SKETCHGNN: GENERATING CAD SKETCHES AS GRAPHS

Sathvik Reddy Chereddy

A THESIS

Presented to the Faculty of Miami University in partial
fulfillment of the requirements
for the degree of

Master of Science

Department of Computer Science & Software Engineering

The Graduate School
Miami University
Oxford, Ohio

2025

Dr. John Femiani, Advisor
Dr. Karen Davis, Reader
Dr. Khodakhast Bibak, Reader

ABSTRACT

Computer-aided design (CAD) is widely used for 3D modeling in many technical fields, yet the creation of 2D sketches remains a manual step in typical CAD modeling workflows. Automatically generating 2D sketches can help users in CAD modeling by reducing their workload and by streamlining the design process. While sketches inherently possess a graph structure, with geometric primitives as nodes and constraints as edges, the application of graph neural networks (GNNs) to this domain remains relatively unexplored. To address this gap, we introduce SketchGNN, a graph diffusion model designed to generate CAD sketches using a joint continuous-discrete diffusion process. Our approach includes a novel discrete diffusion technique, wherein Gaussian-perturbed logits are projected onto the probability simplex via a softmax transformation. This enables our model to express uncertainty in the discrete diffusion process unlike traditional methods. We demonstrate that SketchGNN achieves state-of-the-art performance, reducing the Fréchet Inception Distance (FID) from 16.04 to 7.80 and the negative log-likelihood (NLL) from 84.8 to 81.33.

Table of Contents

List of Tables

List of Figures

Dedication

I dedicate this thesis to my family: my mother Madhuri Latha Chereddy, my father Bhaskara Reddy Chereddy, and my sister Sanvika Reddy Chereddy. I would not have been able to come as far as I have in my academic endeavors without their unconditional and unwavering support. This thesis is the culmination of not just mine, but equally their effort as well. I wish my sister success on her academic career, may she go even farther than I could hope to. I want to thank my mother who truly has been my biggest cheerleader. I am indebted to her for multiple lifetimes over, and I truly can't thank her enough. Lastly, I want to pay homage to my dad who has always been the best coach to me, I only wish I'll be able to repay even a tenth of all he's given to me.

I would like to express my heartfelt gratitude to my advisor, Dr. John Femiani, for taking me under his wing. I also wish to thank my committee members: Dr. Karen Davis, whose database classes remain some of my most vivid memories at Miami; and Dr. Khodkhast Bibak, whose disinterest in movies is only matched by his contagious enthusiasm for cryptography. I am also grateful to Dr. Zulal Sevkli for her zeal in guiding my personal C++ learning and projects, as well as her support of my master's application. I thank Dr. Asaad Althoubi for carrying me through Systems 2, and for his support of my master's application.

I am also indebted to Dr. Daniela Inclezan, who advised me through both my bachelor's and master's degrees, and whose classes were always a joy (especially logic programming). I also want to thank Dr. Eric Bachmann for encouraging me to shoot for the moon in game engine design with ocean simulations and bone animations. I'd like to thank Dr. Kurt Johnson for teaching me web services, and for his support of my master's application. I am grateful to Professor Norm Krumpe for his encouragement of my emulator project and whose classes were always incredibly fun (especially graphics). I would also like to thank Dr. Xianglong Feng whose machine learning course was incredibly helpful in this thesis, along with Dr. Scott Campbell whose courses on APIs/networking played a large part in completing this thesis.

I'd also like to thank Dr. Robert Davis for helping me finally understand probability. Similarly, I'd like to thank Dr. Benjamin Sutcliffe whose Russian folklore class was delightful and plain fun. I'm grateful to Dr. Scott Shreve, Dr. Sara Austin, Dr. Anthony Rapp, and Dr. Alim Sukhtayev for their engaging classes despite the COVID lockdown. Lastly, I wish to thank Dr. Ann Wainscott for her fantastic comparative politics class.

Chapter 1

Introduction

Computer-Aided Design (CAD) is the use of software tools for drafting and editing both 3D models and blueprint documents, and it has become the standard for design in many fields of engineering such as: architecture, civil engineering, mechatronics, and manufacturing [1, 2, 3]. A number of CAD tools have gained popularity among designers, engineers, and users respectively, but in recent years a few tools such as AutoCAD, SolidWorks, and Onshape have emerged as the most widely used CAD software [4, 5, 6]. These tools have significantly enhanced the precision of designs, while also improving the flexibility and speed of the design process. However, despite these benefits, users are still required to meticulously specify and fine-tune design/sketch details to achieve optimal results [2, 7]. This issue has been recognized by others, and several solutions have been proposed to alleviate the issue, with some solutions based on rule-based inference [8, 7] and other solutions based on generative neural networks [3, 2, 9]. Neural network (NN) based solutions have recently outpaced rule-based inference solutions and have become the state-of-the-art (SOA) in terms of performance [2, 3], but are still limited to low fidelity sketches [3]. This work builds off of previous works, particularly Vitruvion [10] and SketchGen [3] by utilizing graph neural networks (GNNs) to generate graph representations of CAD sketches for more diverse and higher fidelity CAD designs.

1.1 Motivation

The primary functionality of CAD software suites is 3D modeling. The 3D models that users create are used for a variety of purposes such as machining/manufacturing parts, designing video game assets, and so forth [1, 11]. CAD tools are particularly popular in engineering disciplines due to their simplicity, precision, and interoperability with many fabrication procedures [3]. The common workflow in CAD modeling is to design 2D sketches that undergo operations such as extrusion, revolution, and lofting to create complex 3D geometry. 2D CAD sketches are blueprints composed of a set of geometric primitives (i.e., lines, arcs, points) and a set of geometric constraints (i.e., coincidence, orthogonality, parallelity) on those primitives.

Although modern CAD software has sped up the design iteration cycle for modeling, it is still required of users to tediously and meticulously work with low level details when designing sketches, which drastically slows down the design iteration by forcing users to dedicate more time to drudgery. Generating CAD sketches can assist users by taking care of tedious design tasks, freeing up their time and mental energy to focus on higher-level constructs. By automating menial design tasks, users can have their workload reduced/streamlined thus assisting their productivity and efficiency. Moreover, user creativity may be boosted by suggesting novel variations of CAD designs; being exposed to various generated CAD designs may inspire users to explore new design possibilities or to draft more diverse ideas. Lastly, multiple CAD designs can be generated simultaneously based on user specifications, allowing users to quickly test several prototypes. This functionality can be particularly useful for engineers and designers who need to iterate through different design options and test their feasibility. By enabling users to explore and test different design options more quickly, CAD sketch generation can accelerate the design process, yielding faster time-to-market for new products and structures.

Other researchers have also identified the design iteration cycle of CAD sketches as a significant bottleneck in productivity, and have proposed several solutions to alleviate this issue which can be bifurcated to rule-based and neural network (NN) based solutions. While rule-based solutions have been proposed, these methods have limitations due to the difficulty of explicitly coding all possible rules for the sheer number of potential user designs. Additionally, these approaches may not generalize well to new design scenarios or accommodate the diversity of design inputs [2]. In contrast, NN solutions have shown promise by capturing relationships between different components of a design, and generating novel designs that satisfy specified constraints far better than existing rule-based methods. Furthermore, using NNs removes the need to explicitly define rules, allowing for more flexible and adaptable solutions that can generalize to different design scenarios. As a result, NNs are now the SOA in the domain of CAD sketch generation.

However, current NN solutions are not robust, accurate, nor reliable enough to facilitate all the aforementioned benefits to CAD design. Currently, generative CAD NNs require an external constraint solver to fix the generated sketch [2], and are limited to simple sketches [3, 10]. Previous research on NNs has focused on utilizing Transformer [2, 3] and Recurrent neural network (RNN) [9] architectures to generate CAD sketches as token sequences. However, to the author's knowledge, no previous research has explored the use of Graph Neural Networks (GNNs) for CAD generation. GNNs have demonstrated an ability to generate novel, diverse, and realistic graphs in a multitude of domains, for instance graph diffusion models are the SOA in molecule generation [12, 13]. This work explores a novel application of GNNs in the domain of CAD sketch generation, motivated by their success in other graph generation tasks.

1.2 Contributions

This work provides, to the author's knowledge, the following contributions:

1. The first application of GNNs to the task of CAD sketch generation.
2. The first application of diffusion to the task of CAD sketch generation.
3. A novel discrete diffusion strategy employing the Gaussian-Softmax distribution.

Chapter 2

Background & Related Work

CAD originated in the 1960s from the aircraft industry, and later cemented itself in various engineering disciplines by the 1980s [1]. Thanks to the increased precision, flexibility, and speed offered by CAD tools such as AutoCAD and Solidworks, CAD design has become the standard for modern manufacturing and engineering [2, 1]. Similar to CAD, modern NNs originated in the 1960s from the seminal work of Rosenblatt, and the field of Machine Learning (ML) was later established by Kohonen in the 1980s [14, 15, 16]. NNs have demonstrated an ability to understand and predict complex phenomenae, and in recent years have revolutionized domains such as image synthesis and natural language processing.

2.1 Computer-Aided Design (CAD)

The primary purpose of CAD tools is the design of 3D models, however the typical workflow for users in modeling any complex 3D geometry is by first designing 2D CAD sketches and then performing operations such as extrusion, lofting, and revolution to extend the 2D planar sketches into 3D volumes. The volumes are then aggregated by the user to form the final model [17]. Extrusion refers to extending a 2D shape into a prism, revolution rotates a 2D shape around a specified axis to trace out a volume, and the loft operation creates a 3D shape by using specified 2D shapes as cross sections. A visualization of the extrude and loft operations are provided in Figure 2.1. As a result, the majority of users' effort and time in CAD modelling is dedicated to the construction of 2D sketches.

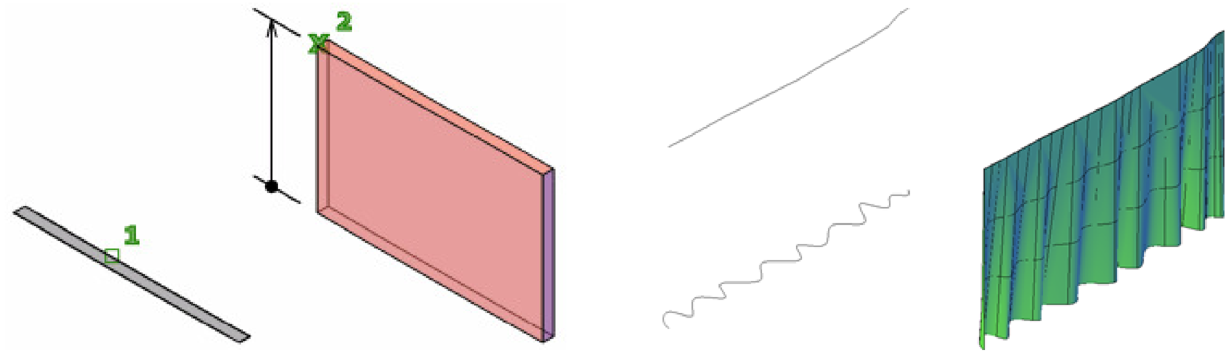


Figure 2.1: A 2D rectangle is extruded into a 3D rectangular prism on the left and a curtain is created from 2 splines by using them as cross sections on the right, created by Autodesk Help. Copyright ©Autodesk, Inc. CC BY-NC-SA 3.0

CAD sketches are simply a collection of geometric primitives and corresponding geometric constraints that express relationships between those primitives. Thus sketches have a natural representation as graphs where the nodes are primitives and the edges are constraints. Figure 2.2 provides a simple sketch and its corresponding graph representation. For brevity, we denote the graph representation of CAD sketches as CAD graphs. Primitives in CAD sketches are often over parameterized due to the ease of performing certain 2D transformations (e.g., rotation, translation) with certain parameters versus others. Table A.1 lists the parameters of each primitive type and their corresponding semantics as used in Onshape. Additionally primitives can be tagged as “constructible” which turns them into construction aids that are not rendered in the final model; judicious use of construction aids along with constraints allows users to precisely codify design intent.

