A Compositional Approach to Generative Modeling of Network Paths

1st Yuanbo Tang, 2nd Naifan Zhang, 3rd Yang Li
Tsinghua Shenzhen International Graduation School, Tsinghua University, ShenZhen, China
{tyb22, znf24}@mails.tsinghua.edu.cn, yangli@sz.tsinghua.edu.cn

***Abstract*—Path generation on network is a crucial research field across various domains, including traffic planning and management, routing optimization in communication networks, and relationship analysis in social networks. Although many studies have used deep learning or generative AI methods to model paths and have achieved promising results, the robustness and interpretability of such models are largely unexplored. This limits the application of path generation algorithms on noisy real-world data and their trustworthiness in downstream tasks. To address this issue, we leverage the insight that it's easy to learn a complex path by decomposing it into representative subpaths, each carrying local routing information. Building on this idea, we propose PathletVAE, a compositional generative model that assembles atomic subpaths from a jointly optimized dictionary. The model, based on a variational auto-encoder framework with a linear decoder, is easy to implement in different scenarios. Moreover, the incorporation of sparse dictionary representation enhances the model's robustness under noise and offers explainability through the statistics and semantics of dictionary subpaths. Our method has demonstrated its effectiveness in multiple scenarios. For urban taxi trajectory generation, it beats strong baselines by 35.4% and 26.3% in terms of the alignment between generated data and the original data on two real-world path datasets. Moreover, we show its novel application scenario in Mixture-of-Experts network pruning, with results that not only significantly outperform baseline method but also demonstrate flexibility in manipulating capabilities across different domains.**

***Index Terms*—compositional learning, pathlet, paths on network, variational autoencoder, model compression**

I. Introduction

Paths on networks are a fundamental data type pervasive across physical and digital domains. They appear in vehicle movements on road networks [28], data packet routing in communication systems [23], and in specific computational contexts such as expert activation paths in Mixture-of-Experts network. Recently, generative models have become a mainstream and advantageous method for the effective modeling of path data [30]. These models work by learning the patterns and distributions inherent in the data, enabling them to perform tasks such as data augmentation and manipulation. For instance, in privacy-preserving path modeling on urban road network, path generation models play a significant role in this context as they are capable of generating synthetic paths that mimic the patterns of real-world datasets [21]. Controlled path generation is also useful for downstream tasks such as routing optimization in communication network [23, 2] and travel route planning [26].

Various methods have been proposed for generating paths on a network. In these approaches, paths are either generated recursively using sequence models such as LSTM [33] or Transformer [29], or as whole entities through generative models, such as diffusion on graph neural networks [10] and masked autoencoders [38]. However, these methods primarily focus on how well the generated paths match the distribution of the training data and their performance in downstream tasks. The generation process, however, lacks interpretability due to the non-linearity of deep networks. It is also challenging to explain the generated paths in terms of their semantic meaning, as the learned representations are dense and not easily interpretable. Few path generation studies address the challenges of noise and missing data in real-world datasets. For instance, traffic path data may suffer from issues like hardware instabilities, environmental interference, signal obstruction, low-frequency sampling, and privacy-related data omissions. These problems can introduce sampling bias, hindering model generalization.

To address the aforementioned challenges, we introduce the concept of compositional learning to the generative modeling of paths, whose basic idea is illustrated in Figure 1. This idea is inspired by a common observation in path datasets: there are numerous shared subpaths that act as fundamental components of paths, akin to how words are the building blocks of a corpus. The original path data can be decomposed into basic units to form a dictionary, from which new paths can be generated by selecting and combining elements. An ensuing question is: what is the optimal decomposition of the path into atomic parts? This question was first studied in trajectory compression. Analogous to the principle of Minimum Description Length (MDL), [7] proposed to reconstruct the dataset using the smallest dictionary and the fewest parts for each path, and refer to the dictionary elements as "pathlets". Building on this idea, we propose a deep generative model that models complex paths via the sparse representation of pathlets. This sparse representation not only enhances the robustness of the model [34, 32], but also makes the path generation process explicit, allowing paths to be explained by the semantics of their containing pathlets.

