing the frequency of their occurrence in the training data.

In formal terms, a probabilistic language model establishes a probability distribution $P(W)$ over word sequences $W = w_1, w_2, ..., w_n$, where each word w_i belongs to a predefined vocabulary. The model calculates the conditional probability of a word given its preceding context, i.e., $P(w_i|w_1, w_2, ..., w_{i-1})$. These models are typically based on statistical methods, such as N-grams, Hidden Markov Models [17], or Maximum Entropy models [19]. In the following we will provide a detailed discussion on N-gram language models, which is the basis of our main approach.

N-gram Language Models

An N-gram is a sequence of N contiguous tokens in a sentence from a corpus. In the context of language modeling, an N-gram language model treats sequences of N tokens using a Markov process. Markov models are statistical models that assume we can predict the probability of some future unit without looking too far into the past. In the case of N-gram language models, the past refers to the last N-1 events in a given sequence, and the event being predicted is the N-th token. For example, when using a trigram ($N = 3$) model to predict the next word, we only consider the last two words in the context rather than the entire sequence leading up to that word. A general equation for this N-gram approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

An N-gram language model learned from a sufficiently large corpus can be used to determine missing words in a phrase or sentence. For example, consider completing the following sentence: "I took my ⟨blank⟩ for a walk." A model trained with sufficient data will yield a much higher probability for the word "dog" completing the sentence than the word "lunch" for example. This

happens because people are much more likely to take their pets for walks as opposed to their lunch. Therefore, sentences where the word "dog" is used are much more likely to be observed within a given corpus. Note that while there are many nouns that would frequently occur after the phrase "I took my", relatively few of them would also precede the phrase "for a walk". As one can see, a language model trained on a sufficiently large corpus will capture natural dependencies between words and phrases that precede or follow them. We leverage on that observation by drawing on the similarities between sequences of words in a corpus of text and sequences of observed points in a corpus of trajectories.

Smoothing Techniques

Smoothing techniques in N-gram language models are methods used to handle the problem of unseen N-grams. These techniques assign a probability to unseen N -grams by redistributing probabilities from observed N -grams, thus improving the overall performance and generalization of the language model. Although various smoothing techniques exist, our discussion will focus on the specific ones used in our experiments.

- **Interpolation Smoothing:** is a technique used to estimate the probability of an N-gram by combining probabilities from lower-order N-grams and higher-order N-grams. For example, in a trigram model, instead of using the probability of just the trigram, probabilities for trigrams, bigrams, and unigrams are computed and each is assigned a weight or coefficient, These weights determine the contribution of each N-gram order to the final probability estimate. Lower-order N-grams, such as unigrams and bigrams, consider fewer preceding words and are more focused on capturing specific dependencies between adjacent words. On the other hand, higher-order N-grams, such as trigrams or higher, take into account a larger context of preceding words, which allows them to capture general patterns. By combining the probabilities of lower-order and higher-order N-grams through interpolation smoothing, the

language model benefits from both types of information.

- **Back-Off Smoothing:** estimates the probabilities of unseen or infrequent N-grams by relying on lower-order N-grams. i.e., when encountering an N-gram with zero frequency or insufficient evidence, the model "backs off" to a lower-order N-gram. By using lower-order N-grams when higher-order context is insufficient, Back-Off Smoothing allows for smoother estimation of probabilities for unseen or sparse N-grams.

Evaluating Language Models

A common evaluation technique of N-gram language models is perplexity. Perplexity provides a quantitative measure of how well a language model predicts a given sequence of words or a test set. Perplexity is calculated based on the probability distribution of predicted words by the model. Formally, it can be defined as the inverse of the probability of a given test set. For a sequence of words $W_1W_2...W_N$,:

$$PP(W) = P(W_1W_2...W_N)^{-1/N}$$

A lower perplexity value suggests that the language model is more certain in its predictions. In other words, if a model assigns a high probability (i.e., low perplexity) to the test set, it means that it is not surprised (or is not perplexed) by the model, indicating a higher level of confidence in generating coherent text. Perplexity serves as a quantitative criterion to compare different language models or variations of the same model.

Perplexity, which is an intrinsic evaluation, is only one type of evaluation method. Another approach is extrinsic evaluation, which involves embedding the model in applications and measuring its performance within a specific task or context. However, extrinsic evaluation, while offering insights into a model's practical use, is very resource-intensive which makes it less practical for researchers.

2.2.2 Neural Network/Deep Learning Language Models

The language models in this category use neural networks, which can be either simple or deep, to learn representations of words and capture complex linguistic patterns and dependencies. By leveraging large amounts of data and complex architectures, neural networks have enabled breakthroughs in language understanding, which significantly advanced NLP tasks such as sentiment analysis, machine translation, question answering, and text generation. Next, we will provide an overview of the advancement of these models in the field of NLP with an explanation for why one model was superseded by another.

Word Embeddings

Word embeddings, such as Word2Vec [12], marked a significant shift in NLP. They represented words as dense vectors by analyzing large amounts of text data, enabling them to capture semantic relationships between words. For example, words like "king" and "queen" would have similar vector representations because they often appear in similar contexts.

However, word embeddings do not explicitly capture contextual information within a specific sentence. Each word's embedding is determined solely based on its local context in the training data; when a word is used in different contexts, it may have different meanings or associations. For example, the word "bank" can refer to the *land beside a river* or *a financial institution* depending on its context. Word embeddings alone cannot capture these contextual nuances. This limitation led to the development of more advanced models like RNNs and Transformers, which explicitly consider contextual information.

Recurrent Neural Networks (RNN)

RNN [20][8] have been widely adopted in NLP for their ability to process sequential data, such as sentences. They introduced the concept of recurrent connections. These connections enable the network to maintain a memory or hidden state that carries information from previous steps, remembering

something about each element as it comes by, and influences the processing of subsequent steps, allowing information to persist across different time steps.

Unlike N-grams which only take into account words within a fixed-size window, RNNs, as they progress through sequential data, have the ability to capture information from each time step by updating their hidden state using the current input and the previous hidden states.

For example, consider the sentence "today she is reading about ⟨blank⟩". Given this sentence, almost any English word could appropriately complete the sentence. However, if we provide more context, since the model has access to the information stored in the hidden state, which includes knowledge of what words have been encountered so far, it may be able to make a more informed choice. For instance, consider the input, "She is interested in learning more about language models. Today, she is reading about ⟨blank⟩". In this example, the model can reasonably discard most words and instead generate predictions closely related to language modeling.

The above example underlines the significance of capturing textual information when dealing with natural language. However, before we extend this model to trajectory data, we must examine whether such a structure holds similar importance in the domain of trajectory data. The experiments detailed in section 4.5.2 aim to provide insight into this very question.

While RNNs somehow addressed the issue of capturing longer textual information, they faced difficulties in learning long-term dependencies due to vanishing or exploding gradient problems. These issues limited their ability to capture long-range dependencies in text data.

Long Short-Term Memory (LSTM)

LSTM [6] was introduced as a solution to the above-mentioned limitation of RNN. LSTMs are a type of RNN architecture with an added memory cell, which allows them to capture long-term dependencies more effectively. The memory cell in LSTMs allows information to flow across multiple time steps, making them better suited for tasks that require understanding and modeling of longer-term dependencies, such as understanding complex sentence struc-

tures or maintaining context over long periods.

By selectively updating and retaining relevant information, through the use of gates (input gate, forget gate, and output gate), LSTMs help prevent the gradients from vanishing or exploding as they propagate through multiple time steps, a challenge faced by traditional RNNs, leading to more stable and effective learning.

While LSTMs were designed to capture long-term dependencies in sequential data, they can still struggle to effectively model dependencies across very long sequences. The information flow in LSTMs occurs through hidden states, and gradients need to propagate through multiple time steps. However, as the sequence length increases, LSTMs may have difficulty retaining and utilizing information from distant time steps.

Transformer Models

Transformers [23] introduced self-attention mechanisms that allow each position to attend to all other positions in the input sequence. By attending to all other positions in the sequence, each token can gather information from the entire input sequence. This enables Transformer models such as BERT [3] to model dependencies across the entire sequence and capture both short-range and long-range dependencies more efficiently than LSTMs. However, Zeng et al. [30] point out that self-attention is "anti-order" to some extent which results in temporal information loss even when using various types of positional encoding techniques. The authors mention that although this is typically not a significant concern for semantic-rich applications like NLP, it can be a problem in data where there is lack of such semantic relationships.

Transformers also introduced the concept of pre-training on large-scale unlabeled data, followed by fine-tuning on specific tasks. This pre-training allows the system to train on a generic database and then specialize it on a specific type of data. In the pre-training phase, the model might come across sentences like: "The longest river in the world is the ⟨MASK⟩ river." Here, the ⟨MASK⟩ token indicates that the model needs to predict the missing word based on the surrounding context. During training, the model is tasked with

predicting the missing words, which in this case is "Nile". By encountering numerous examples like this, the model learns that the Nile river holds a special significance in terms of its length. Then, when fine-tuned on a question-answering task, the model can effectively "reason about" river-related topics, such as answering questions about the longest river.

The preceding explanations provide insights into the evolution of language modeling techniques. It is important to consider which challenges faced by a particular model when applied to natural language are also relevant to trajectory data. For instance, the problem of disambiguating words with multiple meanings is unlikely to surface in trajectory data, as each *word* corresponds to a specific location on the ground. Moreover, our experiments will illustrate that issues related to long-term dependencies are also not relevant in the context of trajectory data.

2.3 Related Work

In this section, we discuss trajectory imputation approaches, organizing them into four groups: geometric methods, probabilistic methods, classic machine learning methods, and neural network/deep learning methods. These categories are not mutually exclusive, and there can be an overlap or combinations of methods within a single approach.

2.3.1 Geometric Methods

Geometric methods typically use interpolation to fill in missing values. These methods can involve basic forms of interpolation, such as linear interpolation that assumes a straight line trajectory between known points, or spline interpolation that considers the curvature of the Earth. However, such an approach, while straightforward, does not take into account changes in the direction of a trajectory. Additionally, it will most often predict points that do not align with actual roads as it lacks mechanisms to account for such geographical constraints.

Elshrif et al. [5] propose TrImpute, a framework that "inserts artificial GPS

points between the real ones in a way that the imputed trajectories end up to be very similar to the case if such trajectories were collected with a much higher sampling rate." The framework consists of three main components: preprocessing, spatial imputation, and temporal imputation. Within the framework, the spatial imputation component introduces the concept of *candidate points* as the set of possible artificial points. The algorithm to generate these points takes into account the direction of the trajectory, i.e., the points should not significantly change the angle from the starting point of the gap towards the direction of the endpoint. Additionally, these candidate points must be within a certain distance from the starting point of the gap to be filled.

They evaluate their idea on a real dataset of taxi trajectories in the city of Doha, Qatar. They compare their results to basic linear interpolation, employing metrics such as Fréchet Accuracy [4], which measures the distance between the imputed segment and the ground truth obtained from raw trajectories, and Open Street Map (OSM) accuracy, which involves matching imputed points with the actual road network. The evaluation involved varying sparsification lengths, ranging from 500m to 2000m. They achieved accuracy rates of over 90%+ for the 500m gaps and 60%+ for the longer 2000m gaps, outperforming the linear interpolation method. In section 4.6, we conduct a comparison between this method and our approach, demonstrating that our approach outperforms it.

Other geometric methods attempt to model the positions of uncertain objects between two sampling points using bounds. These bounds can take the form of geometric objects such as cylinders or beads, providing a representation of possible trajectories. Krumm [9] introduces the concepts of bridgelets, which are "small, spatio-temporal, maximum entropy clouds that model spatial uncertainty over small gaps." To bridge the gap between adjacent cells in both space and time, a "maximum entropy bridgelet" is inserted. This bridgelet encompasses all possible paths between the two cells within the time span, without making any assumptions about the specific paths. The trajectory augmented with these bridgelets serves as a candidate bridge for any pair of points that share the same start cell, end cell, and time span. It uses real

mobility data to mitigate uncertainty and capture observed behavior between locations. The author points out the computational intensity of training in this approach. In contrast, our approach involves a simpler process of calculating word probabilities, eliminating this concern.

2.3.2 Probabilistic Methods

Probabilistic methods leverage statistical analysis to estimate the distribution of possible outcomes based on historical data. One such method is the use of Hidden Markov Models (HMMs).

Simmons et al. [21] use HMMs to predict a driver's intended route and destination based on observed data. The model represents states as pairs of links and goals, and observations are derived from the GPS position, speed, and heading. To reduce the model size, the transition probability function is split into predicting the next link and then predicting the goal destination based on that link. The system's predictions can be used to generate routes or estimate the most likely route based on given goals. This model makes the basic assumption that most routes used by drivers are routines. The authors acknowledge that in scenarios where this assumption is violated, such as in cases where the drivers are delivery people, this approach is less applicable. Unlike their model, which assumes routine driving patterns and requires a comprehensive map database, our approach can handle more diverse driving scenarios such as those by taxi drivers, without reliance on a pre-existing map database.

Qu et al. [16] propose a missing data imputation method based on Probabilistic Principal Component Analysis (PPCA) that combines two key components: Principal Component Analysis (PCA) –to capture the main structure– and Maximum Likelihood Estimation (MLE)– to estimate the missing value. However, this approach applies a linear PCA which makes the assumption that the relationships between variables can be represented by linear combinations, which does not always hold in trajectory data.

2.3.3 Classic Machine Learning Methods

Classic machine-learning methods have also been applied to trajectory imputation. Techniques such as association rules, k-nearest neighbors (kNN), and linear regression, are commonly used to predict missing trajectory points based on observed data patterns. These methods offer an alternative approach to imputation by leveraging pattern recognition and rule-based inference.

Tak et.al. [22] propose a method for imputing missing data in road sections based on the k-nearest neighbors (kNN) algorithm by defining road sections as groups of links with shared traffic properties. The proposed method leverages the strengths of the kNN algorithm, including its ability to capture detailed traffic changes without approximation or smoothing. The algorithm divides the road network into sections, locates missing values in historical and subject data, calculates distances and similarity measures, and generates imputed values by integrating the k nearest neighbors with the highest weighted similarity to the subject data. The authors suggest that a minimum of 400 historical data points is necessary to ensure the quality of the proposed algorithm's imputations. This approach has an advantage over our method in terms of requiring fewer data points. However, it relies on road information, which is not a requirement in our approach.

2.3.4 Deep Learning Methods

In recent years, deep learning approaches have gained significant attention in trajectory data analysis. These models use multi-layered architectures to learn complex patterns and dependencies in the trajectory data.

Wang et al. [24] propose a model called Deep Hybrid Trajectory Recovery model, whose key contribution is the integration of a subsequence-to-sequence neural network model with the Kalman Filter [26] as a post-calibration component, to derive more accurate estimations of cell-level predictions. This detailed calibration step was necessary due to the coarse cell size they use. However, in our case, we use much smaller cells which results in uncertainties that are within acceptable limits.

Another deep learning model, Traj2Traj, proposed by Liao et al. [11], aims at simplifying trajectory data restoration by eliminating the time-consuming map-matching step of previous deep learning models [18]. Their model achieves this by incorporating road network constraints and considering the entire trajectory context, allowing for road-level trajectory point restoration. However, this approach still relies on existing road network information, with an additional step of constructing a road network using data from OpenStreetMap.