Constraints are user specified relationships between primitives that encode geometric requirements in the final 3D model. Such relationships simplify and streamline the design iteration cycle by allowing users to edit individual primitives while preserving geometric requirements. Constraints can be specified on primitives as a whole or individual sub-primitives, such constraints are referred to as sub-primitive constraints. Sub-primitives are simply the start, center, and end points of arcs and lines. Tables A.2 & A.3 list the constraints relevant to this work and their geometric semantics as provided in Onshape. Design intent is primarily conveyed through the constraints that users specify, and as a result require a good deal of attention from users.

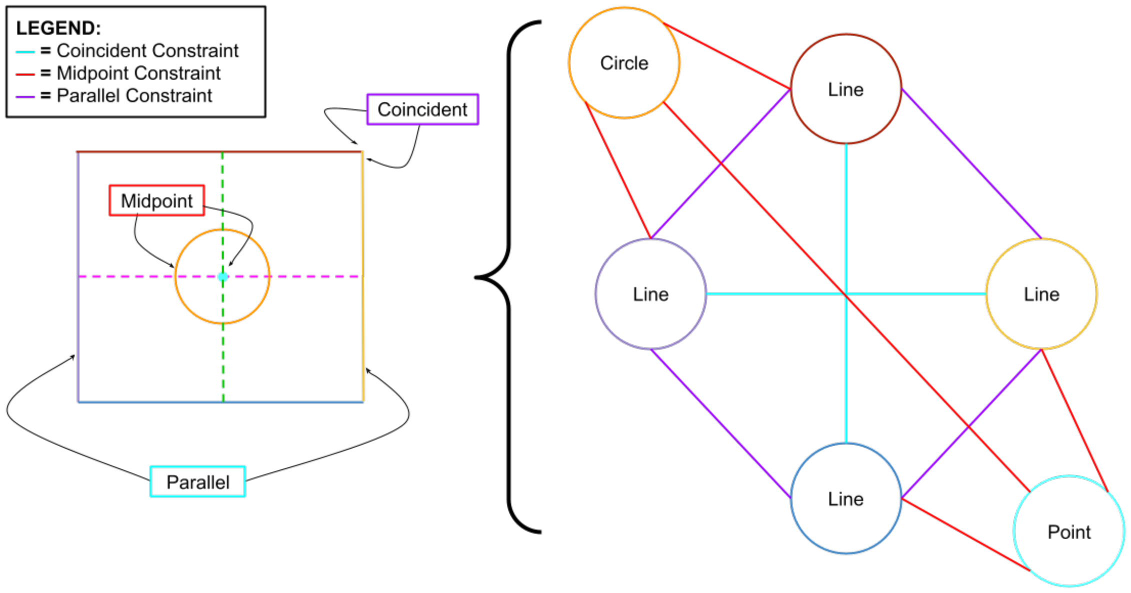


Figure 2.2: A rendering of a CAD sketch where every primitive is assigned a unique color and primitives tagged as construction aids are dotted on the left. The corresponding graph representation is provided on the right where cyan represents coincidence, red a midpoint constraint, and purple a parallel constraint. Construction aids are omitted from the graph for clarity and readability.

The complexity and expressivity of CAD sketches arises from the interaction of primitives and constraints. Often times, as users continue to iterate and amend their designs, constraint conflicts arise where two constraints are mutually incompatible resulting in unsatisfied constraints. Furthermore, due to the sheer expressivity that standard CAD software provide, previously specified geometry may interact in unanticipated ways with newly introduced geometry, as constraints may propagate to other parts of the design in unexpected ways. To alleviate this in some fashion, users typically construct CAD sketches by adding foundational primitives first and building off of them to finish the sketch. As a result, the ordering of primitives encodes very useful semantic information, thus even though all permutations of CAD graphs are equivalent we focus solely on generating the canonical human ordering inline with much of the prior art [9, 3, 10].

2.1.1 Neural Networks

Neural networks are essentially parameterized functions whose parameters are learned through some form of training, almost always being gradient descent based optimization. The functionality of NNs is derived from their ability to approximate any continuous function to any arbitrary degree. This is known as the Universal Approximation Theorem and was proven by Hornik in 1989 [18]. The most basic neural network architecture is known as the Multilayer Perceptron (MLP) or Feed Forward network (FFN), originally introduced by Rosenblatt as an abstraction of biological neural networks [16]. An artificial neuron is just a simple unit of computation that takes a weighted sum of the neurons in the preceding layer with some bias, and uses that as input for a non linear activation function, where the output of the neuron is the result of the activation function. Information, also referred to as signals in ML nomenclature, is passed from the input layer through a series of intermediate layers, also referred to as hidden layers, and then lastly through an output layer [14]. Figure 2.3 demonstrates a simple FFN composed of 4 layers.

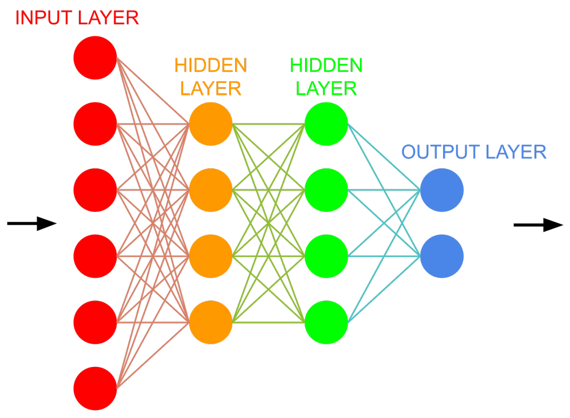


Figure 2.3: A topographical visualization of a 4 layer feed forward network, where every node is a neuron and the weights are represented by edges. The arrows designate the flow of information, typically deeper layers encode higher level information.

We can compactly represent the input of the i^{th} layer as the vector $\vec{x}_{i-1}$ where each layer transforms the input according to $\vec{x}_i = \sigma(W_i\vec{x}_{i-1} + \vec{b}_i)$ where σ is an element-wise nonlinear function, W is a matrix whose entries are parameters/weights to be learned, and $\vec{b}$ is a bias term to be learned. Training a NN is simply determining the optimal values of the weights and biases in each layer for a particular task. The optimal parameters are determined in relation to a loss function, which quantifies the error between the network output and true output. A lower loss means that the network is more apt at producing the correct outputs, and a higher loss means that the network is less adept at producing the correct outputs. The gradient of the loss determines which direction the parameters should be updated to reduce the loss [19]. During training, the NN is fed a large dataset of input data and the loss is calculated, in practice the loss is often not computed using the entire dataset but instead with smaller chunks of the dataset, and gradient descent is applied to reduce the loss. This process is repeated iteratively until the loss is minimized, and the NN is said to have converged [19]. Once the NN is trained, it can be used to make predictions on new data. The most common loss functions are Mean Squared Error (MSE) for regression tasks (continuous variables) which is computed as $\mathcal{L}_{\text{MSE}} = ||\vec{X}_{\text{TRUE}} - \vec{X}_{\text{PRED}}||^2$ and Cross Entropy (CE) for classification tasks (discrete variables/one-hot vectors) which is computed as $\mathcal{L}_{\text{CE}} = -\vec{X}_{\text{TRUE}} \cdot \log \vec{X}_{\text{PRED}}$.

2.1.2 Variational Auto Encoders

Variational Autoencoders, introduced by Kingma et al. [20], are generative models that learn a stochastic bidirectional mapping between data space and, typically, a lower dimensional latent space. They consist of an encoder network that learns the distribution $q(\mathbf{z}|\mathbf{x})$ which stochastically maps data $\mathbf{x}$ to a latent $\mathbf{z}$ such that $\mathbf{z} \sim p_{\text{prior}}(\mathbf{z})$, and a decoder network that learns $p(\mathbf{x}|\mathbf{z})$ which inversely maps latent $\mathbf{z}$ to data $\mathbf{x}$. In simpler terms the encoder stochastically compresses the information of a datapoint into a latent vector and the decoder decompresses the stochastic latent vector into the original input.

It is standard practice in machine learning to make the following assumptions:

$$\begin{aligned} q(\mathbf{z}|\mathbf{x}) &= \mathcal{N}\left(\mu_E\left(\mathbf{x}\right), \sigma_E\left(\mathbf{x}\right)\right), \\ p(\mathbf{z}) &= \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right), \\ p(\mathbf{x}|\mathbf{z}) &= \mathcal{N}\left(\mu_D\left(\mathbf{z}\right), \sigma_{\text{user}}\mathbf{I}\right) \end{aligned}$$

where $\mathcal{N}$ represents the Gaussian distribution [21, 20]. The outputs of the encoder are the mean $\mu_E\left(\mathbf{x}\right)$ and standard deviation $\sigma_E\left(\mathbf{x}\right)$ of a latent Gaussian distribution, the output of the decoder is solely $\mu_D\left(\mathbf{z}\right)$ which is the reconstruction, and σ_{user} is an arbitrary user-specified constant often set to 1. Note that σ_{user} is used solely in log-likelihood computation and that no noise is actually added to $\mu_D\left(\mathbf{z}\right)$ for decoding as doing so would diminish the fidelity of reconstructions [21, 20]. The latent $\mathbf{z}$ is sampled using the reparameterization trick $\mathbf{z} = \mu_E\left(\mathbf{x}\right) + \sigma_E\left(\mathbf{x}\right) \odot \epsilon$ where $\epsilon \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right)$. Figure 2.4 provides a visualization of the compression and decompression process in a VAE architecture.

