Data Analytics and Machine Learning Group
TUM School of Computation, Information and Technology
Technical University of Munich

Trajectory Prediction using Denoising Diffusion Models

Joe Schmit

Thesis for the attainment of the academic degree

Master of Science (M.Sc.)

at the TUM School of Computation, Information and Technology of the Technical University of Munich.

Examiner:
Prof. Dr. Stephan Günnemann

Supervisor:
D. Fuchsgruber
M. Kollovieh
D. Lüdke
V. Dahmen

Submitted:
Munich, 16.10.2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 16.10.2024 Joe Schmit

Abstract

This thesis introduces a novel discrete diffusion framework to predict realistic trajectories on road network graphs.
With advancements in autonomous vehicles and more interconnected cities, trajectory prediction on road networks has become an important research area. Autoregressive models, currently applied to this task, traditionally suffer from issues such as error accumulation and struggle to model long-term dependencies. To address these limitations, we propose a non-autoregressive, diffusion-based framework that generates sequences of edges representing plausible paths within a given graph structure. Our non-autoregressive sampling approach allows for simultaneous prediction of all edges during the denoising process.
Our method splits trajectories into history and future components. We use a GNN model, conditioned on the history, as a backbone for this task. We employ a discrete diffusion process tailored for binary variables and experiment with different transition matrices and noising schedules.
We evaluate our approach on four different datasets: pNEUMA, T-Drive, Geolife, and Mobilität.Leben. We conduct various experiments including standard and conditional future trajectory prediction and trajectory inpainting. In terms of ADE and FDE, our model achieves comparable performance to the autoregressive benchmarks. Notably, the conditional version of our model, which incorporates a predefined future horizon, shows enhanced performance. It outperforms autoregressive benchmarks, successfully generates longer, valid trajectories, and can generate reasonable paths in path planning tasks.

Contents

1 Introduction

1.1 Motivation

As transportation systems evolve and cities become increasingly interconnected, the need for accurate trajectory prediction models that can handle complex road networks is more pressing than ever [1, 2]. Predicting realistic trajectories has manifold applications in different fields, including autonomous vehicles, traffic management, navigation systems, and urban planning.
City road networks are naturally represented as graphs, where intersections are modeled as nodes and road segments as edges. This graph-based representation inherently captures the topology and connectivity of the transportation network, which is crucial for predicting realistic, long-range paths. By modeling road networks as graphs, we can effectively handle the complexity arising from variable traffic conditions and intricate road layouts.

Many existing trajectory prediction models rely on Euclidean space representations, treating trajectories as sequences of spatial coordinates. This approach often fails to capture the underlying road network's structure and constraints, leading to less realistic predictions that may not adhere to actual roadways. Moreover, many Euclidean space approaches concentrate on small, specific areas within the road network and provide predictions only over very short future horizons, typically just a few seconds ahead [3, 4]. Consequently, they are ill-suited for long-range trajectory planning.
Current benchmarks for trajectory prediction on graphs mostly use autoregressive (AR) models [5, 6]. AR models come, however, with their own set of limitations. They suffer from error accumulation once they deviate from the true path during evaluation, thus limiting the quality of their results. Additionally, AR models traditionally struggle to model long-term dependencies, which is especially critical for predicting long trajectories. Lastly, even though AR models generate connected paths by limiting the decision space to neighboring edges, they do not provide the guarantee of sampling acyclic paths. Of course, realistic trajectories are, however, acyclic as people or vehicles usually do not visit the same place twice in the same trajectory.

Hence, why we introduce the first **discrete diffusion based approach for trajectory prediction on graphs**.
Denoising diffusion models, which generate synthetic data by reversing a gradual noise addition process through iterative denoising steps, have recently become a powerful new type of generative model, demonstrating exceptional performance in domains such as image and video data [7, 8]. Combined with the fact that diffusion models have outperformed AR methods in sequential tasks such as speech and music generation and audio enhancement, applying a diffusion-based approach to multi-step trajectory prediction on graphs holds significant promise [9, 10]. By predicting entire trajectories simultaneously, diffusion models operate in a non-AR manner, effectively mitigating the issue of error accumulation inherent in sequential prediction models. Moreover, diffusion-based frameworks are capable of generating a more diverse set of samples than other generative models. This diversity is particularly beneficial in real-world navigation scenarios, where constantly changing traffic patterns require adaptable and varied route predictions. Lastly, diffusion models facilitate the straightforward incorporation of conditioning, which has been shown to significantly enhance the performance of generative models, allowing for more accurate and context-aware trajectory predictions.

1.2 Thesis Structure

The thesis continues with Chapter 2, introducing the relevant theoretical background, which includes graph theory, Machine Learning (ML) techniques on graphs, trajectory prediction, and diffusion models. The diffusion models section first introduces continuous diffusion and builds upon this foundation to finally present discrete diffusion and its peculiarities.
Chapter 3 presents related work, including research on trajectory prediction in Euclidean space and on graphs. It additionally showcases the advantages of discrete diffusion in certain data domains, such as image and graph data.
In Chapter 4, we introduce our method, detailing the data processing, the model architecture we use, and the different diffusion configurations we experiment with.
The datasets that we use to evaluate our method are put forward in Chapter 5, along with some basic statistics to grasp their differences.
In Chapter 6, we present the results our method achieves in a variety of experiments.
Lastly, Chapter 7 concludes the thesis and provides an outlook to future work in this space.

2 Theoretical Background

2.1 Graphs

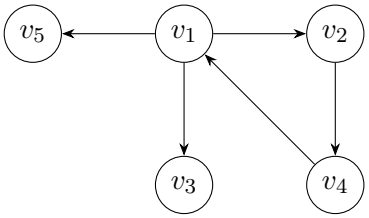
2.1.1 Graph Theory

Graphs, as defined in Def. 2.1.1, are versatile data structures, good at modeling a variety of scenarios and interactions. They concisely represent complex networks such as the World Wide Web, with webpages interconnected by links; social media platforms, with relationships among users; road networks comprising streets and intersections; and molecular structures, where atoms are joined by chemical bonds [11, 12, 13, 14]. To accommodate the diversity and specificity of these applications, graphs are categorized into several types that capture different characteristics of data and relationships.

Definition 2.1.1 (Graph) *A graph is a pair $G = (\mathcal{V}, \mathcal{E})$, where:*

- *$\mathcal{V}$ is a set of vertices or nodes $(v_1, v_2, \ldots, v_{|\mathcal{V}|})$*
- *$\mathcal{E}$ is a set of pairs of vertices $\{v_1, v_2\}$ called edges*

As such there exist directed graphs (Fig. 2.1), where the edges are **ordered** pairs of vertices: $\mathcal{E} \subseteq \{(u, v) \mid (u, v) \in \mathcal{V}^2 \text{ and } u \neq v\}$. In that case, a relationship exists uniquely from the start node to the end node, with no reciprocal relationship from the end node back to the start node. This is in contrast to undirected graphs with **unordered** pairs of vertices, where each connection between nodes is reciprocal.
Another tool to represent more complex data using graph structures consists in assigning weights to edges. For example, a link to a popular website that is used more often than other links to less popular websites should be assigned a higher weight in return by search algorithms on the web. Another example of enriching the expressive power of graphs entails labeling or classifying vertices. This classification enables the use of ML to uncover patterns in large graphs or to classify new nodes. We will explore ML on graphs in further detail in Subsec. 2.1.2.



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.1 An unweighted directed graph and its adjacency matrix.

A graph is concisely defined by its adjacency matrix, as seen in Fig. 2.1. The adjacency matrix, as defined in Def. 2.1.2, offers a compact representation of any graph and facilitates computations with and analysis of them. The adjacency matrix of undirected graphs is consequently symmetric.

Definition 2.1.2 (Adjacency Matrix) *An adjacency matrix $\boldsymbol{A}$ is a square matrix used to represent a finite graph. Its entries $\boldsymbol{A}_{i,j}$ indicate whether pairs of vertices are adjacent in a graph.*

$$\boldsymbol{A}_{i,j} = \begin{cases} w_{i,j} & \text{if there exists an edge with weight } w \text{ between vertices i and j,} \\ 0 & \text{else} \end{cases}$$

2.1.2 Graph Neural Networks

Motivation

With the concise theoretical background provided in Sec. 2.1.1 we can now explore how and why ML algorithms are applied on graphs.
Real graphs are, in fact, not random, but they exhibit patterns that can be observed, learned, and, in return, leveraged to accomplish varying tasks [15]. Examples of such patterns include a power law distribution for the average node degree or the so-called "Small World Phenomenon" [16].
These patterns are not just theoretical curiosities but are an integral part of various ML algorithms on graphs. Node classification and ranking harness the structural information of underlying graphs to sort and categorize nodes within social networks or the web, for example. Clustering algorithms also make use of nodes' tendencies to form groups, which is helpful in community detection, for instance.
Link prediction is another classic task on graph data, predicting potential links between nodes based on their attributes and existing connections. This is essential in collaborative filtering for recommender systems, where it forecasts possible relationships or product preferences [17].
Moreover, the PageRank algorithm, the foundation of Google Search, showcases the practicality of graph theory in web search by using link structures to determine the significance of web pages, thereby enhancing search result relevancy and accuracy [18].
Traditional ML techniques may, however, not be optimal for uncovering these patterns in graph data owing to several challenges. These include the potential absence of non-structural features, scalability issues on larger graphs (with computational complexity often scaling with $\mathcal{O}(|V|^2)$), and the variability in data length caused by differing numbers of neighbors for each node. This is why specialized architectures, so-called Graph Neural Networks (GNNs), have been devised to overcome these challenges [19]. We will explore their theoretical foundation and evolution in the following paragraphs.

Graph Neural Networks

GNNs rely on the concept of differentiable message passing (DMP) as defined in Algorithm 1. It allows us to learn an embedding h_v for each node v, based on its features x_v and its neighborhood $\mathcal{N}(v)$. The final embedding h_v^L is thus recursively defined in terms of the embeddings of its neighbors. As such, the number of layers L in a GNN defines the L-hop neighborhood influencing this representation of each node. Finally, this learned representation h_v can further downstream be used for tasks such as classification or link prediction.

Algorithm 1 Differentiable Message Passing on Graphs

Input: Graph $G = (\mathcal{V}, \mathcal{E})$ with node features $\{x_v\}$ for each $v \in \mathcal{V}$ and edge weights $w_{u,v}$
Output: Node representations $\{h_v^{(L)}\}$ after L layers
procedure DMP($G, \{x_v\}, L$)
    for $v \in V$ **do**
        $h_v^{(0)} \leftarrow x_v$ ▷ Initialization
    end for
    for $l = 1$ **to** L **do**
        for $v \in V$ **do**
            $m_v^{(l)} \leftarrow \sum_{u \in \mathcal{N}(v)} f(h_u^{(l-1)}, h_v^{(l-1)}, w_{u,v})$ ▷ Aggregate messages from neighbors
            $h_v^{(l)} \leftarrow g(m_v^{(l)}, h_v^{(l-1)})$ ▷ Update node embedding
        end for
    end for
    return $\{h_v^{(L)}\}$
end procedure

Here, f represents the message function, and g is the update function. Both are differentiable functions, such as neural networks with learnable parameters optimized through backpropagation. A common choice for f is a weighted average as in Eq. 2.1. The update function g, on the other hand, can be chosen as a simple neural network such as in Eq. 2.2.

$$m_v^{(l)} = \sum_{u \in \mathcal{N}(v)} \frac{1}{d_v} (W^{(l)} h_u^{(l-1)} + b^{(l)}) \tag{2.1}$$

$$h_v^{(l)} = \sigma \left(Q^{(l)} h_v^{(l-1)} + p^{(l)} + m_v^{(l)} \right) \tag{2.2}$$

where $W^{(l)}$, $b^{(l)}$, $Q^{(l)}$, and $p^{(l)}$ are trainable parameters of layer l and σ is a non-linearity.

Graph Convolutional Networks

Building on the principles of DMP, Graph Convolutional Networks (GCNs) refine the process by mimicking convolution operations used in traditional convolutional neural networks, but adapted for graph data [20, 21]. GCNs simplify the message aggregation mechanism, typically using a spectral or spatial approach to efficiently combine neighborhood features, as seen in Eq. 2.3.

$$H^{(l)} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{\frac{1}{2}} H^{(l-1)} W^{(l-1)} \right) \tag{2.3}$$

where $H^{(l)}$ is a matrix of stacked node embeddings h at layer l,
$\tilde{A} = A + I$ is the sum of the adjacency matrix and the identity matrix, and
$\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$ is the degree matrix of the graph.

Graph Attention Networks

To capitalize on the power of the self-attention mechanism, Graph Attention Networks (GATs) were introduced [22, 23]. Their architecture does not rely on spectral approaches such as GCNs but is purely attention-based. The attention mechanism allows the model to weigh the importance of nodes differently, thus achieving better performance than GNNs and GCNs [24]. In its most general form, the GAT layer would attend to all the nodes in the graph to update the embedding of a given node, thus abandoning all structural information. This is, however, not beneficial as the structure of a graph and the relationships between nodes can hold a great amount of information. As a result, the GAT layer uses masked attention, with the corresponding attention scores $\alpha_{u,v}$, to only attend to neighboring nodes, as seen in Eq. 2.4, to update the embeddings h_v.

$$h_v^{(l)} = \sigma \left(\sum_{u \in \mathcal{N}(v)} \alpha_{u,v} W h_u^{(l-1)} \right) \tag{2.4}$$

where W is a parameterized weight matrix matrix.

The attention scores $\alpha_{u,v}$ between nodes u and v can be computed according to Eq. 2.5. Visually, this calculation is displayed in Fig. 2.2.

$$\alpha_{u,v} = \frac{\exp\left(\text{LeakyReLU}\left(a^T \left[W h_v \parallel W h_u\right]\right)\right)}{\sum_{k \in \mathcal{N}_v} \exp\left(\text{LeakyReLU}\left(a^T \left[W h_v \parallel W h_k\right]\right)\right)} \tag{2.5}$$

where a is a weight vector of a single layer Multilayer Perceptron (MLP) and
$\parallel$ is the concatenation operation.

To minimize noisy updates and thus stabilize the learning process, the GAT layer gets further refined by utilizing multi-head attention [22]. Multi-head attention uses K independent weight matrices to compute K embeddings for each node. These representations are then concatenated or averaged to get a more stable embedding for each node, as calculated in Eq. 2.6 and visualized in Fig. 2.2.

$$
\begin{aligned}
h_v^{(l)} &= \|_{k=1}^{K} \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k} W^{k} h_u^{(l-1)}\right) \\
h_v^{(l)} &= \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k} W^{k} h_u^{(l-1)}\right)
\end{aligned}
\tag{2.6}
$$

One additional beneficial property of the attention mechanism, as presented, is that it is highly parallelizable. This makes the calculation of the attention weights between node-neighbor pairs highly efficient.

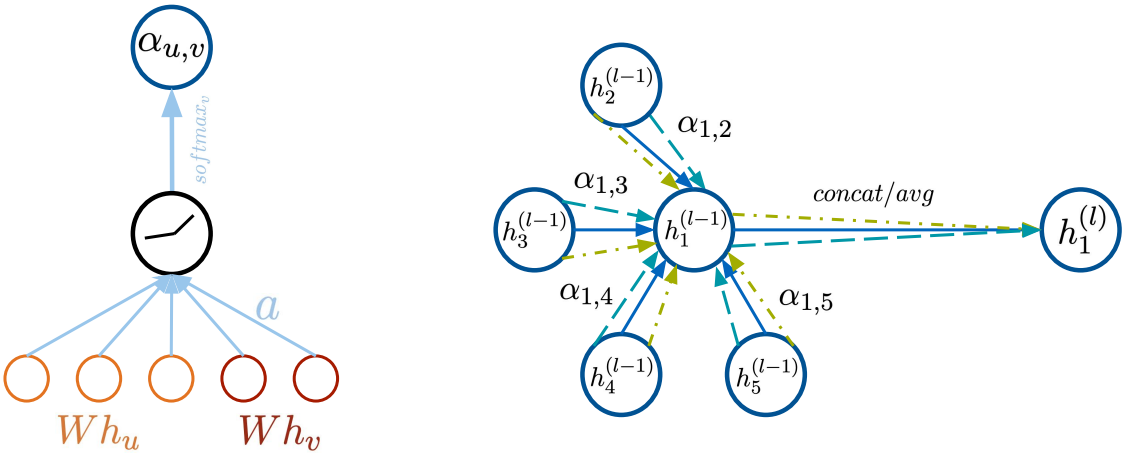


Figure 2.2 Left: The attention mechanism from Eq. 2.5 parameterized by the weight vector a and using a LeakyReLU activation. Right: The multi-head attention from Eq. 2.6 with $K = 3$ heads.

2.2 Trajectory and Prediction

Motivation

Recent advances in autonomous driving and robotics have elevated trajectory prediction to a critical research domain, essential for developing efficient, fast, and safe navigation strategies [1, 2]. The increasing prevalence of autonomous vehicles and automated systems underscores the importance of accurately modeling human-robot interactions [25]. Additionally, steering autonomous vehicles and other traffic participants through the dynamic conditions of urban streets efficiently and sustainably is crucial for reducing both travel times and greenhouse gas emissions [26, 27, 28].

Problem Description

Trajectory prediction can be applied in various settings, each addressing distinct challenges:

- Given a historical movement pattern, the task is to predict the future positions of an entity after a specified time or traveled distance.
- Given a starting point and a destination, the objective consists of predicting a trajectory that optimally connects them, considering varying criteria such as travel time, distance, or congestion avoidance.

Models for trajectory prediction can be developed for and applied in the continuous Euclidean space to predict coordinates or on discrete graphs to predict the next nodes, for example. We will next explore the distinct challenges and different methodologies applied in each setting.

Trajectory Prediction in Euclidean Space

In Euclidean space, trajectory prediction usually consists of predicting continuous 2D coordinates. There exist different ML approaches to this problem, depending on the data modality and environment setting at hand. For instance, Long Short-Term Memory (LSTM) models get used in combination with time series data from vehicle sensors to predict the future positions of vehicles [29]. Additionally, there also exist probabilistic methods, such as Gaussian Mixture Models, that rely on GPS data [30]. Other approaches, however, mostly rely on video recordings of moving people or vehicles over a short period. Hence, common ML architectures for handling images, such as Convolutional Neural Networks (CNNs) or denoising diffusion models, have been widely utilized [3, 4, 31].