Our generative framework, named PathletVAE, models the path generation process using a Variational Autoencoder (VAE) component and a linear decoder component. The VAE models the distribution of binary representation vectors, indicating whether a pathlet is used. While the linear decoder con-

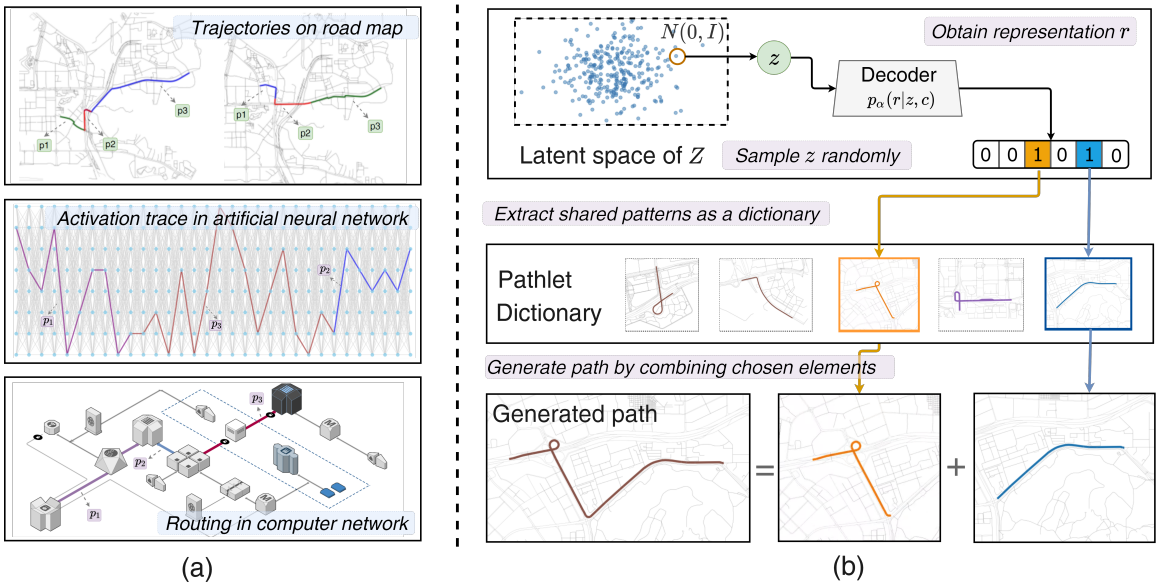

Fig. 1. The left side shows that in different application scenarios, paths on networks can often be decomposed into combinations of multiple basic units. The right side takes urban road network scenario as an example to present the fundamental idea of our method: by learning shared patterns in the data to obtain a dictionary and then generating by recombining elements from the dictionary, the inherent sparsity makes the algorithm more robust and understandable.

verts these vectors into full paths. During the training process, the pathlet dictionary and the VAE model for representation vectors are jointly learned. Several regularization constraints are proposed to ensure the optimality of the learned dictionary.

We applied the proposed method to two specific scenarios. The first scenario involves paths in city road networks. We evaluated our method on two real-world datasets and simulated scenarios under varying noise levels. The experimental results show that the data generated by our method exhibits significantly higher similarity to real data compared to other methods. Furthermore, we validated the effectiveness of our method in downstream tasks such as conditional generation. The second scenario focuses on token routing in Mixture of Experts (MoE) networks. By applying our path generative model, we discovered inherent patterns in token routing related to the tokens and expert weights. Learning these patterns enhanced the interpretability of MoE models, enabled more effective model compression, and allowed us to adjust the model's capabilities in specific domains.

Overall, our main contributions can be summarized as follows:

- We propose a path generation framework that combines the strong distribution fitting capabilities of deep learning models with the robustness of sparse combinatorial learning, leveraging the advantages of both approaches. Additionally, we comprehensively evaluate the performance of the method on real-world datasets.
- Experimental results in the urban road network scenarios indicate that our method achieves better Jensen-Shannon Divergence (JSD) scores than previous approaches. Moreover, we examined the algorithm's utility in downstream tasks, visualized important pathlets, and discussed the interpretability of the method.
- Using our path generative model, we uncover fundamental patterns in token routing that are influenced by both the tokens and the weights assigned to experts. Understanding these patterns not only enhances the interpretability of MoE models but also facilitates effective model compression and enables the customization of the model's capabilities for specific domains.