VAEs are trained to minimize the negative evidence lower bound (ELBO), in other words, the loss function of the VAE is $\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{REC}} + \mathcal{L}_{\text{KLD}}$. The first term $\mathcal{L}_{\text{REC}}$ is the reconstruction loss (e.g., MSE, CE) between the decoder output and the ground truth. The second term $\mathcal{L}_{\text{KLD}}$ is the Kullback-Leibler divergence (KLD loss) between the encoded and prior distribution for the latent. The KLD loss is calculated as $\mathcal{L}_{\text{KLD}} = \mathbf{1} \cdot (\sigma_E^2 + \mu_E^2 - \mathbf{1} - 2\log\sigma_E)$. The reconstruction loss coerces the encoder to capture important high level information in the latent vector and the decoder to take that high level information and accurately reconstruct the original graph. The KLD loss applies only to the encoder and coerces the latents to follow an isotropic Gaussian distribution so new data can be generated by passing in a random vector into the decoder.

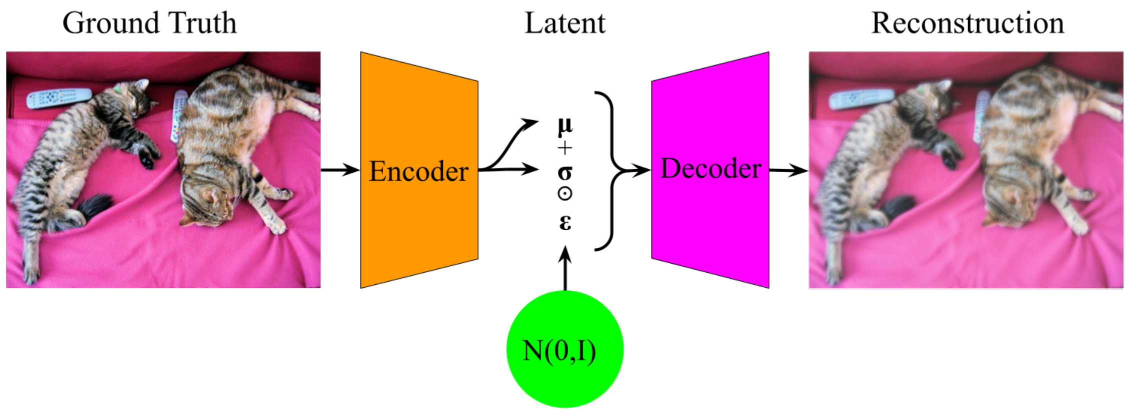


Figure 2.4: The encoder takes in an image and predicts the mean μ_E and the standard deviation σ_E of a Gaussian distribution. The latent is sampled from the Gaussian distribution via the reparameterization trick $\mathbf{z} = \mu_E(\mathbf{x}) + \sigma_E(\mathbf{x}) \odot \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The decoder takes the perturbed latent and attempts to reconstruct the original image.

2.1.3 Diffusion Models

Diffusion models, first introduced by Sohl-Dickstein et al. in 2015 [22] and later popularized by Ho et al. in 2020 [23], represent a class of generative models that learn to simulate the reverse of a gradual noising process. These models are inspired by the principles of thermodynamics, where a data point is slowly corrupted by noise, and the model learns to revert it back to its original state. A visualization of the process is provided in Figure 2.5.

Continuous Diffusion

The forward noising process is a Markov chain that gradually adds Gaussian noise to a data point over a series of T timesteps. For a given clean data point $\mathbf{x}_0$, the process produces a sequence of increasingly noisy versions $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$. This is achieved through a transition distribution defined as

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\,\mathbf{I}\right) \iff \mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{(1-\alpha_t)}\epsilon \tag{2.1}$$

here $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and $\alpha_0, \alpha_1, \ldots, \alpha_T$ is a noise schedule that controls the amount of noise added at each timestep.

A noise schedule is a sequence of monotonically decreasing scalar values satisfying $\alpha_0 = 1$ and $\alpha_T = 0$. Figure 2.6 illustrates the widely adapted cosine variance schedule introduced by Nichol & Dhariwal [24], that we also adopt in this work.

The goal of the forward process is to gradually obscure the original data in a controlled manner, reaching a point where $\mathbf{x}_T$ is essentially pure noise. Conveniently, instead of iteratively noising a datapoint we can in one shot sample $\mathbf{x}_t$ for any arbitrary t from the original datapoint via the cumulative transition distribution

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\overline{\alpha_t}}\mathbf{x}_0, (1-\overline{\alpha_t})\,\mathbf{I}\right) \iff \mathbf{x}_t = \sqrt{\overline{\alpha_t}}\mathbf{x}_0 + \sqrt{(1-\overline{\alpha_t})}\epsilon \tag{2.2}$$

where $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$.

The reverse process at a high level is simply gradually denoising a noisy sample by interpolating it with the clean datapoint. It is given by the distribution

$$p\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{\sqrt{a_t}\left(1-\bar{a}_{t-1}\right)\mathbf{x}_t + \sqrt{\bar{a}_{t-1}}\left(1-a_t\right)\mathbf{x}_0}{1-\bar{a}_t}, \frac{\left(1-a_t\right)\left(1-\bar{a}_{t-1}\right)}{1-\bar{a}_t}\mathbf{I}\right) \tag{2.3}$$

$$\mathbf{x}_{t-1} = \frac{\sqrt{a_t}\left(1-\bar{a}_{t-1}\right)\mathbf{x}_t + \sqrt{\bar{a}_{t-1}}\left(1-a_t\right)\mathbf{x}_0}{1-\bar{a}_t} + \sqrt{\frac{\left(1-a_t\right)\left(1-\bar{a}_{t-1}\right)}{1-\bar{a}_t}}\epsilon \tag{2.4}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. Therefore, the reverse process can be learned by training a network to predict the clean data point given a noisy sample. More explicitly, the denoiser network learns the function $\mathbf{x}_0(\mathbf{x}_t, t)$. New data can be generated by passing in pure noise into the network and iteratively denoising it by plugging the input and network prediction into the reverse distribution. Accordingly, continuous diffusion models are trained only to reconstruct the noiseless datapoint given some noisy input, thus MSE loss is almost always used.

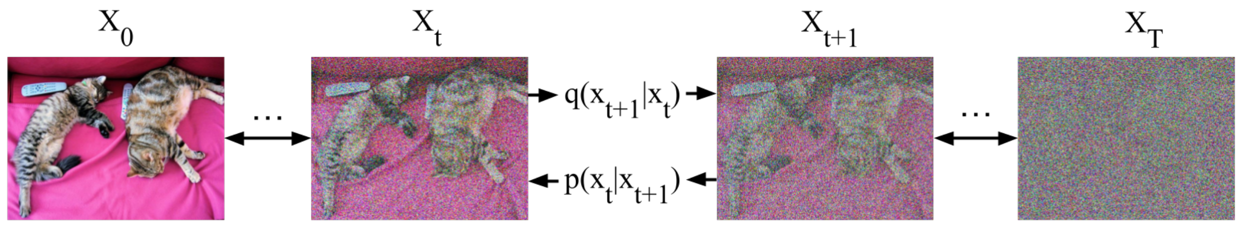


Figure 2.5: Noise is iteratively added to the data until it is indistinguishable from pure noise according to the forward distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. The reverse distribution $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ gradually removes noise, and is iteratively applied to transform pure noise into novel data.

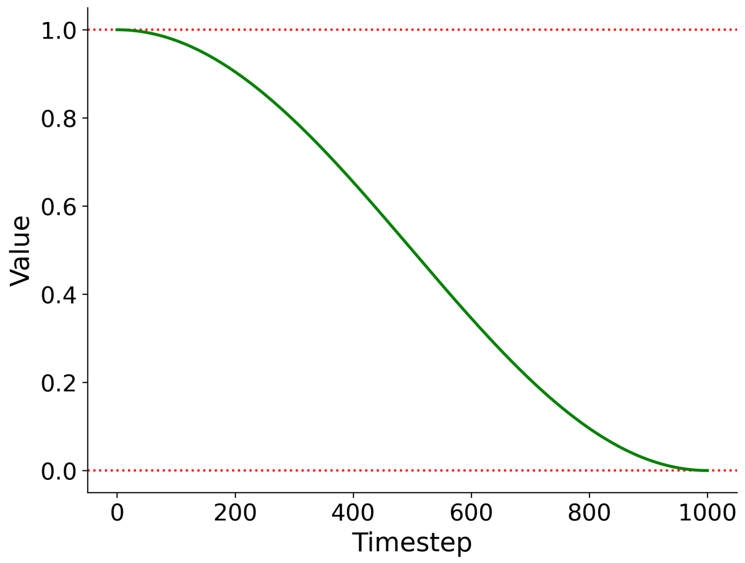


Figure 2.6: The green curve plots the cosine variance schedule $\overline{a}_t = \cos(\frac{\pi t}{2T})$ proposed by Nichol & Dhariwal over $T = 1000$ steps. This particular schedule is very popular since it ensures information is not destroyed unnecessarily early in the forward process. We can recover the non-cumulative variance via $a_t = \overline{a}_t / \overline{a}_{t-1}$

Discrete Diffusion

Analogous to continuous diffusion, a discrete diffusion framework utilizing the categorical distribution was introduced by Hoogeboom et al. [25] and later extended by Austin et al. [26]. The forward noising process for discrete diffusion is a discrete Markov chain that jumbles the state of a onehot vector over a series of T timesteps. This is achieved through a transition defined as $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{C}(\mathbf{x}_t; a_t\mathbf{x}_{t-1} + (1 - a_t)\mathbf{1}/D)$, here $\mathcal{C}(x; \mathbf{p})$ is the categorical distribution, $a_0, a_1, \ldots, a_T$ is a variance schedule, and D is the dimension of $\mathbf{x}$. Again, akin to continuous diffusion, there is a cumulative transition distribution and reverse transition distribution that are both Categorical distributions, however, as these details are not directly relevant to this work, we do not elaborate further. Initial experiments using this discrete noising scheme yielded poor performance. We attribute this limitation to the categorical framework's inability to represent superposition. In this context, superposition refers to a probability vector representing a combination of multiple states. However, the categorical distribution inherently produces one-hot vectors at each iteration of both the forward and reverse processes, collapsing the model's representation to a single discrete state. As a result, the model cannot maintain partial confidence across multiple categories. Other prior art addressed this issue by embedding discrete variables into a continuous space and then performing continuous diffusion, some used learnable embeddings [27, 28, 29] and others used nonlearnable embeddings [30, 31]. Instead, we propose a new discrete diffusion strategy that showed far better performance for generating CAD sketches and allows for superposition, allowing the model to consider multiple possible primitive types at the same time. Our approach is a simple augmentation of continuous diffusion using the softmax function such that the entire diffusion process is constrained to the probability simplex. Specifically, the softmax function is defined as

$$\mathrm{softmax}(\mathbf{x}) = \frac{\exp(\mathbf{x})}{\mathbf{1} \cdot \exp(\mathbf{x})}$$

which is simply a mapping to turn unnormalized logits to a probability vector and the forward noising process is

$$\mathbf{x}_{t+1} = \mathrm{softmax}\left\{\sqrt{a_{t+1}} \log(\mathbf{x}_t) + \sqrt{1 - a_{t+1}}\epsilon\right\} \tag{2.5}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The cumulative transition and reverse process are analogously

$$\mathbf{x}_t = \mathrm{softmax}\left\{\sqrt{\overline{a}_t} \log(\mathbf{x}_0) + \sqrt{1 - \overline{a}_t}\epsilon\right\} \tag{2.6}$$

$$\mathbf{x}_{t-1} = \mathrm{softmax}\left\{\mu_{t-1} + \sqrt{\frac{(1 - a_t)(1 - \overline{a}_{t-1})}{1 - \overline{a}_t}}\epsilon\right\} \tag{2.7}$$

$$\mu_{t-1} = \frac{\sqrt{a_t}(1 - \overline{a}_{t-1}) \log \mathbf{x}_t + \sqrt{\overline{a}_{t-1}}(1 - a_t) \log \mathbf{x}_0}{1 - \overline{a}_t}$$

Intuitively, we perform continuous diffusion in log-space and then project back onto the simplex via the softmax function. The distribution that describes this transition is the Logistic Normal distribution [32], however we simply refer to it as the Gaussian-Softmax distribution. A detailed derivation of the formulae listed above is provided in A.0.1 & A.0.3. We again prioritize reconstruction and we accordingly employ CE loss. The inference and training procedures for our proposed discrete diffusion framework are exactly analogous to the inference and training procedures for continuous diffusion. Note, however, that we have to initially label smooth $\mathbf{x}_0$ to a near one-hot representation to avoid singularities at the start of the diffusion process. This can be done via $\mathbf{x}'_0 = k\mathbf{x}_0 + \frac{1}{D}(1 - k)$ for some k close to 1 and D being the dimension of $\mathbf{x}_0$.