Trajectory Prediction on Graphs

On graphs, the task of trajectory prediction boils down to a classification problem predicting the next edge or the next node in an already existing sub-trajectory. Unlike the continuous Euclidean space, the decision space here is confined to a discrete set of nodes or edges. Additionally, most current AR approaches limit the decision space further to only include neighbors of the current sub-trajectory. This might suggest a simpler problem, but it can also lead to significant challenges. At first glance, this seems to be an easier problem to solve, thanks to the discretized decision space. However, small prediction errors on graphs can result in completely invalid and illogical trajectories. For example, incorrectly predicting a sequence of nodes could lead to a cyclical path, which would be highly unusual and impractical for a vehicle or a person who typically move from a start to an endpoint.
There currently exists a large variety of approaches, including probabilistic ones, such as Markov chains. More complex ML approaches have also been explored, including Recurrent Neural Networks (RNNs), GNNs, or algorithms relying on simplicial complexes [5, 6, 32]. We will present these methods and their respective contributions and limitations in further detail in Subsec. 3.1.2.

2.3 Denoising Diffusion Models

Generative models have recently become powerful tools for producing diverse samples with high fidelity across different data modalities, including images and videos, text, audio, or chemical molecules [33, 34, 35, 36, 37, 38]. Traditional generative architectures include Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), AR models, and Normalizing Flows. Each of these architectures has its strengths and limitations. For instance, GANs are known for producing high-quality samples but often suffer from mode collapse [39]. In addition to mode collapse, GANs often fail to capture the full diversity of the data distribution, which complicates the training process. AR models can model complex distributions but are computationally intensive and accumulate errors during sampling due to their sequential nature [40].
Denoising diffusion models are a family of latent variable generative models that have demonstrated remarkable performance in generating high-quality samples in all of the data domains mentioned above [7, 41, 42, 43, 44]. Moreover, diffusion models offer several advantages over other generative methods. They achieve state-of-the-art results in image generation, producing samples with high authenticity and diversity [8]. They are less prone to mode collapse compared to GANs, effectively capturing all modes of the data distribution. Finally, thanks to the noising process during training, no adversarial training is necessary for diffusion models.
The idea behind diffusion models is rooted in non-equilibrium thermodynamics, where Markov chains are used to incrementally transition one distribution into another [45, 46]. The task of learning in the setting of diffusion models consists of learning the transitions of these Markov chains as opposed to learning a

distribution over the whole dataset, as is the case in classical ML.
In the following sub-chapters, we will explore the motivation behind the theoretical foundation and the different components of denoising diffusion models.

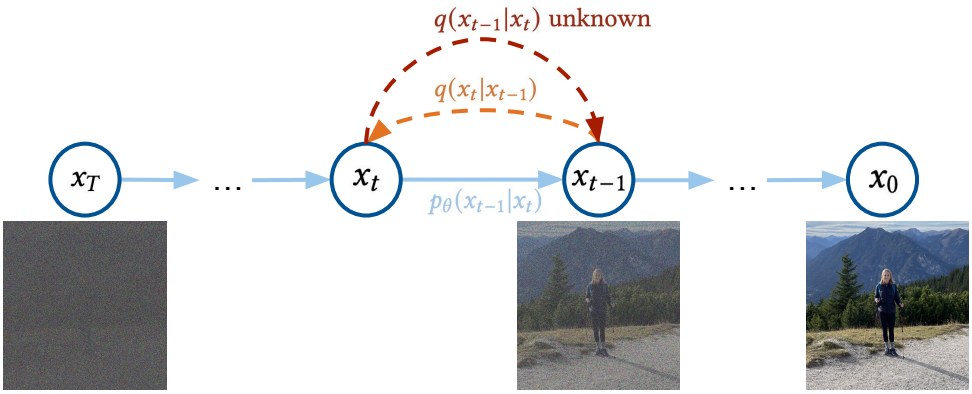


Figure 2.3 Visualization of the forward and reverse processes of a diffusion model

2.3.1 Forward Diffusion Process

Denoising diffusion models consist of two distinct processes: the forward or noising process q and the reverse or denoising process p_θ as depicted in Fig. 2.3. The forward process is a Markov chain defined to incrementally add noise to a datapoint $x_0 \sim q$. In traditional continuous diffusion models, this simply involves adding Gaussian noise to the data, where the variance, denoted by $\beta_t \in (0, 1)$, follows a predefined schedule. As this process progresses, the original data x_0 gradually loses its distinctive features. With a sufficiently large number of diffusion steps T, the data x_0 is ultimately transformed into an isotropic Gaussian distribution x_T. We can thus represent the Markovian noising process q as follows:

$$q(x_1, \ldots, x_T) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{2.7}$$

$$q(x_t \mid x_{t-1}) := \mathcal{N}(x_{t-1}; \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t \boldsymbol{I}) \tag{2.8}$$

Eq. 2.8, in combination with the reparameterization trick, allows us to formulate a noisy latent x_t at any arbitrary timestep t directly conditioned on x_0. With $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$:

$$\begin{aligned} x_t &= \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_{t-1} \\ &= \sqrt{\alpha_t}\, (\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \epsilon_{t-2}) + \sqrt{1-\alpha_t}\, \epsilon_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \sqrt{\alpha_t - \alpha_t \alpha_{t-1}}\, \epsilon_{t-2} + \sqrt{1-\alpha_t}\, \epsilon_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\, \bar{\epsilon}_{t-2} \\ &= \ldots \\ &= \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \end{aligned} \tag{2.9}$$

with $\epsilon, \epsilon_{t-1}, \epsilon_{t-2}, \cdots \sim \mathcal{N}(0, 1)$ and $\bar{\epsilon}_{t-2} = \epsilon_{t-1} + \epsilon_{t-2}$ the sum of two Gaussians.

We can now express the marginal:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0,\, (1-\bar{\alpha}_t)\, \boldsymbol{I}) \tag{2.10}$$

Noise Schedule

The variance β_t of the noise added during the forward process is restricted to a range between 0 and 1. How one chooses the size of β_t depends on the data modality at hand. However, as the sample becomes noisier with increasing diffusion timestep t, we can typically afford larger increments in variance. This allows for a reduction in the total number of diffusion steps T while still retaining the isotropic nature of x_T, thereby decreasing both the training and sampling time required. To achieve this increasing variance magnitude as t increases, there exist two commonly used schedules:

- Linear schedule [7]
- Cosine schedule [47]

As the name suggests, when using the **linear** schedule, the variance β_t of the added noise at step t increases linearly from a starting value β_0 to a maximum of β_T.
The **cosine** schedule on the other hand is expressed in terms of $\bar{\alpha}_t$ as seen in Eq. 2.11.

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{\pi}{2} \cdot \frac{\frac{t}{T} + s}{1 + s}\right)^2 \tag{2.11}$$

A small offset s is used to prevent the noise level at $t = 0$ from being too small, as this has been shown to hinder the learning process [47]. Now we can calculate the variance at step t as $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$.
Visualizing $\bar{\alpha}_t$ helps to understand the key differences between both noise schedules. When looking at Fig. 2.4, we can observe that in the middle part of the diffusion process, $\bar{\alpha}_t$ generated by the cosine schedule has a linear drop-off. In comparison, $\bar{\alpha}_t$ produced by the linear schedule have a much steeper drop-off in the middle section and reach their minimum much faster. This translates to adding more noise in a faster way which can be detrimental to learning and the generation of samples by the diffusion model.

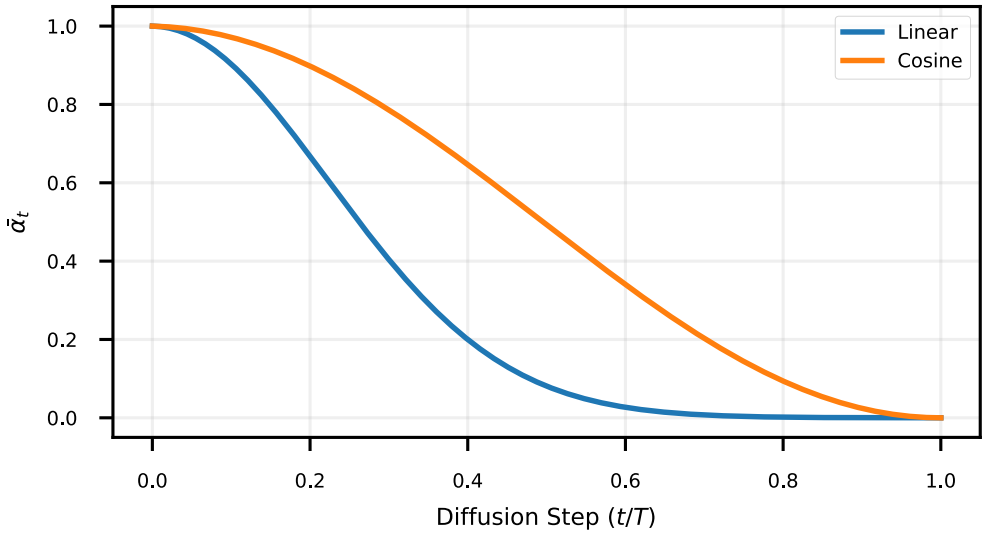


Figure 2.4 Comparison of the evolution of $\bar{\alpha}_t$ during the diffusion process between a linear and a cosine schedule

2.3.2 Reverse Process and Training

If we knew $q(x_{t-1} \mid x_t)$, we could sample some white noise x_T, run the reverse process, and thus generate a sample from $q(x_0)$. The reverse distribution $q(x_{t-1} \mid x_t)$ is however unknown and depends on the whole data distribution which is why we want to train a neural network p_θ as defined in Eq. 2.12 to approximate it.

$$p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2.12}$$

This approach aligns with the VAE framework and leads the loss function we aim to optimize in traditional continuous diffusion models. Specifically, if we consider $x_1, \ldots, x_T$ as different latent space representations of the model, then the combination of p_θ and q corresponds to a VAE. Consequently, during training, we focus on optimizing the Evidence Lower Bound (ELBO) in Eq. 2.13 as a surrogate for minimizing the negative log-likelihood (NLL).

$$\begin{aligned}\mathbb{E}[-\log(p_\theta(x_0))] &\leq \mathbb{E}_q\left[-\log\left(\frac{p_\theta(x_0,\ldots,x_T)}{q(x_1,\ldots,x_T \mid x_0)}\right)\right] \\ &= \mathbb{E}_q\left[-\log\left(p(x_T)\right) - \sum_{t\geq 1}\log\left(\frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})}\right)\right] \\ &=: \mathcal{L}\end{aligned} \tag{2.13}$$

Using the Kullback-Leibler divergence ($\mathbb{KL}$), we can express $\mathcal{L}$ as a sum of analytically computable terms $\mathcal{L}_0$, $\mathcal{L}_{t-1}$, and $\mathcal{L}_T$ as follows:

$$\mathcal{L} := \mathbb{E}_q\left[\mathcal{L}_0 + \sum_{t>1}\mathcal{L}_{t-1} + \mathcal{L}_T\right] \tag{2.14}$$

$$\mathcal{L}_0 := -\log(p_\theta(x_0 \mid x_1)) \tag{2.15}$$

$$\mathcal{L}_{t-1} := \mathbb{KL}(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)) \tag{2.16}$$

$$\mathcal{L}_T := \mathbb{KL}(q(x_T \mid x_0) \parallel p(x_T)) \tag{2.17}$$

We use the fact that the $\mathbb{KL}$ divergence between two normal distributions can be calculated in closed form as seen in Theorem 2.3.1. Additionally, the forward process posterior $q(x_{t-1} \mid x_t, x_0)$ is tractable when conditioned on the data x_0. In fact, Eq. 2.10, in combination with Bayes' theorem, allows us to calculate a closed-form expression for the posterior [48]:

$$\begin{aligned}q(x_{t-1} \mid x_t,\, x_0) &= \frac{q(x_t \mid x_{t-1},\, x_0)\, q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)} \\ &\propto \exp\left(-\frac{1}{2}\left(\frac{(x_t - \sqrt{\alpha_t}\, x_{t-1})^2}{\beta_t} + \frac{(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\, x_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(x_t - \sqrt{\bar{\alpha}_t}\, x_0)^2}{1-\bar{\alpha}_t}\right)\right) \\ &= \ldots \\ &= \mathcal{N}(x_{t-1};\, \tilde{\mu}_t(x_0, x_t),\, \tilde{\beta}_t\, \boldsymbol{I}),\end{aligned} \tag{2.18}$$

$$\text{with} \quad \tilde{\mu}_t(x_0, x_t) = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right) \quad \text{and} \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{2.19}$$

During training we want to train the neural network from Eq. 2.12 to predict $\mu_\theta(x_t, t)$ close to $\tilde{\mu}_t(x_0, x_t)$. Alternatively, given that x_t is available during training and using the reparameterization trick, the neural network could learn to predict the noise ϵ_t at timestep t. So by setting $\Sigma_\theta(x_t, t) = \sigma_t^2\, \boldsymbol{I}$ as in [7], using Eq. 2.10 to express x_t, and Eq. 2.19 to get the predicted forward process posterior mean, we can parameterize $\mathcal{L}_{t-1}$ as follows:

$$\begin{aligned}\mathcal{L}_{t-1} &= \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\right\|^2\right] \\ &= \ldots \\ &= \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\,\alpha_t(1-\bar{\alpha}_t)}\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon,\, t)\right\|^2\right]\end{aligned} \tag{2.20}$$

The expression found for $\mathcal{L}_{t-1}$ in Eq. 2.20 is differentiable with respect to θ and optimized during training of the diffusion model. $\mathcal{L}_T$, on the other hand, does not depend on θ and is a constant close to 0, because $q(x_T \mid x_0) \approx \mathcal{N}(0, 1)$. $\mathcal{L}_0$ can be modeled with a separate discrete decoder, depending on the data modality, derived from $\mathcal{N}(x_0;\, \mu_\theta(x_1, 1),\, \Sigma_\theta(x_1, 1))$ [7].

Theorem 2.3.1 *Let p and q be two multivariate normal distributions of dimension k with means $\boldsymbol{\mu}_p$ and $\boldsymbol{\mu}_q$ and (non-singular) covariance matrices Σ_p and Σ_q. Then we have that*

$$\mathbb{KL}(p \parallel q) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_q^{-1}\Sigma_p\right) - k + (\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^T \Sigma_q^{-1} (\boldsymbol{\mu}_q - \boldsymbol{\mu}_p) + \log\left(\frac{\det \Sigma_q}{\det \Sigma_p}\right)\right) \tag{2.21}$$

2.3.3 Sampling

Sampling from a trained denoising diffusion model is straightforward. First a noised latent $x_T \sim \mathcal{N}(0, \boldsymbol{I})$ is sampled. For every timestep $t \in [T, ..., 1)$, x_T is then incrementally denoised by sending it through the neural network p_θ, as seen in Eq. 2.22.

$$x_{t-1} = \frac{1}{\alpha_t}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z \quad \text{,with} \quad z \sim \mathcal{N}(0,\, \boldsymbol{I}) \tag{2.22}$$

At the last step, $t = 1$, $z = 0$, and the final sample x_0 is returned.

2.3.4 Discrete Denoising Diffusion Models

Initial discrete diffusion models, emerging as extensions of continuous denoising diffusion techniques, focused on binary variables [46]. The scope of these models has expanded to encompass categorical random variables to enhance capabilities in tasks such as text and image segmentation [49].
The first diffusion model in discrete state spaces able to outperform non-AR baselines for text generation and to generate image samples of high quality is D3PM or Discrete Denoising Diffusion Probabilistic Model [50]. We will now introduce the concept and the theoretical background of discrete diffusion based on D3PM.
In continuous diffusion, the forward and (learned) reverse processes are modeled to be continuous distributions $q(x_t \mid x_{t-1} = \mathcal{N}(x_t;\, \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t \boldsymbol{I})$ and $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1};\, \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t))$. In discrete diffusion, on the other hand, both processes now follow categorical distributions. In fact, if we consider discrete random variables $x_t, x_{t-1} \in [1, \ldots, K]$, where K is the number of classes, we can represent the forward process q as follows:

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) = \mathrm{Cat}(\boldsymbol{x}_t;\, p = \boldsymbol{x}_{t-1}\boldsymbol{Q}_t) \tag{2.23}$$

with $\boldsymbol{x}$ is the one-hot encoded row vector of x and
$[\boldsymbol{Q}_t]_{i,j} = q(x_t = j \mid x_{t-1} = i)$ the transition probability from class i to class j

$\mathrm{Cat}(\boldsymbol{x}; p)$ refers to a categorical distribution over $\boldsymbol{x}$ with a probability vector p and consequently $\boldsymbol{Q}_t \in \mathbb{R}^{K \times K}$. Similar to Eq. 2.10 we can express the marginal for an arbitrary diffusion timestep t with respect to the data $\boldsymbol{x}_0$ as follows:

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathrm{Cat}(\boldsymbol{x}_t;\, p = \boldsymbol{x}_0\bar{\boldsymbol{Q}}_t) \quad \text{with} \quad \bar{\boldsymbol{Q}}_t = \boldsymbol{Q}_0\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_t \tag{2.24}$$

When we condition on the data $\boldsymbol{x}_0$, we can use Bayes' theorem and the Markovian property of our forward process to formulate the forward process posterior $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t,\, \boldsymbol{x}_0)$ in a tractable way:

$$\begin{aligned} q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t,\, \boldsymbol{x}_0) &= \frac{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1},\, \boldsymbol{x}_0)\, q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)} \\ &= \frac{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})\, q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)} \\ &= \mathrm{Cat}\left(\boldsymbol{x}_{t-1};\, p = \frac{\boldsymbol{x}_t\boldsymbol{Q}_t^T \odot \boldsymbol{x}_0\bar{\boldsymbol{Q}}_{t-1}}{\boldsymbol{x}_0\bar{\boldsymbol{Q}}_t\boldsymbol{x}_t^T}\right) \end{aligned} \tag{2.25}$$

The transition matrices $\boldsymbol{Q}_t$ must be designed to fulfill certain conditions for the diffusion process to work as expected.

- $\boldsymbol{Q}_t$ must be row-stochastic (rows sum to 1), to conserve the probability mass.
- The cumulative product $\bar{\boldsymbol{Q}}_t$ must converge to a known stationary distribution as $t \to T$. The stationary distribution is used for the initialization of $\boldsymbol{x}_T$ during sampling.