II. Related Work

A. Path generative model on networks.

Path generation on networks is a fundamental problem in many domains, where the goal is to generate one or more paths on a given network. The generated paths should often meet objectives such as minimizing the distance of distribution between true data and generated data [30], while adhering to constraints like feasibility (e.g., connectivity), and diversity (e.g., generating multiple distinct paths). For example, in traffic simulation, it is used to create path datasets that replicate real-world traffic patterns, enabling urban mobility analysis and transportation optimization [8, 20]. In path prediction [19], it involves forecasting future paths based on initial conditions, which is critical for autonomous driving and pedestrian safety.

Path generation techniques can be grouped into two broad categories: traditional methods and data-driven methods. Classical methods for path generation on networks are primarily based on search and optimization techniques. Search-based methods include well-known algorithms such as Dijkstra's algorithm [11] for single-source shortest paths, A-star [16] for heuristic-based pathfinding, and Bellman-Ford [5] for handling networks with negative edge weights. These methods are robust and theoretically grounded, making them suitable for

generating optimal paths under static conditions. Optimization-based approaches, such as dynamic programming and minimum spanning tree algorithms, have also been widely used for solving global optimization problems on networks. While these methods are effective for small-scale or static networks, they face significant limitations in dynamic or large-scale environments. Specifically, their computational complexity grows rapidly with the size of the network, and they often struggle to adapt to real-time changes or generate diverse paths.

To address the limitations of classical methods, data-driven approaches have emerged as powerful alternatives for path generation on networks. Generative models based on Generative Adversarial Networks [14, 9] , Variational Autoencoders [22] and diffusion architecture [26, 38] have been widely used to generate diverse paths. On the other hand, sequence modeling techniques, such as Long Short-Term Memory networks [35, 33] and Transformers [29], further enhance path generation by capturing temporal and global dependencies. LSTMs are effective for sequential path data, while Transformers, with their self-attention mechanisms, excel at modeling global relationships and generating complex paths. Additionally, some methods utilize graph structures for modeling, such as Graph Neural Networks and Graph Attention Networks [10], which aim to capture the graphical relationships and structural information present in the data. However, these data-driven methods, often seen as "black-box" models, lack transparency and interpretability in the path generation process [13]. This limits their application in critical tasks like traffic navigation and robotics, where understanding the rationale behind generated paths is essential, especially when paths deviate from expectations.

B. Compositional learning.

Compositional learning has emerged as a versatile paradigm for addressing complex problems and processing intricate data structures. The core principle of compositional learning is to decompose a problem or dataset into smaller, manageable components and then integrate these components to achieve a globally optimal solution or improve the model's performance on unseen data distributions [27, 12]. These advantages have therefore attracted the attention of researchers in fields such as computer vision [17, 18] and reinforcement learning [15].

However, selecting appropriate combination methods is complex and requires careful alignment with task objectives and data characteristics. In the field of compositional learning for path data, early work includes the introduction of the pathlet learning, initially proposed by [7] and further developed in subsequent studies [3]. For example, [28] used a randomized rounding algorithm to further optimize the dictionary and explored the effectiveness of path representation across multiple downstream tasks. During the decomposition stage, previous methods employed selection-based strategies, which were time-consuming. Our approach, based on learning rather than selection, significantly improves efficiency. Additionally, our method not only focuses on decomposing the dataset to learn the pathlet dictionary but also can generate new paths by combining elements from the learned dictionary.

III. Preliminary

A. Terminology.

Given a path dataset X on a network that can be formed as a directed graph $G = (E, V)$, a path $x \in X$ is defined as a sequence of edges e on G, denoted by $x = \{e_1, e_2, ..., e_n\}$. Similarly, a pathlet p also refers to a sequence of edges e on G and all pathlets together form a dictionary.

B. Problem statement (Path Generation).

Given a path dataset X mapped on a road network G, the goal of the path generation task is to learn a generative model M using X, which can synthesize path data Y that satisfies the following:

- *Authenticity*: The generated paths Y retain the spatiotemporal properties and distribution of the original path dataset X.
- *Utility*: The generated paths and the generative model M can benefit various downstream tasks.