2.2 Related Work

The most relevant prior art to this work is Vitruvion, proposed by Seff et al. [10]. Vitruvion is a generative model for CAD sketches based on the transformer architecture, generating CAD sketches as sequences of tokens. The transformer architecture, originally introduced by Vaswani et al. [33] for natural language processing, has demonstrated state-of-the-art performance by enabling neural networks to focus on specific words or phrases via attention mechanisms. Seff et al. chose to represent CAD sketches as token sequences to address the heterogeneous nature of CAD sketches, where different primitives require varying numbers of parameters. To further manage this heterogeneity, their work was limited to a subset of primitive types namely: circles, arcs, lines, and points, and a select group of constraints: coincident, horizontal, vertical, parallel, perpendicular, and midpoint. Seff et al. converted CAD sketch graphs from the SketchGraphs dataset into token sequences by embedding each primitive independently. This embedding was achieved by summing the class embeddings of the primitive type with the embeddings of the 6-bit uniformly quantized parameter values. A similar embedding process was applied to constraints, where sub-primitive references and constraint types were independently embedded and then summed. Their model was trained using CE loss to predict both the type and quantized parameter values.

Another closely related work is SketchGen, proposed by Para et al. [3]. Like Seff et al., Para et al. introduced SketchGen as a generative model for CAD sketches based on the transformer architecture. However, unlike Vitruvion, SketchGen employed a custom grammar to represent CAD sketches as token sequences. Despite this difference, they also restricted their scope to the same primitive types: circles, arcs, lines, and points, as well as the same constraints: coincident, horizontal, vertical, parallel, perpendicular, and midpoint. Para et al. generated separate token sequences for primitives and constraints by applying a custom grammar parser to the list of entities. Their neural network was trained using CE loss, similar to the approach used by Seff et al.

SketchGraphs by Seff et al. [9] is another related work that introduced the dataset that all subsequent research has used. They provide a baseline model utilizing a Recurrent Neural Network (RNN) architecture. RNNs are not the state of the art in CAD sketch generation, and their follow up work Vitruvion has switched to the transformer architecture as well. Additionally, the baseline model does not predict the parameter values for primitives and instead relies on an off the shelf CAD constraint solver to fill in the parameter values.

Han et al. [30] also proposed a discrete diffusion procedure based on the Gaussian-Softmax distribution, however their procedure differs from ours in 2 major aspects. The first is that they use a heuristic reverse transition which they define to be

$$\mathbf{x}_{t-1} = \operatorname{softmax}\left\{\sqrt{a_{t-1}}\log \mathbf{x}_0 + K\sqrt{1-a_{t-1}}\epsilon\right\}$$

This is in contrast to our work for which we derived the reverse transition from the posterior distribution of the discrete forward process, justifying our approach from a probabilistic perspective. Another limitation of their heuristic reverse process is that too much information is destroyed since the cumulative transition is used at each step in the reverse process, which they address by introducing an almost one-hot projection scheme after each reverse step, but as a result the benefits of superposition were greatly diminished. The second aspect that differentiates our work is that we also propose a variance modification formula in Equation 3.1, that ensures a gradual forward process. In our early experiments we found weighting the noise added by an arbitrary constant K to abruptly destroy information during the forward process.

Simonovsky & Komodakis [13] introduced GraphVAE, a generative model for molecule synthesis that employs a variational autoencoder architecture. In GraphVAE, a graph is represented by an adjacency matrix, an edge attribute tensor, and a node attribute tensor. The adjacency matrix encodes the connections between nodes. The edge attribute tensor contains additional information about each edge, such as the type of chemical bond represented using a one-hot vector. Similarly, the node attribute tensor includes further details about each node, such as the atomic element, also represented with a one-hot vector. GraphVAE uses a feedforward network with edge-conditioned graph convolutions for the encoder. The decoder is a simple multi-layer perceptron that takes the latent vector as input and outputs the predicted adjacency matrix, predicted edge attribute tensor, and predicted node attribute tensor. GraphVAE was trained using CE loss and achieved decent results on the task of molecule synthesis.

Lastly, In their work, Vignac et al. [12] introduced Digress, a generative graph diffusion model for molecule synthesis that outperformed other models in its class. Like GraphVAE, Digress employs node and edge attribute tensors, where the attributes of each node and edge are represented by one-hot vectors. Unlike GraphVAE, however, Digress does not utilize an explicit adjacency matrix; instead, it uses the edge attribute tensor directly. Digress uses the categorical discrete diffusion framework proposed by Hoogeboom et al. [25]. The model was trained using CE loss and achieved state-of-the-art results on the task of molecule synthesis.

Chapter 3

Methodology

In this work, we begin by processing the SketchGraphs dataset in 2 stages. For the first stage we filter out simple CAD sketches, for which automation provides limited benefits, and large CAD sketches due to time and resource constraints. In the second stage we simplify primitive parameterizations, to avoid redundancy, and remove duplicate sketches from the dataset to avoid biasing our models. Initially, we explored a VAE as a generative approach for sketch synthesis, however, we found that our VAE produced suboptimal generations. Consequently, we pivoted to a diffusion-based generative model, which demonstrated superior performance in generating realistic and coherent sketches. Despite this shift, for the sake of completeness and to provide a comprehensive account of our research process, we include the methodology and findings from our VAE experiments. Ultimately, we train both of our models on the processed SketchGraphs dataset.

3.1 Dataset

We use the CAD sketch dataset introduced in SketchGraphs by Seff et al. [9]. The dataset is comprised of 15 million human-created CAD sketches scraped from Onshape, a cloud centric CAD platform. The majority of sketches in the dataset have less than 8 primitives comprising approximately 84% of the dataset, they are very simple box-like shapes and are almost identical to one another, Figure 3.2 illustrates such simple sketches. Figure 3.1 provides the proportion of sketches in relation to the number of primitives they contain. We also report, in Table 3.1, statistics on the frequency of each primitive type and constraint type in the dataset.

In accordance we chose to filter out all sketches with less than 8 primitives as this would heavily bias our model to generate simple boxes. Additionally due to resource and time constraints we chose to discard all sketches with more than 16 primitives, and retain only the most important primitive types namely: Line, Circle, Arc, and Point. Doing so is not very detrimental, however, since it appears that only a minority 9.98% of CAD sketches are comprised of more than 16 primitives. After filtering the primitives, we then kept the 8 most frequent constraints out of the subset namely: Coincident, Horizontal, Vertical, Parallel, Perpendicular, Tangent, Midpoint, and Equal. Thus even after filtering we keep 6 out of the 8 most frequent constraints in the raw dataset. After filtering we are left with 2.1 million CAD sketches. Para et al. similarly filter CAD sketches but discarded those with more than 24 primitives, whereas we discard sketches with more than 16 [3].

We also noted that the primitives in the raw dataset are overparameterized, and so we

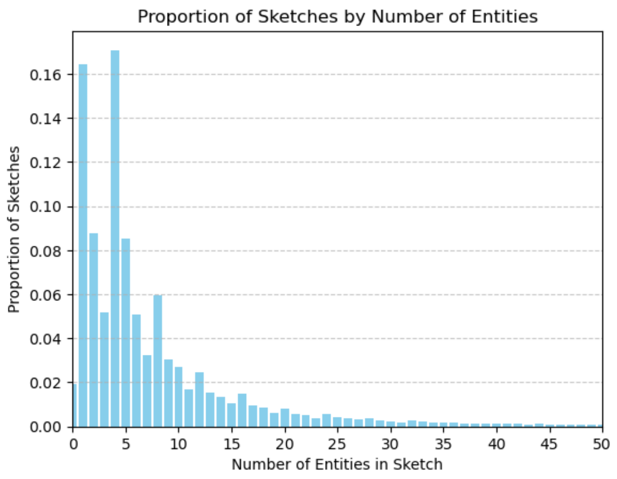


Figure 3.1: We plot the proportion of sketches in the SketchGraphs dataset with the number of primitives they contain. The histogram indicates that OnShape users tend to create and use CAD sketches with less than 16 primitives.

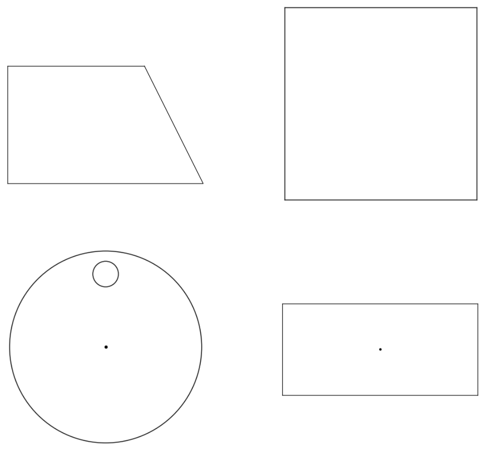


Figure 3.2: The majority of the SketchGraphs dataset is comprised of very simple sketches like these, so we exclude such sketches to ensure our model will learn to generate more useful and diverse CAD sketches.

simplified them to avoid redundancy. The reparameterizations are listed in Table A.4. In addition we noted that the scales of the CAD sketches varied greatly from being only a few millimeters in dimensions to being as large as several meters, so we perform a normalization procedure where sketches are rescaled, such that their bounding boxes have a side length of 2 meters, and shifted to align their center of mass onto the origin. Lastly, since the SketchGraphs dataset was constructed by retrieving public CAD sketches from the OnShape cloud platform, several sketches are duplicates of one another. This arises primarily from users creating sketches by following tutorials, reusing previously created sketches, or using collaboration tools provided by OnShape to copy sketches from each other. We perform a deduplication procedure to avoid biasing to such sketches, where we group identical sketches together and keep only 1 sketch from each group. After the deduplication procedure we are left with 1.4 million CAD sketches, which is sufficient for this work. Seff et al. perform a similar normalization and deduplication procedure [10].