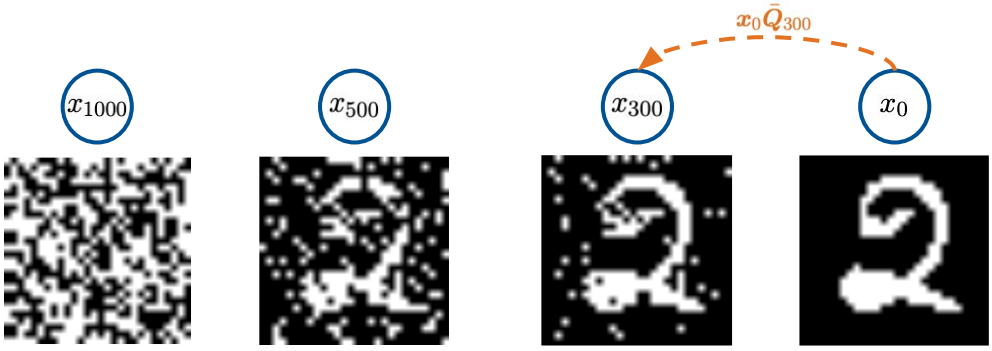


Figure 2.5 Discrete diffusion process applied on a binary image from the MNIST dataset with $T = 1000$ diffusion timesteps [51].

One simple way to ensure the second condition is to make $\boldsymbol{Q}_t$ doubly stochastic (rows and columns sum to 1) with strictly positive entries. This renders $\boldsymbol{Q}_t$ irreducible and aperiodic and lets $\bar{\boldsymbol{Q}}_t$ converge to a uniform distribution over all classes K. There exist different options for designing $\boldsymbol{Q}_t$ and for how it evolves as t increases, which we will examine in closer detail in Sec. 3.2. A demonstration of the discrete diffusion process in action is provided in Fig. 2.5.
We can now express a parameterized version of the learned reverse process $p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$. In fact, we use a neural network to predict $\tilde{\boldsymbol{x}}_0$ given the noised sample $\boldsymbol{x}_t$ and combine it with Eq. 2.25 to get Eq. 2.26.

$$p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) \propto \sum_{\tilde{\boldsymbol{x}}_0} q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t \mid \tilde{\boldsymbol{x}}_0)\, p_\theta(\tilde{\boldsymbol{x}}_0 \mid \boldsymbol{x}_t) \tag{2.26}$$

During training, we again want to optimize the ELBO loss, although different loss functions have also been proposed.

3 Related Work

3.1 Trajectory Prediction

Trajectory prediction, as mentioned in Sec. 2.2, is a complex task with different promising approaches. In the following two subsections, we will explore the different methods that are related to this work in closer detail and compare respective advantages, disadvantages, and potential shortcomings. Here, we distinguish two modalities in which to perform trajectory prediction:

- Euclidean space
- Graphs

3.1.1 Trajectory Prediction in the Euclidean Space

In this section, we will focus on two approaches that use diffusion models to predict a future trajectory given a past trajectory.
A first example is the **Motion Indeterminacy Model (MID)**, which aims to model and predict pedestrian trajectories using video data [3]. The authors argue that human behavior is indeterminate and multi-modal (i.e., it depends on the behavior of other humans or objects in the immediate surroundings). Hence, why they devised a diffusion-based stochastic trajectory prediction framework to predict future trajectories of humans.
To achieve this, video data is used to simultaneously track multiple people in a specific place over a certain period of time. Using the timestamps of the respective observations, so-called scenes are created which contain the trajectories of humans that are present in a video frame at the same time. To capture the interactions between different people in that scene, MID uses the Trajectron++ social encoder [52]. In combination with the past trajectory, this social embedding forms an encoding f. MID is composed of this temporal-social encoder and a transformer-based decoder model to capture temporal dependencies. A standard continuous diffusion process, as described in Subsec. 2.3.1 and Subsec. 2.3.2, is applied to noise and denoise the trajectories. This leads to Eq. 2.22 for sampling trajectories with the slight modification that the neural network $\epsilon_\theta(x_t, t, f)$ now also takes the encoding f as a condition. MID achieves state-of-the-art performance on two widely used benchmark datasets.

Another diffusion-based approach to trajectory prediction in Euclidean space is presented in **Diffusion-Based Environment-Aware Trajectory Prediction** [4]. While the MID model focuses on predicting pedestrian trajectories, this work addresses the complex task of forecasting vehicle trajectories in different traffic environments, which introduces its own unique challenges due to vehicle dynamics and interactions. Similarly to MID, this approach focuses on modeling interactions between vehicles occupying the same scene. They achieve this by training GNNs on sequences of graphs $G_1, G_2, \ldots, G_n$, centered around a target vehicle v_0. These graphs represent inter-agent relationships over n time steps and capture their spatiotemporal dependencies. Again, an encoding (or condition) is calculated by combining the past trajectory, inter-vehicular interactions, and map-based information. Using this encoding, a conditional diffusion model is then trained to predict the future trajectories of vehicles in a scene. One of the main contributions of this work is the generation of 'physically feasible' trajectories by incorporating differential motion constraints. Thus, the predicted trajectories must adhere to certain physical laws, such as a prescribed maximum tire grip.
Their approach shows efficacy for trajectories on highways and trajectories on and around roundabouts.

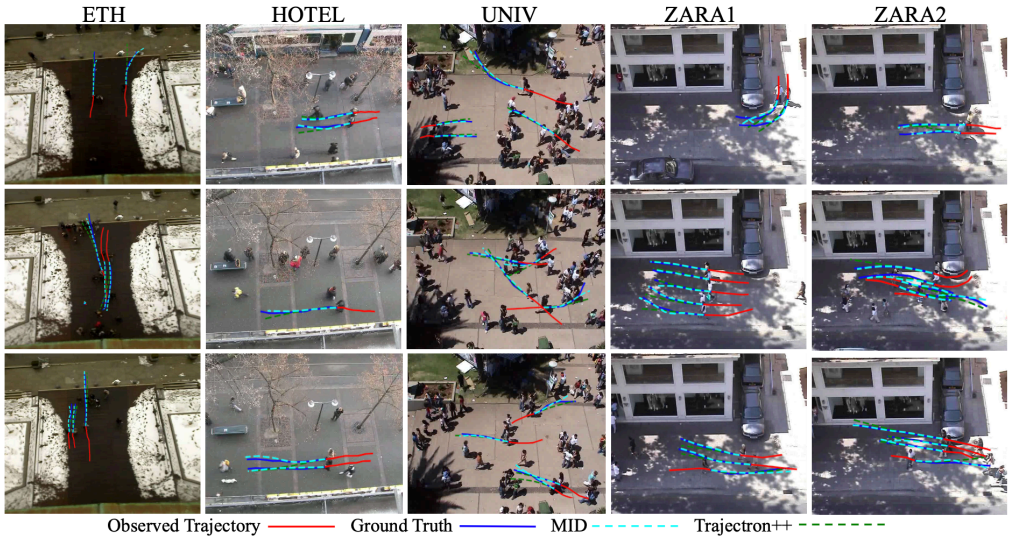


Figure 3.1 Visualized predicted and ground truth trajectories of MID. (Figure taken from [3])

Both of these approaches share two main shortcomings. For one, in real road networks, not all of the space around a vehicle or a person is available. Vehicles can only travel along roads, and pedestrians cannot surmount obstacles like buildings in their surroundings. While this may not pose a significant issue when predicting a short future horizon, it becomes relevant when forecasting long trajectories. This leads to the second limitation of the aforementioned models: their relatively short prediction horizon. MID and the environment-aware model only aim to predict the next 4.8 or 5 seconds, respectively. Therefore, they are not necessarily suited for forecasting long, realistic trajectories that span several roads, for example.

3.1.2 Trajectory Prediction on Graphs

The presented shortcomings are partially addressed by modeling trajectories on graphs, thus limiting the decision space to actual roads and allowing longer-range predictions. For trajectory prediction on graphs, we will present the leading AR methods that exist. Furthermore, we will showcase one non-AR approach that relies on simplicial complexes [53].
The most basic AR method for trajectory prediction is a simple **Markov chain**. For every possible prefix (sequence of node or edge indices), we count the number of times each possible next node or edge appears in the training data. As a result, at each node, we choose the successor node or edge, which is the most likely. The most obvious limitation of this is the fact that, for unseen prefixes, the Markov chain simply generates a random walk.
More complex methods based on RNNs also exist, such as the Constrained State Space Recurrent Neural Network (**CSSRNN**) [5]. This model allows for the inclusion of more features, thus enhancing its expressive power over a simple Markov chain. The authors use a standard Recurrent Neural Network approach with the modification of constraining the decision space to neighboring nodes or edges of the current node or edge. CSSRNN comfortably outperforms the Markov chain approach on road network datasets.
Gretel is a GCN based approach that adds one more constraint over CSSRNN: a trajectory should be "non-backtracking" [6]. This means that once an edge is traversed, the model cannot predict traversing the same edge in the reverse direction immediately afterward. While this is a useful property, it will not prevent

longer cycles from being sampled. The Gretel model encodes the directionality of past trajectories to then predict the next edges of a trajectory in an AR fashion. Thanks to the graph-based model architecture and the non-backtracking condition, Gretel outperforms CSSRNN.
Lastly, we also want to present a non-AR approach to trajectory prediction. **SCoNe**, or Simplicial Complex Net, is a GCN-based model that uses simplicial complexes as its backbone to represent data flows on graphs and subgraphs. Simplicial complexes enable message passing along higher-dimensional cells (simplices), allowing the model to capture higher-order relationships among groups of nodes beyond just pairwise interactions between nodes. By including these higher-order interactions, ScoNe is able to outperform both a Markov chain and CSSRNN.
All of the approaches described above show some weaknesses or limitations when it comes to trajectory prediction. All AR methods (Markov chain, CSSRNN, and Gretel) suffer from error accumulation over long prediction horizons [40]. Moreover, due to their sequential nature, AR models often struggle to effectively capture long-term dependencies during training, making it challenging to model extended temporal relationships in trajectories. Additionally, all of the presented approaches lead to low diversity in samples. Our diffusion-based approach presented in Chapter 4, aims to address these issues.

3.2 Discrete Diffusion

Discrete diffusion models, introduced in Subsec. 2.3.4, extend diffusion models from continuous to discrete state spaces. They now play an important role in generative AI for discrete data, enabling the modeling of textual, categorical, and graph-structured information that continuous models struggle to represent effectively. The following sections explore the advantages that **D3PM** introduces over previous discrete diffusion methods and continuous diffusion models [50]. Furthermore, we focus on the central role that discrete diffusion has played in graph generation, exemplified by the **DiGress** model, which leverages discrete diffusion processes to generate complex graph structures [44].

3.2.1 Discrete Denoising Diffusion Models (D3PM)

D3PM introduces several contributions that improve its performance:

- Auxiliary loss function
- Domain dependent transition matrices $\boldsymbol{Q}_t$

Inspired by a hybrid loss function which was previously introduced for continuous diffusion and showed promising results, the authors of D3PM combine the ELBO loss of Eq. 2.13 with an auxiliary objective for the $\boldsymbol{x}_0$ parameterized reverse process [47]. This leads to an alternative loss function $\mathcal{L}_\lambda$ as described in Eq. 3.1.

$$\mathcal{L}_\lambda = \mathcal{L} + \lambda \, \mathbb{E}_{q(\boldsymbol{x}_0)} \left[\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[-\log \left(p_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t) \right) \right] \right] \tag{3.1}$$

In addition to this improvement, D3PM introduces transition matrices $\boldsymbol{Q}_t$ which can incorporate structural information of the data at hand. For training on image data the authors devised and experimented with three different transition matrices:

- **Uniform**: $\boldsymbol{Q}_t = \alpha_t \boldsymbol{I} + \frac{(1-\alpha_t)}{K} \mathbb{1}\mathbb{1}^T$. Since $\boldsymbol{Q}_t$ is doubly stochastic and only has positive entries, the rows of $\bar{\boldsymbol{Q}}_t$ will converge to a uniform distribution as t approaches T.
- **Absorbing State**: Inspired by BERT and other Conditional Masked Language Models, this transition matrix contains an absorbing state called m [54]. $\boldsymbol{Q}_t = \alpha_t \boldsymbol{I} + (1-\alpha_t)\mathbb{1}e_m^T$ where e_m is a vector with a one at index m and zeros everywhere else. For RGB images, the absorbing state m is set to $K//2 = 128$.

- **Discretized Gaussian**: This transition matrix ensures that transitions between classes that are close together are more likely. It leverages the structure of images where neighboring pixel values are usually similar. By making transitions between similar classes more probable, the matrix captures the natural smoothness and local continuity present in images, where pixels close in value tend to be spatially adjacent.

$$[\boldsymbol{Q}_t]_{i,j} = \begin{cases} \dfrac{\exp\left(-\dfrac{4|i-j|^2}{(K-1)^2\beta_t}\right)}{\sum_{n=-(K-1)}^{K-1} \exp\left(-\dfrac{4n^2}{(K-1)^2\beta_t}\right)} & \text{if } i \neq j, \\ 1 - \sum_{\substack{l=0 \\ l\neq i}}^{K-1} [\boldsymbol{Q}_t]_{i,l} & \text{if } i = j. \end{cases}$$

With these advancements, D3PM has been evaluated on text and image data. On the CIFAR-10 dataset, D3PM manages to achieve competitive performance as continuous diffusion models [55]. Notably, the discretized Gaussian transition matrix far outperforms the uniform and absorbing transition matrices, motivating a careful choice of $\boldsymbol{Q}_t$ depending on the data and task at hand. Additionally, optimizing the alternative loss function of Eq. 3.1 improves the performance of D3PM drastically.

3.2.2 Graph Diffusion Models

Beyond image or text generation, discrete diffusion is especially well suited for the task of graph generation, which itself is crucial, for example, in drug discovery. Initial applications of diffusion models for graph generation relied on embedding the graphs in a continuous space [56, 57]. These methods then added Gaussian noise to node features and the adjacency matrix, which destroyed the graph's sparsity and made it difficult to properly represent its structural information. Hence, the choice to apply discrete diffusion to graph generation.
Seminal research in this area introduced **DiGress**, a discrete diffusion model for generating graphs [44]. We will now explore this model in closer detail and present its main contributions. The diffusion process in DiGress works on categorical edges and nodes and makes successive graph edits such as edge additions or deletions and node or edge category changes. While DiGress, like other generative models for graph data, struggles to detect substructures such as cycles in a graph due to the message passing algorithm, the discrete nature of the noise model in DiGress allows the addition of spectral and structural features for training and sampling, which improves its performance significantly. Another major contribution consists in the design of a novel transition matrix that converges to the marginal prior categorical distribution of edges and nodes. This change makes the training significantly easier, as noisy graphs are closer to realistic graphs when compared to noisy graphs obtained by a uniform transition matrix, for example (see Fig. 3.2). This further underlines the importance of choosing a well-suited transition matrix for the data at hand. DiGress achieves state-of-the-art performance in graph generation on common molecular benchmark datasets.

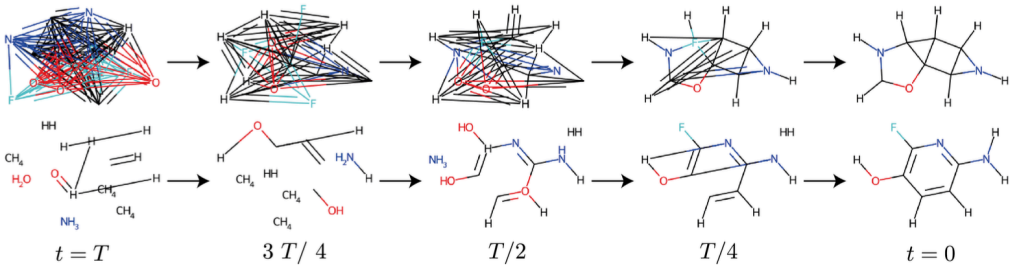


Figure 3.2 Top: Reverse process from a DiGress model trained with uniform transition matrices. Bottom: Reverse process from a model trained with marginal prior matrices. (Figure taken from [44])

4 Method

In this chapter, we introduce a novel approach for predicting realistic trajectories on graphs of road networks by applying a discrete diffusion model. Our goal is to generate sequences of edges that accurately reflect plausible paths within a given graph structure. By leveraging discrete diffusion processes, we aim to model the stochastic nature of trajectory generation while capturing long-term dependencies and complex patterns inherent in graph-based data.
The methods presented address the limitations of traditional AR models, such as error accumulation and difficulty in modeling long-term dependencies.

4.1 Data Representation

4.1.1 Trajectories

Our approach focuses on edges rather than vertices because edges represent the connections or movements between nodes, directly modeling the transitions that constitute a trajectory. While representing trajectories as sequences of vertices is common, using edges captures the same path information by emphasizing the actual traversed links in the graph. This edge-based representation is largely equivalent to a vertex-based one in terms of reconstructing the trajectory. Nevertheless, it's important to note that trajectories consisting solely of a single node (with no outgoing edges) cannot be represented in an edge-based framework, which is a potential limitation of this approach.
The first step in processing these trajectories consists of splitting them into a history and a future part. So, given a trajectory with edge indices $[e_1, e_2, \ldots, e_l]$, we generate a history of length h and a future of length f: $[e_1, e_2, \ldots, e_h]$ and $[e_{h+1}, e_{h+2}, \ldots, e_{h+f}]$. If the trajectory is shorter than $h + f$, we pad the future part with -1. The splitting process is visualized in Fig. 4.1. This future part is what we want to teach our model to predict, given the history part.

As we want to perform discrete binary diffusion (see Sec. 4.3) on the future part of our trajectory, we binarize both the history and future part into vectors $\boldsymbol{h}_{\text{bin}}$ and $\boldsymbol{f}_{\text{bin}}$ of length $|\mathcal{E}|$, where $\mathcal{E}$ corresponds to the set of edges in the underlying graph G, as shown in Eq. 4.1.

$$
\begin{aligned}
[\boldsymbol{h}_{\text{bin}}]_i &= \begin{cases} 1 & \text{if } i \in [e_1, e_2, \ldots, e_h], \\ 0 & \text{else.} \end{cases} \\
[\boldsymbol{f}_{\text{bin}}]_i &= \begin{cases} 1 & \text{if } i \in [e_{h+1}, e_{h+2}, \ldots, e_{h+f}], \\ 0 & \text{else.} \end{cases}
\end{aligned}
\tag{4.1}
$$

4.1.2 Graph

Since our approach is centered around edges, we do not use the graph G in its standard form. Instead, we transform it into a linegraph as defined in Def. 4.1.1, which is better suited to this approach and reduces the number of calculations needed on G for modeling edge sequences.
To represent the linegraph's structure and the connectivity of the edges, we build an edge index $\boldsymbol{E}$ as defined in Def. 4.1.2. $\boldsymbol{E}$ is later used as an input to the GNN (see Sec. 4.2) to inform the neural network of the graph's structure.
Since we do not always have the necessary information to determine whether the road at hand is a one-way or two-way street, we use an undirected graph.