Musleh et al. [14], the only work that uses language models, propose TrajBERT, a framework for "an efficient and practical solution for almost all fundamental trajectory analysis problems" by changing the core of the BERT system itself to effectively handle trajectory data as primary data type. The framework consists of three layers. The first layer called the *Data Layer*, preprocesses input trajectories to handle challenges related to data quality and availability. The second layer, which is the *"BERT-Like" Layer*, takes the preprocessed training trajectories and learns a unified trajectory model. This layer incorporates three components, Spatial Tokenization, Spatial Embedding and Spatial Attention designed to "understand spatial characteristics during the learning process". The third *Fine tuning layer* tunes a simple model according to the task at hand. To assess the framework's performance, the authors conducted a trajectory imputation experiment, achieving mean and median distances of 37.9 and 38.9 meters, respectively, between the imputed and actual points. However, the specific gap size for achieving this level of accuracy was not disclosed in their paper.

Let us delve a little deeper into the spatial embedding and spatial attention components of the BERT-Like layer of this approach. In BERT, these layers convert tokens into vectors to capture semantic and contextual information in text. This is particularly useful in tasks like Word Sense Disambiguation, where words with multiple meanings can be distinguished based on the context in which they occur. In the context of spatial data, the authors customize the BERT-like layer to leverage unique spatial properties. They mention that the embedding of spatial tokens considers factors like trajectories and spatial attributes such as proximity to roads or other geospatial features. Yet, it

remains unclear how this benefits spatial tokens, as their meaning does not change with context, i.e., a single token consistently represents one specific location. Moreover, the problem of temporal loss information associated with self-attention mechanisms as pointed out in [30] raises the question of whether such a model is the best choice for trajectory data, where order plays a very important role.

One of the main advantages of BERT is its pre-training on diverse types of data, enabling it to generalize its knowledge to new data. In a subsequent paper [15], the authors introduce KAMEL, a scalable BERT-based system designed specifically for trajectory imputation. This system incorporates a partitioning component that handles batches of trajectories during training. It makes a decision based on the geographic location: if the model already contains trajectories for that specific location, it enriches the existing model; if the trajectories cover an entirely new geographic area, it creates a new model. While this approach can create a comprehensive system for trajectories across a large geographic area, since there are no inherent relationships between trajectories in different cities, it is not clear how the model benefits from such pre-training.

Chapter 3

Proposed Approaches

This chapter focuses on two key aspects: the conversion of trajectories into sentences and the use of N-grams to complete trajectories.

3.1 Trajectories as Sentences

We use a grid-based representation of the (2D) space to convert trajectories into sentences. We start by overlaying a 2-D grid to cover the entire area encompassing all trajectories in our dataset, as illustrated in Figure 3.1b. Within this grid, each cell will have an identifier that can be likened to a unique word or token in a finite vocabulary (Figure 3.1c). To represent individual points within a trajectory, we use the identifiers of the cells in which the points fall. This representation enables us to treat a full trajectory similar to a complete "sentence," where each word corresponds to a cell in the grid.

The two trajectories shown in Figure 3.1a can then be read as:

- Green trajectory: (0,1), (1,1), (3,1), (4,1), (5,2), (5,3), (5,4), (6,4), (8,4), (9,4), (11,4), (12,4), (12,2), (13,3), (15,3)
- Blue trajectory: (15,0), (14,0), (13,0), (11,0), (10,1), (9,0), (8,0), (7,0), (7,1), (6,2), (6,3), (5,3), (4,4), (3,5)

Much like how language modeling can be used to predict missing words in incomplete sentences, we can apply a similar concept to incomplete trajectories. When predicting these "missing words", we are predicting the corre-

sponding grid cells within our trajectory grid. The centroids of these cells are assigned to be the missing points in the trajectory.

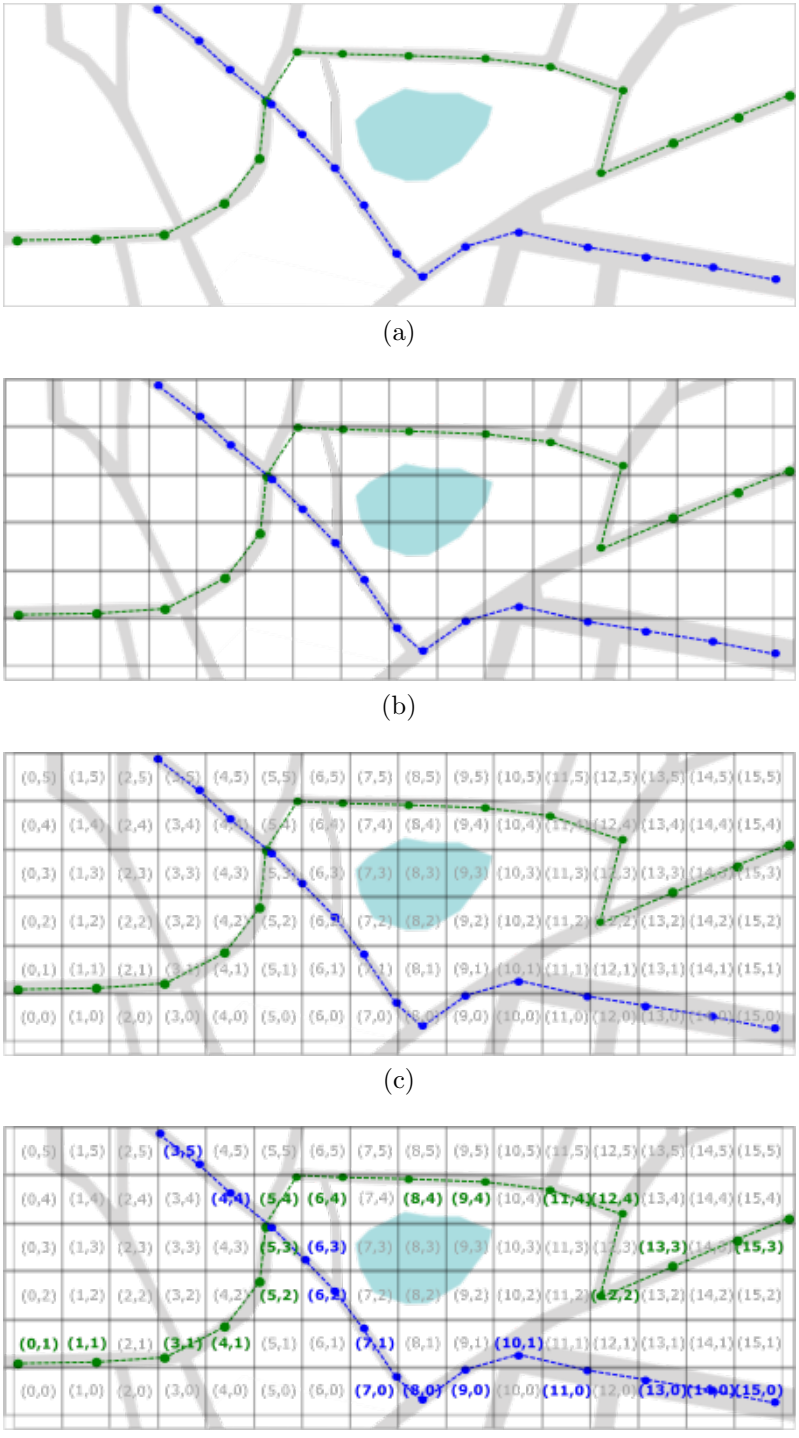


Figure 3.1: Converting trajectories to "sentences"

A key parameter in this mapping is the grid size, as this will define the precision of the approximation for the missing points. A finer-grained grid provides a higher level of detail as well as a larger vocabulary and longer sentences. On the other hand, a coarser grid may lead to loss of information when multiple points in a trajectory are mapped to the same grid cell. There is clearly a trade-off, which we will investigate later in Section 4.5.1.

3.2 Imputation using N-grams

In the same way that we can find the most probable words to complete a sentence in natural language, we can use N-gram language models to approximate points to complete a trajectory. To achieve this, we can make use of N-grams in two different ways. The first approach, PaLMTo-Generative, which is more rooted in language modeling, involves generating missing points between a gap, one by one, until the trajectory is complete. The second approach, PaLMTo-Lookup, directly searches through our N-grams to identify those that can bridge the gap in our trajectories, effectively treating the collection of N-grams as repositories. Following is a detailed explanation of each method with a comparison of the two methods provided in section 4.5.6

3.2.1 PaLMTo-Generative

In this method, we follow the traditional use of language modeling and use the N-grams to predict the next token given previous (N-1) tokens which we refer to as **context**.

We create N-grams from sentences by dividing them into N-token groups. As an example with English words for clarity, with $N = 3$ (trigrams), from the sentence *"my blue sweater is missing,"* we get the trigrams *"my blue sweater"*, *"blue sweater is"* and *"sweater is missing"*.

We then use these trigrams to predict the most likely tokens for trajectory completion, relying on the two previous tokens to impute the next one. For example, if we have *"my"* and *"blue"* as the previous tokens, we look for trigrams starting with *"my blue"* and pick the third token according to their

probability. This process can be performed from both sides of a gap in a trajectory.

Let us assume there is a gap that needs to be filled in a trajectory. PaLMTo-Generative approach involves iteratively predicting a point from each side, meaning that in each iteration, we predict one word from the left side and another one from the right side. To achieve this, we need to create N-grams from both our original trajectories and their reversed counterparts.

This process is repeated until the predicted points from both ends coincide or are sufficiently close. The maximum number of iterations for each trajectory is determined by the average gap between consecutive points in the complete trajectories within our dataset. For example, if this average gap is 120m, with each round of prediction generating two points (one from each side), we should not require more than three iterations to bridge a gap of 500m.

Next, we will explore three different approaches under PaLMTo-Generative, labeled as PaLMTo-G-1-1, PaLMTo-G-3-1, and PaLMTo-G-3-3 for differentiation.

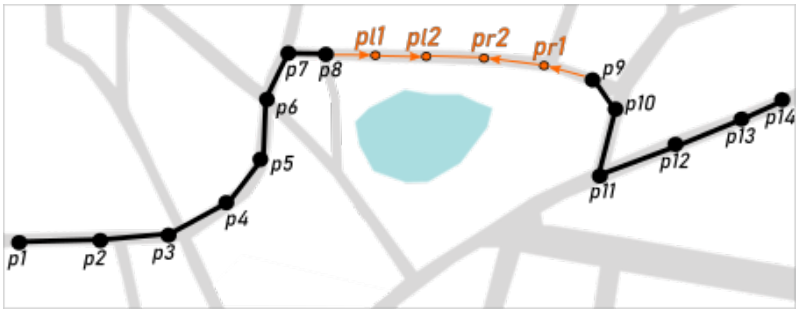
PaLMTo-G-1-1 - Imputation using Probability

In the first approach, we simply select the most probable next token based on N-1 previous tokens from each side until the generated points are close enough. This method lacks a mechanism to "steer" the imputed points from each side towards one another. This may not have a significant impact on short gaps or when there are limited path options (Figure 3.2a). However, it becomes challenging when dealing with larger gaps or in situations where there are multiple likely options for the path to take. (Figure 3.2b)

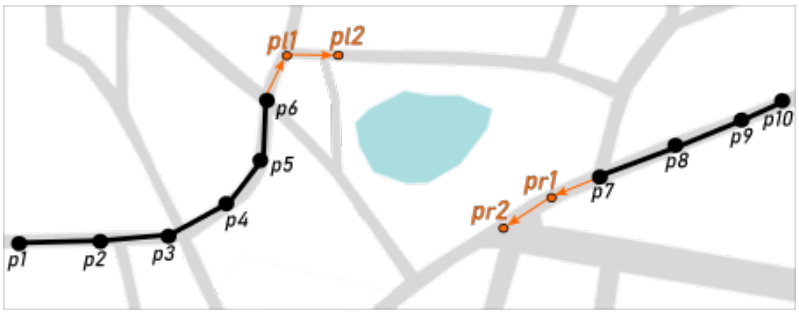
PaLMTo-G-3-1 - Imputation using Probability and Distance

In the second approach, rather than selecting the single most probable next token from each side, we select the three most probable "next points" using the previous points, for example, $p5$ and $p6$ from the left side and $p7$ and $p8$ from the right side on Figure 3.3a can be used as previous points.

We opt for three options because there can be more than one path the route



(a)



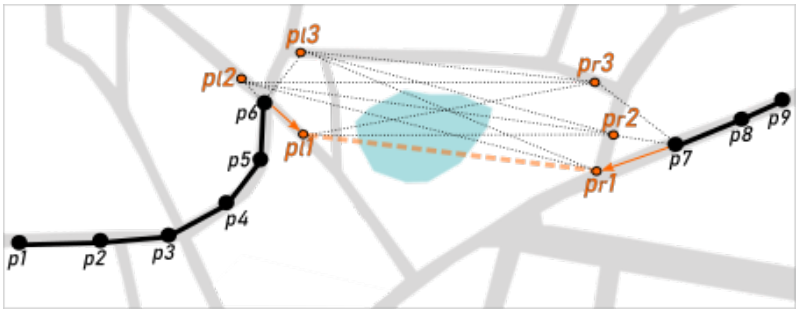
(b)

Figure 3.2: PaLMTo-G-1-1 - Imputation using Probability

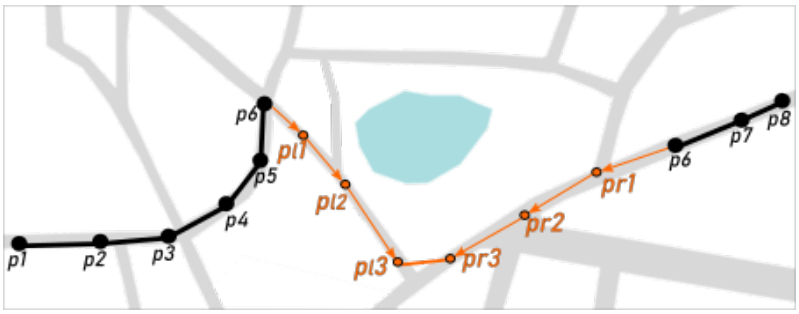
can possibly take. Take Figure 3.3a as an example; it shows three possible next points for each side. Some of these points, such as points $pl1$, $pl2$, $pl3$ are on completely different paths. The most likely tokens here could be $pl2$ on the left and $pr1$ on the right. But in the figure, one can see that these points head in different directions and are not likely to come close enough to fill the gap.

If we instead generate multiple possible next tokens and then calculate the distance between these points, we may get better candidates i.e. candidates that are on paths likely to meet. Here, we are making the assumption that points that are closer to one another are more likely to be heading toward each other than those that are far apart. There may be exceptions, but for the most part, this method gives us better choices than simply picking the most likely token.

Let us assume the three most probable "next points" from the left and the right side are $pl1$, $pl2$ and $pl3$, and $pr1$, $pr2$ and $pr3$, respectively (Figure 3.3a). Next, we choose the closest pair between those points in either side, let us say that that pair is $pl1$ and $pr1$ (connected by the dashed line in Figure 3.3a).