Subsequently, the final CAD sketches were converted into graph representations for use in our models. For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the nodes are represented as tensors $\mathcal{V} \in \mathbb{R}^{n \times d_n}$ and the edges as tensors $\mathcal{E} \in \mathbb{R}^{n \times n \times d_e}$, where n is the maximum number of nodes, d_n is the dimension of node features, and d_e is the dimension of edge features. For our purposes $n = 16$, $d_n = 20$, and $d_e = 17$. Figure 3.3 shows a visualization of the features for each node and edge. We split our dataset of 1.4 million CAD sketches into 3 subsets: 90% is used for training, 5% is used for validation, and 5% is reserved for testing.

Primitive and Constraint type frequencies in raw dataset

Primitive	%
Line	68.47
Circle	9.97
Arc	9.45
Point	8.58
Spline	2.57
Ellipse	0.08
Constraint	**%**
Coincident	42.17
Projected	9.71
Distance	6.72
Horizontal	6.45
Mirror	5.54
Vertical	4.78
Parallel	4.37
Length	3.68
Perpendicular	3.24
Tangent	2.94

Primitive and Constraint type frequencies in filtered dataset

Primitive	%
Line	76.64
Circle	7.06
Arc	7.02
Point	9.28
Constraint	**%**
Coincident	49.94
Horizontal	12.19
Vertical	6.93
Parallel	11.80
Perpendicular	7.76
Tangent	5.10
Midpoint	3.27
Equal	3.01

Table 3.1: Here we present the frequency of each primitive type and constraint type in the SketchGraphs dataset on the left and we present statistics on the filtered dataset on the right. Splines and Ellipses make up a negligible portion of the dataset. We also note that the frequencies of the various primitive and constraint types are roughly preserved in the filtered dataset.

3.2 VAE for CAD Graph Generation

We first trained a Variational Autoencoder (VAE) since it is a well-established architecture for generative tasks. The encoder and decoder networks incorporate the graph attention module proposed in DiGress, due to its strong performance in other generative tasks [12]. Figure A.1 illustrates the architectural details of both the encoder and the decoder. The encoder maps CAD graphs into a latent space, while the decoder reconstructs CAD graphs from the latent representations. The latent space in our model is $\mathbb{R}^{1024}$ represented as 1024-dimensional vectors. As is standard, we choose the isotropic Gaussian distribution as our prior for the latent space. The encoder predicts the mean and log-variance (logvar) of a multivariate Gaussian distribution, which parameterizes the latent space. Latent vectors are then sampled from this distribution during both training and inference. These sampled latents serve as input to the decoder, enabling the generation of CAD graphs. We use MSE loss for the node parameter values and CE loss for the node type, node constructible, edge type, and edge subnode types. We choose the prior distribution of the latents to be

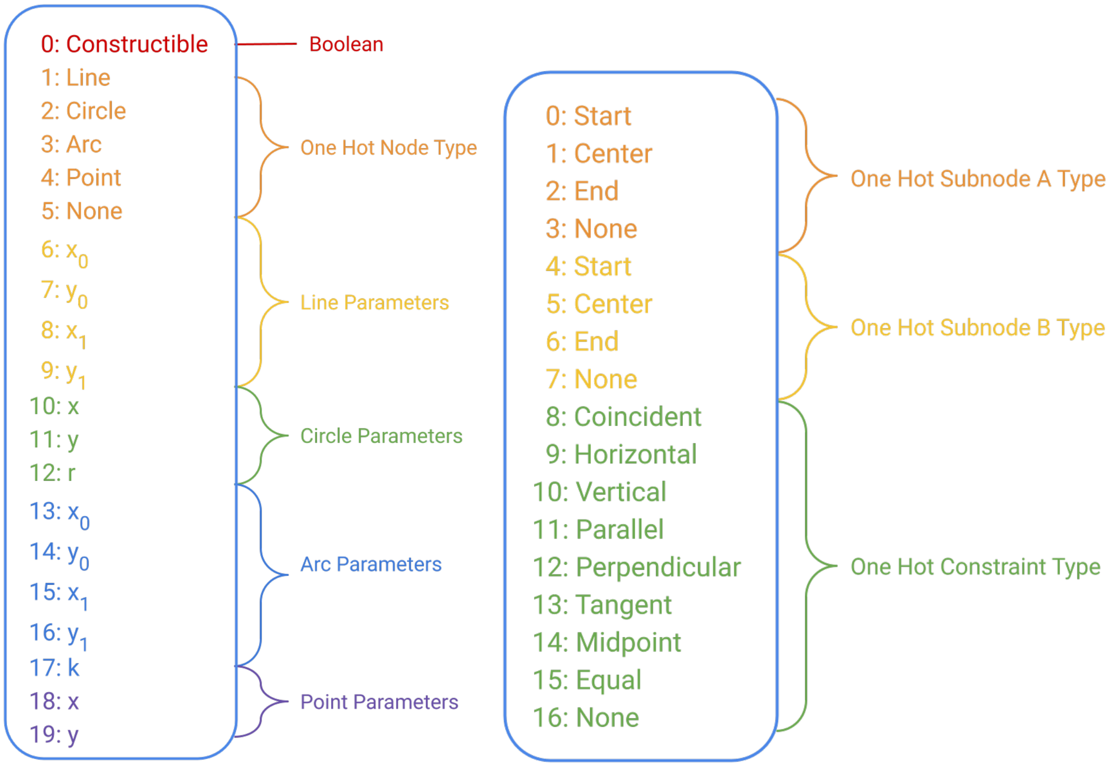


Figure 3.3: Node features on left and Edge features on right. Definitions of parameters is provided in A.4. All parameters inconsistent with the primitive type are zeroed out. Subnode A and B specify whether a constraint applies to the start, end, or center point of the relevant primitives, none means the whole primitive is affected by the constraint.

a standard Gaussian distribution so the KLD loss coerces the encoder to predict the mean and logvar to be **0**.

3.2.1 Training Procedure

The encoder and decoder networks were jointly trained for 70 epochs with a batch size of 8×256 distributed across 8 NVIDIA A30 GPUs. During training, we observed a severe adversarial relationship between the reconstruction loss and the KLD loss. This conflict arose because the KLD loss drives the latent space towards a standard Gaussian distribution, thereby reducing the encoder's capacity to encode meaningful information into the latent representations. Consequently, this behavior hindered the reconstruction of CAD graphs with high fidelity. To mitigate this issue, we applied a scaling factor of 1×10^{-3} to the KLD loss term to reduce its impact during training, additionally we also scaled the MSE loss term by a factor of 16. This adjustment improved graph reconstructions by forcing the model to prioritize reconstruction quality, but as a result we can't generate novel CAD graphs by simply feeding a random standard Gaussian vector into the decoder. A diagram of the training pipeline is provided in Figure 3.4, and pseudocode is provided in Algorithm

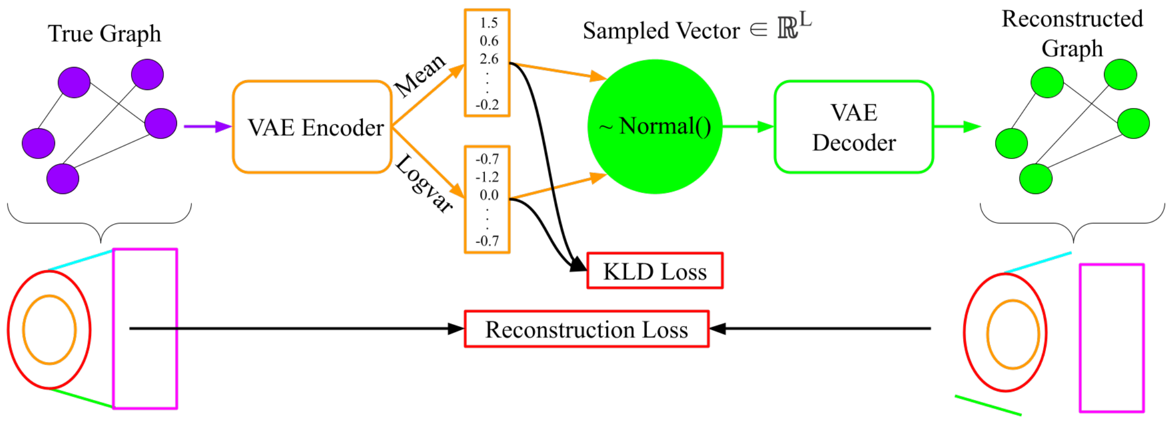


Figure 3.4: Training pipeline for VAE model. The encoder produces a mean μ and logvar $\log \sigma^2$, then the latent $\mathbf{z}$ is sampled via $\mathbf{z} = \mu + \sigma \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. The latent vector is fed into the decoder to reproduce the encoded graph.

1.

Algorithm 1 VAE Training Procedure

```
Require: Training data $X$, neural networks for encoder $E_\phi(\mathcal{V}, \mathcal{E})$ and decoder $D_\theta(z)$, number of epochs $N$, and KLD loss weight $\lambda$
 1: for epoch = 1 to $N$ do
 2:     for each graph $(\mathcal{V}, \mathcal{E}) \in X$ do
 3:         Calculate mean and standard deviation $\mu, \sigma = E_\phi(\mathcal{V}, \mathcal{E})$
 4:         Sample latent $\mathbf{z} = \mu + \sigma \odot \epsilon, \epsilon \sim \mathcal{N}(\vec{0}, \mathbf{I})$
 5:         Reconstruct graph $(\mathcal{V}', \mathcal{E}') = D_\theta(\mathbf{z})$
 6:         Compute reconstruction loss: $\mathcal{L}_{\text{recon}}(\mathcal{V}', \mathcal{E}', \mathcal{V}, \mathcal{E})$
 7:         Compute KL divergence: $\mathcal{L}_{\text{KL}} = \frac{1}{2}\sum_i(\sigma_i^2 + \mu_i^2 - 1 - 2\log\sigma_i)$
 8:         Compute total loss: $\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda\mathcal{L}_{\text{KL}}$
 9:         Update $\phi$ and $\theta$ using gradient descent on $\mathcal{L}$
10:     end for
11: end for
```

3.2.2 VAE Inference Sampler

To extend the generative capability of our baseline model, we trained a separate continuous diffusion-based sampler network after training the VAE. This sampler generates latent vectors that the decoder transforms into CAD graphs, independent of the encoder. The diffusion sampler was trained for 1000 epochs with a batch size of 8×1024, also utilizing 8 NVIDIA A30 GPUs for efficient training. The sampler network is a simple 32 layer feed forward network with residual connections. Since, the sampler is tasked with generating

Algorithm 2 VAE Inference Procedure

```
Require: Trained neural networks for sampler $S_\phi(z)$ and decoder $D_\theta(z)$, and variance schedule $a_0, \ldots, a_T$
1: Sample seed $\mathbf{z}_t \sim \mathcal{N}(\vec{0}, \mathbf{I})$
2: for $t = T - 1$ to $0$ do
3:     Predict true latent $\mathbf{z}'_0 = S(\mathbf{z}_t, t)$
       $$\mathbf{z}_{t-1} = \frac{\sqrt{a_t}(1-\bar{a}_{t-1})\mathbf{z}_t + \sqrt{\bar{a}_{t-1}}(1-a_t)\mathbf{z}'_0}{1-\bar{a}_t} + \sqrt{\frac{(1-a_t)(1-\bar{a}_{t-1})}{1-\bar{a}_t}}\epsilon, \epsilon \sim \mathcal{N}(\vec{0}, \mathbf{I})$$
4: end for
5: Reconstruct graph $(\mathcal{V}', \mathcal{E}') = D_\theta(\mathbf{z_0})$
```

latents for the decoder, we employ MSE loss between the predicted latent and true latent vectors for training. The generation pipeline of the VAE is illustrated in Figure 3.5 and pseudocode is provided in Algorithm 2.