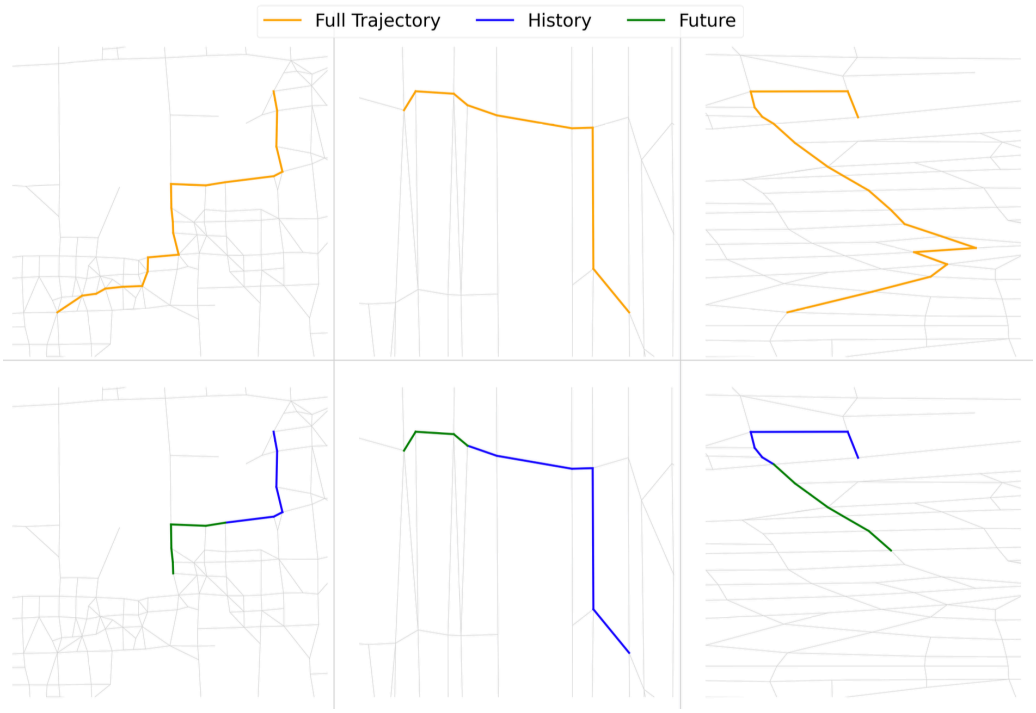


Figure 4.1 Example of trajectory splitting with a $h = f = 5$.

Definition 4.1.1 (Linegraph) *Given a graph $G = (\mathcal{V}, \mathcal{E})$, the linegraph $L(G)$ is defined as follows:*

- *Each vertex in $L(G)$ corresponds to an edge in G.*

  $\mathcal{V}_{L(G)} = \{e_i \mid e_i \in \mathcal{E}\}$

- *Two vertices e_i and e_j in $L(G)$ are adjacent if and only if their corresponding edges in G share a common vertex (are incident to the same vertex).*

  $\mathcal{E}_{L(G)} = \{(e_i, e_j) \mid e_i \text{ and } e_j \text{ share a common vertex in } G\}$

Definition 4.1.2 (Edge Index) *Given a linegraph $L(G)$, the edge index $\boldsymbol{E}$ is a matrix of shape $[2, |\mathcal{E}'|]$, where $|\mathcal{E}'|$ is the number of edges in $L(G)$. Each column represents an edge between two nodes in $L(G)$, indicating that their corresponding edges in G are adjacent.*

4.1.3 Feature Matrix

For each trajectory, we additionally compute feature matrix $\boldsymbol{F}$ of shape $[|\mathcal{E}|, n]$, where n corresponds to the number of features we calculate for each edge. We experiment with different combinations of features listed below:

- $\boldsymbol{h}_{\text{bin}}$, which serves as a condition to inform the model about the past trajectory.
- $\boldsymbol{f}_{\text{bin}_t}$, which is the noised future trajectory as obtained by the diffusion process explained in Sec. 4.3 and is always included.
- 2D coordinates of all the edges in G. The coordinates are normalized to a $[0, 1]$ range.
- Sinusoidal positional encoding of the edges in the history to inform the model about the direction of the trajectory.
- Length or norm of each edge.

- Distance from the last point in the history to the middle of all other edges in G.
- Cosine of the angle between the history vector (connecting the starting and the end point of the history) and all other edges in G.
- Total number of predicted edges in the future trajectory.

4.2 Model Architecture

We use a GAT-based model, including a residual connection, which is implemented in PyTorch [58]. An overview of the model architecture is provided in Fig. 4.2. To effectively leverage the structural information inherent in graphs and the trajectories on them, a specialized graph-based architecture is essential. We chose to build our model around graph attention because it has been shown to outperform or match other GNN methods in capturing complex relational patterns [23]. In addition, we opted to include residual connections at each GAT layer to get a richer and more global feature representation of edges beyond the immediate neighborhood of the current trajectory.
The diffusion timestep t is fed to the model after going through sinusoidal positional encoding. In fact, the resulting encoding $\tilde{\boldsymbol{t}}$ is concatenated with the feature matrix $\boldsymbol{F}$. This temporal information is necessary to inform the model where, in the diffusion process, the current input $\boldsymbol{f}_{\text{bin}t}$ is located.
The edge representation obtained by the GAT layers is finally fed into a MLP to get the binary class logits $\boldsymbol{f}^*$. The model is trained by minimizing the Binary Cross Entropy (BCE) loss between $\boldsymbol{f}_{\text{bin}}$ and $\boldsymbol{f}^*$ with the widely used Adam optimizer [59].

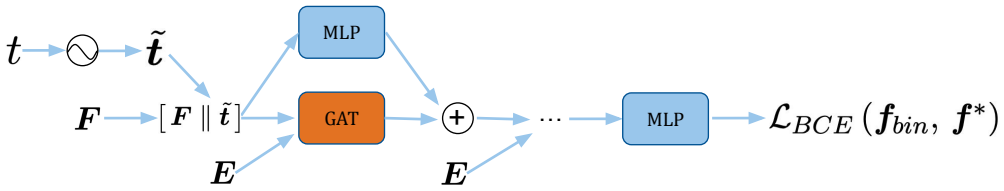


Figure 4.2 Graphical overview of our residual, GAT-based model architecture.

4.3 Discrete Diffusion Process

Given that we binarize our input data $\boldsymbol{f}_{\text{bin}}$, we devise a discrete diffusion process for binary variables inspired by the D3PM approach [50]. As a result, the transition matrices $\boldsymbol{Q}_t$ are 2x2 matrices.
Motivated by recent advancements for tuning the noise schedule and designing different transition matrices, we experiment with different choices for both [44, 47].
We train and evaluate our model with two different **noising schedules**, namely the linear and cosine schedules, as described in Subsec. 2.3.1.
Additionally, we adopted two different types of transition matrices:

- **Uniform**: $\boldsymbol{Q}_t = \begin{bmatrix} \alpha_t & 1-\alpha_t \\ 1-\alpha_t & \alpha_t \end{bmatrix}$. Since $\boldsymbol{Q}_t$ is doubly stochastic, the rows of $\bar{\boldsymbol{Q}}_t$ will converge to a stationary uniform distribution as $t \to T$.
- **Marginal Prior**: $\boldsymbol{Q}_t = \begin{bmatrix} \alpha_t + (1-\alpha_t)p_0 & (1-\alpha_t)p_1 \\ (1-\alpha_t)p_0 & \alpha_t + (1-\alpha_t)p_1 \end{bmatrix}$, where p_0 and p_1 are the respective prior class probabilities. Since we clip and split the trajectories using fixed history and future lengths h and f, we have that $p_1 = \frac{f}{|\mathcal{E}|}$ and $p_0 = 1 - p_1$. Thus $\bar{\boldsymbol{Q}}_t$ will converge to $\begin{bmatrix} p_0 & p_1 \\ p_0 & p_1 \end{bmatrix}$ as $t \to T$.

One of the primary advantages of diffusion models is their ability to perform non-AR sampling. This means that edges are not sampled sequentially but rather simultaneously during the denoising process. Depending on the transition matrix used, we initially sample noisy edges according to the probability of the rows of $\bar{Q}_T$ and use this as input for the denoising process, which gradually removes noise to generate realistic future trajectories.

5 Data

In this chapter, we will introduce the different datasets on which we evaluate our method. Additionally, we will present some statistical insights to convey their differences.
In total, we have four datasets at our disposal:

- **T-Drive** [60]: A collection of trajectories recorded by various taxi drivers in Beijing, China.
- **Geolife** [61]: A dataset of trajectories of commuters, pedestrians, and other vehicles in Beijing, China.
- **pNEUMA** [62, 63]: A dataset of trajectories of vehicles recorded by drones in Athens, Greece.
- **Mobilität.Leben (MoLe)**: A dataset containing trajectories of cyclists in the city of Munich, Germany. It was conceived in the context of the Mobilität.Leben project between TUM and Hochschule für Politik.

For T-Drive and Geolife, GPS loggers were used to collect the positions of the vehicles or pedestrians. These GPS traces were then mapped to a grid of roads, obtained from OpenStreetMap, by a hidden Markov map matching model [64]. The trajectories in pNEUMA, on the other hand, were recorded using camera-equipped drones. Thus, matching the trajectories to the road network was much simpler.
Fig. 5.1 shows an overview of the whole street network for each dataset (T-Drive and Geolife use the same graph, as they were both recorded in Beijing).

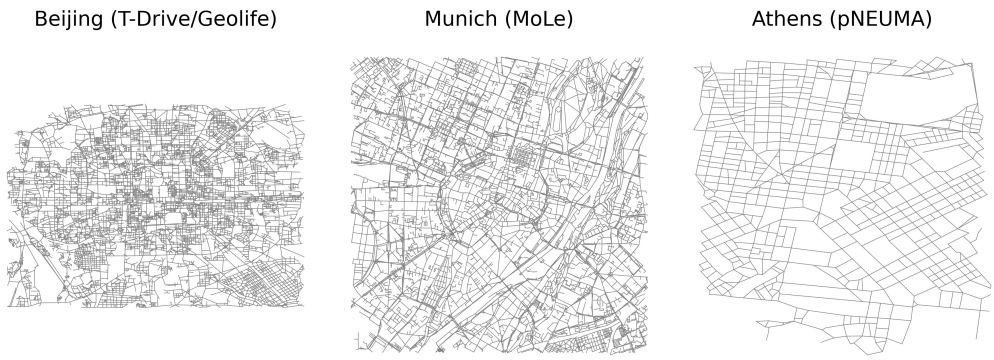


Figure 5.1 Street network graphs of the different datasets.

Table 5.1 Dataset Overview

Dataset	Nodes	Edges	Trajectories	Avg. Trajectory Length [Edges]	Used Edges [%]
T-Drive	9,812	16,784	6,996	23.78	61.76
Geolife	9,812	16,784	27,408	12.08	53.29
MoLe	8,742	12,806	2,603	23.65	38.13
pNEUMA	825	1,378	76,015	10.62	25.18

Basic information for the graphs and trajectories of the four datasets is listed in Tab. 5.1. These statistics reveal quite large differences between them. Whereas the graphs of the T-Drive, Geolife, and MoLe are of similar size in terms of number of edges and vertices, the graph for pNEUMA is an order of magnitude

smaller. Moreover, on pNEUMA, we have by far the most amount of available data, which is also more concentrated around a certain area of Athens, as can be seen by the fraction of used edges of the graph as well as in Fig. 5.3. In terms of trajectory length, T-Drive and MoLe contain, on average, quite long paths, whereas the Geolife pNEUMA datasets contain shorter paths. Fig. 5.2 underlines how different the path length distributions are between the respective datasets. Whereas the trajectory lengths in Geolife and pNEUMA are relatively concentrated, T-Drive and MoLe contain more diverse path lengths. These differences will play an important role in explaining some of the results in Chapter 6.

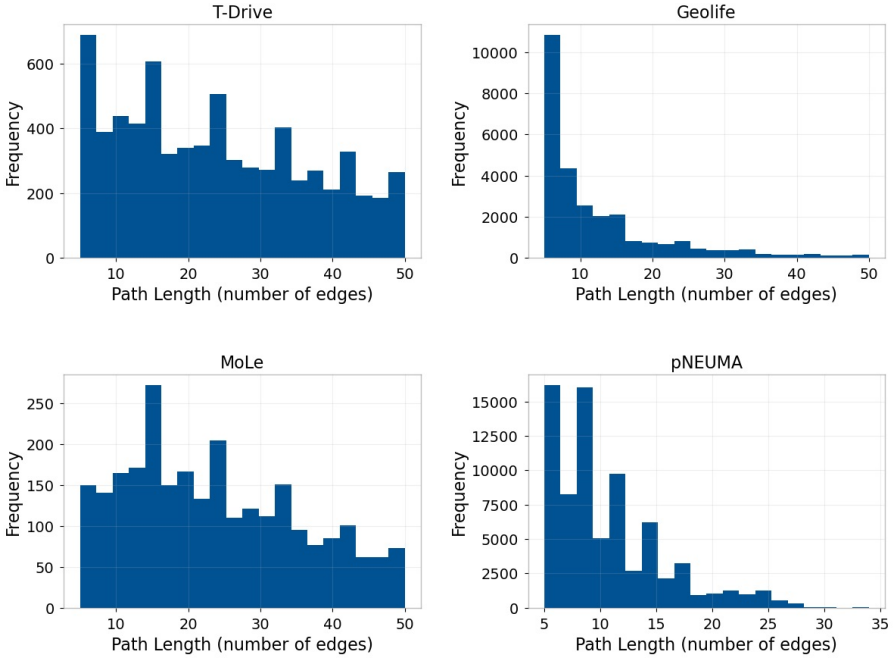


Figure 5.2 Distribution of the path lengths in the different datasets.

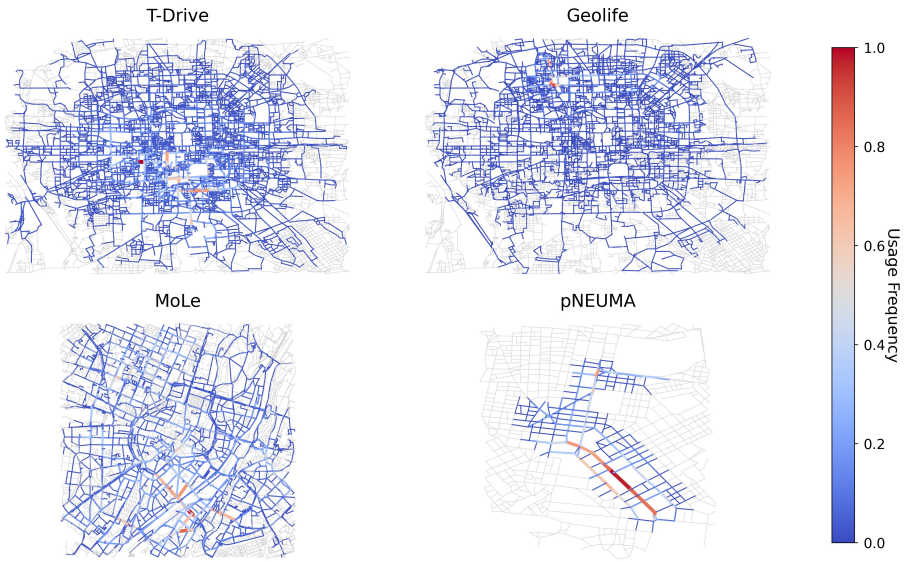


Figure 5.3 Heatmap representing how often each edge has been traversed by the trajectories in the respective datasets.

6 Experiments and Results

This chapter details the results obtained from applying our diffusion-based trajectory prediction model in different experimental settings. Our experiments are designed to assess the performance of both the standard model and a conditional model in predicting future trajectories within graph-based environments. We evaluate how our model's performance, in terms of ADE and FDE, fares against different AR benchmarks and assess how the choice of the transition matrices and noising schedule affects our results. Finally, we investigate our model's ability to generate longer paths and to plan trajectories from a given start point to a given endpoint.

6.1 Future Trajectory Prediction

Benchmarks

We compare the performance of our diffusion approach to common AR benchmarks to evaluate its effectiveness in trajectory prediction. One such benchmark is a **Markov** chain model, which predicts the most likely next edge based on the current edge's neighboring edges. The probability of selecting a neighboring edge is determined by the frequency of that transition occurring in the training data. Specifically, we calculate transition probabilities to adjacent edges for each sequence of edges by normalizing the observed transition counts from the training set.
We additionally build a **RNN**-based model, similar to CSSRNN as described in Subsec. 3.1.2. This means we constrain the decision space to the current neighbors of the last edge in the trajectory. We build three different versions, based either on RNN layers, LSTM layers, or Gated Recurrent Units (GRU) layers. Given that the implementation with LSTM layers outperformed the others, we use this version as the benchmark.

Metrics

To evaluate the predictive performance of models that address this task, there exist two commonly used metrics: Average Displacement Error (ADE) and Final Displacement Error (FDE). ADE measures the average distance between the predicted positions and the actual positions throughout the trajectory. FDE, on the other hand, specifically calculates the deviation between the predicted trajectory and the actual trajectory at their respective endpoints.

Results

We train and evaluate our standard diffusion model using two different future horizon lengths to assess its performance. First, we experiment with a short prediction horizon of $f = 2$ to determine whether our discrete diffusion approach enables the model to sample edges adjacent (or close) to the past trajectory and to learn the directionality of movement. This initial validation step is essential because, unlike current AR methods, our approach does not restrict the sampling space for the next edges to the immediate neighbors. By using a short horizon, we can, however, also verify if the model can accurately predict the next few moves of an agent on the graph, given its past trajectory.
After confirming the approach's effectiveness with $f = 2$, we extend the experiments to a longer horizon of $f = 10$ to investigate the model's ability to learn long-range dependencies and to sample edges that form longer, valid trajectories.
First, we evaluate the one-shot predictions of our diffusion model using ADE, as this metric enables a fair comparison with benchmark models.