Besides the above two criteria that are common to any generative models, an ideal model M should also have:

- *Robustness and Interpretability*: The algorithm can effectively learn data distribution from noisy or incomplete dataset and its internal structure, behavior, and outcomes are easy to understand.

IV. Methodology

In this section, we first introduce the probabilistic graphical model used for modeling the path generation process. Subsequently, we provide detailed descriptions of the design of the objective function and key components within the framework. Finally, we describe how the generative model is applied in downstream tasks.

A. Graphical model.

As previously described, based on the assumption that common shared mobility patterns exist within the path dataset, we can consider this set of motion patterns as a dictionary, and each path sample can be viewed as a combination of elements from this dictionary. Figure 2 provides an overview of the graphical model: On the one hand, data sample x is obtained by linearly combining elements from dictionary D using representation vector r. On the other hand, the probability distribution of the representation vector r is learned by transforming from a standard normal distribution. Besides, the characteristics of the x can be further controlled by incorporating conditional information such as departure time into the generation of r.

In this setting, a complete generative process consists of three steps: First, a high-dimensional vector z is randomly sampled from the normal distribution. Second, r is obtained through a transform function. Here we use functions $p_\alpha(r|z)$ and $p_\beta(z|r)$ as bridges to transform between r and z respectively where α and β are learnable parameters. Finally, r will

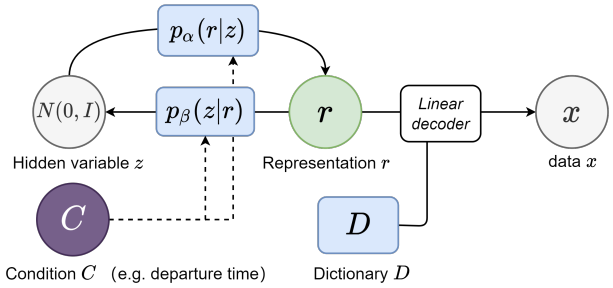

Fig. 2. Illustration of the graphical model which describes the generative process of path.

be projected into a new space using D to generate the sample x ($x = Dr$). The design principle of reconstructing paths by combining semantically meaningful path segments enhances the transparency of the model's behavior, thereby providing a certain level of interpretability.

B. Vectorization and Linear decoder.

From the perspective of vectors and matrices, each path with variable-length can be transformed into a binary vector x with the size of $|E| * 1$, where $|E|$ refers to the number of edges in G and each element of x corresponds to an edge in this network. Each entry $x_i = 1$ if e_i is covered by this path, and $x_i = 0$ otherwise. Similarly, the binary vector r is used to store which elements from the dictionary D are selected. The dictionary is a collection containing pathlets, which can also be standardized into matrix D. The dictionary D serves the role of spatial transformation: The path vector x can be obtained by linearly decoding the representation vector r using D ($x = Dr$). For multiple paths and their corresponding representation vectors, vector r and vector x naturally transform into matrix R and matrix X, respectively. To ensure the accuracy of reconstruction, it is required that $DR = X$.

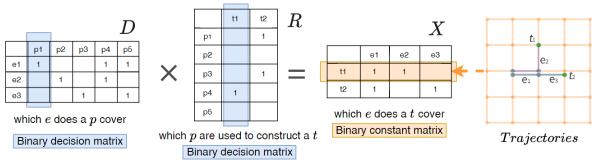

Fig. 3. Illustration of the matrices D, X, R: X is constant matrix generated from path dataset; D refers to dictionary matrix; R is decision matrix where each column corresponds to representation vector.

C. Objective function and Problem formulation.

As illustrated in Figure 2, in order to model the complete path generative process, the objective function should consist of two parts: 1) the VAE component, which aims to model the distribution of the representation vector r, and 2) the dictionary learning component, which encourages the model to learn an efficient and compact dictionary.

1) Generative Model using VAE: In our framework, we adopt a binary version of the VAE architecture proposed by [37]. By employing Bernoulli distribution sampling instead of Gaussian sampling, the generated samples will be inherently binary. As described in Equation below, z is sampled from a normal distribution and used as the input to the neural network. The neural network outputs n and p as the parameters for the Bernoulli distribution, from which the final result is sampled. This architecture is inherently suitable for binary data. $f_{\alpha^n}(\boldsymbol{z})$ and $f_{\alpha^p}(\boldsymbol{z})$ represent neural networks that generate n and p respectively. In our proposed framework, we implement these using fully connected neural networks.