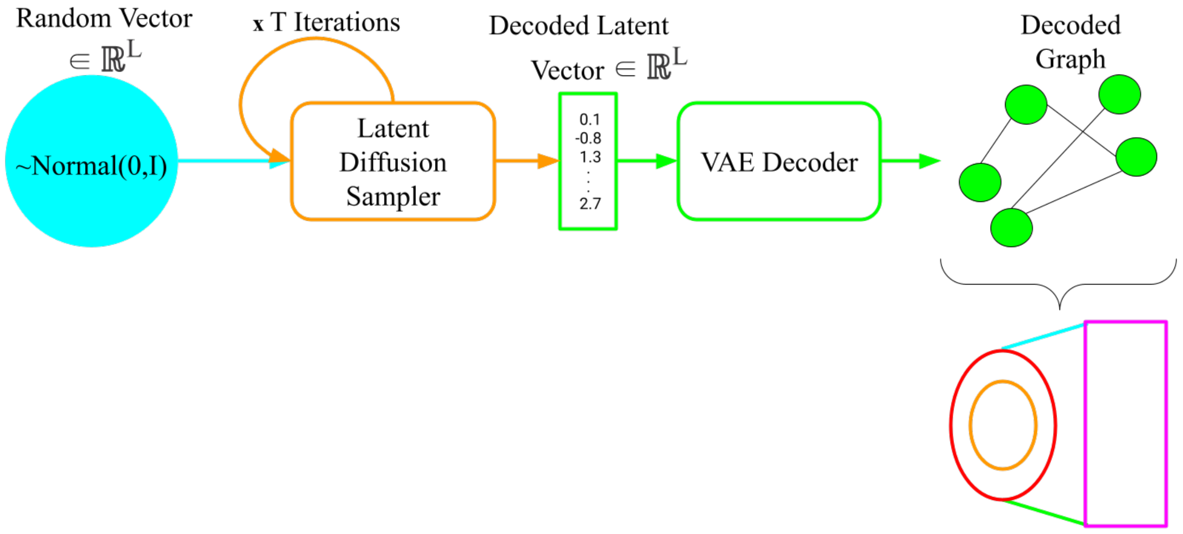


Figure 3.5: Inference pipeline for VAE model. The latent sampler network generates a latent vector that can be then fed into the decoder to decode into a CAD graph.

3.3 Diffusion Model for CAD Graph Generation

Motivated by state-of-the-art performance of diffusion models across a variety of generative tasks, we present the first application of diffusion to the task of CAD graph generation. The architecture of our denoiser network is a slight variant of the Diffusion Transformer (DiT) architecture, as introduced by Peebles & Xie [34]. This choice was motivated by the strong performance of DiT in image generation tasks, and our experiments confirmed its suitability for primitive generation. A schematic of the network architecture is depicted in Figure A.2.

3.3.1 Forward Process

The denoiser network is trained to reconstruct the original noiseless graph $\mathcal{G}_0$ from a noisy graph $\mathcal{G}_t$ conditioned on the diffusion timestep t. We define the total diffusion process across 1500 steps, where $t = 0$ corresponds to the original graph without noise, and $t = 1500$ represents pure noise. We use the aforementioned cosine variance schedule illustrated in Figure 2.6. The forward diffusion process independently noises each variable x in the graph via:

$$x_{t,c} = \sqrt{\overline{a_t}} x_{0,c} + \sqrt{1 - \overline{a_t}}\epsilon$$
$$x_{t,d} = \text{softmax}\left\{ \sqrt{\overline{b_t}} \log\left(kx_{0,d} + \frac{1-k}{D}\right) + \sqrt{1 - \overline{b_t}}\epsilon \right\}$$

where $x_{-,c}$ are continuous variables (i.e., primitive parameters), $x_{-,d}$ are discrete variables (i.e., primitive types, edge types, edge subnodes, and construct booleans), D is the dimension of the respective one-hot vector, $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, k is a user-defined smoothing constant near 1 as to avoid singularities at the start of the diffusion process which we set $k = .99$. Essentially, all continuous variables undergo continuous Gaussian diffusion and all discrete variables undergo Guassian-Softmax diffusion. Note that we expand boolean values into a one-hot representation for the forward noising and reverse denoising processes.

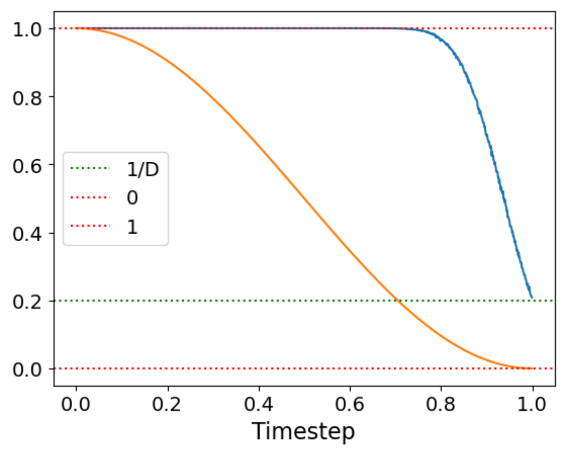
For discrete diffusion, variance schedules can not be used as is, since the softmax projection distorts the effect of the injected noise. To address this, we propose the following adjustment where the argmax of the noised class label follows the Categorical distribution $\text{argmax}(\mathbf{x}_t) \sim \mathcal{C}(\overline{a_t}\mathbf{x}_0 + (1 - \overline{a_t})/D)$ achieved via the augmentation:

$$\overline{b_t} = \frac{f(\overline{a_t})^2}{f(\overline{a_t})^2 + f(k)^2} \quad \text{where} \quad f(x) = \log\left(\frac{1-x}{(D-1)x+1}\right) \tag{3.1}$$

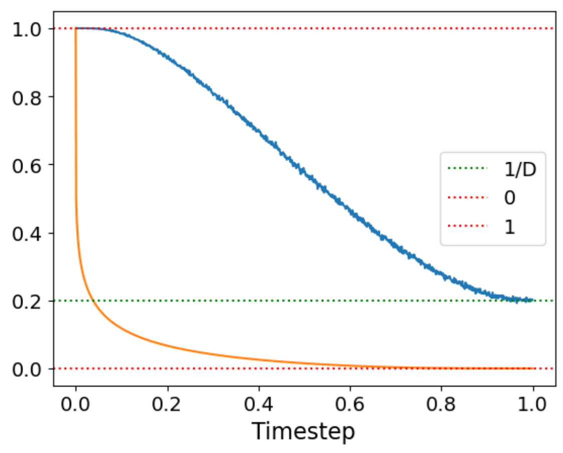
Figure 3.6 highlights the importance of the proposed variance schedule augmentation, demonstrating its advantages over directly using the raw variance schedule. A derivation of the augmentation scheme is provided in A.0.4

3.3.2 Training Procedure

Training was performed over 1000 epochs with a batch size of 8×512, distributed across 8 NVIDIA A30 GPUs. A constant learning rate of 1×10^{-4} was employed throughout. To prevent bias toward predicting zeros, we use the ground truth primitive type to mask irrelevant parameters before computing the MSE loss. This approach ensures the model simultaneously predicts the optimal parameters for all possible primitive types while assigning a confidence score (i.e., probability) to each. For example, if the true primitive type is



(a) Cosine variance used directly in discrete forward process.



(b) Cosine variance augmented by 3.1 in discrete forward process.

Figure 3.6: The orange curves are the raw and augmented variance schedules, and the blue curves are the respective probabilities that the argmax does not transition to another index at that timestep. The augmented variance schedule provides a more gradual discrete forward process than the raw variance schedule.

a line, the model still predicts the parameters for the other types, such as circles or arcs, while assigning probabilities that reflect its confidence in each prediction. A diagram of the training pipeline is presented in Figure 3.7, and pseudocode is provided in Algorithm 3. The denoiser network was trained using the same reconstruction loss that the VAE was trained with. Similarly, we found it beneficial for training convergence to weigh the MSE loss by $\min(c, \text{SNR}(t))$, where $\text{SNR}(t) = \overline{a}_t/(1-\overline{a}_t)$, following the methodology of Hang et al. [35]. Additionally, to improve training convergence, we deviated from the conventional approach of sampling timesteps from a uniform distribution. Instead, we adopted the methodology proposed by Wang et al. [36], to sample timesteps from a piecewise distribution, for our work it is:

$$A(x) = \begin{cases} 1.6 & \text{if } x \leq 750, \\ 0.4 & \text{if } x > 750 \end{cases} \tag{3.2}$$

3.3.3 Inference Procedure

Since we noise all continuous and discrete variables independently, for the reverse process we denoise all variables independently as well. We use the same reverse transitions defined in Equations 2.4 & 2.7. Before each continuous reverse step, we weight the predicted primitive parameters by their corresponding rescaled predicted type probabilities. This adjustment is necessary because irrelevant parameters are masked during training, prompting the model to predict optimal parameters for all types simultaneously. Since the model is trained to expect irrelevant parameters to approach zero at smaller timesteps, discrepancies may arise

Algorithm 3 Diffusion Training Procedure

```
Require: Training data $X$, neural network for denoiser $M_\theta(\mathcal{V}, \mathcal{E})$, number of epochs $N$
Ensure: Trainable parameters $\theta$
 1: for epoch = 1 to $N$ do
 2:     for each graph $(\mathcal{V}, \mathcal{E}) \in X$ do
 3:         Sample timestep $t \sim A(x)$ provided in 3.2
 4:         Noise graph $(\mathcal{V}_t, \mathcal{E}_t) = \text{noise}(\mathcal{V}, \mathcal{E}, t)$
 5:         Reconstruct graph $(\mathcal{V}', \mathcal{E}') = M_\theta(\mathcal{V}_t, \mathcal{E}_t)$
 6:         Mask predicted parameters $\mathcal{V}'_c$ with corresponding ground truth primitive type
            probabilities $\mathcal{V}_d$
                $\mathcal{V}'_c \leftarrow \mathcal{V}'_c * \mathcal{V}_d$
 7:         Compute reconstruction loss: $\mathcal{L}_{\text{recon}}(\mathcal{V}', \mathcal{E}', \mathcal{V}, \mathcal{E})$
 8:         Update $\theta$ using gradient descent on $\mathcal{L}$
 9:     end for
10: end for
```