As shown in Tab. 6.1, our generative approach achieves ADE performance comparable to current AR benchmark models. For longer prediction horizons ($f = 10$), our diffusion model even outperforms the AR benchmarks on the T-Drive, pNEUMA, and MoLe datasets in terms of ADE. For $f = 2$, the performance of the diffusion model is comparable to that of the AR benchmarks.
Here and moving forward, we present the results of our best-performing configurations for each dataset and future horizon f. On datasets that live on large graphs (T-Drive, Geolife, and MoLe), we use a combination of uniform transition matrices and a linear noising schedule for $f = 2$. For larger prediction horizons f, the marginal prior transition matrices are best suited to generate satisfactory results. Both on T-Drive and on Geolife, the linear noise schedule slightly outperforms the cosine schedule. On MoLe, the opposite is true. On pNEUMA, due to the significant differences in the size of its graph and the amount of available training data, we utilize the prior transition matrices in combination with a cosine noising schedule for all prediction horizons f. We shed more light on the importance of the transition matrix and noising schedule in Subsec. 6.2.3.

Table 6.1 One-Shot ADE comparison across datasets and different future horizons f.

Dataset	f	Valid Ratio (ours)	Diffusion (ours)	Markov	RNN
T-Drive	2	0.757	**0.00453**	0.0068	0.0052
	10	0.437	**0.01613**	0.02414	0.0198
Geolife	2	0.904	**0.00381**	0.00572	0.00398
	10	0.664	0.01765	0.01946	**0.01677**
MoLe	2	0.81	0.00753	0.01217	**0.00601**
	10	0.28	**0.02945**	0.04329	0.03533
pNEUMA	2	0.986	0.01777	0.01697	**0.00866**
	10	0.704	**0.05088**	0.0794	0.06078

Qualitatively, Fig. 6.1 and Fig. 6.2 display sample results on pNEUMA and T-Drive, both for short and longer future horizons.
From this qualitative evaluation, it is evident that we have to caveat the quantitative results (e.g., ADE) of our model. Even though for $f = 2$ we achieve great qualitative results, with realistic and valid trajectories, especially for larger f and on datasets that live on large graphs (such as T-Drive in this case), our model struggles to produce valid paths for each trajectory. This is a major drawback of our approach compared to AR methods, which, due to their implementation, only produce connected paths. Additionally, the ADE metric does not take this limitation into account, hence why the qualitative assessment is crucial here. However, on the pNEUMA dataset, the problem of invalid paths is less pronounced because we have more data and a smaller graph, which facilitates better learning and path connectivity. Because calculating the FDE for invalid paths is ill-defined, we omit it here.
Therefore, in Subsec. 6.2.1, we investigate whether our model can generate valid paths on these challenging datasets—characterized by large graphs and limited data availability—when given multiple sampling attempts.

6.2 Ablation Studies

6.2.1 Valid Paths

As mentioned above, we will now evaluate whether our diffusion model is able to predict valid longer future trajectories ($f = 10$) when given multiple tries. For this, we generate ten samples for each input path. If any of these ten samples fulfills the validity conditions (connected, acyclic, no splits into multiple trajectories), we save it and use it for comparing our quantitative results to those of the AR benchmarks, as seen in Tab. 6.2. If multiple valid paths are found, we select the longest one, to counter the tendency of selecting

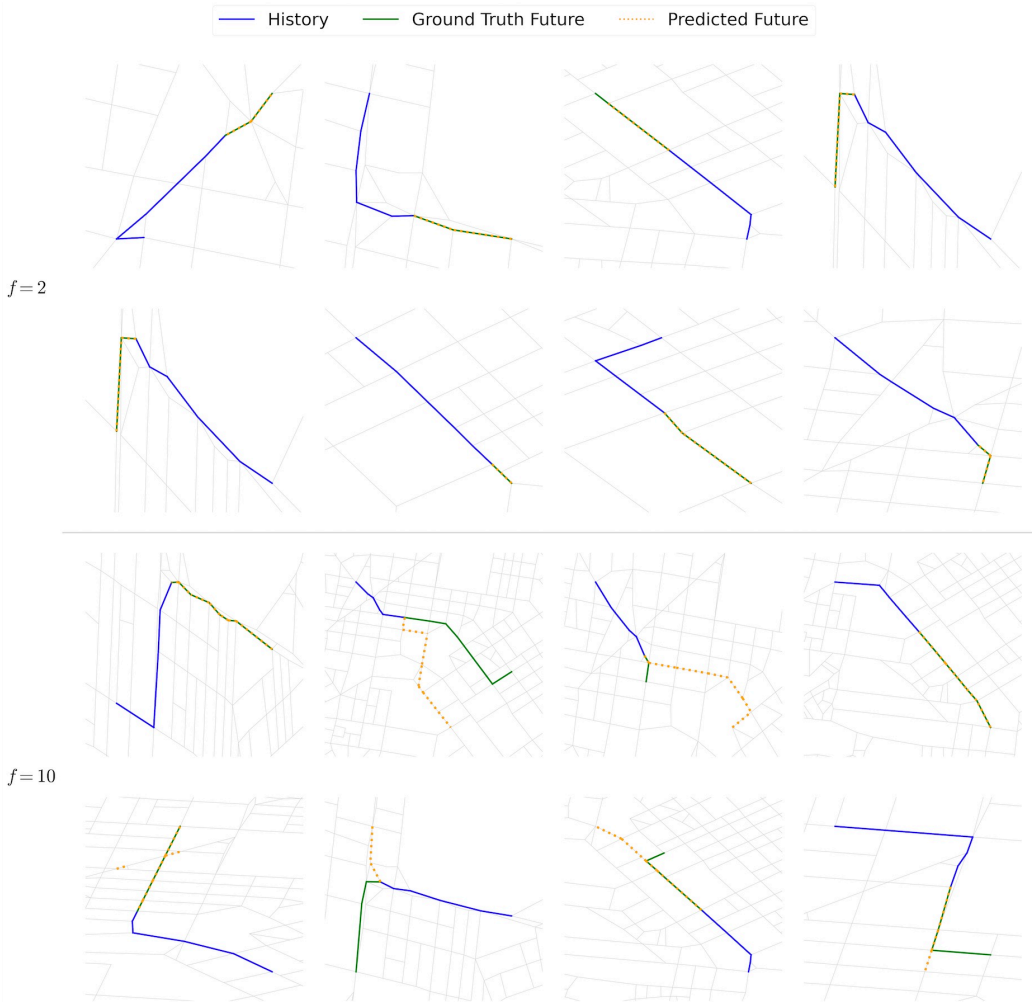


Figure 6.1 Exemplary samples generated by our standard diffusion model on pNEUMA.

shorter trajectories. Since for valid paths, we have a defined start and endpoint, we can now also evaluate the FDE metric. The **Valid** column in Tab. 6.2 indicates the fraction of trajectories for which our model was able to generate at least one valid sample out of ten attempts.
Our model is thus capable in roughly 90% of cases to generate valid trajectories that can be usefully analyzed and compared to results from AR benchmarks. On the more complex datasets such as T-Drive and MoLe, we sacrifice, however, some of the path length when selecting valid paths only as revealed by Fig. 6.5. This indicates that on these datasets, our diffusion model only has a limited ability to generate longer paths. On pNEUMA and Geolife, on the other hand, we do not encounter this issue and are, in fact, able to find even longer valid paths compared to the one-shot samples. This is most likely due to the fact that on these datasets with more available training data, our model can generally sample more valid paths (see Tab. 6.1). Selecting the longest one then leads to increased path lengths. The fact that the sampled paths on pNEUMA are, in general, shorter than those of T-Drive is simply due to the shorter trajectories in the training data of pNEUMA (see Tab. 5.1).
In terms of ADE and FDE, our diffusion approach now trails the AR benchmarks slightly for valid paths for $f = 10$. One of the causes for this drop in performance lies in the fact, that the crude method of simply selecting the longest valid trajectory can lead to large errors because the ground truth and the predicted path have very different lengths, respectively. This is also observable in Fig. 6.3 and Fig. 6.4.
Because our model runs into similar issues on MoLe as on T-Drive when dealing with long prediction horizons as seen in Fig. 6.2 and Fig. A.3, we conduct the qualitative assessment on MoLe for valid paths. Fig. 6.3 shows that, indeed, our model can generate slightly longer valid paths, even on complex datasets, but is, in general, not able to accurately predict the whole length of the ground truth future trajectory. Fig. 6.4

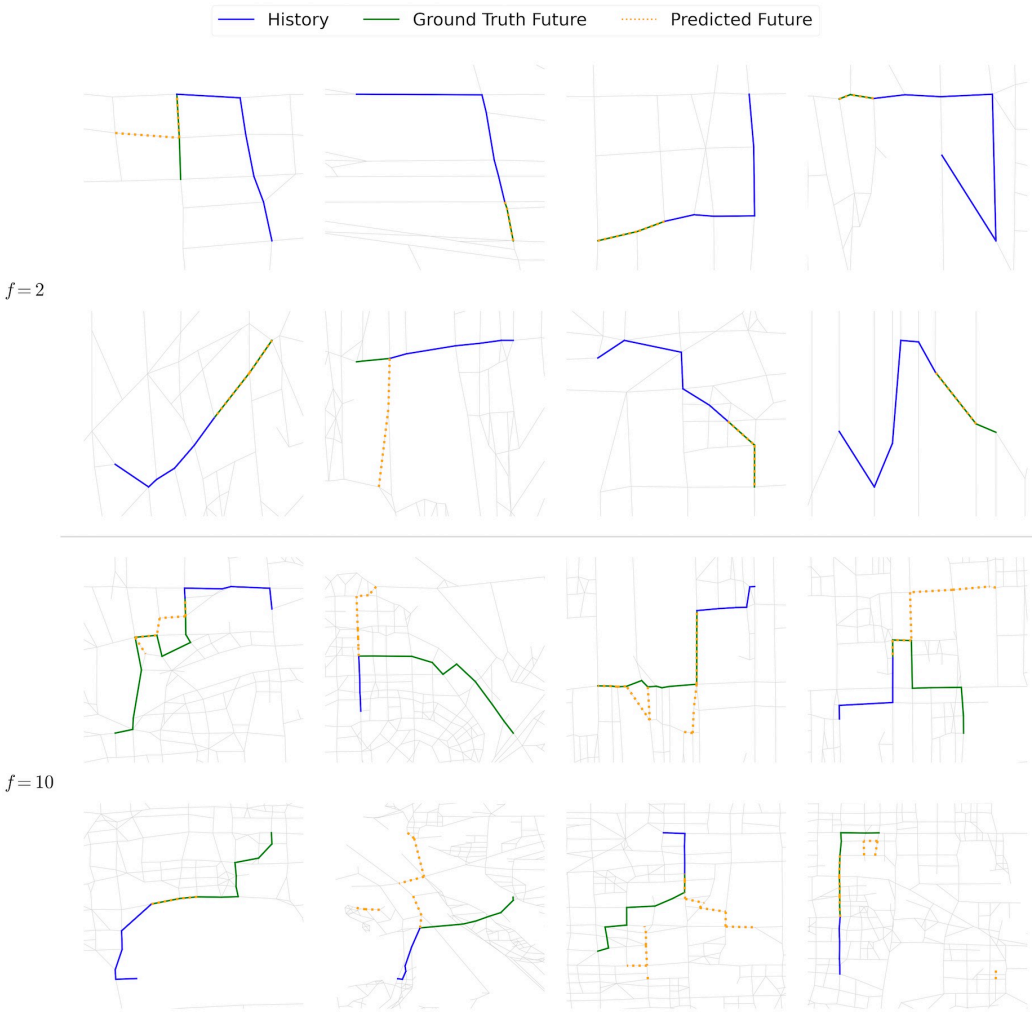


Figure 6.2 Exemplary samples generated by our standard diffusion model on T-Drive.

displays a similar picture for the valid paths generated on Geolife.

In addition to the ability to select valid paths, generating multiple samples allows us to verify the diversity of the samples our model produces as well as their general trend. In Fig. 6.6, we can observe that our approach is able to achieve a good level of diversity in its samples while capturing the trend of the trajectory as well.
As the qualitative assessment and the worse quantitative results in Tab. 6.2 show, we should try to increase the performance of our model by getting the correct future length. Hence, why we introduce a conditional model in Subsec. 6.2.2.

6.2.2 Conditional Model

Conditioning has been shown to significantly enhance the performance of diffusion models across various data domains, such as images and videos [65, 66, 67]. Implementing conditioning is straightforward and can even be done post-training. However, it introduces a trade-off between sample diversity and quality, requiring careful calibration to achieve optimal results.
Hence, above the standard model, we also experiment with a conditional discrete diffusion model. In our case, the condition is simply the size of the future horizon and is introduced as an additional feature in $\boldsymbol{F}$. In combination with the number of already predicted edges, this enables the model to easily sample the desired number of edges in the future path.
In a first step, we choose the actual ground truth future length of each trajectory $\tilde{f}$ as the condition and

Table 6.2 Valid Paths: Comparison of ADE and FDE across datasets and future horizons $f = 10$.

Dataset	Valid	ADE			FDE		
		Diffusion	Markov	RNN	Diffusion	Markov	RNN
T-Drive	0.966	0.0322	0.02414	**0.01980**	0.04598	0.04321	**0.03828**
Geolife	0.993	0.02864	0.01946	**0.01677**	0.03867	**0.02828**	0.02876
MoLe	0.873	0.05754	0.04329	**0.03533**	0.08901	**0.07684**	0.07692
pNEUMA	0.995	0.06733	0.07940	**0.06078**	0.13227	0.12846	**0.11398**

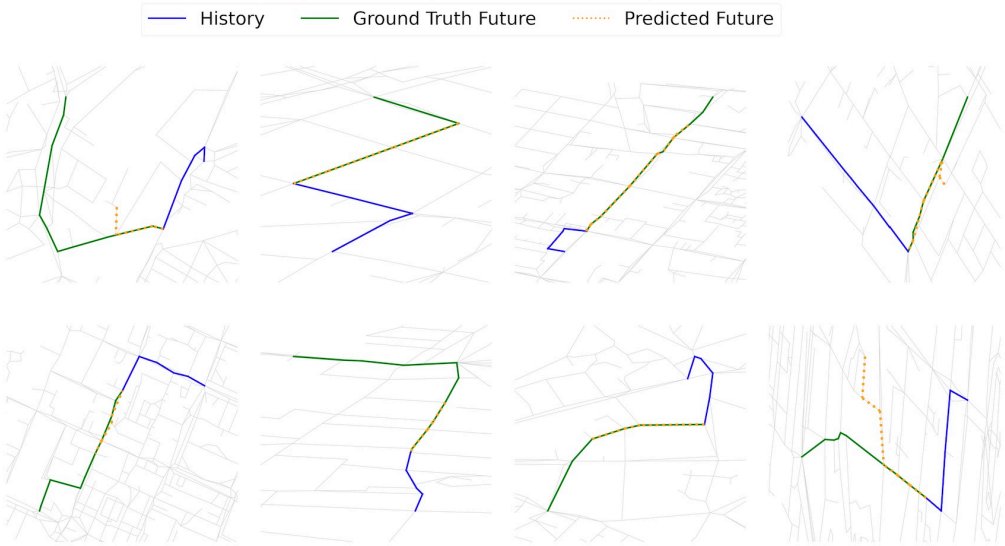


Figure 6.3 Exemplary valid samples generated by our diffusion model on MoLe with $f = 10$.

divide it by $|\mathcal{E}|$ to ensure that all features have a similar magnitude. If we use the marginal prior transition matrix, we redefine p_1 with the average of $\tilde{f}$ divided by $|\mathcal{E}|$. Due to the fact that $\tilde{f}$ is relatively large, the combination of marginal prior transition matrices and a cosine noise schedule leads to the best results.
On pNEUMA, the conditional model performs very well qualitatively (see Fig. 6.7). As we can see, it is usually able to sample the correct path length and further improves the prediction accuracy over the standard model. Moreover, the model's ability of generating valid samples is also bettered.
The quantitative comparison is not straightforward since the future length f differs between the models. But, considering that the generated sample lengths for $f = 10$ in the standard model and for $f = \tilde{f}$ in the conditional model are similar, we still present the quantitative results in Tab. 6.3. A conditional model thus also outperforms a standard diffusion model for our task. In fact, the conditional model also outperforms the AR benchmarks, both qualitatively and quantitatively, in terms of ADE. Additionally, it becomes clear that with the condition, the diffusion model is able to accurately learn and sample the correct path lengths.
On Geolife, the conditional model also improves slightly over the standard diffusion model. However, for this task, the uniform transition matrix in combination with a cosine noise schedule performs best, especially when we consider the valid paths only. In this setting, it achieves an ADE of 0.0086 and an FDE of 0.00983 with an average sample length of 2.918 edges, thus outperforming the standard diffusion model as well as the AR benchmarks with $f = 10$. Qualitatively, Fig. 6.8 shows valid paths generated by our conditional model on Geolife. Even though most samples are rather short, their directionality is correct and their accuracy is high.
On the other datasets, the conditional model was not able to comprehensively improve over the standard model. This might again be a consequence of the poorer data quality, the lower amount of available data, and the size of the underlying graphs.

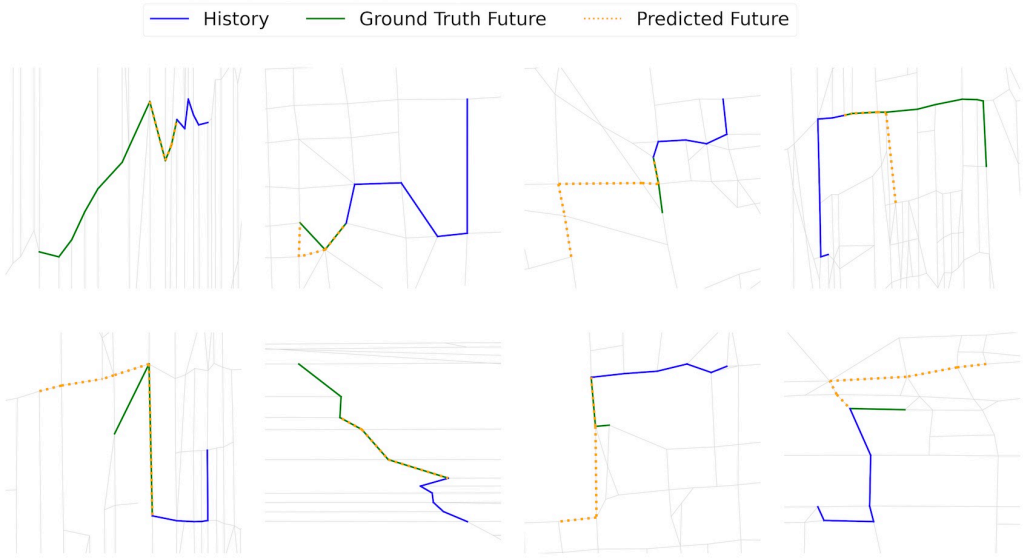


Figure 6.4 Exemplary valid samples generated by our diffusion model on Geolife with $f = 10$.