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$$

$$\boldsymbol{n} = \exp(f_{\alpha^n}(\boldsymbol{z})), \boldsymbol{p} = \text{sigmoid}(f_{\alpha^p}(\boldsymbol{z}))$$

$$\boldsymbol{r} \sim \text{Bernoulli}(\mathbf{1} - \boldsymbol{p^n})$$

The variational lower bound (ELBO) of Binary VAE model can be written as:

$$\mathbb{E}_{\mathrm{q}(\boldsymbol{z}|\boldsymbol{r})}\left[\log \mathrm{p}\left(\boldsymbol{r} \mid \boldsymbol{z}\right)\right] - \mathrm{KL}\left[\mathrm{q}\left(\boldsymbol{z} \mid \boldsymbol{r}\right) \| \mathrm{p}\left(\boldsymbol{z}\right)\right] \quad (1)$$

The first term refers to the reconstruction loss, which measures the discrepancy between the original input r and its reconstruction $\hat{r}$ from the latent space. Based on the definition of Bernoulli distribution, it can be further written as:

$$\sum_j \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{r})}\left[\boldsymbol{r}_j \log(\mathbf{1} - \boldsymbol{p^n})_j + (1 - \boldsymbol{r}_j) \log(\boldsymbol{p^n})_j\right] \quad (2)$$

The second component is the Kullback-Leibler (KL) divergence, denoted as D_{KL}, which calculates the divergence between the learned distribution of latent variables $\mathrm{q}\left(\boldsymbol{z} \mid \boldsymbol{r}\right)$ and a prior distribution $\mathrm{p}\left(\boldsymbol{z}\right)$, generally chosen to be a standard normal distribution $N(0, I)$. This divergence encourages the latent space to follow the predefined structure, which is crucial for generating coherent new data. We employ ELBO as the optimization objective L_{VAE} for the VAE component.

2) Compositional learning: We use a compositional learning based approach to automatically learn the pathlet dictionary and representation from dataset. We adopted the same objective function design as in previous work [28], which contains three parts:

I. The size of the dictionary: A smaller dictionary contains less redundant information and is therefore more desirable. For any row in R, if it contains non-zero elements, it indicates that the corresponding pathlet is used. Therefore, the size of dictionary can be written as $\sum \mathbf{1}\{max(R_{i,:}) > 0\}$. Here, $R_{i,:}$ refers to the i-th row of R. Since R is binary, this is equivalent to summing the infinity norm of each row $\sum ||R_{i,:}||_\infty$.

II. The average number of elements required to reconstruct paths: As we mentioned before, each entry $r_i = 1$ if p_i is selected for generating a path, and $r_i = 0$ otherwise. Therefore, $||r||_1$ refers to the number of elements required to reconstruct a path. Furthermore, the average number of elements required to reconstruct paths in dataset X

can be written using matrix format $||R||_1/N$. Here, N represents the number of samples in the dataset.

III. The average reconstruction error of the path from dataset: For each path x we want to minimize the difference $||x - Dr||_1$ between reconstruction one and original one. Therefore, the average reconstruction error can be written as $||X - DR||_1/N$ for dataset.

To summarize, the overall objective function can be expressed as:

$$L_{\text{dict}} = \lambda_1 \sum ||R_{i,:}||_\infty + \frac{\lambda_2||R||_1 + \lambda_3||X - DR||_1}{N} \quad (3)$$

Here λ_1,λ_2,λ_3 are hyperparameters that control the trade-off between three losses.

3) Problem formulation: The constraints of this problem come from the fact that both the dictionary matrix and the elements of the representation vectors are binary. Therefore, taking into account the two parts of the path generation process, the problem can be formulated as:

$$\min_{\alpha,\beta,D,R} L_{\text{VAE}} + L_{\text{dict}} \quad (4)$$

$$s.t. D_{i,j} \in \{0,1\}, R_{i,j} \in \{0,1\}$$

We have formulated the problem into an optimization problem with binary constraints, whose exact solution is intractable. Therefore, our approach first relaxes the original problem into a convex optimization problem, followed by Bernoulli sampling to obtain the final solution. Additionally, because traditional VAE assumes variables are continuous, we employ a binary version of the VAE to model the distribution of binary representation vector r.