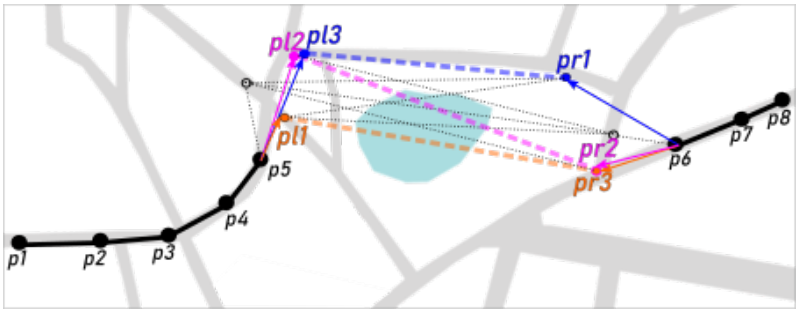
(a)



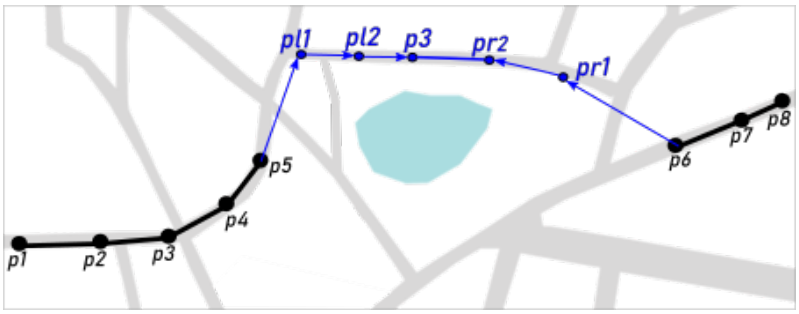
(b)

Figure 3.3: PaLMTo-G-3-1 - Imputation using Probability and Distance

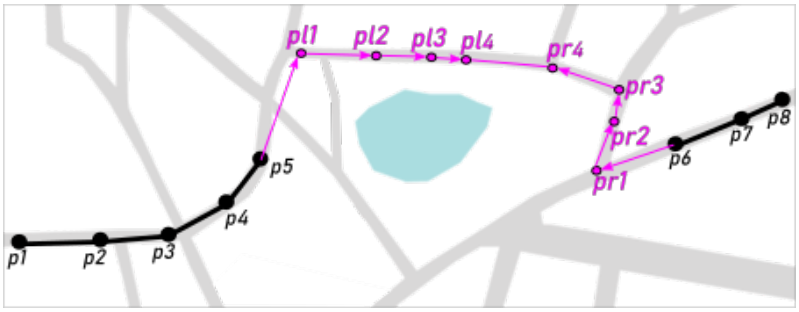
Then, we repeat the same process using a trigram on the left side with $p6$ and $pl1$ and a trigram on the right side with $p7$ and $pr1$ and find a new closest pair. We do this process until the points from either side are sufficiently close, and therefore completing the imputation process (Figure 3.3b)



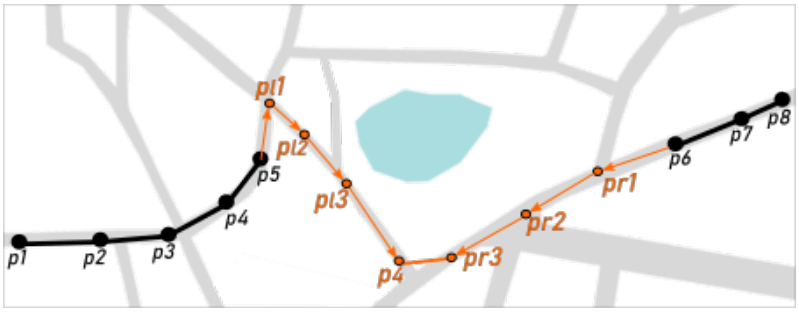
(a) Initial points for 3 alternative options



(b) Option 1



(c) Option 2



(d) Option 3

Figure 3.4: PaLMTo-G-3-3 - Imputation using Probability, Distance and Perplexity

PaLMTo-G-3-3 - Imputation using Probability, Distance and Perplexity

The final approach is an extension of the preceding method, where instead of selecting only the closest pair, we choose the three closest pairs, and for each of these pairs, we apply PaLMTo-G-3-1. This results in three potential paths for bridging the gap. From these three possibilities, we select the one with the lowest perplexity (2.2.1).

Similar to PaLMTo-G-3-1, this is a way to explore the possibility of finding better candidates. Using distance along with probability does result in better predictions, i.e., points that move towards one another. However, it also has the risk of 'overshooting' the next points. For example, in Figure 3.4a, points $pl3 - pr1$ (shown in blue) are the closest pairs, but $pr2$ or $pr3$ clearly would have been the better picks since they better capture the road network. Using perplexity to pick the most likely route among multiple options is a way to tie it all back to probability which language models rely on. By selecting the option with the lowest perplexity, we are choosing the path that is deemed most probable for successfully completing the trajectory. For instance, Figure 3.4b and 3.4c follow similar paths but 3.4c aligns more closely with actual driving paths, which makes it more likely to have a lower perplexity compared to 3.4b.

In all the methods discussed above, we use backoff smoothing i.e., when higher N-grams are unable to find a match, we back off to lower N-grams (section 2.2.1). Additionally, when we have a context size of C, it means that we consider a range of contexts starting from the immediate preceding token (context size 1) up to the C preceding tokens. For instance, if our context size is set to 3, this means that we take into account the information from the previous token ($C = 1$), the two tokens before the gap ($C = 2$), and the three tokens before the gap ($C = 3$). We use linear interpolation (section 2.2.1) of these ranges of contexts where different weights are assigned to each context size. These weights are tuned using a separate dataset.

3.2.2 PaLMTo-Lookup

This method uses N-grams essentially as storage mechanisms, taking advantage of the highly repetitive nature of trajectory data. For instance, in our Porto dataset, we have 15.6M GPS points, in a city with a road network spanning 965 km. Estimating the required number of points to cover this entire network based on the average gap between consecutive points at around 7800, we find that there are approximately 2000 trajectories for each road segment. With this many variations of trajectories for a single road segment, it is expected that there will be many similar N-grams, even as the N-gram size grows.

In contrast to PaLMTo-Generative which only needs to store unigrams, bigrams and trigrams, this approach requires storing a more extensive range of N-grams. The size of N grows as the size of the gap to be completed increases.

Using the same examples from Figure 3.4a in the previous section, to fill the gap between points $p5$ and $p6$, we search through our entire N-gram corpus for all N-grams that start with $p5$ and end with $p6$ and select the one with the highest probability. The advantage here is that we are able to impute all the points necessary to fill the gap at once, making this method significantly faster.

However, this approach becomes increasingly memory-intensive as the gap size to be completed increases. For instance, to complete a trajectory with a 2km gap, we may need to create N-grams as large as 16-grams, and this number further increases with larger gap sizes.

Chapter 4

Experiments

4.1 Datasets

This study uses three different datasets. Two of these are real-world taxi datasets from the cities of Porto, Portugal [13] and Beijing, China [28][29]. The third dataset is synthetic and was generated within the city of San Francisco, United States[1]. The Porto dataset is used in two ways: one encompassing the entire city and the other focusing on a small but dense (in terms of roads) region within each city.

These datasets differ in terms of the density of the trajectories, their underlying road structure, and city size. A detailed discussion of all the datasets is presented in the subsequent sections providing insights into their respective features. Additionally, a detailed comparison of all four datasets is presented in table 4.1.

4.1.1 Porto

Porto is a small city covering an area of 41.4 sq km. The selected dense area within the city (shown in square in Figure 4.1), spans an area of 3.5 sq km. The roads have a maze-like layout with irregular intersections.

The dataset consists of data collected from 442 taxis over the course of a year. Each data sample within the dataset represents one complete trip, with

[1]https://sfsp.mpi-inf.mpg.de/

the time stamp provided for the starting GPS coordinate of each trip, and subsequent GPS points recorded at regular 15-second intervals.

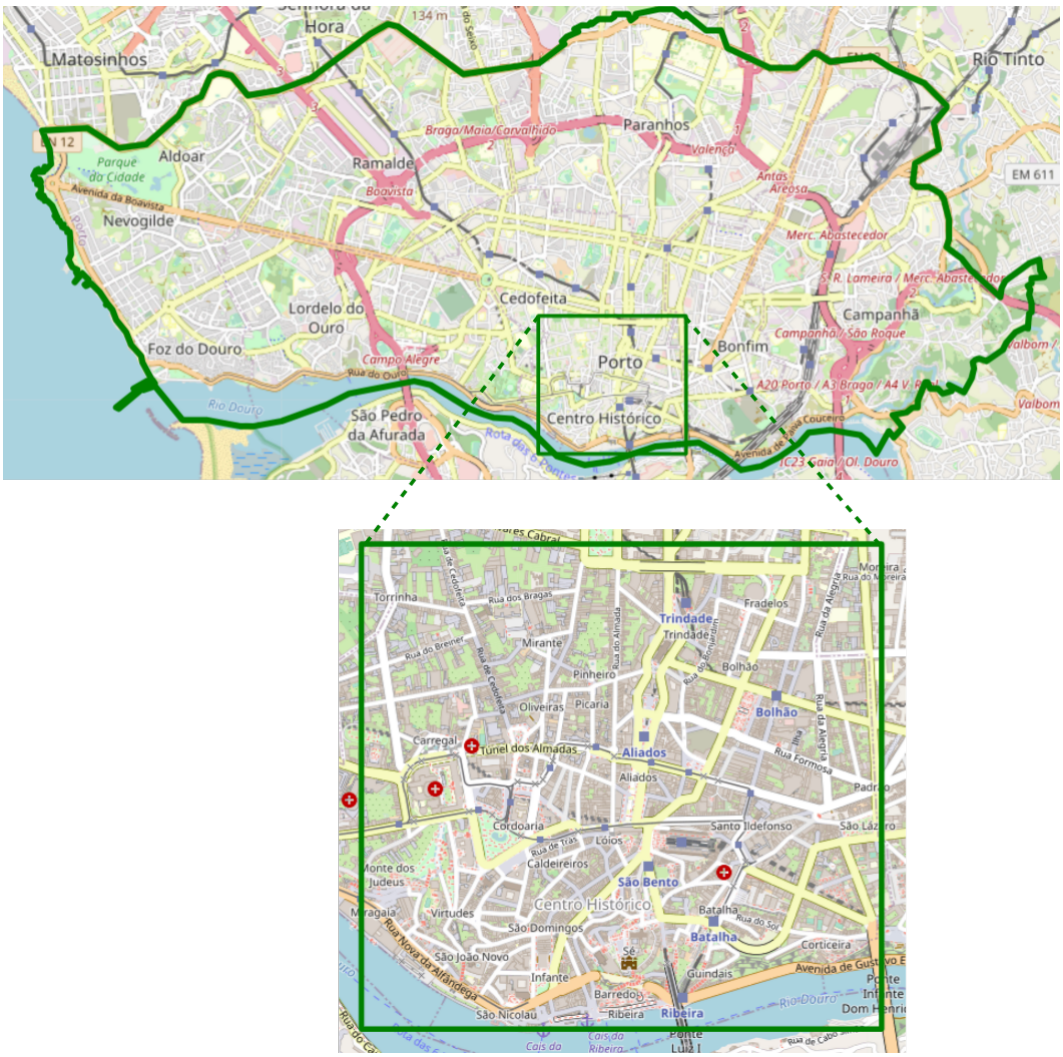


Figure 4.1: Porto city (top), selected dense area (bottom)

4.1.2 San Francisco (synthetic)

The second dataset used in our experiments is a synthetic dataset created using OpenStreetMap in the city of San Francisco (Figure 4.2). This dataset is only spatial and does not include timestamps for the trajectories. San Francisco is nearly three times larger than Porto, covering an area of 121.4 square kilometers. Its road network is characterized by a grid-like pattern with perpendicular intersections.

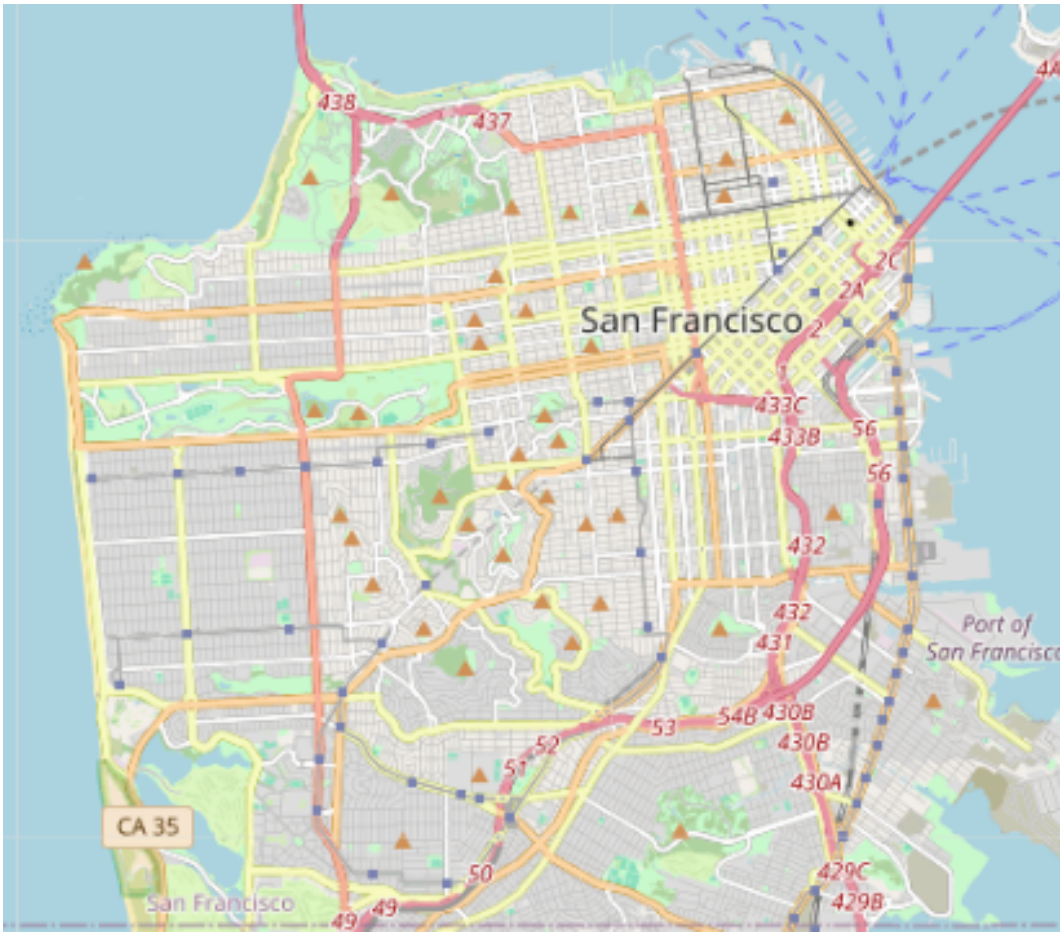


Figure 4.2: San Francisco

4.1.3 Beijing

The third dataset is a real-world taxi trajectory dataset from the city of Beijing, one of the largest urban centers in the world (Figure 4.3). Covering an expansive area of 16,411 square kilometers, Beijing surpasses San Francisco's area by over 100 times and Porto's size by nearly 400 times. The dataset consists of data collected from 10,357 taxis over the course of one week. The road network in Beijing exhibits a mix of both grid-like and maze-like characteristics.