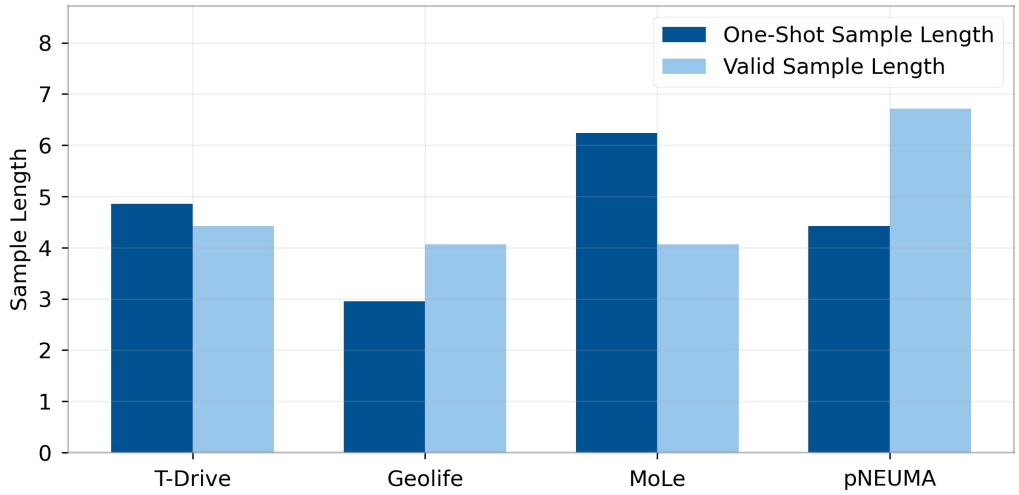


Figure 6.5 Differences in sample length for $f = 10$ between one-shot samples and valid samples.

Generating Long Paths on pNEUMA

Given that the average path length in pNEUMA is much shorter than in T-Drive or MoLe, for example, we want to investigate the reason behind the improvement of the conditional model on pNEUMA:

- Is it due to the fact that our diffusion model (standard and conditional) cannot produce long, valid paths?
- Is it due to the data issues on T-Drive, Geolife, and MoLe mentioned above?

To find out, we try out different conditional future lengths above $\tilde{f}$, forcing the model to predict longer paths than present in the pNEUMA dataset and evaluate its performance in those scenarios. We experiment with conditional future lengths of 10, 15, and 20.
As Fig. 6.9 clearly shows, our diffusion approach is capable of generating long, valid trajectories on pNEUMA. The fact that the conditional future lengths we experiment with exceed the longest paths present in pNEUMA further validates our method.
Fig. 6.10 clearly shows the impact of the condition on the sample length-a clear indication that our model is able to learn and sample the desired length. Moreover, it shows that, when given multiple attempts, the

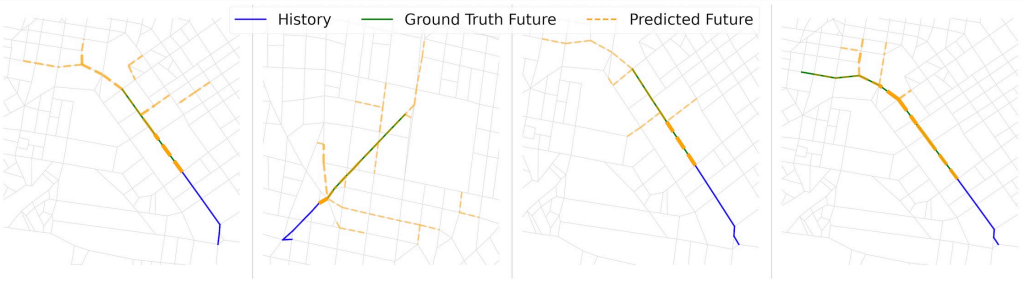


Figure 6.6 Density plot for trajectories with $f = 10$ on pNEUMA. The thickness of the sampled edges indicates how often they have been sampled in all of the generated ten samples.

Table 6.3 Comparison between the one-shot results of the standard model and the conditional model on pNEUMA.

Model	f	F1 Score	ADE	Avg. Pred. Length	Valid Ratio
Standard	10	0.472	0.05088	4.419	0.704
Conditional	$\tilde{f}$	**0.777**	**0.01467**	5.169	**0.804**

diffusion model can even generate very long valid paths up to $f_{cond} = 20$. However, we also observe that the quality and the diversity of the samples decrease as we increase f_{cond}.
We thus conclude that the improvement of the conditional model over the standard model on pNEUMA is in fact not due to our model's inability to generate long connected paths. The reason that the conditional model does not deliver improved results on T-Drive and MoLe, for example, is thus most likely the lack and the lower quality of data available.

6.2.3 Review of Approach

Model Architecture

To motivate the choice of the complex GAT-based model, we investigate how a pure MLP approach fares against a graph-oriented architecture on pNEUMA (our highest-quality dataset). The results listed in Tab. 6.4 show that a GAT based architecture comfortably outperforms an MLP-only model in all relevant metrics and for different prediction horizons. This confirms our choice to use GAT layers as they can capture and represent a notion of graph structure.
The advantage of the GAT-based model is also evident in qualitative assessments. By examining sample outputs from the MLP-only model (see Fig. A.2), it becomes apparent that the MLP architecture struggles to produce connected paths due to its inability to leverage the structural information inherent in the graph. In contrast, the GAT-based model effectively utilizes the graph structure to generate connected paths (see Fig. 6.1 for a direct comparison to the MLP figure mentioned above) by incorporating relational information between edges through attention mechanisms. This capability allows the model to sample valid trajectories that adhere to the topology of the graph. Consequently, we focus exclusively on the GAT-based approach in our subsequent experiments, as it demonstrates superior performance both quantitatively and qualitatively.

Table 6.4 Comparison between MLP-only and GAT-based models trained on pNEUMA with marginal prior transition matrices and a cosine noising schedule. Results on the validation set.

Architecture	f	F1 Score	ADE	Avg. Pred. Length	Valid Ratio
MLP	2	0.43	0.04515	1.359	0.735
GAT+MLP		**0.646**	**0.01747**	1.277	**0.984**
MLP	10	0.191	0.19629	5.885	0.055
GAT+MLP		**0.456**	**0.05032**	4.599	**0.672**

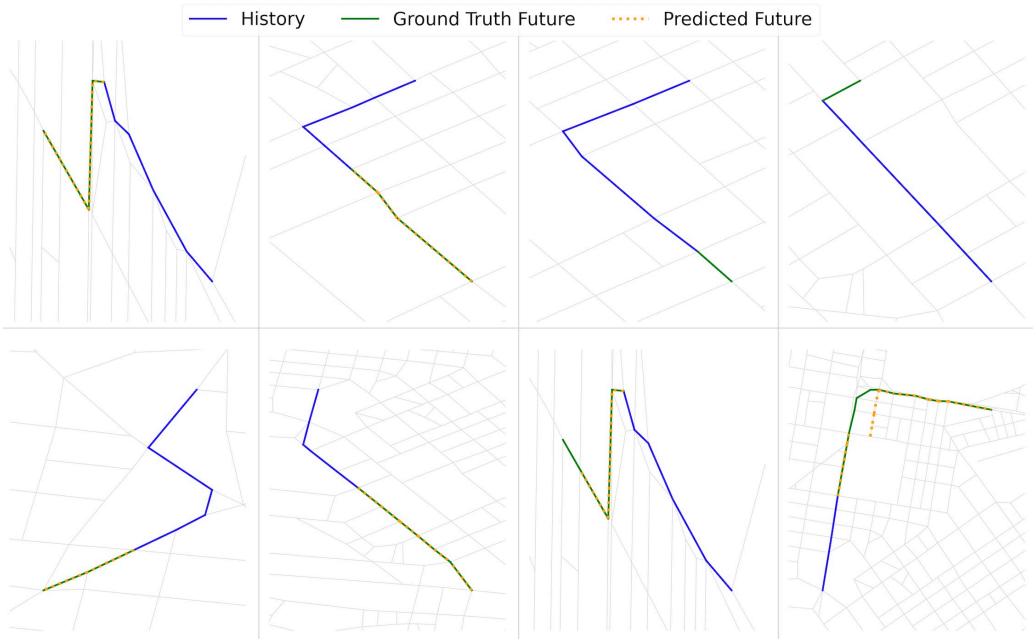


Figure 6.7 Exemplary samples generated by our conditional diffusion model on pNEUMA with $f_{cond} = \tilde{f}$.

Transition Matrices and Noising Schedules

Next, we will compare how the choice of the transition matrices and the noising schedules affects results. Tab. A.2 presents the one-shot results for each combination of our two transition matrices (Uniform and Marginal Prior) and our two noising schedules (linear and cosine) for two representative datasets.
It turns out that the optimal combination of transition matrix type and noising schedule is highly dependent on the size of the graph and the future prediction horizon f. Here, T-Drive serves as an exemplary dataset on a large graph, and pNEUMA as an example of a dataset that lives on a smaller graph. On large graphs with a small prediction horizon, the diffusion approach struggles to sample any edges if we train it with a prior transition matrix. Hence, in this case, the uniform type is preferable. If we increase the prediction horizon, our marginal distribution changes, and the application of the marginal prior transition matrices is beneficial. More specifically, using the uniform transition matrix prevents our model from generating valid paths, as opposed to the model trained with a marginal prior transition matrix.
On smaller graphs, the model is not as sensitive to the choice of the transition matrix type. However, for longer prediction horizons, the prior transition matrix is again better suited for generating valid paths. It seems that, when starting from uniform noise, the model struggles to 'turn off' enough edges over the denoising process, which leads to disconnected and thus invalid paths, as seen in Fig. 6.11. This visualization also clearly shows that starting from the prior distribution helps the denoising process to converge faster and generate more valid samples.
The noising schedule does not seem to play a major role itself, as both schedule types lead to similar results.

6.2.4 Trajectory Inpainting

Finally, we evaluate our model's performance on the task of trajectory inpainting or path planning. In this scenario, the model is provided with only the start and endpoints of each trajectory as the "history" input, and it must infer the intermediate edges that constitute the ground truth. This task is particularly compelling because it directly corresponds to the real-world challenge of navigation.
For this task, we again utilize the conditional model with $f_{cond} = \tilde{f}$ and evaluate it on pNEUMA and on Geolife. The results of this experiment can be seen in Fig. 6.12 and Fig. 6.13. Here, our diffusion model performs exceptionally well on pNEUMA, being able to find valid connections between the start and

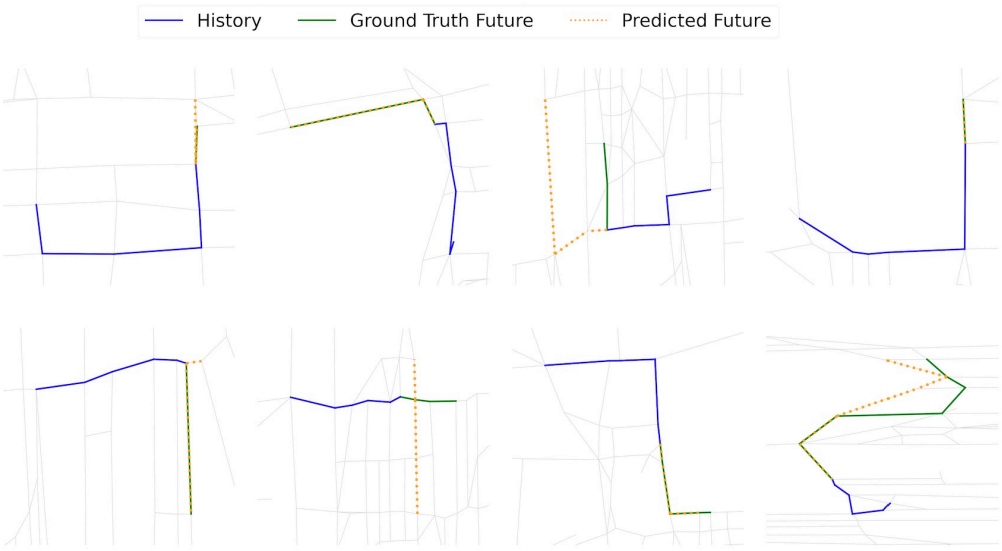


Figure 6.8 Exemplary valid samples generated by our conditional diffusion model on Geolife with $f_{cond} = \tilde{f}$.

endpoints of each trajectory. On Geolife, we observe once again, that our diffusion approach struggles to always find connected paths. Hence, why we show the valid paths (valid ratio of 75%) only for Geolife as well in Fig. 6.14. In this scenario, our model can generate high-quality samples that connect start and end points most of the time.
One drawback that becomes apparent in Fig. 6.15 is the relative lack of diversity that the model produces. In part, this is due to the use of the conditional model. Since most start-to-end trajectories follow the shortest path, our model tends to also create the shortest path, thus limiting diversity.

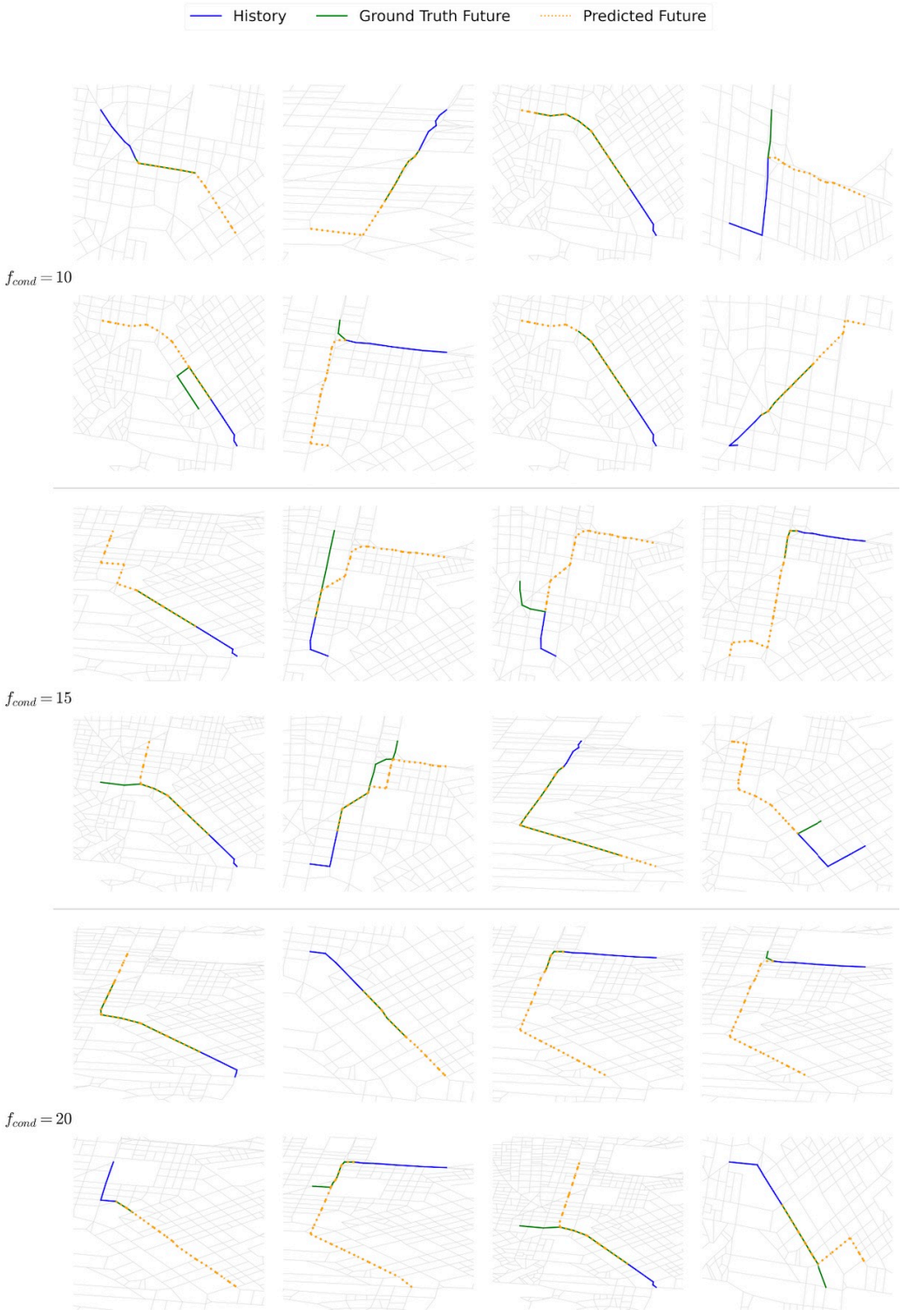


Figure 6.9 Exemplary valid samples generated by our conditional diffusion model on pNEUMA with different conditions.

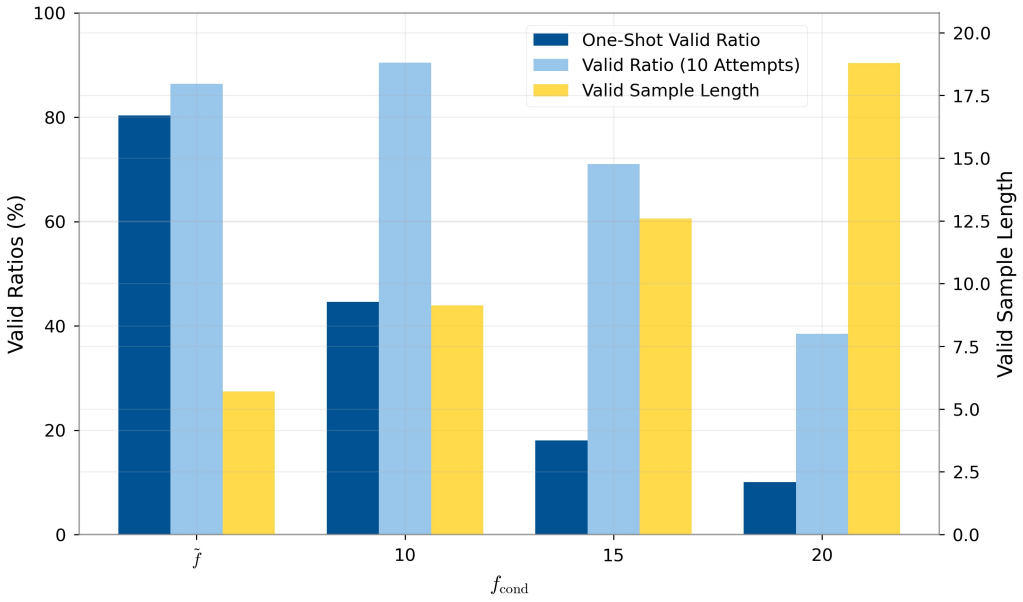


Figure 6.10 Valid sample ratios and sample lengths generated by our conditional model on pNEUMA.