Algorithm 4 Diffusion Inference Procedure

```
Require: neural network for denoiser $M_\theta(\mathcal{V}, \mathcal{E})$
 1: Sample latent: discrete variables from the isotropic standard Gaussian-Softmax and
    continuous variables from the isotropic standard Gaussian
        $(\mathcal{V}_T, \mathcal{E}_T) \sim \mathcal{N}(\vec{0}, \mathbf{I}) || \text{softmax}\{\mathcal{N}(\vec{0}, \mathbf{I})\}$
 2: for $t = T - 1$ to 1 do
 3:     Predict noiseless graph $(\mathcal{V}', \mathcal{E}') = M_\theta(\mathcal{V}_t, \mathcal{E}_t)$
 4:     Weight predicted parameters $\mathcal{V}'_c$ by corresponding predicted primitive type probabil-
        ities $\mathcal{V}'_d$, rescaled such that the maximum element is exactly 1
            $\mathcal{V}'_c \leftarrow \mathcal{V}'_c * \mathcal{V}'_d / \max(\mathcal{V}'_d)$
 5:     Interpolate as given by Equations 2.4 & 2.7
            $\mathcal{V}_{t-1}, \mathcal{E}_{t-1} = \text{interpolate}(\mathcal{V}_t, \mathcal{V}'), \text{interpolate}(\mathcal{E}_t, \mathcal{E}')$
 6: end for
```

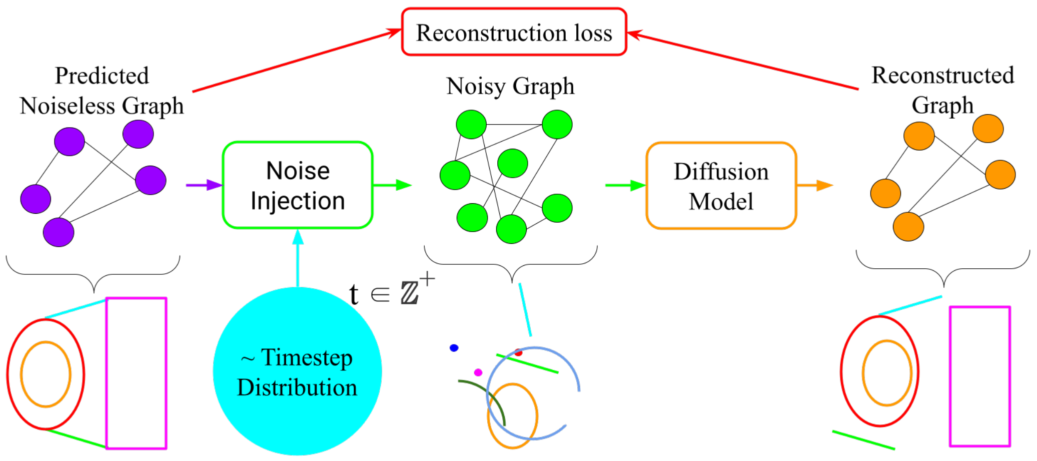


Figure 3.7: Training pipeline for Diffusion model. A graph is noised to a random timestep and then passed into the diffusion model. The diffusion model is trained to denoise the graph.

during the reverse process. Applying this weighting ensures that the model's predictions better align with its expectation of noisy primitives, particularly at smaller timesteps. We rescale the predicted type probabilities by dividing each probability vector with its maximum element. This rescaling prevents parameter values from decaying to zero throughout the reverse diffusion process, ensuring more accurate predictions of noisy primitives, particularly at smaller timesteps. A visualization of the generation pipeline is provided in Figure 3.8, and pseudocode is provided in Algorithm 4.

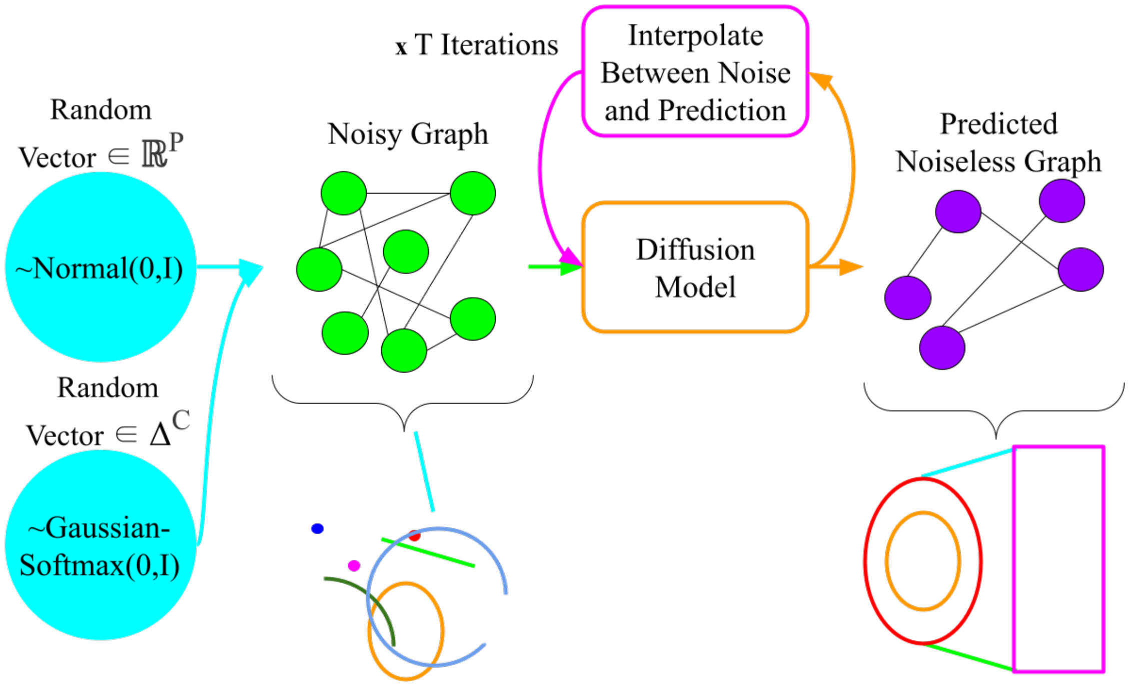

x T Iterations
Interpolate Between Noise and Prediction
Random Vector $\in \mathbb{R}^P$
Noisy Graph
Predicted Noiseless Graph
~Normal(0,I)
Diffusion Model
Random Vector $\in \Delta^C$
~Gaussian-Softmax(0,I)

Figure 3.8: Inference pipeline for Diffusion model. The model iteratively and gradually interpolates between the noisy variables and its predictions over T steps.

Chapter 4
Results

4.1 Baselines

For our baselines we choose the most relevant prior art, namely: SketchGraphs by Seff et al., SketchGen by Para et al., and Vitruvion by Seff et al., since they are all methods pertaining to generating CAD sketches. SketchGraphs was the first work in this domain and importantly introduced the dataset that all subsequent work has utilized, however it could not generate primitive parameters and relied on an external constraint solver provided by Onshape to estimate the values. SketchGen and Vitruvion are subsequent works that aimed to generate primitive parameters. SketchGen used a custom grammar to represent primitives as token sequences, which neccessitated the primitive parameters to be quantized. Vitruvion is a follow up work that similarly generates CAD sketches as token sequences, but they did not write their own custom grammar and instead opted to generate simplified primitive parameterizations. Vitruvion was the SOA, so we'll be comparing our work against theirs the most. Unlike prior art we chose not to quantize primitive parameters and treat them as continuous values. We justify this approach by noting that this allows for a simpler implementation and also allows our model to directly consider the distance, direction, and other geometric information between primitives.

4.1.1 Negative Log-Likelihood

We present the negative log-likelihood (NLL) on the test set to measure the aptitude of our model in learning the distribution of CAD graphs, provided in Table 4.1. A lower NLL indicates better generalization and improved model performance, though it is not necessarily indicative of sample quality. For diffusion models and VAEs, computing the exact NLL is intractable, therefore, we follow the standard approach of approximating the NLL using the ELBO, which satisfies the inequality ELBO $\geq$ NLL. As a result, we present the ELBO for our VAE and diffusion model, however despite this, our diffusion model achieved SOA NLL for CAD sketch generation. We note that the NLL for constraint generation seems to be inaccurate due to the non-sparse representation of constraints in our approach. The vast majority of graph edges are the null type, which biases the model toward predicting null constraints, artificially skewing the constraint NLL lower. Importantly, this issue does not affect the generation of primitives—the most critical component of a CAD sketch—where our model has still achieved SOA NLL.

Method	Bits/Sketch$\downarrow$	Bits/Primitive$\downarrow$	Bits/Constraint$\downarrow$
Diffusion (Ours)	**81.33**	**5.08**	**0.0031**
VAE (Ours)	334.19	_	_
Vitruvion	84.80	8.19	0.82
SketchGen	88.22	8.60	0.61
SketchGraphs	158.90	_	2.42

Table 4.1: The NLL for each model is reported in bits, where a lower NLL is better. For our VAE, we provide only sketch-level statistics as the ELBO must be calculated jointly over latents, nodes, and edges. For SketchGraphs, since primitive parameters are not predicted, there is no corresponding NLL for primitives. Notably, our diffusion model achieved SOA NLL per primitive and per sketch.

4.1.2 Sample Quality

Here we present the Fréchet Inception Distance (FID), precision, and recall as metrics to evaluate the fidelity of generated CAD graphs, as shown in Table 4.2. Fidelity measures how closely synthetic CAD graphs resemble real CAD graphs. Standard image generation metrics are employed to assess sample quality; however, since constraints are immaterial, these metrics evaluate only the fidelity of primitives. A lower FID score indicates higher fidelity samples, and our diffusion model achieves state-of-the-art (SOA) FID. Higher precision reflects a closer resemblance between generated and real samples, minimizing irrelevant or low-quality outputs, while higher recall indicates greater diversity in generated samples, capturing the full range of real data variations. Precision and recall often exhibit an inverse relationship, where improvements in one may lead to reductions in the other. As is standard in image generation literature, we compute these metrics using InceptionV3 trained on ImageNet over 10K CAD graphs from our test set.

Method	FID$\downarrow$	Precision$\uparrow$	Recall$\uparrow$
Diffusion (Ours)	**7.80**	0.233	**0.251**
VAE (Ours)	93.34	0.134	0.033
Vitruvion	16.04	**0.294**	0.176

Table 4.2: The FID, precision, and recall are presented for unconditional primitive generation. A lower FID is better, a higher precision is better, and a higher recall is better. We note that our diffusion model has achieved SOA FID.