D. Training details

To solve the aforementioned optimization problem, we have designed a two-step algorithm. To be specific, we first relax the integer optimization constraints to interval constraints, then obtain a decimal solution using gradient descent, and finally convert it into a binary solution through probabilistic rounding method.

Details are provided in Algorithm 1: Given the matrix X composed of path datasets, we first initialize the dictionary D as a matrix identical to matrix X. The representation vector matrix R is initialized as an identity matrix. Then, we calculate the gradients for α, β, R, D and update them repeatedly until the objective function converges. The representation matrix and the dictionary matrix are then converted into the desired results using a rounding algorithm.

Algorithm 1 Training Procedure

Input: X: path matrix; ϵ, θ: hyperparameters; L: loss function; δ: learning rate;

Output: Solution R^r and D^r

Step1, we compute the fractional solution R^* and D^* using gradient descend.
Initial $R_0 = I$; initial $D_0 = X$, initial $\Delta = +\infty$
Randomly initialize the parameters α, β of the VAE model.
while $\Delta < \epsilon$ **do**
  Compute loss: $L_k = L(\alpha_k, \beta_k, R_k, D_k)$ by Eq 3-5
  Compute gradient and update the parameters

$$R_{k+1} = R_k - \delta \nabla_{R_k} L_k \quad D_{k+1} = D_k - \delta \nabla_{D_k} L_k$$
$$\alpha_{k+1} = \alpha_k - \delta \nabla_{\alpha_k} L_k \quad \beta_{k+1} = \beta_k - \delta \nabla_{\beta_k} L_k$$

  Clip the result to ensure that every element in R_k and D_k has a value between 0 and 1
  $\Delta = |L_k - L_{k-1}|$
end while
#Step2, round solution R^r and D^r based on R^*, D^*.
Sample R with $P(R_{i,j} = 1) = min(1, \theta R^*_{i,j})$
Sample D with $P(D_{i,j} = 1) = min(1, \theta D^*_{i,j})$

E. Post-processing for Generated paths

Our approach essentially involves learning the distribution of a given dataset and then sampling from it to generate new paths. Therefore, an unavoidable issue is that the connectivity of the generated paths is not strictly guaranteed. To address this problem, we propose a greedy algorithm that uses road network information to post-process the generated paths to ensure connectivity.

The detailed steps are explained in Algorithm 2: Given a network G and a generated path on it, assume it is not a completely connected path but a set of separate segments. Our goal is to complete these segments into a full path. The basic idea of the algorithm is: For a set of separate segments, each time find the two segments with the smallest connection cost (The connection cost is defined as the length of the shortest path in the G that connects these two segments), merge them into a new segment, and repeat this process until a complete path is obtained.

F. Downstream tasks

Once the generative model has been obtained, it captures the distribution of the data. This section will demonstrate how our proposed generative model can be conveniently applied to various downstream tasks. Due to space limitations, we will use two applications as examples: conditional generation and generation with noisy paths.

1) Conditional path generation: Compared to unconditional generation, conditional generation allows the specification of certain conditions or attributes, thereby generating data that meets specific requirements. This control capability is particularly useful when data with specific characteristics is needed.

This capability can be achieved in our framework with a simple modification: replacing the VAE with a CVAE. The figure below illustrates a specific scenario where, for example, the first half and departure time of the path are provided as conditions along with the input, and the model generates the

Algorithm 2 Post-processing Procedure

Require: List of path fragments `segments`; Network G; Cost function `cost(path)`
Ensure: Merged path `merged_path`

while Number of fragments in `segments` > 1 **do**
  Initialize minimum cost `min_cost` $\leftarrow \infty$
  Initialize best fragment pair `best_pair` $\leftarrow$ `None`
  for Each pair of fragments (s_i, s_j) in `segments`, where $i \neq j$ **do**
    Compute the minimum cost to connect s_i and s_j:
    Use the shortest path algorithm to find the shortest path between the start or end points of s_i and s_j
    Obtain the minimum cost `cost`
    if `cost` $<$ `min_cost` **then**
      Update `min_cost` $\leftarrow$ `cost`
      Update `best_pair` $\leftarrow (s_i, s_j)$
      Record the shortest path
    end if
  end for
  Merge `best_pair` using shortest path
  Add `new_segment` to `segments`
  Remove s_i and s_j from `segments`
end while
return `merged_path`

complete path. For the encoding method, we map the departure time to 24 hourly intervals and use one-hot encoding. In this case, the framework can perform the path prediction task.