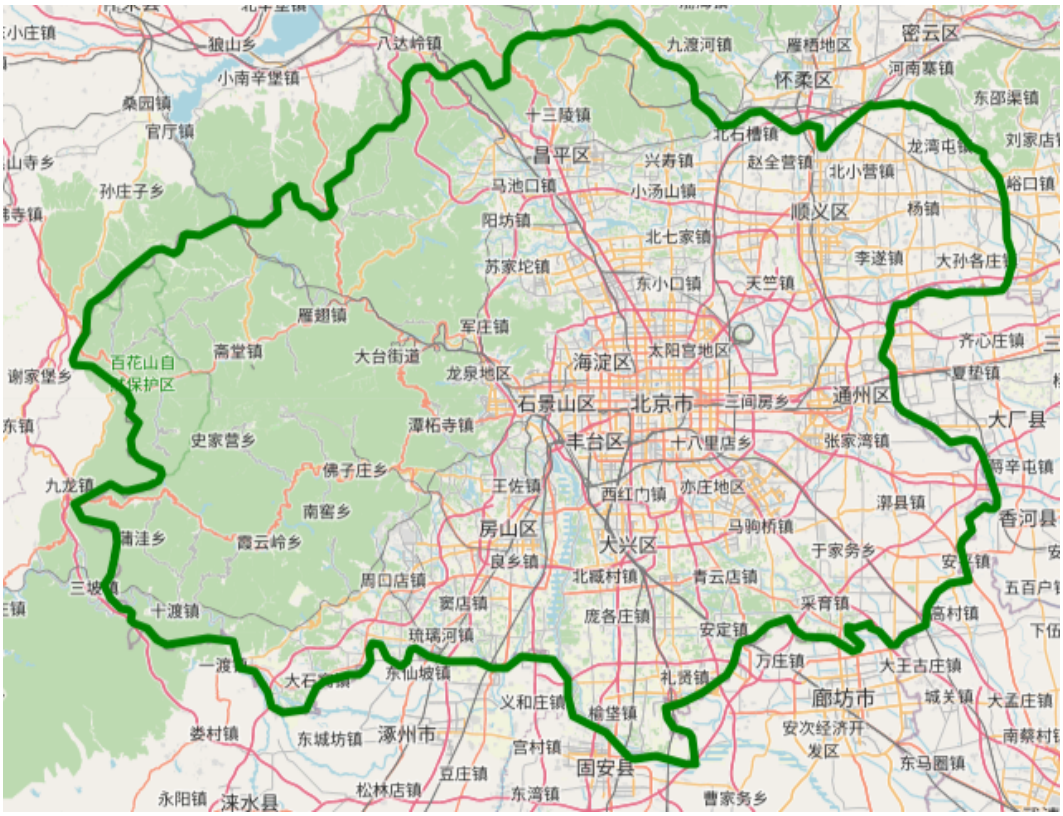


Figure 4.3: Beijing

	Porto (Full)	**Porto (Small)**	**San Francisco**	**Beijing**
Area	41.42 sq.km	3.5 sq.km	121.4 sq.km	16,411 sq.km
Number of points	15,615,943	3,937,735	2,121,662	17,679,695
Points per sq.km	370,925	1,125,067	12,554	914
Average distance between points	118m	118m	85m	464m

Table 4.1: Comparison of the four datasets

We can analyze these datasets in terms of their size, density, and characteristics of trajectories:

Size and Density: The Porto dataset contains a total of 15 million points with about 4 million points located within the smaller selected region.

The smaller region has 3 times as many points per square kilometer compared to the full city. The San Francisco dataset contains a total of 2 million points spread across an area of 121.4 square kilometers. This translates to an average density of approximately 38,000 points per square kilometer. On the other hand, the Porto dataset consists of 15 million points covering an area of 41.42 square kilometers. This results in a higher density of around 370,000 points per square kilometer making it significantly denser than the San Francisco dataset, with a higher concentration of points in a smaller area. The Beijing dataset is significantly less dense compared to both the Porto and San Francisco datasets. It contains fewer than 1000 points per square kilometer.

Average distance between points: When examining the average trajectory length, we find that the San Francisco dataset has an average trajectory length of 3.8 kilometers, and the Porto dataset is similar at 3.4 kilometers. Although the trajectory lengths are comparable, the Porto dataset has nearly four times the number of points per trajectory compared to the San Francisco dataset. This indicates that the trajectories in Porto are more finely sampled and have a higher level of detail. The Beijing dataset, on the other hand, has an average of nearly 500m between consecutive points, making it significantly more sparse than the Porto and San Francisco datasets.

Our objective in using datasets with varying characteristics is to gain insight into their respective impacts on the quality of the predictions. Does having a dense dataset (in terms of points per square kilometer) give better results? Does the size of the city influence our choice of grid size? In the following sections we aim to answer these and other similar questions.

4.2 N-gram Trends in Trajectory Data and Natural Language

In this section, we compare how the number of N-grams behaves in trajectory data and natural language data[2].

[2]https://blog.research.google/2006/08/all-our-n-gram-are-belong-to-you.html

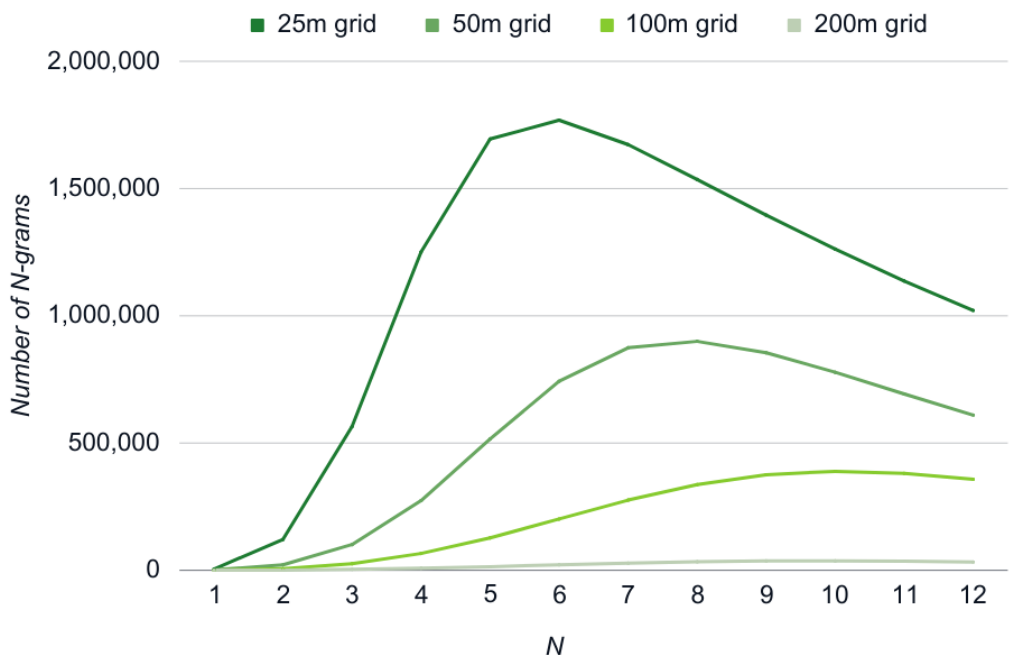


Figure 4.4: N-gram trends in trajectory data

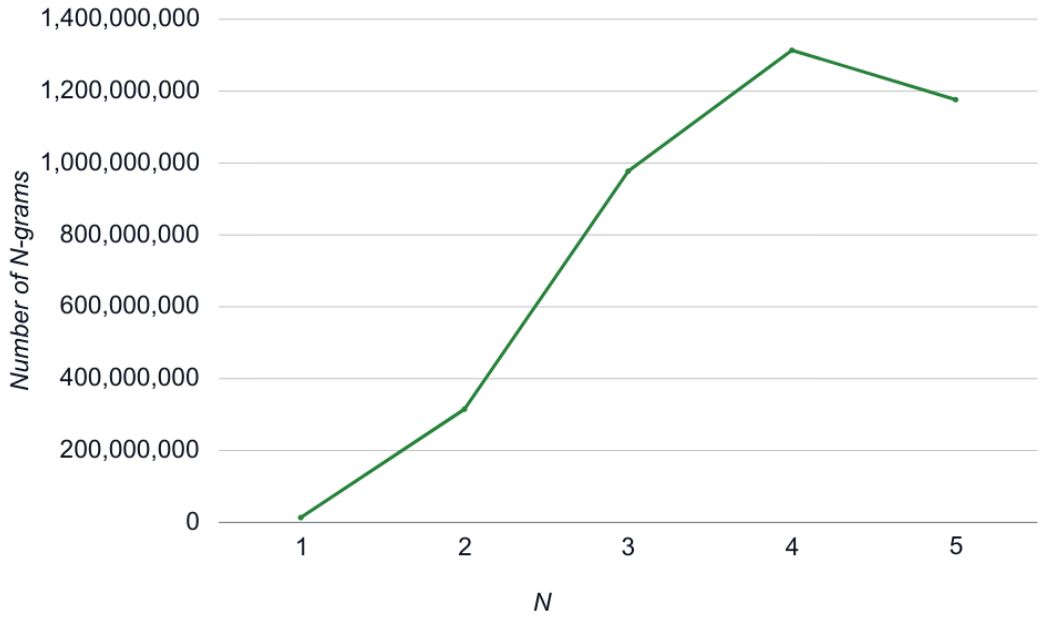


Figure 4.5: N-gram trends in natural language

In both cases, we notice a similar trend (Figure 4.4 and 4.5). As we increase the value of N, the number of unique N-grams initially goes up. However, as N becomes very large, the number of unique N-grams starts to decline because many of the longer sequences of tokens may not appear frequently enough in the dataset to generate a significant number of unique N-grams.

The choice of grid size is one of the factors that affect these N-gram trends in trajectory data. In Figure 4.4, we can observe that a smaller grid size of 25 meters results in the highest number of unique N-grams. This is because smaller grids generally yield a larger vocabulary compared to larger grid sizes.

Moreover, the point at which the number of unique N-grams begins to decline varies for different grid sizes. In larger grid sizes, N becomes much larger due to the reduced vocabulary size, as even longer sequences tend to repeat frequently in these settings. In simple terms, larger grids mean fewer different movements to make, so longer sequences of movements start to repeat more often.

This might be one factor to help us decide between the two methods: PaLMTo-Lookup and PaLMTo-Generative. PaLMTo-Lookup relies on repetitions, and a flatter curve (a curve where the peak point occurs at a larger N) is favorable for this approach because it signifies longer sequence repetitions. However, this flatter curve is associated with larger grid sizes (100m and 200m), which come with their own set of limitations, as we will explore in section 4.5.1.

4.3 Preprocessing

The first step in the preprocessing is removing "excessively" incomplete trajectories. Any trajectory with a gap exceeding a distance d between consecutive points is considered incomplete and removed from the dataset. The value of d varies depending on the specific dataset in use. For instance, in the case of the Porto dataset, we set d to 500 meters. This choice is based on the data collection frequency (which was 15 seconds for this dataset), under the assumption that it would be unlikely for any vehicle to travel a distance exceeding 500

meters during this 15-second interval.

Next, we process the converted trajectories (i.e., sentences). In the context of trajectory dataset, conventional NLP preprocessing steps such as tokenization and stemming are inapplicable. This is due to the fact that the *sentences* in the dataset are created in tokenized forms, and the tokens which are simple tuples of numerics, do not have natural language characteristics such as prefixes, root words, etc.

The only preprocessing we apply to the converted trajectories is removing repeated consecutive tokens in each trajectory. The reason for removing this repetition is to avoid imputed tokens from repeating previously used tokens. By removing repetitions, we acknowledge the potential loss of the temporal aspect of the data. However, at this stage, our primary objective is spatial imputation, and the repeated tokens do not serve any purpose.

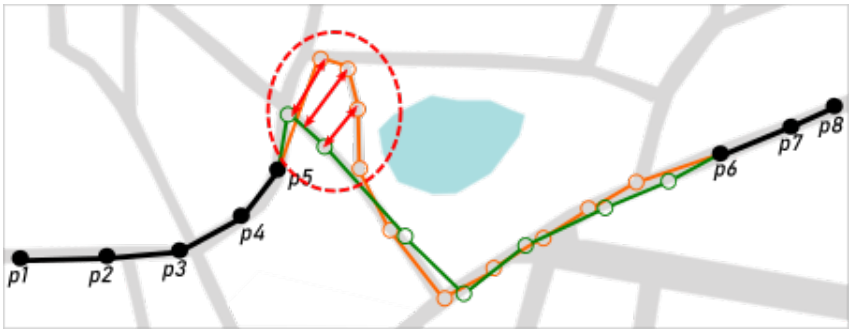
We initially divided the processed dataset into training and test data in a 90:10 ratio. Within this 10% allocated for testing, we randomly select 1000 test trajectories. In section 4.5.4, we conduct experiments to investigate how reducing the training dataset size to a substantially smaller size affects the results.

4.4 Evaluation Metrics

To evaluate the quality of the predictions, we use precision and recall metrics. The precision measures the proximity of the imputed points to the original line segment (created from the original points) (Figure 4.6a) and answers the question "What is the percentage of imputed points that fall within a specific threshold of the original line segment?"

Recall, on the other hand, measures the proximity of the original points to the imputed segment (created from the imputed points) (Figure 4.6b) and addresses the question "What is the percentage of the original points that are within a specific threshold of the imputed line segment"?

In essence, precision helps us measure how many points we imputed correctly, while recall assesses how many points we successfully recovered. A value



(a) precision

(b) recall

Figure 4.6: (a) Precision - predicted points (*orange*) that are outside the specified threshold from the actual path (*green*) are circled in red. (b) - Recall - actual points (*green*) that failed to be recovered (are outside the specified threshold from the predicted path) are circled in red.

closer to 100% in both cases indicates good results, signifying higher quality in our predictions.

We report the F1-score for a threshold of 25m in all experiments, except in section 4.6 where we provide results for both 50m, which is similar to the threshold used in [5], and 25m thresholds.

In our experiments, we compare the computation times of the different approaches discussed in Chapter 3. These experiments were conducted in Google Colab using virtual machines with Intel(R) Xeon(R) CPU running at 2.20GHz. We calculated the average time based on 10 runs of each experiment.

4.5 Experiments with Different Parameters

In our experiments, we conducted tests with various parameter settings for the N-gram size, the grid resolution, and the dataset size. The following

subsections present the outcomes of these different experiments along with a rationale for the selection of each parameter.

4.5.1 Grid Size

In our first experiment, we focused on evaluating the performance of different grid sizes, specifically, grid sizes of 25 meters, 50 meters, and 100 meters, using data from downtown Porto.

There is a direct correlation between grid size and the quality of the predictions, where an increase in grid size corresponds to a decrease in F1-score. Notably, this decline becomes more pronounced when transitioning from a 50m grid to a 100m grid compared to the shift from a 25m grid to a 50m grid (Figure 4.7). This decrease occurs despite the larger grid sizes resulting in more predictions that are within the same cell as the removed point. The reason for this is the fact that what we predict is a cell, not a point, and the centroid of the predicted cell is assigned as the prediction point. In cases of larger grid sizes, this can result in a significant distance between the actual point and the predicted point, which is less of an issue with smaller grid sizes (Figure 4.8). In these cases, if the actual point and the predicted point fall within the same cell, the maximum distance is at most 17.68 meters and 35.36 meters for the 25m and 50m grids, respectively. In contrast, the 100m grid can result in distances of up to 70.71 meters.