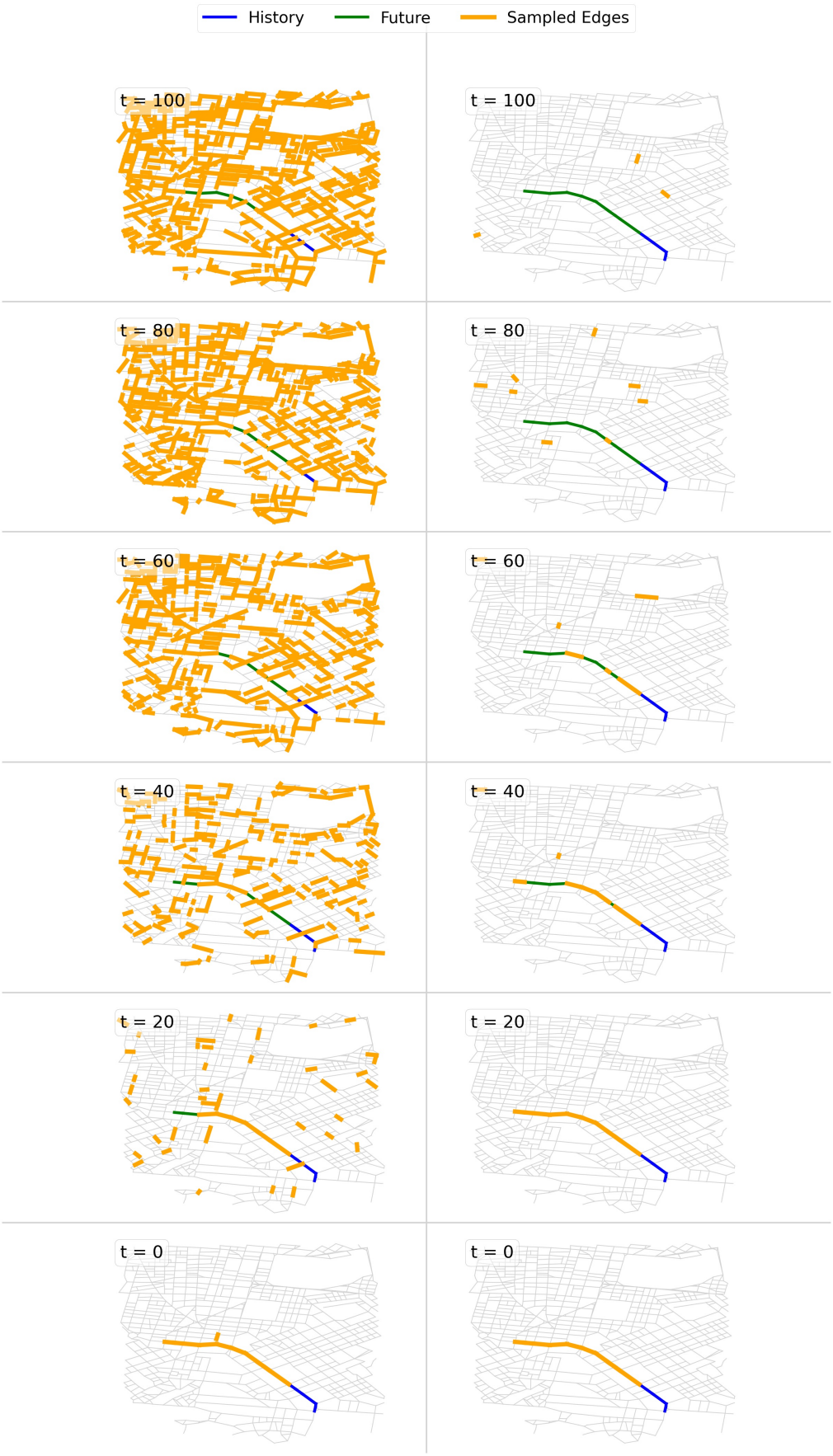


Figure 6.11 Comparison of the denoising processes with the uniform transition matrices (left) and the marginal prior transition matrices (right) on the pNEUMA dataset with $f = 10$. Both models have been trained with a cosine noise schedule.

Figure 6.12 Exemplary samples generated by our conditional model, for the trajectory inpainting experiment on pNEUMA.

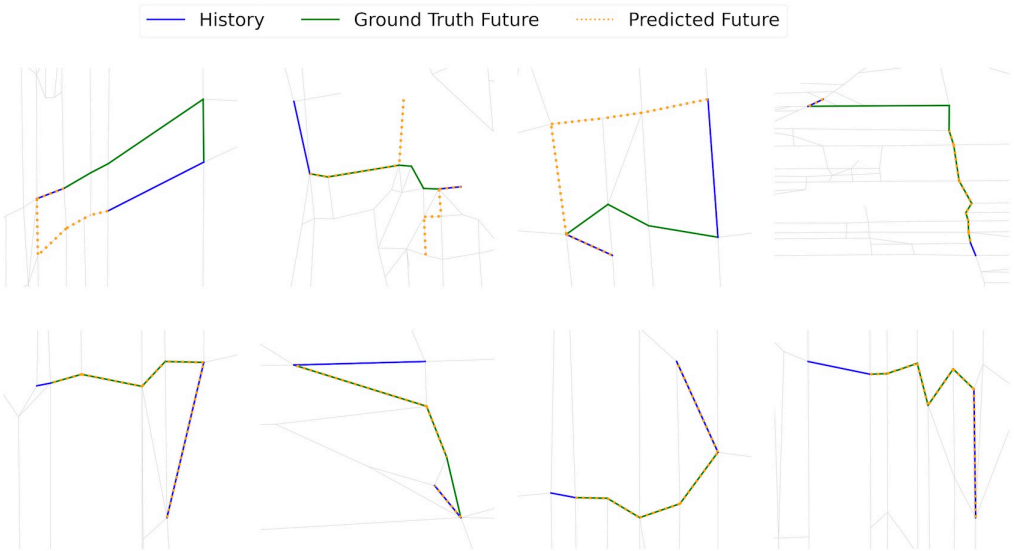


Figure 6.13 Exemplary samples generated by our conditional model, for the trajectory inpainting experiment on Geolife.

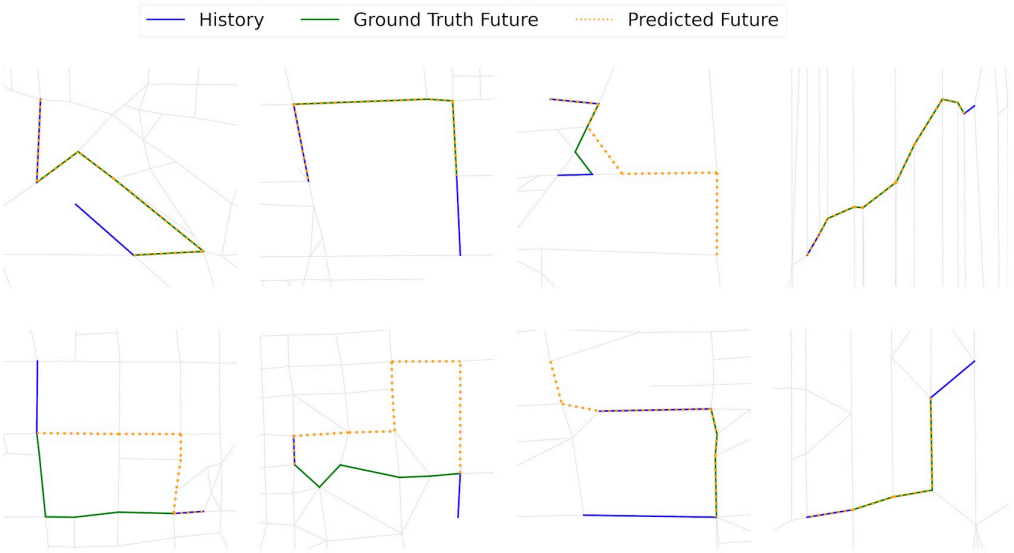


Figure 6.14 Exemplary valid samples generated by our conditional model, for the trajectory inpainting experiment on Geolife.

Figure 6.15 Density plot for the trajectory inpainting experiment on pNEUMA. The thickness of the sampled edges indicates how often they have been sampled in all of the generated ten samples.

7 Conclusion and Outlook

From our results and experiments in Chapter 6, we can conclude that for predicting small future horizons (i.e., $f = 2$), our diffusion approach produces high-quality results on all four datasets we experiment with. Quantitatively, we can match or slightly outperform AR benchmarks on datasets that live on large graphs. On pNEUMA, on the other hand, the AR models achieve better ADE for $f = 2$ since our diffusion approach produces paths that are too short on average.
For longer predictions (i.e., $f = 10$), we observe a different picture. Our diffusion approach now achieves better ADE than AR models on pNEUMA, T-Drive, and MoLe. With this increase in sampling length, the biggest shortcoming of our method reveals itself: the inability to generate valid paths for some trajectories. This issue is most prominent in datasets where the data quality and especially quantity is too low, such as T-Drive and MoLe. Two other aspects that likely plague our model's performance on these datasets are the longer trajectories on average and the wide variety in trajectory lengths. While the size of the graph appears to influence our model's performance—as evidenced by our best results on the smaller pNEUMA network—the fact that we also achieve good results on Geolife, which has a large road network but ample data, suggests that insufficient data on large graphs (like T-Drive and MoLe) plays a more significant role in limiting our model's performance than the graph size itself. To mitigate these issues, future approaches could focus on combining data sources that live on the same graph, such as T-Drive and Geolife, to build a higher-performance model. Additionally, if ample data is available, selecting trajectories that fall into a certain range in terms of length might lead to enhanced performance while trading off the diversity of the results though.
Solving the issue of invalid paths by producing multiple samples and selecting the longest valid one leads to a different tradeoff that gets clear when looking at quantitative results: selecting valid paths leads to a decrease in quantitative performance. Therefore, a potential area for future work is developing a more effective post-processing algorithm for the raw samples. Instead of merely selecting the longest valid sample from multiple raw outputs, algorithmically generating valid paths may be possible by correcting or reconstructing invalid raw samples. Alternatively, working on the model architecture and clever feature engineering could help solve the issues of invalid paths, too. Similar to DiGress, including spectral and structural features such as the cycle count or the number of disconnected edge sequences in a trajectory might help the model generate more valid paths during the denoising process [44]. Additionally, Spatio-Spectral GNNs or S2GNNs have recently been shown to enable global and efficient information propagation on graphs, thus underlining the importance of spectral features in GNNs for modeling long-range interactions [68].
Another insight we gather is that conditioning leads to improved performance if we have enough data. As a result, our conditional diffusion model outperforms the standard diffusion model and the AR benchmarks, both on pNEUMA and Geolife. Moreover, tweaking the condition enables the model to generate cohesive paths that are longer than any trajectories present in the training data.
Lastly, our diffusion approach performs qualitatively well on path-planning tasks, but the samples lack diversity and usually revert to predicting the shortest path. A quantitative comparison to AR methods is also reserved for future work.

In conclusion, we show that a discrete diffusion approach is able to compete or even outperform current AR benchmarks for trajectory prediction on graphs if enough training data is available. This work can serve as a starting point for developing more powerful diffusion approaches that ultimately solve the remaining issue of invalid paths and thus represent a superior alternative to AR methods.

A Appendix

A.1 Experimental Setup

Training and sampling was performed on different types of GPUs:

- **NVIDIA GeForce GTX 1080 Ti** with 11GB of memory
- **NVIDIA GeForce RTX 2080 Ti** with 11GB of memory
- **NVIDIA Tesla V100** with 32GB of memory

We implemented our model in PyTorch with 3 GAT layers (and 3 attention heads), including residual connections, both with a hidden size of 64. We used a positional encoding dimension of 16 for both the timestep encoding and the embedding of the edge indices.
On T-Drive, Geolife, and MoLe, we trained the model for 80 epochs. On pNEUMA, we trained the model for 60 epochs because, due to the smaller graph and the vast amount of data, we observed faster convergence. For the optimization, we always used the Adam algorithm, a learning rate of 0.009 in all cases except for $f = 2$ on Geolife. Here, the training proved to be very noisy hence, we opted for a smaller learning rate of 0.003. To stabilize training further, we used gradient clipping as well as exponential learning rate decay after 40 epochs. During training, we used a batch size of 64 for pNEUMA and 6 for the other datasets (T-Drive, MoLe, Geolife) due to their larger underlying graph and, thus, larger edge index.
We used $T = 100$ diffusion timesteps along with $\beta_0 = 1e - 5$ and $\beta_T = 0.09$ for the linear noise schedule.

A.2 Additional Experimental Results

In this section of the Appendix, we will briefly present some additional interesting results from our various experiments.

A.2.1 Feature Selection

As described in Subsec. 4.1.3, we designed a feature matrix $\boldsymbol{F}$ to contain information about the trajectory history, edge data such as edge coordinates and their length, but also variable topological features such as the distance of each edge to the last point of the history or the angle of each edge to the history. While calculating these features for each trajectory is somewhat expensive, especially on larger graphs, the information is essential for our model. Fig. A.1 shows some qualitative results on pNEUMA when we omit every feature except the binary history and the edge coordinates. While for $f = 2$, the model is still able to generate good and partially realistic predictions, for larger future horizons, it fails completely when compared to the results using the full feature matrix in Fig. 6.1.
Quantitatively, the improvement of including topological features is laid out in Tab. A.1.

A.2.2 Model Architecture

We chose to use a GAT-based model to include structural information of the underlying graph in our model. The importance of using this architecture over a simple MLP model gets clear once we compare the results of both approaches quantitatively. But also qualitatively, the MLP model fares much worse compared to the GAT model, especially for longer future horizons, as is showcased in Fig. A.2.

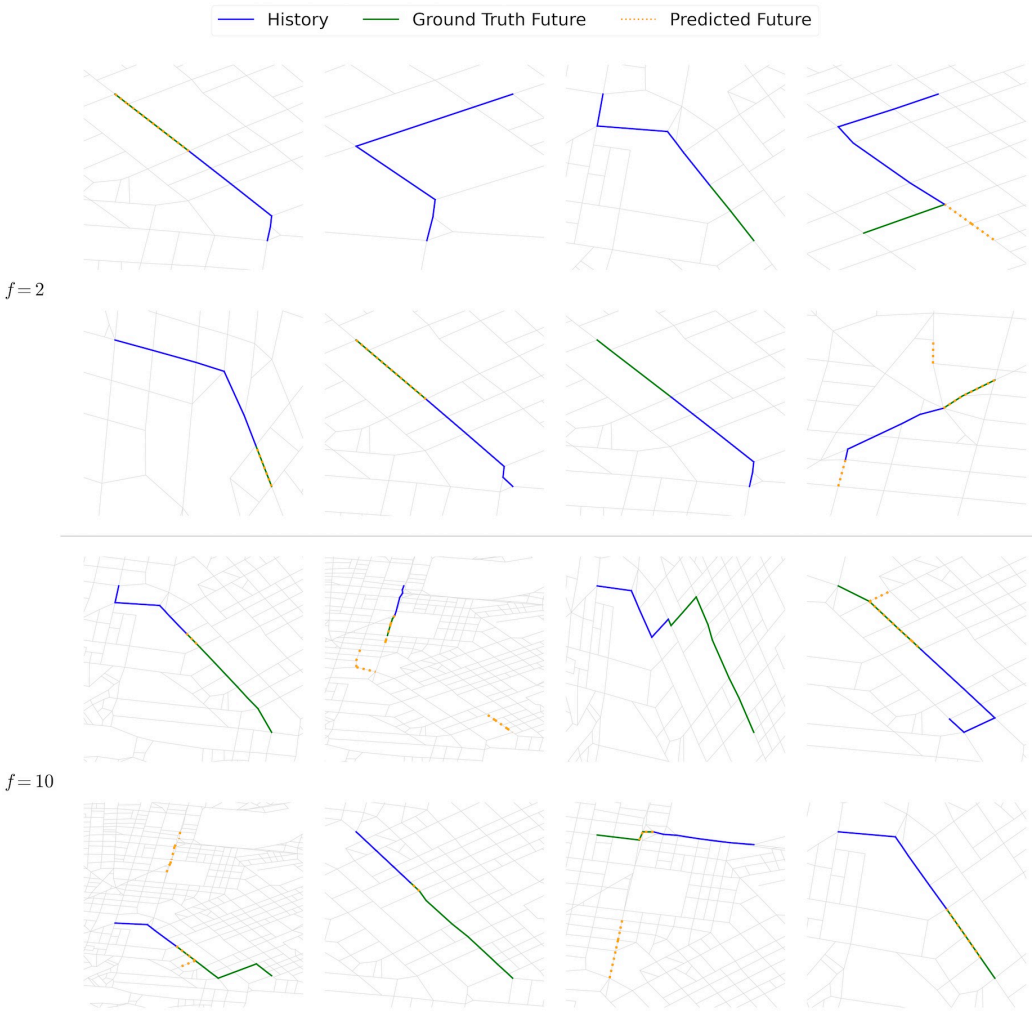


Figure A.1 Exemplary samples generated by our standard diffusion model on pNEUMA with a reduced feature set containing only the binary history and the edge coordinates.

A.2.3 Transition Matrices and Noising Schedule

Choosing a suitable transition matrix and noising configuration is an important step in designing discrete diffusion models. Tab. A.2 shows large differences in the results for different choices of these diffusion parameters. There is also a clear dependence between f, the size of the graph, and the transition matrix.

A.2.4 Additional Visualizations

In this section, we will showcase some more visualizations to underline our work. Fig. A.3 and Fig. A.4 show some more samples generated by our standard diffusion approach on MoLe and on Geolife respectively for different future horizons f.

Table A.1 Comparison between models trained with different feature sets on pNEUMA with $f = 10$, marginal prior transition matrices, and a cosine noise schedule.

Features	F1 Score	ADE	Avg. Pred. Length	Valid Ratio
[h_{bin}, coordinates]	0.308	0.14882	4.589	0.51
[h_{bin}, coordinates, PE]	0.296	0.13303	4.502	0.565
[h_{bin}, coordinates, PE, length]	0.321	0.14768	5.106	0.461
[h_{bin}, coordinates, PE, angle]	0.407	0.09298	4.087	0.643
[h_{bin}, coordinates, PE, distance]	0.439	0.05244	3.738	0.717
[h_{bin}, coordinates, PE, length, angle, distance]	0.5	0.03719	3.529	0.834

Table A.2 Comparison between models trained with different transition matrices and different noising schedules. Results are obtained on the validation set.