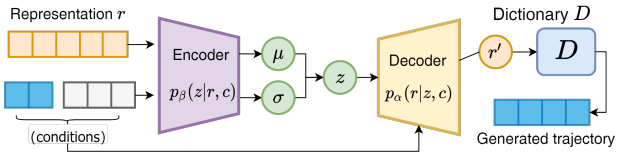

Fig. 4. Architecture of CVAE.

2) Generation with noisy paths: As mentioned earlier, our framework incorporates the concept of pathlet dictionary and the sparse combination of elements within the dictionary for data generation. This design not only provides interpretability but also introduces sparsity, making the model inherently robust to noise and suitable for data denoising tasks. Specifically, after completing the model training, the dictionary D is fixed as one of the model parameters. For a new noisy data x, denoising consists of two steps:

(1) Map x to a new representation space using D. To be specific, representation vector r is obtained by solving:

$$\min_{r_i \in \{0,1\}} \quad \lambda_2 ||r||_1 + \lambda_3 ||x - Dr||_1 \tag{5}$$

This problem can be viewed as a simplified version of the original problem because the dictionary is fixed at this moment. We solve it using the same strategy described before: first get the optimal fractional solution r^* using gradient descent and then round it to get the final binary solution r^-.

(2) Once we get r^*, then the denoised data $\hat{x}$ can be obtained by multiplying D and r^-.

V. Experiment in Urban Road Networks

In this section, we conduct extensive experiments to comprehensively evaluate the proposed framework from four aspects: effectiveness, efficiency, robustness, and interpretability, and compare it with previous methods. We first briefly introduce the datasets, experimental setup, and evaluation protocol. After that, we will attempt to answer the following research questions:

RQ1: Is our proposed generative model capable of learning the true data distribution and generating datasets that closely approximate the true distribution?
RQ2: Can our algorithm be used on noisy or incomplete data? To what extent of noise can it operate normally?
RQ3: Can the model be conveniently applied to different downstream tasks while achieving good performance?
RQ4: What does the representation vector generated signify? Can it be visualized or understood in an intuitive way?
RQ5: How efficient is the proposed method?

A. Experiment setup

1) Datasets: To evaluate the proposed method, we utilized two real-world taxi datasets. Key statistical data are shown in Table I. Shenzhen dataset contains approximately 510k dense paths generated by 14k taxi cabs in Shenzhen, China [36]. Porto dataset describes paths performed by 442 taxis running in the city of Porto, Portugal [25]. Each taxi reports its location every 15s. This dataset is used for the path Prediction Challenge@ ECML/PKDD 2015.

TABLE I
Statistic of path datasets.

Name	#paths	Avg.#Points	Avg.time gap
Porto	1.2M	60.20	15.00Sec.
Shenzhen	510K	43.96	21.66Sec.

2) Data Preprocess: For these two datasets, we remove paths with less than 20 GPS sample points and use the method proposed in [24] to convert path to a series of edges on roadmap. Then the matrices D and X are generated as described in Methodology.

3) Noise Simulation: To validate the robustness of our method, we chose to manually introduce varying levels of noise into the original dataset. Specifically, we added a node named "unknown" to the map, which is connected to all other nodes. For each path x in the dataset, the probability that each node is randomly replaced by the "unknown" node follows a Bernoulli distribution with parameter p_{noise}. By adjusting p_{noise}, we can simulate noisy datasets at different noise levels.

4) Other Settings: We randomly sampled 30% paths as our test dataset, and use the rest 70% as the training dataset. Our method is implemented in Python and trained using an Nvidia A40 GPU. All experiments are run on the Ubuntu 20.04 operating system with an Intel Xeon Gold 6330 CPU.