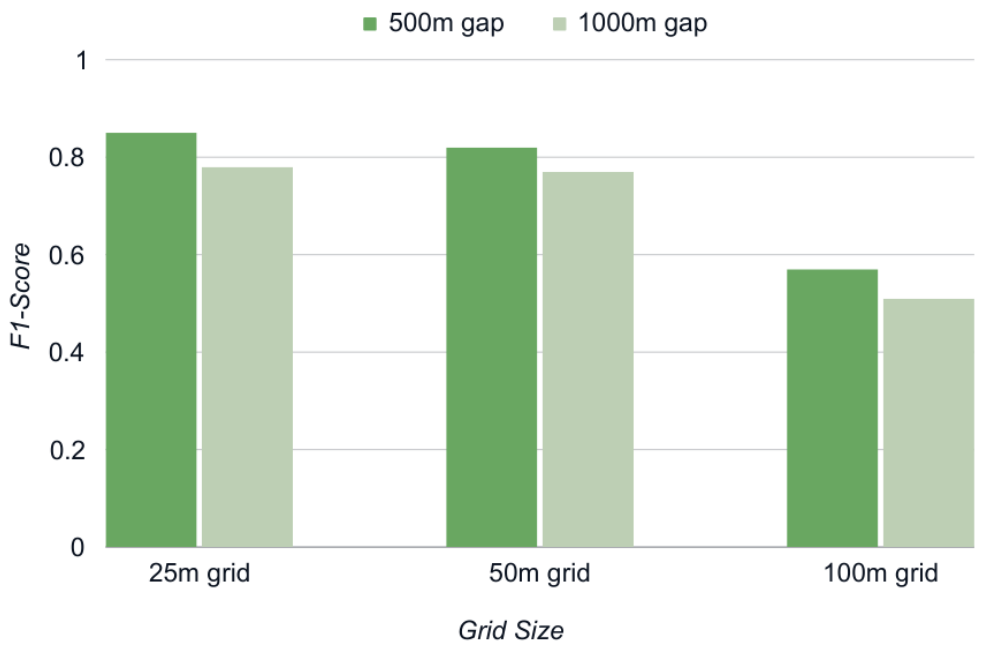


Figure 4.7: Grid size vs. F1-score

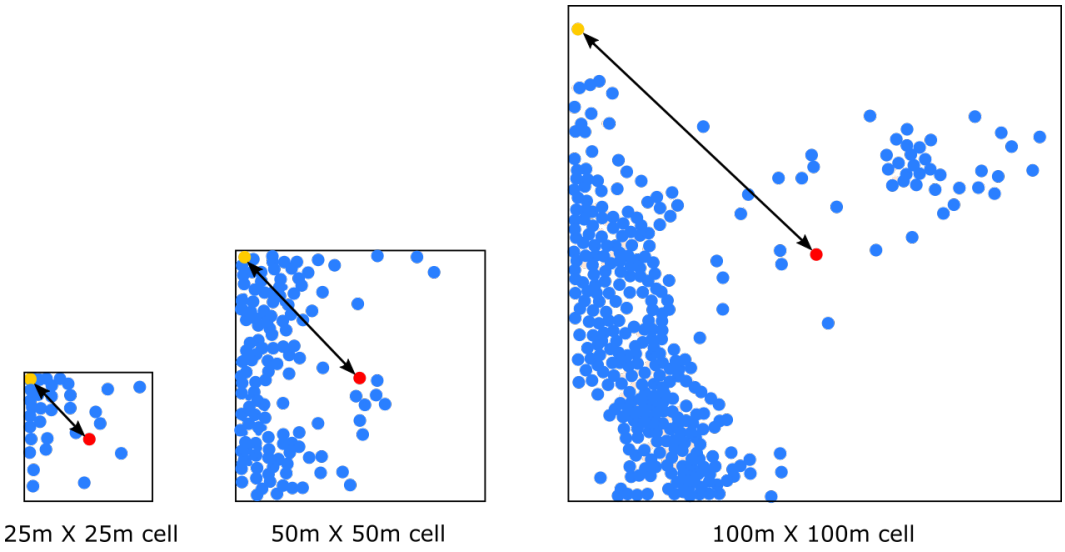


Figure 4.8: The impact of grid cell size on prediction quality. As cell size increases, the distance between the actual point and the predicted point (center of the cell) may also increase, highlighting the trade-off between grid granularity and prediction precision

In addition to the quality of the predictions, we also took into account the

execution time associated with these grid sizes. There is an inverse relationship between grid size and execution time (Figure 4.9). Larger grid sizes resulted in faster execution times, which can be attributed to the reduced number of unique N-grams when using a larger grid size (Figure 4.10).

When it comes to selecting the optimal grid size, we face a trade-off between quality and execution time. As we change the grid size from 25m to 50m, and then from 50m to 100m, the execution time shows an approximate 30% decrease in both transitions. It is important to note that this percentage is based on the downtown Porto area, which is relatively small. The impact could be even more significant when working with data from a larger region.

If we prioritize quality, the 25m grid appears to be the preferable choice. However, the marginal F1-score improvement of less than 5% does not justify the 30% increase in execution time. In contrast, the use of a 50m grid over a 100m one can be rationalized by the 26% boost in F1-score it offers.

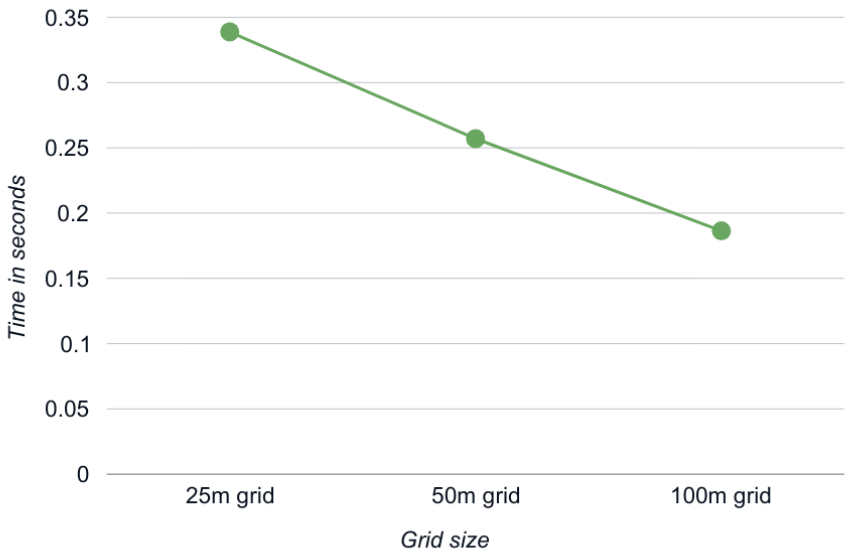


Figure 4.9: Grid size vs. execution time (to fill a gap of 1km)

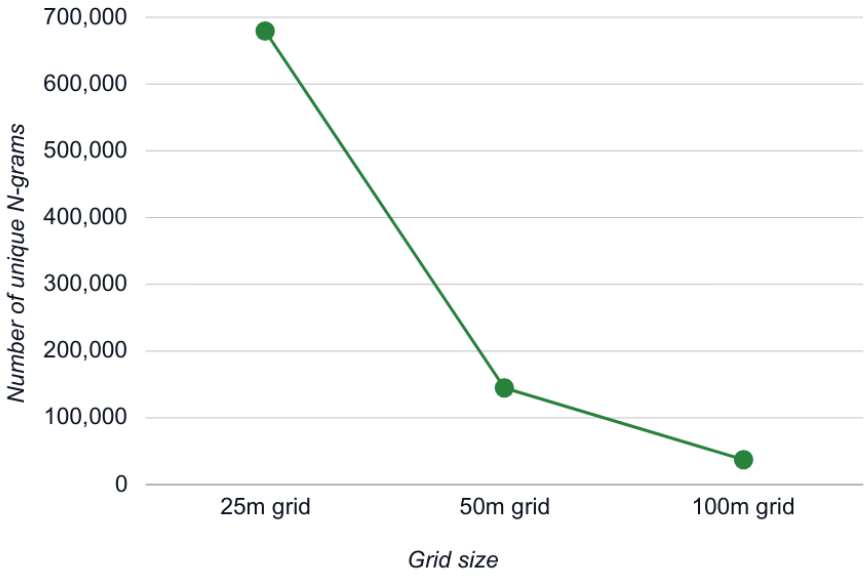


Figure 4.10: Grid size vs. N-gram count

Based on these results, we have decided to use a grid size of 50 meters for the subsequent experiments.

4.5.2 Context Size

The second experiment focuses on investigating the impact of context size in the task of predicting the next token. "Given the preceding N-1 tokens, i.e., context size of N-1, what is the most likely next token?". We started by using bigrams, which provide a context size of one token, aiming to predict the most likely next token based on just the immediately preceding token. We increased the N-gram size to up to 6-grams, which would provide a context size of five tokens, to predict the next one. The main objective of this experiment was to understand whether increasing the context size would lead to an improvement in the quality of the predictions.

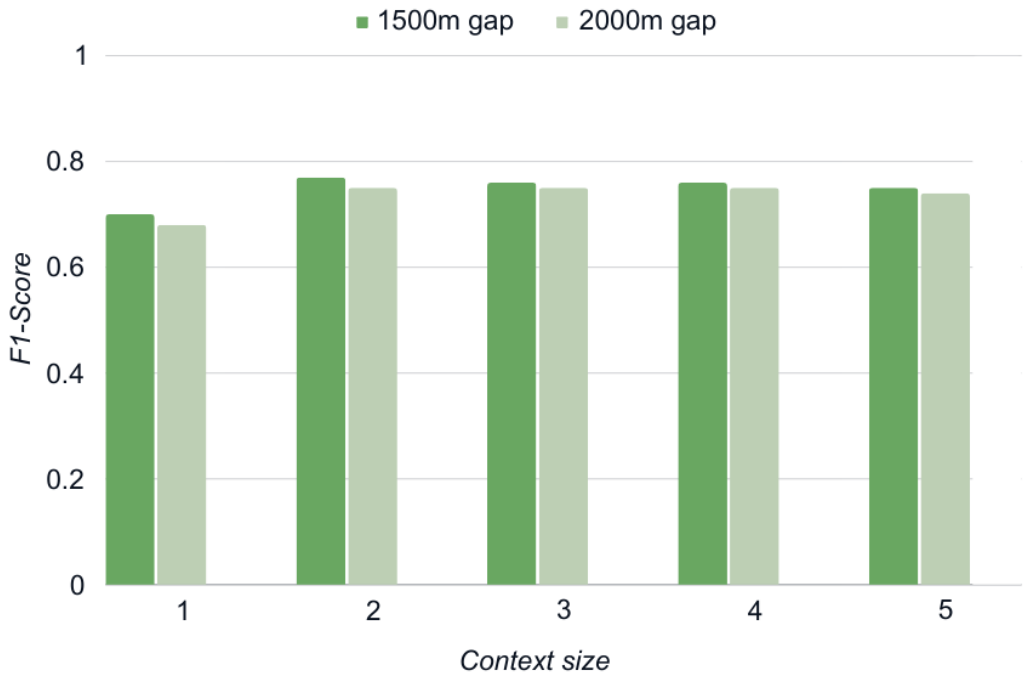


Figure 4.11: Context size vs. F1-score

The results of these experiments (Figure 4.11) show that using two previous tokens instead of just one improves prediction quality. However, increasing the context size beyond this point does not provide further improvements. For instance, in the 2km gap, increasing the context from 1 to 2 results in a 7% increase in F1-score. However, the additional context, up to a context size of 5, does not result in any improvement, in fact can slightly decrease it.

We can examine the reason for this outcome in Figure 4.12. Given a single point, we lack the means to detect the direction from which the vehicle is approaching. Therefore, given only $p8$, points $p7$, $p9$ and $p10$ will all be considered as potential next points (Figure 4.12b), as there is no way of determining the trajectory direction. However, based on the direction of the trajectory shown on the diagram, it appears highly unlikely that $p7$ could be the actual next point. By adding one more token to our context, i.e., using $p7$ and $p8$ (Figure 4.12c), we gain a clearer understanding that the vehicle is moving from right to left, narrowing the selection to $p9$ and $p10$ as possible

next points.

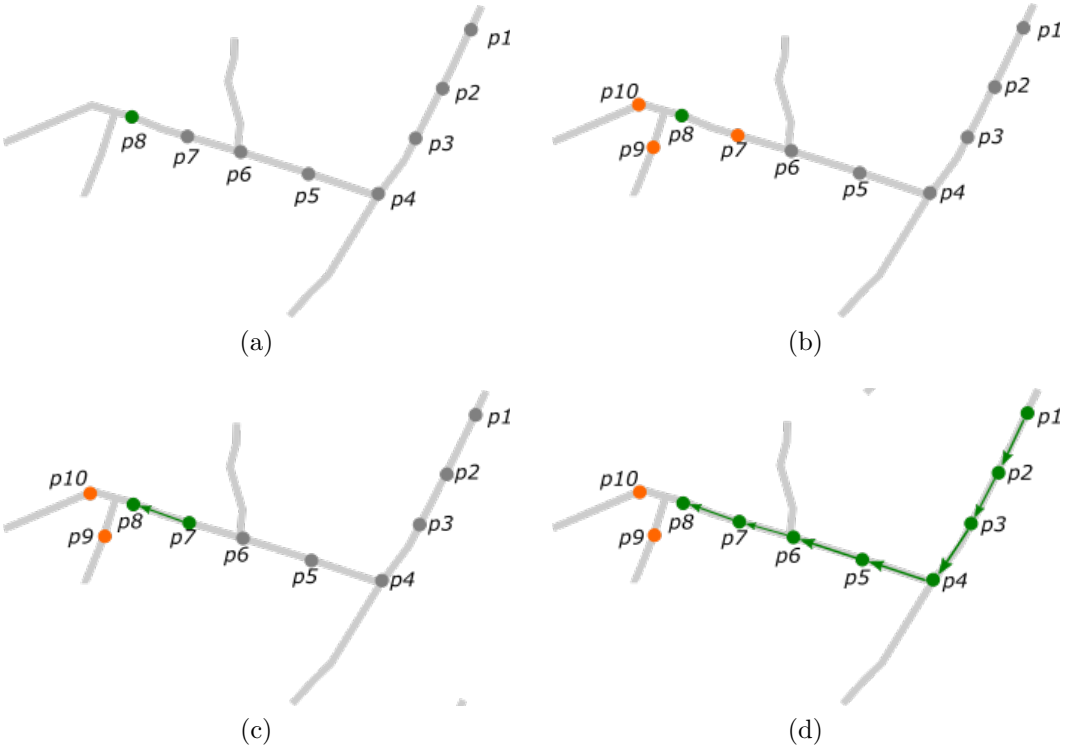


Figure 4.12: (a) shows a context of one point (green), the objective is to predict the next point. (b) and (c) show the possible next points (orange), with context sizes of one and two points, respectively. (d) shows how the increase in context does not change the potential next points.

However, beyond this point, further expansion of the context size does not provide us with any additional usable information. Even when we include all points from $p1$ to $p8$ in the context (Figure 4.12d), the likely next points remain $p9$ or $p10$. This explains why we do not observe any improvement in prediction quality with an increase in context size.

This is an aspect that is worth paying attention to. In natural language, as shown in the example within section 2.2.2, context plays a critical role in determining the next word. Words that appeared several words or even sentences ago can provide important information about what the next word should be. However, in the domain of trajectory data, our expectations differ. The results of our experiments align with this expectation, showing that knowing the complete trajectory of a vehicle does not provide any additional value to the

prediction of the next point compared to what can be provided by the last two previous tokens.

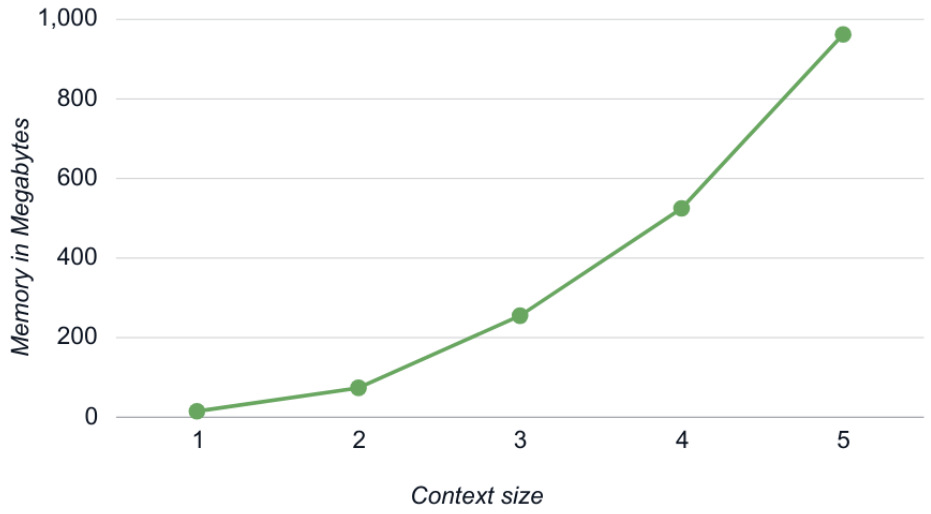


Figure 4.13: Context size vs. memory

Expanding the context size beyond 2 not only fails to improve the quality but it also significantly increases the memory required to store the N-grams (Figure 4.13), which makes larger N-grams less appealing.