Dataset	f	Transition Matrix	Noising	F1 Score	ADE	Avg. Pred. Length	Valid Ratio
T-Drive	2	Marginal Prior	Linear	0	0.0068	0	**1**
			Cosine	0	0.0068	0	**1**
		Uniform	Linear	**0.472**	**0.00451**	2.309	0.756
			Cosine	0.429	0.00528	2.216	0.747
T-Drive	10	Marginal Prior	Linear	0.141	**0.01539**	4.868	**0.435**
			Cosine	**0.173**	0.02304	9.267	0.281
		Uniform	Linear	0.155	0.02588	16.944	0.031
			Cosine	0.137	0.02999	15.132	0.031
pNEUMA	2	Marginal Prior	Linear	0.636	0.01842	1.258	0.99
			Cosine	**0.646**	**0.01747**	1.277	0.984
		Uniform	Linear	0	0.03393	0	**1**
			Cosine	0.605	0.01888	1.706	0.839
pNEUMA	10	Marginal Prior	Linear	0.424	0.04712	3.483	**0.709**
			Cosine	**0.456**	0.05032	4.599	0.672
		Uniform	Linear	0.427	0.06768	5.283	0.374
			Cosine	0.45	**0.03956**	3.84	0.669

Figure A.2 Exemplary samples generated by our standard MLP-only diffusion model on pNEUMA.

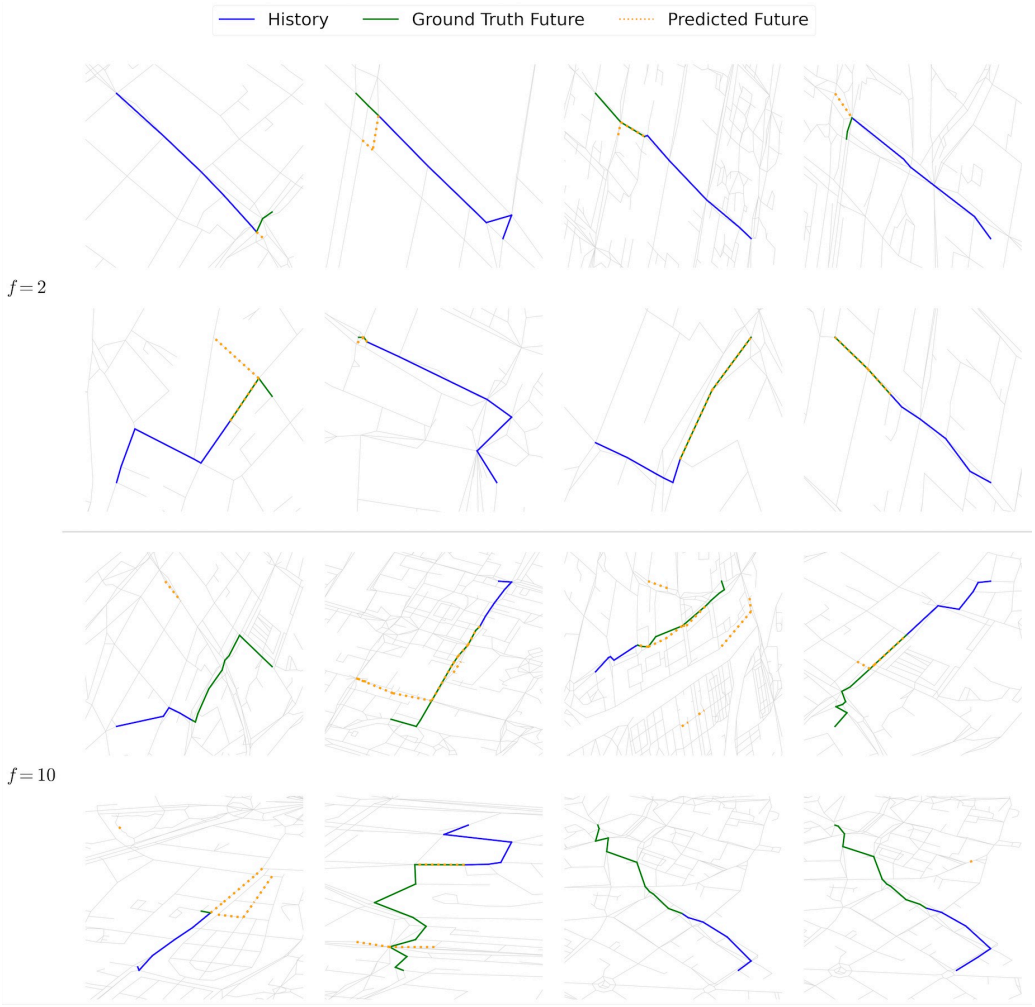


Figure A.3 Exemplary samples generated by our standard diffusion model on MoLe.

Figure A.4 Exemplary samples generated by our standard diffusion model on Geolife.

Glossary

Average Displacement Error Average distance between predicted positions and ground truth positions. 25–27, 29, 31, 39

Artificial Intelligence The simulation of human intelligence processes by machines, especially computer systems, enabling them to perform tasks that typically require human intelligence such as visual perception, speech recognition, decision-making, and language translation. 15

Autoregressive Referring to a model or process where the current output depends on its own previous outputs or states, commonly used in time series analysis and sequence modeling. 1, 7, 11, 14, 15, 19, 22, 25–27, 29, 39

Binary Cross Entropy A loss function used in binary classification problems to measure the difference between predicted probabilities and true labels, calculated as the negative average of the log-likelihood of the correct class. 21

Convolutional Neural Network A class of deep neural networks that use convolutional layers to automatically learn hierarchical spatial features from input data, particularly effective for processing grid-like data such as images. text. 7

Differentiable Message Passing A framework used in graph neural networks where messages are passed between nodes through differentiable functions, allowing end-to-end training via gradient-based optimization. 4, 5

Evidence Lower Bound A lower bound on the log marginal likelihood (evidence) of observed data, used in variational inference to approximate complex probability distributions by maximizing this bound. 10, 12, 15

Final Displacement Error Distance between the last predicted position and the last ground truth position. 25–27, 29

Generative Adversarial Network A class of machine learning frameworks where two neural networks, a generator and a discriminator, are trained simultaneously through adversarial processes, enabling the generation of new data samples that resemble the training data. 7

Graph Attention Network A neural network model that extends graph convolutional networks by incorporating attention mechanisms, allowing for adaptive weighting of neighbor nodes during feature aggregation. 5, 6, 21, 31, 41

Graph Convolutional Network A neural network architecture that generalizes convolutional operations to graph-structured data, enabling the aggregation of feature information from a node's local neighborhood. 5, 14, 15

Graph Neural Network A type of neural network designed to process data represented as graphs by leveraging the graph structure to learn node, edge, or graph-level representations. v, 4, 5, 7, 13, 21, 39

Global Positioning System A satellite-based navigation system used to determine precise location on Earth's surface, consisting of a constellation of 24 satellites orbiting the planet and receivers on Earth that can triangulate position based on signals from these satellites. 7, 23

$\mathbb{KL}$ A statistical distance that measures how one distribution p differs from a distribution q. It is defined as $\mathbb{KL}(p \parallel q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$. 10

Long Short-Term Memory Type of recurrent neural network that alleviates the vanishing gradient problem. 7, 25

Multi-Layer Perceptron A type of feedforward artificial neural network consisting of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one and uses activation functions to learn non-linear patterns. 5, 21, 31, 41

Machine Learning A discipline that combines statistics and algorithms to allow systems to learn from data autonomously and make data-driven decisions.. 2–4, 7, 8

Negative Log-Likelihood A loss function used in statistical models and machine learning, representing the negative logarithm of the likelihood function; minimizing the NLL corresponds to maximizing the likelihood of the observed data under the model. 10

Recurrent Neural Network A class of neural networks that process sequential data by using recurrent connections to retain information from previous inputs, enabling the modeling of temporal dependencies in sequences like time series or natural language. text. 7, 14

Variational Autoencoder A generative model that encodes input data into a probabilistic latent space and decodes it back to the original space, enabling the generation of new data samples by sampling from the latent space distribution. 7, 10

Bibliography

[1] Y. Huang et al. “A Survey on Trajectory-Prediction Methods for Autonomous Driving”. In: IEEE Transactions on Intelligent Vehicles 7.3 (2022), pp. 652–674. DOI: 10.1109/TIV.2022.3167103.

[2] B. Paden et al. “A survey of motion planning and control techniques for self-driving urban vehicles”. English (US). In: IEEE Transactions on Intelligent Vehicles 1.1 (2016), pp. 33–55. ISSN: 2379-8858. DOI: 10.1109/TIV.2016.2578706.

[3] T. Gu et al. Stochastic Trajectory Prediction via Motion Indeterminacy Diffusion. 2022. arXiv: 2203.13777 [cs.CV].

[4] T. Westny, B. Olofsson, and E. Frisk. Diffusion-Based Environment-Aware Trajectory Prediction. 2024. arXiv: 2403.11643 [cs.CV].

[5] H. Wu et al. “Modeling Trajectories with Recurrent Neural Networks”. en. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 3083–3090. ISBN: 978-0-9992411-0-3. DOI: 10.24963/ijcai.2017/430.

[6] J.-B. Cordonnier and A. Loukas. Extrapolating paths with graph neural networks. arXiv:1903.07518 [cs, stat]. Mar. 2019.

[7] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. 2020. arXiv: 2006.11239 [cs.LG].

[8] P. Dhariwal and A. Nichol. Diffusion Models Beat GANs on Image Synthesis. 2021. arXiv: 2105.05233 [cs.LG].

[9] Z. Kong et al. DiffWave: A Versatile Diffusion Model for Audio Synthesis. 2021. arXiv: 2009.09761 [eess.AS].

[10] Z. Chang, G. A. Koulieris, and H. P. H. Shum. On the Design Fundamentals of Diffusion Models: A Survey. 2023. arXiv: 2306.04542 [cs.LG].

[11] A. Broder et al. “Graph structure in the web”. In: Computer networks 33.1-6 (2000), pp. 309–320.

[12] J. Ugander et al. The Anatomy of the Facebook Social Graph. 2011. arXiv: 1111.4503 [cs.SI].

[13] R. C. Thomson and D. E. Richardson. “A graph theory approach to road network generalisation”. In: Proceeding of the 17th international cartographic conference. 1995, pp. 1871–1880.

[14] S. Kearnes et al. “Molecular graph convolutions: moving beyond fingerprints”. In: Journal of computer-aided molecular design 30 (2016), pp. 595–608.

[15] A.-L. Barabasi. “The New Science of Networks”. In: J. Artificial Societies and Social Simulation 6 (Mar. 2003). DOI: 10.2307/20033300.

[16] D. J. Watts and S. H. Strogatz. “Collective dynamics of ‘small-world’ networks”. en. In: 393 (1998).

[17] P. Resnick et al. “GroupLens: an open architecture for collaborative filtering of netnews”. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work. CSCW ’94. New York, NY, USA: Association for Computing Machinery, Oct. 1994, pp. 175–186. ISBN: 978-0-89791-689-9. DOI: 10.1145/192844.192905.

[18] L. Page et al. “The PageRank Citation Ranking : Bringing Order to the Web”. In: The Web Conference. 1999.

[19] F. Scarselli et al. “The Graph Neural Network Model”. In: IEEE Transactions on Neural Networks 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[20] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: Proceedings of the IEEE 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[21] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. 2017. arXiv: 1609.02907 [cs.LG].

[22] A. Vaswani et al. Attention Is All You Need. 2023. arXiv: 1706.03762 [cs.CL].

[23] P. Veličković et al. Graph Attention Networks. 2018. arXiv: 1710.10903 [stat.ML].

[24] J. Zhou et al. Graph Neural Networks: A Review of Methods and Applications. 2021. arXiv: 1812.08434 [cs.LG].

[25] T. B. Sheridan. “Human–Robot Interaction: Status and Challenges”. In: Human Factors 58.4 (2016). PMID: 27098262, pp. 525–532. DOI: 10.1177/0018720816644364. eprint: https://doi.org/10.1177/0018720816644364.

[26] D. J. Fagnant and K. Kockelman. “Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations”. In: Transportation Research Part A: Policy and Practice 77 (2015), pp. 167–181. ISSN: 0965-8564. DOI: https://doi.org/10.1016/j.tra.2015.04.003.

[27] A. Chehri and H. T. Mouftah. “Autonomous vehicles in the sustainable cities, the beginning of a green adventure”. In: Sustainable Cities and Society 51 (2019), p. 101751.

[28] L. D. Burns. “A vision of our transport future”. In: Nature 497.7448 (2013), pp. 181–182.

[29] F. Altché and A. de La Fortelle. “An LSTM network for highway trajectory prediction”. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). 2017, pp. 353–359. DOI: 10.1109/ITSC.2017.8317913.

[30] J. Wiest et al. “Probabilistic trajectory prediction with Gaussian mixture models”. In: 2012 IEEE Intelligent Vehicles Symposium. 2012, pp. 141–146. DOI: 10.1109/IVS.2012.6232277.

[31] N. Nikhil and B. Tran Morris. “Convolutional Neural Network for Trajectory Prediction”. In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops. Sept. 2018.

[32] T. M. Roddenberry, N. Glaze, and S. Segarra. Principled Simplicial Neural Networks for Trajectory Prediction. arXiv:2102.10058 [cs]. June 2021.

[33] T. B. Brown et al. Language Models are Few-Shot Learners. 2020. arXiv: 2005.14165 [cs.CL].

[34] I. J. Goodfellow et al. Generative Adversarial Networks. 2014. arXiv: 1406.2661 [stat.ML].

[35] A. Brock, J. Donahue, and K. Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. 2019. arXiv: 1809.11096 [cs.LG].

[36] A. van den Oord et al. WaveNet: A Generative Model for Raw Audio. 2016. arXiv: 1609.03499 [cs.SD].

[37] P. Dhariwal et al. Jukebox: A Generative Model for Music. 2020. arXiv: 2005.00341 [eess.AS].

[38] H. Tang et al. Molecular Generative Adversarial Network with Multi-Property Optimization. 2024. arXiv: 2404.00081 [q-bio.BM].

[39] Z. Zhang, M. Li, and J. Yu. “On the convergence and mode collapse of GAN”. In: SIGGRAPH Asia 2018 Technical Briefs. SA ’18. Tokyo, Japan: Association for Computing Machinery, 2018. ISBN: 9781450360623. DOI: 10.1145/3283254.3283282.

[40] B. Li et al. “Multitask Non-Autoregressive Model for Human Motion Prediction”. In: IEEE Transactions on Image Processing 30 (2021), pp. 2562–2574. DOI: 10.1109/TIP.2020.3038362.

[41] Y. Song and S. Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. 2020. arXiv: 1907.05600 [cs.LG].

[42] S. Gong et al. DiffuSeq: Sequence to Sequence Text Generation with Diffusion Models. 2023. arXiv: 2210.08933 [cs.CL].

[43] X. L. Li et al. Diffusion-LM Improves Controllable Text Generation. 2022. arXiv: 2205.14217 [cs.CL].

[44] C. Vignac et al. DiGress: Discrete Denoising diffusion for graph generation. 2023. arXiv: 2209.14734 [cs.LG].

[45] C. Jarzynski. "Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach". In: Physical Review E 56.5 (Nov. 1997), pp. 5018–5035. ISSN: 1095-3787. DOI: 10.1103/physreve.56.5018.

[46] J. Sohl-Dickstein et al. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. 2015. arXiv: 1503.03585 [cs.LG].

[47] A. Nichol and P. Dhariwal. Improved Denoising Diffusion Probabilistic Models. 2021. arXiv: 2102.09672 [cs.LG].

[48] L. Weng. "What are diffusion models?" In: lilianweng.github.io (July 2021).

[49] E. Hoogeboom et al. Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions. 2021. arXiv: 2102.05379 [stat.ML].

[50] J. Austin et al. Structured Denoising Diffusion Models in Discrete State-Spaces. 2023. arXiv: 2107.03006 [cs.LG].

[51] Y. LeCun, C. Cortes, and C. Burges. "MNIST handwritten digit database". In: ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2 (2010).

[52] T. Salzmann et al. Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data. 2021. arXiv: 2001.03093 [cs.RO].

[53] M. T. Schaub et al. Signal processing on simplicial complexes. 2022. arXiv: 2106.07471 [eess.SP].

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

[55] A. Krizhevsky, V. Nair, and G. Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: ().

[56] C. Niu et al. Permutation Invariant Graph Generation via Score-Based Generative Modeling. 2020. arXiv: 2003.00638 [cs.LG].

[57] J. Jo, S. Lee, and S. J. Hwang. "Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations". In: International Conference on Machine Learning. 2022.

[58] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., 2019, pp. 8024–8035.

[59] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. 2017. arXiv: 1412.6980 [cs.LG].

[60] J. Yuan et al. "T-drive: Driving directions based on taxi trajectories". In: Nov. 2010, pp. 99–108. DOI: 10.1145/1869790.1869807.

[61] Y. Zheng et al. Geolife GPS trajectory dataset - User Guide. Geolife GPS trajectories 1.1. July 2011.

[62] N. Geroliminis and E. M. Barmpounakis. "pNEUMA dataset". en. In: (2024). Publisher: Zenodo. DOI: 10.5281/zenodo.10491409.

[63] E. Barmpounakis and N. Geroliminis. “On the new era of urban traffic monitoring with massive drone data: The pNEUMA large-scale field experiment”. In: Transportation Research Part C: Emerging Technologies 111 (2020), pp. 50–71. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2019.11.023.

[64] P. Newson and J. Krumm. “Hidden Markov map matching through noise and sparseness”. en. In: Seattle Washington: ACM, Nov. 2009, pp. 336–343. ISBN: 978-1-60558-649-6. DOI: 10.1145/1653771.1653818.

[65] J. Ho and T. Salimans. Classifier-Free Diffusion Guidance. 2022. arXiv: 2207.12598 [cs.LG].

[66] J. Ho et al. Video Diffusion Models. 2022. arXiv: 2204.03458 [cs.CV].

[67] J. Song, C. Meng, and S. Ermon. Denoising Diffusion Implicit Models. 2022. arXiv: 2010.02502 [cs.LG].

[68] S. Geisler et al. “Spatio-Spectral Graph Neural Networks”. In: arXiv preprint arXiv:2405.19121 (2024).