When it comes to the computation time of the different N-gram sizes, we see that a context size of one, which has the lowest F1-score compared to the larger context sizes, also consumes more time (Figure 4.14). This is primarily because using bigrams (a context size of 1) generates a lot more possibilities for the next points in comparison to using a larger context size, such as trigrams or higher. Considering all these factors, opting for a context size of 2, (i.e., using trigrams), seems to provide a good compromise between effectiveness and efficiency.

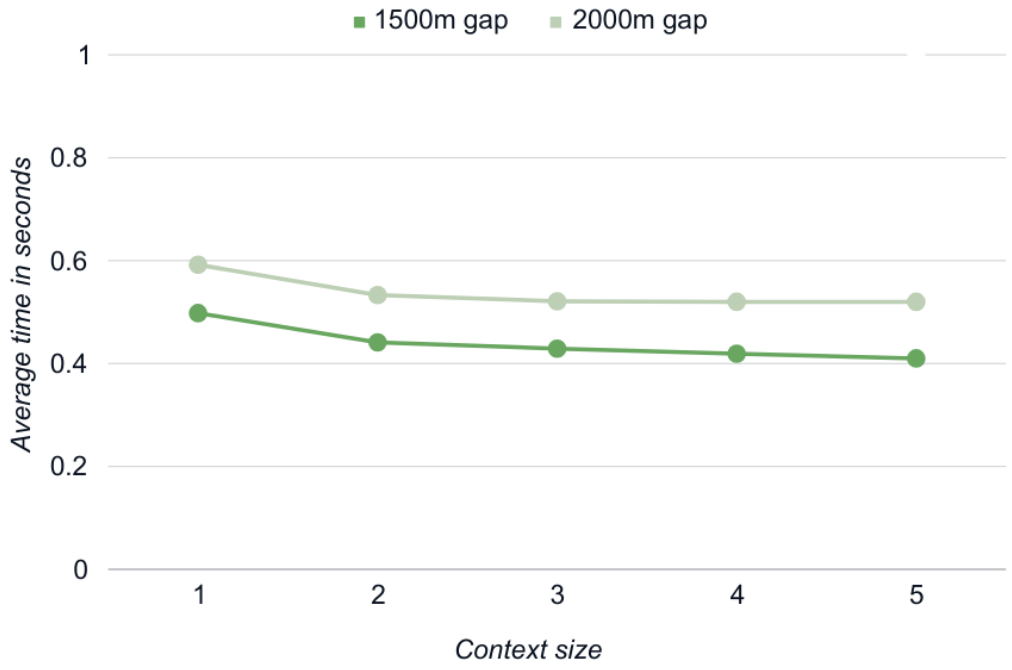


Figure 4.14: Context size vs. time

4.5.3 Comparing PaLMTo-G-1-1, PaLMTo-G-3-1 and PaLMTo-G-3-3

In this section, we examine the performance of the three different methods explained in Section 3.2.1 under PaLMTo-Generative approach. We analyze these methods in terms of their effects on both quality of predictions and execution time.

The first approach, which simply predicts the most probable next point by alternating between each side of the gap, has the lowest F1-score—which is to be expected (Figure 4.15). Under this approach, each side of the gap generates paths that tend to follow the most frequently used routes, without taking into account the destination or endpoint. Consequently, these two paths may never meet, as is the case with the examples in Figure 4.17 – shown in red. (We use linear interpolation to connect the gaps if they fail to meet.)

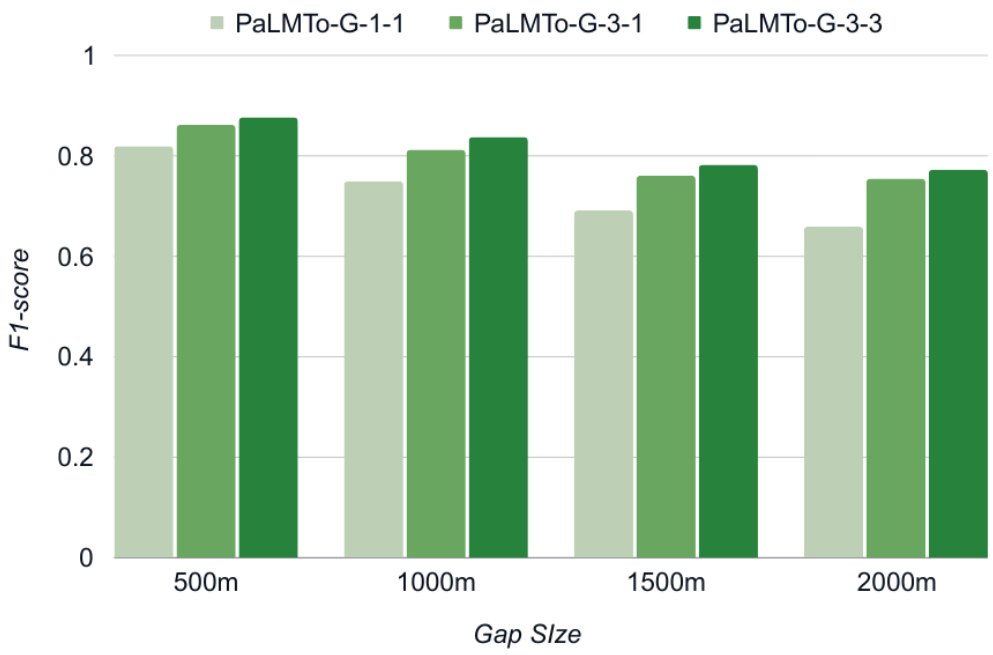


Figure 4.15: F1-Score of the 3 different methods of the PaLMTo-Generative approach

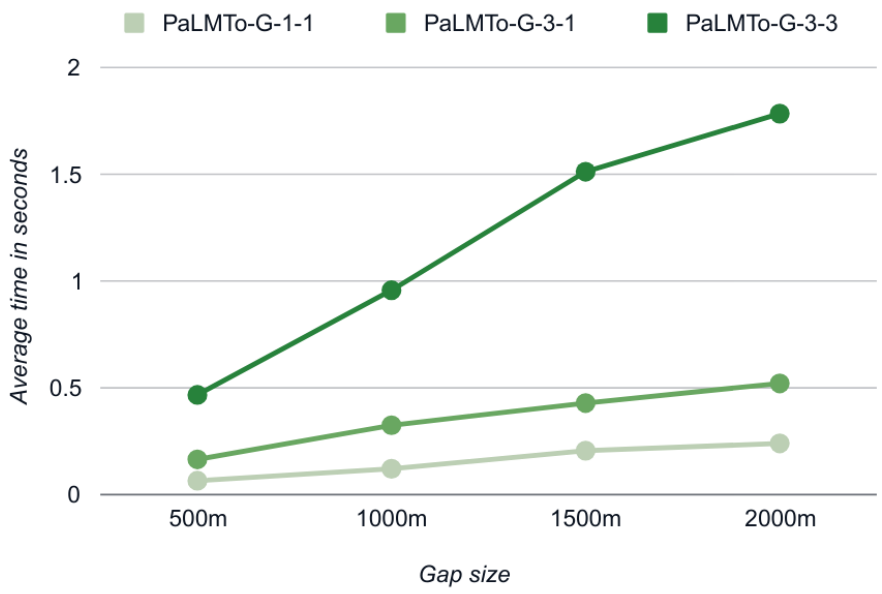
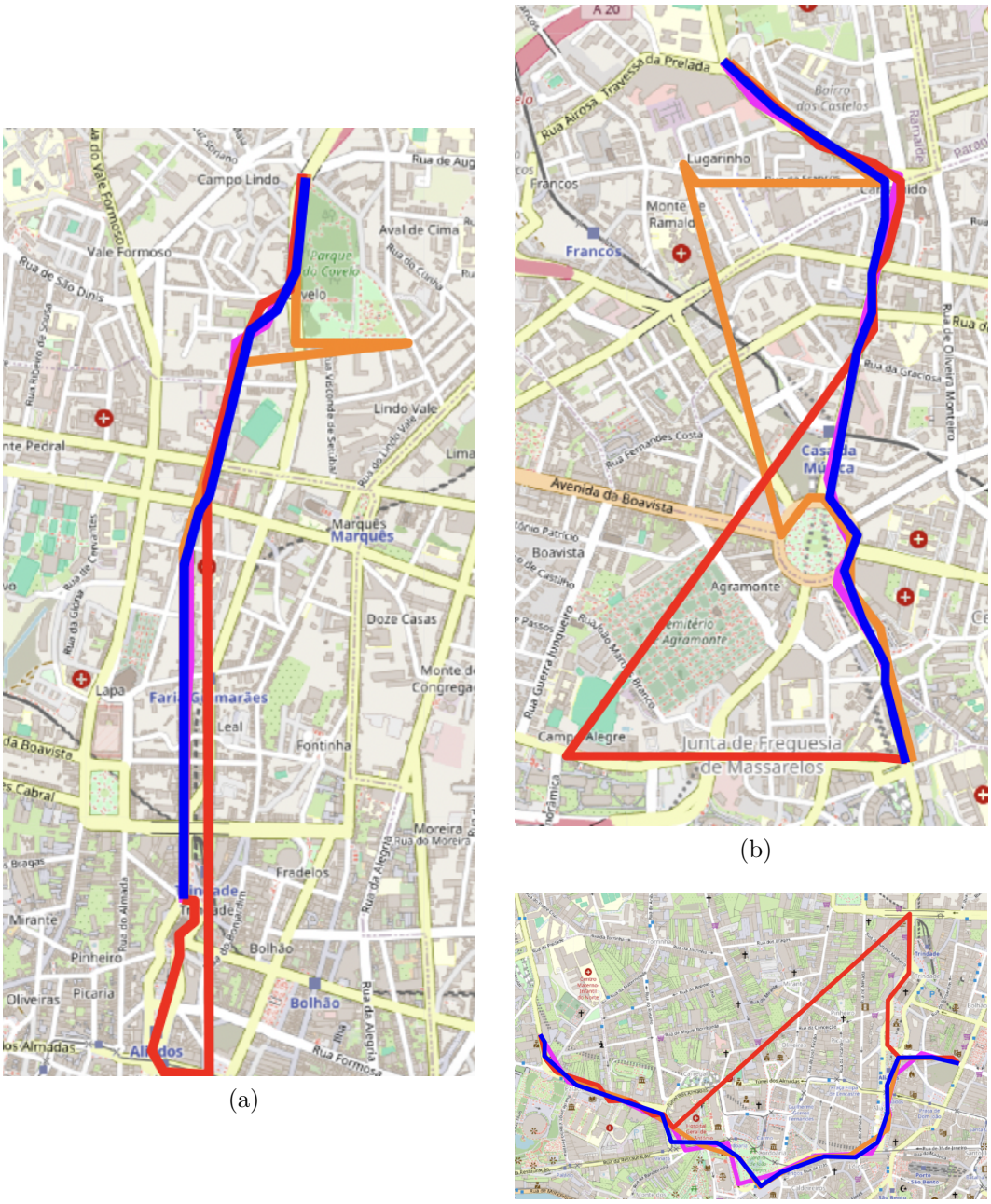


Figure 4.16: Execution time of the 3 different methods of the PaLMTo-Generative approach

The second method, which selects the three most probable tokens from each side and keeps those closest to one another, shows an improved F1-score compared to the initial approach.



(a)

(b)

(c)

Figure 4.17: Real world examples of the 3 different methods - Original path (*magenta*), PaLMTo-G-1-1 (*red*), PaLMTo-G-3-1 (*orange*), PaLMTo-G-3-3 (*blue*)

The final method, which generates three potential paths and selects one based on perplexity, achieves the highest F1-score, albeit with a marginal difference from the second method. However, the computational time for this

approach is considerably longer (Figure 4.16). Given that the increase in F1-score is minimal (Figure 4.15) and the execution time significantly longer, opting for the second method seems more practical.

Nevertheless, when we consider real-world examples of these three methods, a different perspective emerges. For instance, in Figure 4.17a with the PaLMTo-G-3-1 approach (shown in violet color), only a few points fall beyond the 25-meter threshold. While these few "wrong predictions" might not substantially affect the overall F1-score, they result in a path that would be impractical for real-world usage.

4.5.4 Dataset

In this section, we take a closer look at how the dataset impacts prediction quality. We focus on two key factors: the size of the dataset and the sparsity of the data.

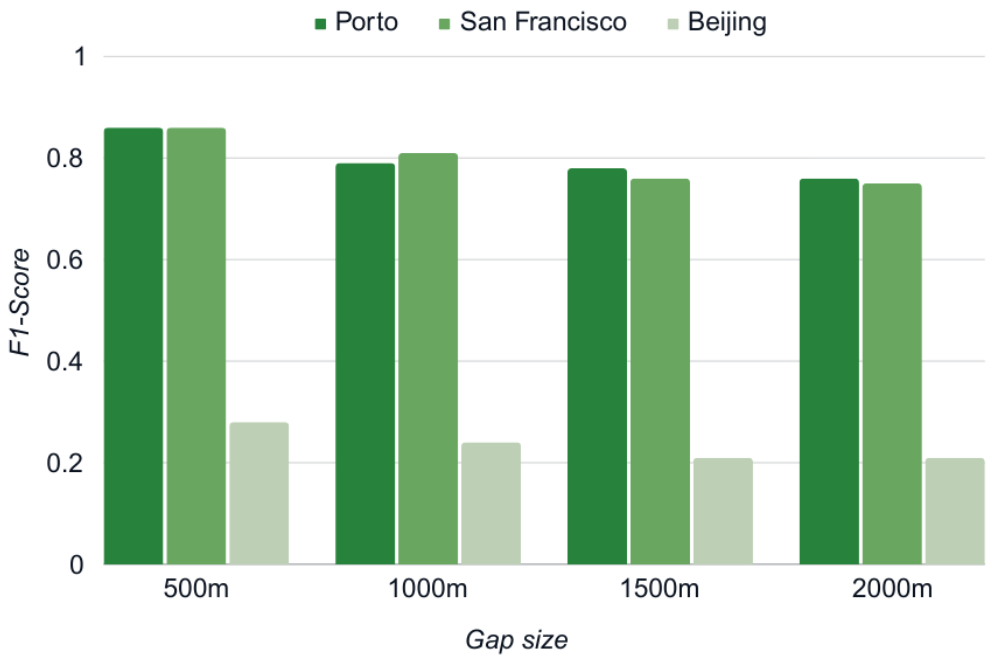


Figure 4.18: F1-score of predictions for the three datasets: Porto, San Francisco and Beijing

We first check how sparsity affects the quality by comparing the results obtained from the 3 different datasets. As previously detailed in Section 2, Porto and San Francisco exhibit a similar level of sparsity, with only a 30m difference between the two. In contrast, the dataset from Beijing is significantly sparser, being approximately 5 times more sparse than the datasets from Porto and San Francisco (Figure 4.19).

Porto and San Francisco datasets exhibit promising outcomes, whereas the results for Beijing are less favorable (Figure 4.18).

Figure 4.19: The average gap between consecutive points for the three datasets: Porto, San Francisco and Beijing

However, it is important to note that this decline in performance for the Beijing dataset is not solely attributable to inaccuracies in the imputed points. In fact, in some cases, the imputed trajectory may represent a more plausible path than the original, highly sparse trajectory (Figure 4.20). Nonetheless, since our evaluation is based on a comparison against the original trajectory, the F1-score of the imputed path may appear to be less than optimal, overall

falling below 50%.

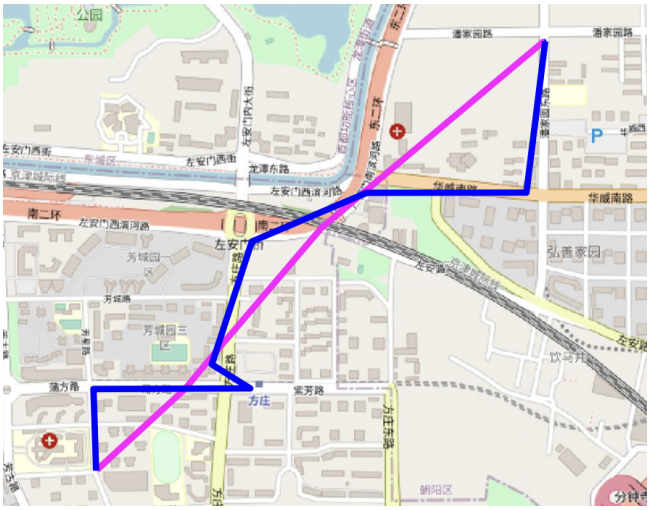


Figure 4.20: Path from the predicted points (*blue*), path from original points (*magenta*)

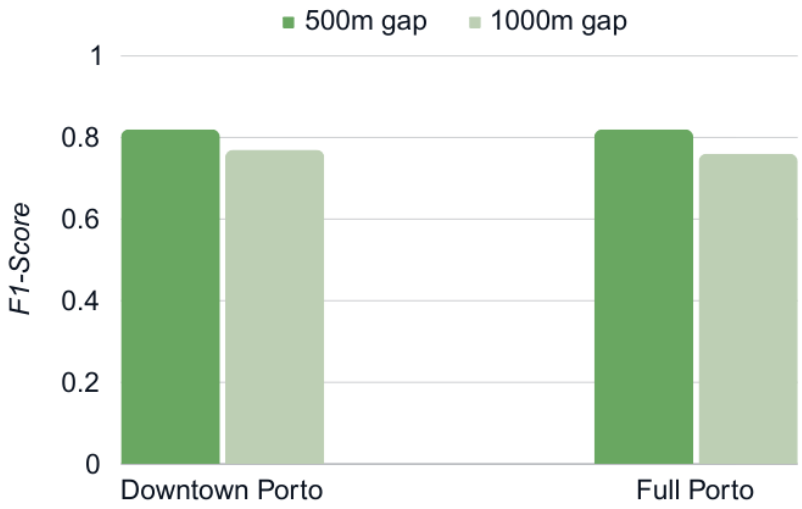


Figure 4.21: F1-score of predictions in dense region in Porto vs. the entire city

Next, we examine how the size of the dataset impacts prediction quality. To do this, we compare results from the smaller Porto region with those from

the entire city. In the smaller region, we have over three times more points per square kilometer compared to the entire city. However, the results from these two datasets are very similar (Figure 4.21). In other words, having a larger dataset for the full Porto area does not seem to lead to an improved F1-score.

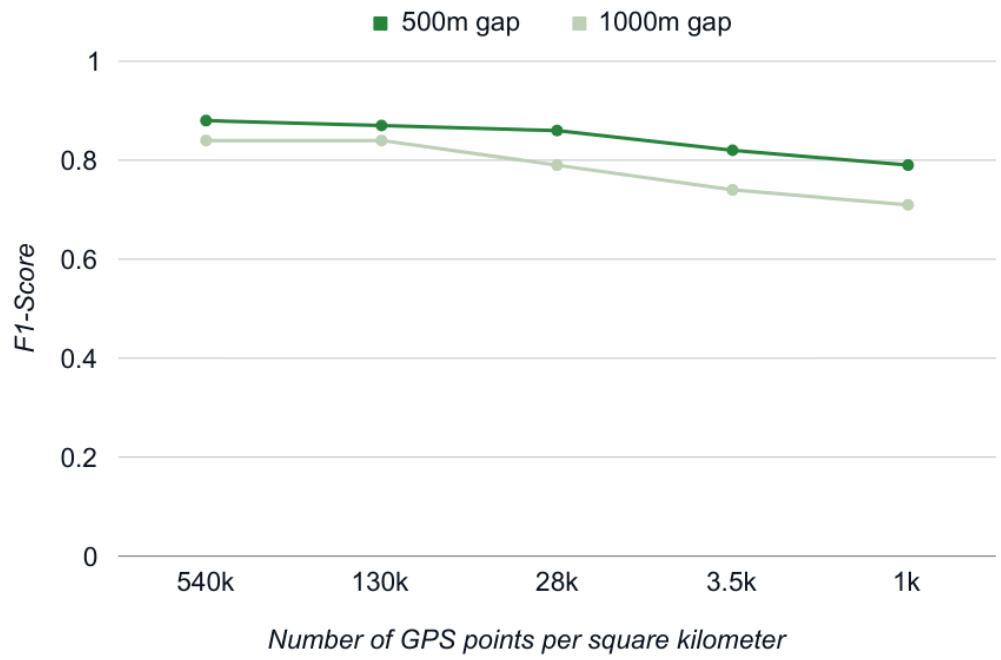


Figure 4.22: Minimal decline in F1-score when the size of the dataset decreases

To explore this further, we conducted tests on the small Porto area by reducing the dataset size to 1/2, 1/4, 1/8 th, and so on, of the available data. Figure 4.22 shows that there is only a marginal decline in F1-score with each reduction in dataset size. Even when the dataset is as small as 1000 points per square kilometer, compared to the over 1 million we have, the decrease in F1-score remains relatively small.

This highlights another advantage of probabilistic language models: they are not as data-intensive as larger language models.

4.5.5 Gap Size

In this section, we analyze how gap size influences prediction qualty. We conducted experiments using gap sizes of 500m, 1000m, 1500m, and 2000m.

As expected, F1-score decreases as the gap size increases. However, this decline in F1-score does not necessarily indicate a failure of the model to handle larger gaps effectively. Instead, it reflects the challenges posed by larger gaps. With larger gaps, there is a higher likelihood of multiple possible routes connecting two points.

In such cases, our model identifies the most likely route, but there is a chance that the test trajectory did not follow that specific route. Consequently, even if the predicted route is a valid one, with all predicted points situated on actual roads (Figure 4.23), it is still considered an incorrect prediction due to the potential mismatch with the test trajectory. This difference is the primary factor contributing to the drop in F1-score as the gap size increases.

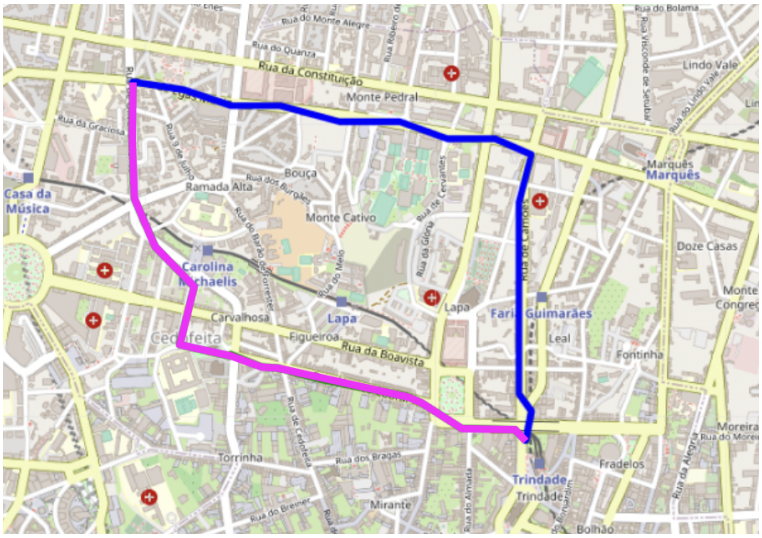


Figure 4.23: The predicted path (*blue*) differs from the actual path (*magenta*), but it is a viable route.

4.5.6 PaLMTo-Lookup vs. PaLMTo-Generative method

In this section, we analyze the performance of the two approaches, PaLMTo-Lookup with PaLMTo-Generative, explained in Section 3.2. The experiment aims to show the trade-off between time and memory required to achieve

similar results for both methods.

We already have results from PaLMTo-Generative, which only requires storing unigrams, bigrams and trigrams. To achieve similar outcomes with the PaLMTo-Lookup, we have to store larger N-grams, and these N-grams get larger as the gap size increases. For instance, to bridge a 500m gap, we need to store up to 6-grams, and for a 1km gap, this size increases to 12-grams. As shown in Figure 4.26, the memory needed to store these N-grams gets significantly larger as the gap size grows.

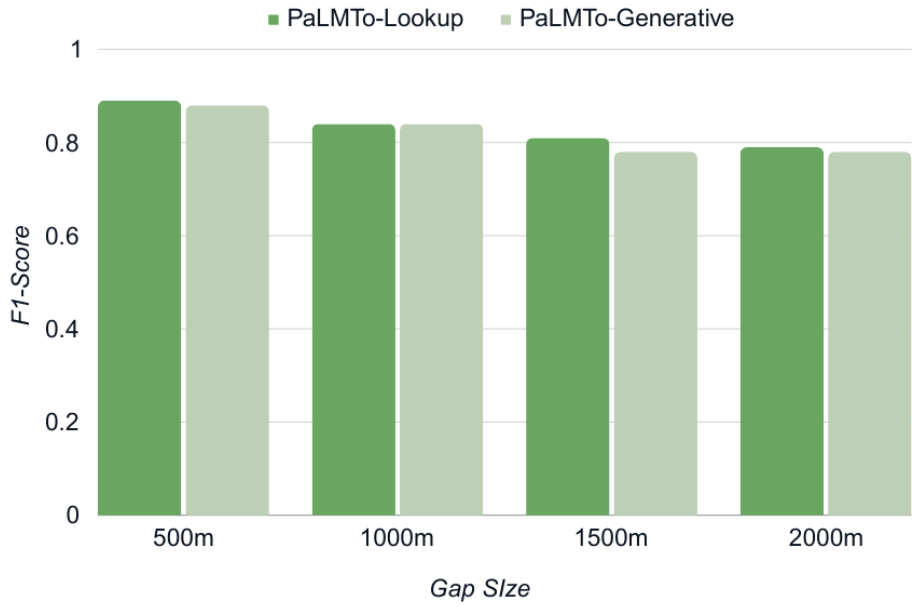


Figure 4.24: PaLMTo-Lookup vs. PaLMTo-Generative (F1-score)

The advantage of the PaLMTo-Lookup is that it fills the gap all at once, making it much faster than PaLMTo-Generative. The execution time is a fraction of that required by PaLMTo-Generative and remains constant for all gap sizes. (Figure 4.27)

On the other hand, with PaLMTo-Generative, we only need to store N-grams up to 3-grams, resulting in a significantly smaller and constant memory requirement, irrespective of gap size.

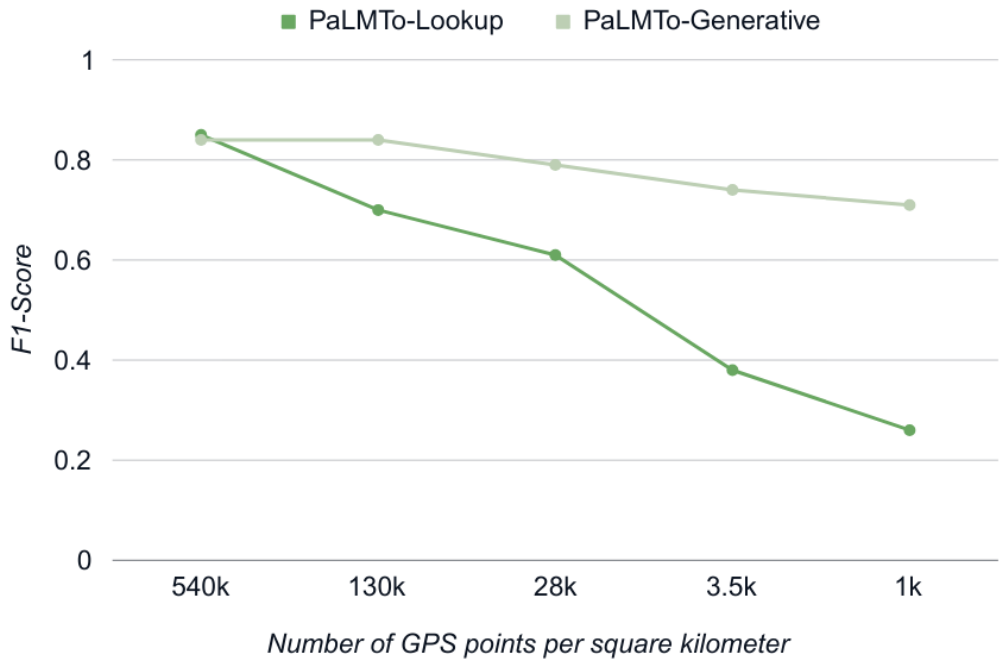


Figure 4.25: PaLMTo-Lookup vs. PaLMTo-Generative (data size)

As explained in Section 3.2.2, PaLMTo-Lookup relies on the repetitive nature of trajectory datasets. As a result, it requires a substantial amount of data to yield satisfactory results.

Figure 4.25 shows how dataset size affects the performance of both methods. When using PaLMTo-Generative, even a significant reduction in data size to just 1/20th of the complete dataset did not result in a substantial decrease in F1-score. With as few as 1000 points per square kilometer, we were able to achieve an F1-score above 70% in bridging a 1-kilometer gap.

However, when using PaLMTo-Lookup, reducing the dataset size led to a sharper decline in F1-score. The score fell below 40% when the dataset contained only 3.5k points per square kilometer whereas the PaLMTo-Generative method had an F1-score of 75% for the same dataset size.

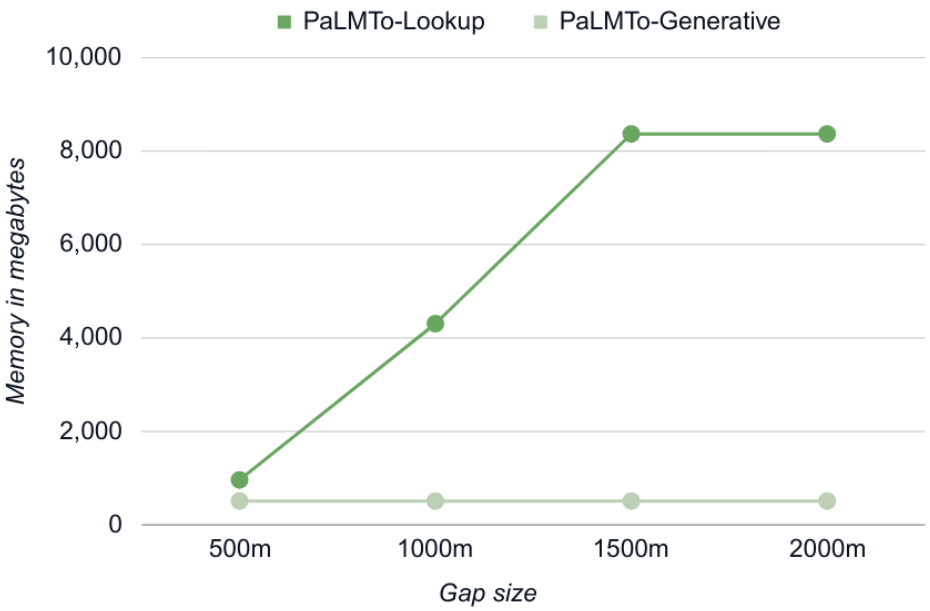


Figure 4.26: PaLMTo-Lookup vs. PaLMTo-Generative (memory)

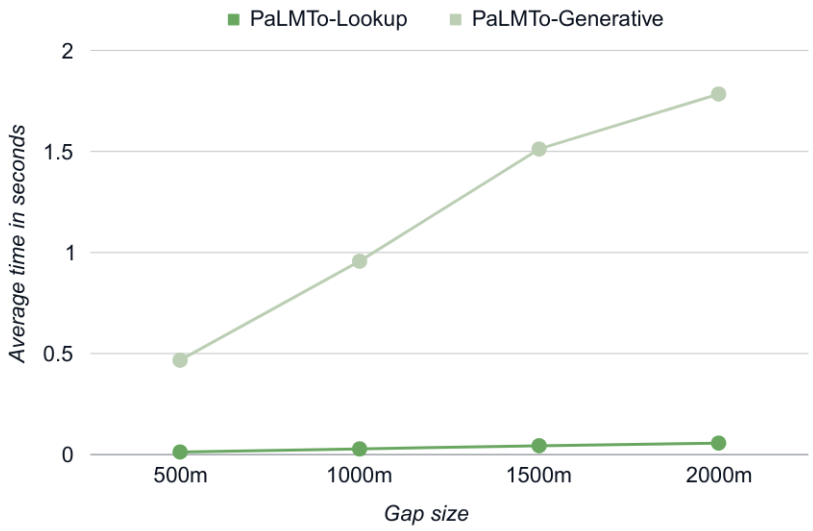


Figure 4.27: PaLMTo-Lookup vs. PaLMTo-Generative (time)

4.6 PaLMTo vs. TrImpute

We compared the results of our experiments with TrImpute [5], an approach discussed in section 2.3.1. TrImpute aims to address both spatial and temporal imputation challenges, while our approach focuses solely on spatial imputation. However, during our experiments, we observed that its temporal imputation simply duplicates the data from the previous timestamp. Compared to our approach, which requires less data than large language models, TrImpute requires even less, which is a major advantage over our approach.

Our results show that when imputing a gap of 2km, our approach achieved an F1-score of 85% (with a 50m threshold), which was a noticeable improvement to the 69% achieved by TrImpute (Figure 4.28).

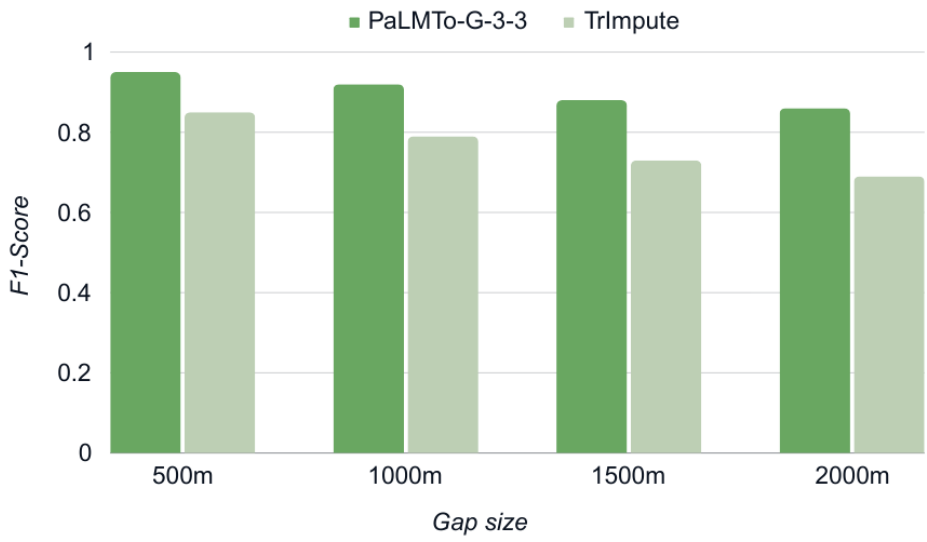


Figure 4.28: PaLMTo vs. TrImpute (50m threshold)

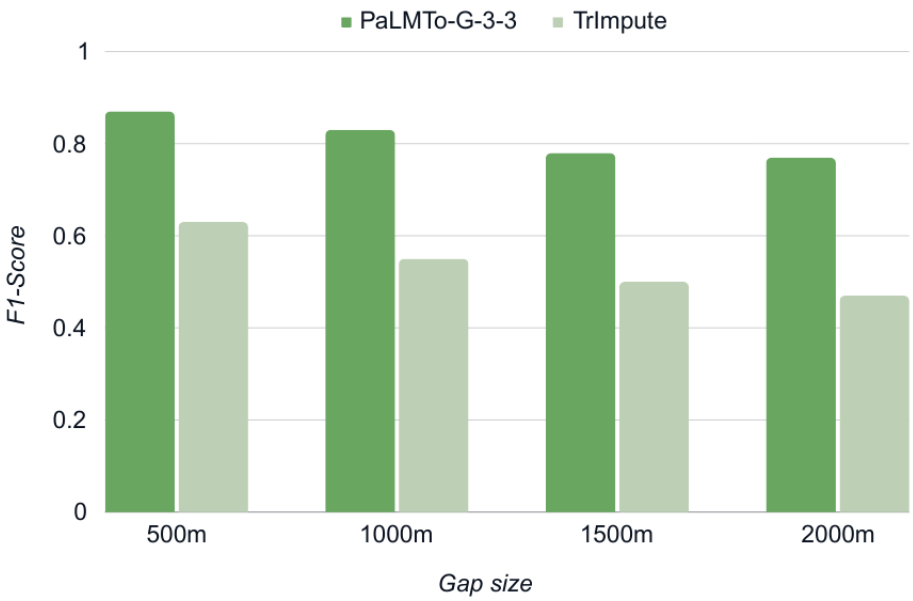
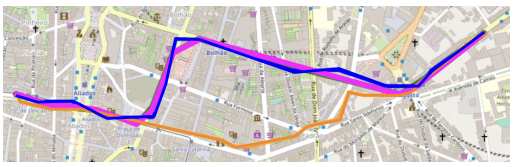


Figure 4.29: PaLMTo vs. TrImpute (25m threshold)

Furthermore, when we employed a stringent quality measure, checking how many of the imputed points were within 25 meters of the original trajectory, our approach achieved a 75% F1-score. In contrast, TrImpute's fell below 50% (Figure 4.29), showing the reliability of the Probabilistic Language Model approach in accurately imputing trajectory data, especially in scenarios demanding a more precise spatial prediction.

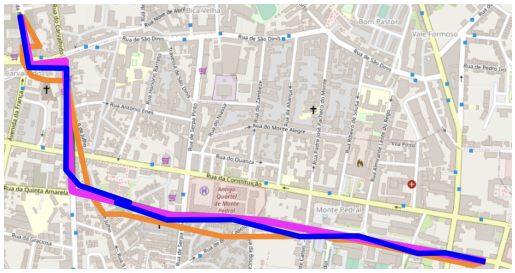
Figure 4.30 provides real-world examples illustrating the performance of PaLMTo and TrImpute. In cases where TrImpute encounters difficulty imputing points, it resorts to linear interpolation, as can be seen in Figure 4.30, a method that we also use if the generated points fail to get close enough to close the gap. However, such situations occur less frequently in our method than in TrImpute. Both approaches face the challenge of generating routes that are viable yet different from the original trajectory. Lastly, even when the routes generated by both methods closely resemble each other, PaLMTo's output is very close to the original trajectory, and by extension, the road network, in comparison to TrImpute.

(a)



(b)

(c)



(d)

(e)

Figure 4.30: Real world examples comparing the performance of PaLMTo with TrImpute - Original path (*magenta*), PaLMTo (*blue*), TrImpute (*orange*)

Chapter 5

Conclusion and Future Work

5.1 Summary

In conclusion, our research demonstrates the effectiveness of probabilistic language models in trajectory imputation. We investigated the impact of cell size within the grid system that is used to convert trajectories into sentences. We explored different approaches, using language models both as a means to generate points to complete gaps and as repositories for quick data retrieval, to understand the benefits and drawbacks of each one. Our exploration of various N-gram sizes allowed us to understand how longer context sizes impact trajectory completion. Additionally, we experimented with datasets of varying density to understand how it affects the trajectory completion task.

The following are key summaries of the experiments discussed in the previous chapter.

a. Can PLMs solve the trajectory imputation problem?

As shown with the results in section 4.6, PLMs are a very effective method to solve the trajectory imputation problem. They deliver good results while avoiding the computational and data-intensive demands associated with large language models. Moreover, the method can address a range of trajectory completion scenarios, from filling small gaps in trajectories to providing full route information.

b. What are the trade-offs between grid size and quality of predictions?

Smaller grid sizes offer better accuracy but the associated execution time may not always justify their use, particularly in real-time applications. In contrast, due to their smaller vocabulary size, larger grid sizes provide faster execution times, but they come at the cost of reduced accuracy.

c. How does the choice of N-gram size affect the quality of predictions?

Accuracy improves with an increasing context size up to a certain point (a context size of 2), beyond which further expansion provides no additional benefit. Unlike natural language, a longer context does not improve prediction accuracy in the domain of trajectory data.

d. What are the trade-offs between N-gram size and efficiency (memory/computation time)?

As we increase the N-gram size, we do save computation time because the number of possible next points we need to evaluate decreases. However, the trade-off is that the storage requirement grows rapidly as we transition from lower N-grams to higher ones. It is also important to note that, except for the transition from bigrams to trigrams, there is no significant improvement in accuracy with larger N-grams. Therefore, the minor time savings become irrelevant, and it appears that trigrams offer the best balance between efficiency and accuracy.

e. How do methods that rely solely on probability, those incorporating probability with distance, and those incorporating probability with distance and perplexity compare with one another?

Among the three different methods explored, the one that uses perplexity to choose the best path among the three options achieved the highest accuracy. However, the second method, which generates a single path based on the distance between the predicted points, offers a more efficient balance between accuracy and execution time. The method

that relies solely on probability, although the fastest, had the lowest accuracy.

f. How does the dataset (size, sparsity) affect the quality of the predictions?

Dataset sparsity significantly affects accuracy, with Beijing data having lower accuracy due to greater sparsity. However, significantly reducing the dataset size had minimal impact on accuracy, highlighting the efficiency of probabilistic language models for smaller datasets.

g. How does the gap size affect the quality of the predictions?

Larger gap sizes lead to reduced accuracy mainly due to multiple potential routes between points, leading to predicted routes that are plausible but differ from the actual test trajectories.

h. Can a database-like "lookup" on previous trajectories be as effective as generating points using PLMs "as usual"?

Due to the repetitive nature of trajectory data, especially in large datasets, it is possible to identify N-grams that can fill small gaps without generating each point using the language model. While this approach may save time, it incurs a substantial memory overhead. In contrast, generating each point using the language model ensures the flexibility to address gaps of any size while requiring only a fraction of the memory required to store large N-grams.

5.2 Future work

While it is indisputable that large language models have made significant strides in addressing long-term dependency issues inherent in natural languages, we raise the question of their relevance in trajectory data where such issues are not applicable.

However, it is important to note that trajectory imputation represents just a small facet of the broader spectrum of trajectory-related tasks. Future

research could explore the shared characteristics between natural language and trajectory data that warrant the use of large language models in other trajectory-related tasks.

Additionally, there are areas within our own model that need further investigation and improvement. One such challenge is the loss of accuracy when working with larger grid sizes, which is needed to efficiently handle trajectory data over extensive geographic areas. Our research has explored the effects of grid size within the context of a rectangular grid. It is important to further investigate how different grid shapes might influence accuracy. Furthermore, the temporal aspect of spatio-temporal data remains a critical dimension that our current work has yet to incorporate.

Another possible area of future work is investigating the application of this approach for generating synthetic datasets. This could prove to be useful in situations where detailed road network maps are not available.

References

[1] C. Calenge, S. Dray, and M. Royer-Carenzi, "The concept of animals' trajectories from a data analysis perspective," *Ecological informatics*, vol. 4, no. 1, pp. 34–41, 2009.

[2] B. Y. Chen *et al.*, "Map-matching algorithm for large-scale low-frequency floating car data," *Intl. J. of Geographical Information Science*, vol. 28, no. 1, pp. 22–38, 2014.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Association for Computational Linguistics, 2018, pp. 4171–4186.

[4] T. Eiter and H. Mannila, "Computing discrete fréchet distance," Technische Universit at Wien, Tech. Rep. CD-TR 94/64, 1994.

[5] M. M. Elshrif, K. Isufaj, and M. F. Mokbel, "Network-less trajectory imputation," in *Proceedings of the 30th International Conference on Advances in Geographic Information Systems (Seattle, Washington) (SIGSPATIAL '22)*, Association for Computing Machinery, New York, NY, USA, 2022, pp. 1–10.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] Y.-L. Hsueh *et al.*, "Map matching for low-sampling-rate gps trajectories by exploring real-time moving directions," *Information Sciences*, vol. 433, pp. 55–69, 2018.

[8] M. Jordan, "Serial order: A parallel distributed processing approach," California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, Tech. Rep. ICS 8604, 1986.

[9] J. Krumm, "Maximum entropy bridgelets for trajectory completion," in *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, 2022, pp. 1–8.

[10] M. Li, X. Ye, S. Zhang, X. Tang, and Z. Shen, "A framework of comparative urban trajectory analysis," *Environment and Planning B: Urban Analytics and City Science*, vol. 45, no. 3, pp. 489–507, 2018.

[11] L. Liao, Y. Lin, W. Li, F. Zou, and L. Luo, "Traj2traj: A road network constrained spatiotemporal interpolation model for traffic trajectory restoration," *Transactions in GIS*, 2023.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[13] W. K. moreiraMatias, *Ecml/pkdd 15: Taxi trip time prediction (ii)*, 2015. [Online]. Available: `https://kaggle.com/competitions/pkdd-15-taxi-trip-time-prediction-ii`.

[14] M. Musleh, "Towards a unified deep model for trajectory analysis," in *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, 2022, pp. 1–2.

[15] M. Musleh and M. Mokbel, "A demonstration of kamel: A scalable bert-based system for trajectory imputation," in *Companion of the 2023 International Conference on Management of Data*, 2023, pp. 191–194.

[16] L. Qu, L. Li, Y. Zhang, and J. Hu, "Ppca-based missing data imputation for traffic flow volume: A systematical approach," *IEEE Transactions on intelligent transportation systems*, vol. 10, no. 3, pp. 512–522, 2009.

[17] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[18] H. Ren, S. Ruan, Y. Li, *et al.*, "Mtrajrec: Map-constrained trajectory recovery via seq2seq multi-task learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1410–1419.

[19] R. Rosenfeld *et al.*, "A maximum entropy approach to adaptive statistical language modelling," *Computer speech and language*, vol. 10, no. 3, p. 187, 1996.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep. ICS 8504, 1985.

[21] R. Simmons, B. Browning, Y. Zhang, and V. Sadekar, "Learning to predict driver route and destination intent," in *2006 IEEE intelligent transportation systems conference*, IEEE, 2006, pp. 127–132.

[22] S. Tak, S. Woo, and H. Yeo, "Data-driven imputation method for traffic data in sectional units of road links," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 6, pp. 1762–1771, 2016.

[23] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017.

[24] J. Wang, N. Wu, X. Lu, W. X. Zhao, and K. Feng, "Deep trajectory recovery with fine-grained calibration using kalman filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 3, pp. 921–934, 2019.

[25] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 25–34.

[26] G. Welch, G. Bishop, *et al.*, *An introduction to the kalman filter*, University of North Carolina at Chapel Hill, 1995.

[27] Z. Xiao, Y. Wang, K. Fu, and F. Wu, "Identifying different transportation modes from trajectory data using tree-based ensemble classifiers," *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, p. 57, 2017.

[28] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 316–324.

[29] J. Yuan, Y. Zheng, C. Zhang, *et al.*, "T-drive: Driving directions based on taxi trajectories," in *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*, 2010, pp. 99–108.

[30] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" *arXiv preprint arXiv:2205.13504*, 2022.

[31] Y. Zheng, "Trajectory data mining: An overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, pp. 1–41, 2015.