4.1.3 Qualitative Results

We also provide model generations to illustrate perceptual quality, as evaluating CAD sketches is still an open problem. In Figure 4.1 we provide 3 columns of rendered CAD sketches from the SketchGraphs dataset, our diffusion model, and Vitruvion respectively.

We also provide some diffusion trajectories to visualize the sampling process of our diffusion model in Figure 4.2. We don't show generations from our VAE due to subpar results.

4.2 Remarks

We believe there are two major reasons as to why our model has improved generative capabilities. The first reason is that our diffusion model has access to the full context of the CAD graph at any given step, unlike Vitruvion and SketchGen which have a limited view due their autoregressive nature. Since, Vitruvion and SketchGen append one node to the graph at a time, the models start off with an empty graph with zero context of how the node will fit in with the yet to be generated graph. Seff et al., make note of this in their work where they discuss how their model has better performance near the end of the sampling process where it has a larger context window. This is in contrast to our graph diffusion model which always has full context and can thus better model the relationships between nodes.

The second reason we hypothesize as to why our model performs better is our diffusion approach. Unlike autoregressive models which can only influence each node once, namely during each generation step, our diffusion based approach allows our model to influence nodes over the whole reverse process. Although the downside is that the sampling takes much longer, the benefit is that our model gets multiple chances to correct the nodes. This works in tandem with our focus on generating CAD graphs, since the model has multiple chances to see how any particular node/primitive fits in with the rest of the sketch, our model is better able to exactly pinpoint the semantics of primitives.

4.3 Failure Modes

We identified two primary failure modes in our model. The first failure mode is where the model produces no discernible shape. This typically occurs when the model fails to interpret the relationships between primitives within a sketch and is unable to effectively denoise it. A lesser offshoot of this issue is that the model will often not terminate arcs and lines in a cohesive manner resulting in gaps and deformities. The second common failure involves generating constraints that are frequently unsatisfiable or, if satisfiable, fail to accurately represent the intended relationships between primitives. When an external constraint solver, such as Onshape's, attempts to enforce these constraints, the resulting sketch structure is often compromised. Although we were able to improve primitive generation, constraint generation performance is still poor. Similar sentiments were voiced by Para et al. and Seff et al. in their respective works SketchGen and Vitruvion, where they found generated constraints often did not improve the fidelity of sketches and instead even reduced the fidelity of sketches when a constraint solver attempted to enforce the constraints [3, 10]. We provide visualizations of each failure case in Figure A.3.

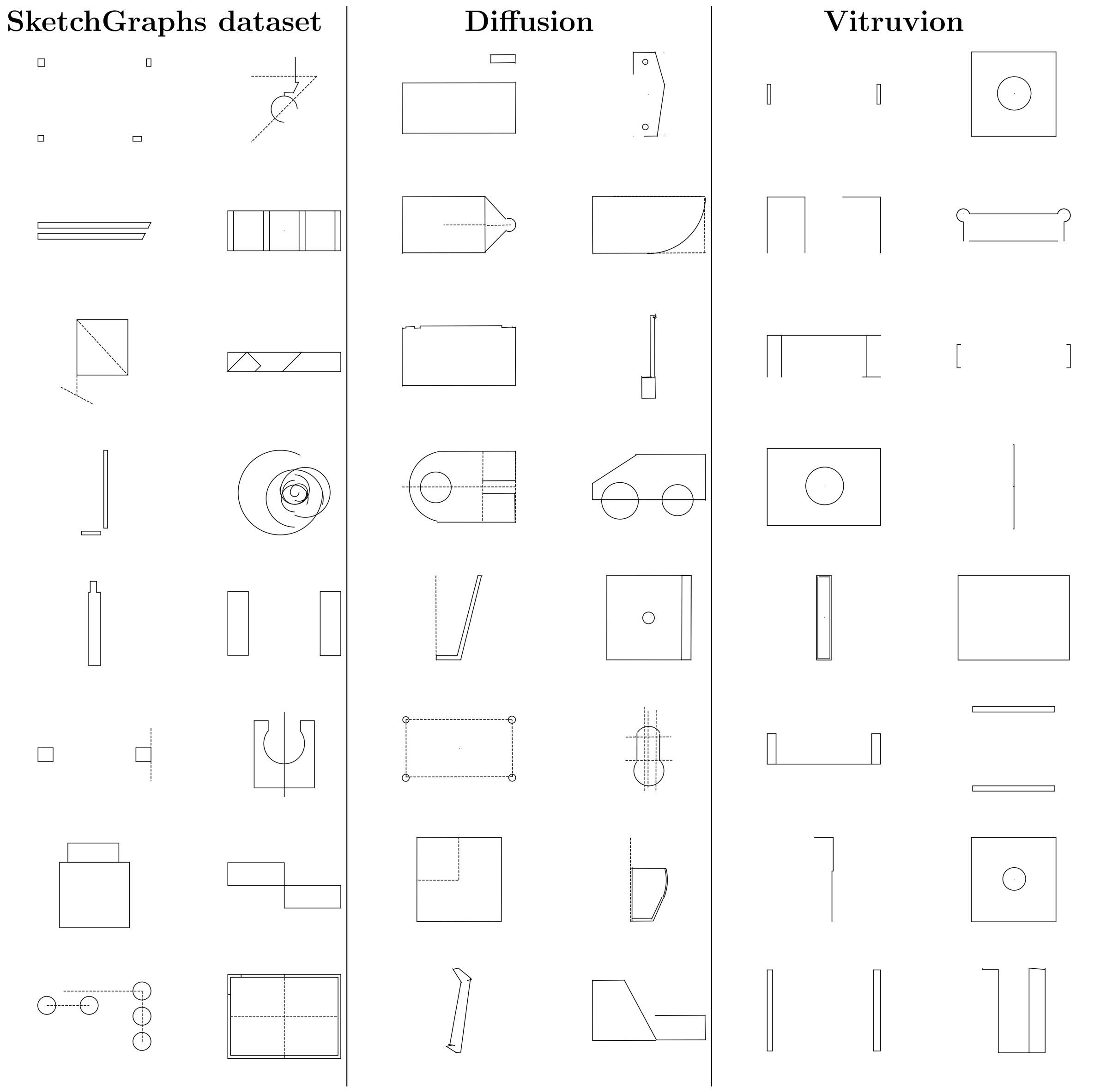


Figure 4.1: Random examples from the SketchGraphs dataset (left), random samples from our unconditional diffusion model (center), and random samples from Vitruvion (right). Although Vitruvion generates higher precision CAD sketches, the generated samples are not very diverse, indicating mode collapse where the model fixated on generating very specific types of sketches. Our diffusion model generates more diverse samples, but in exchange does not have great precision in the generations. This behavior is reflected in the precision and recall scores, where Vitruvion got a higher precision but lower recall score, and our diffusion model got a higher recall but lower precision score.

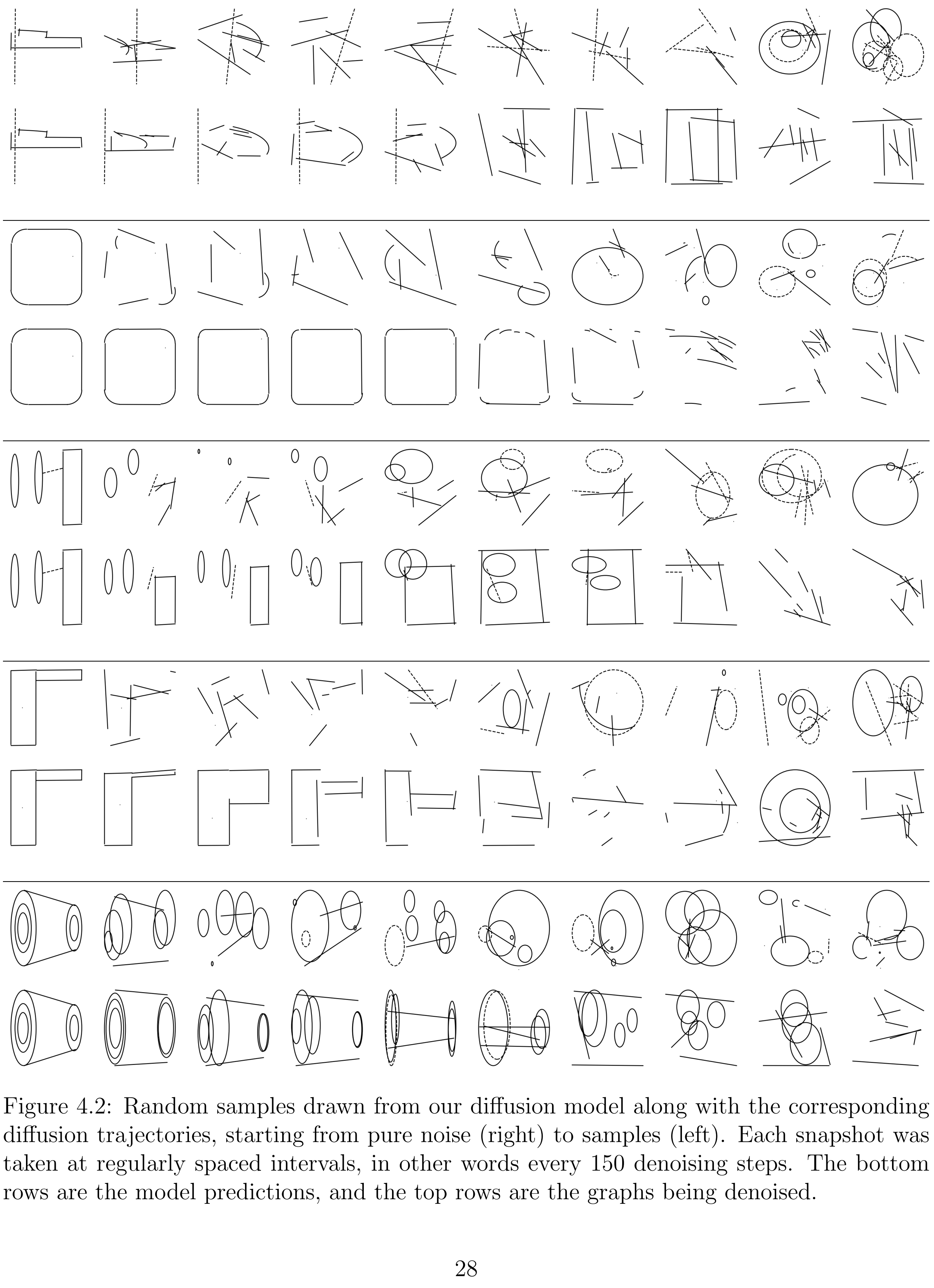


Figure 4.2: Random samples drawn from our diffusion model along with the corresponding diffusion trajectories, starting from pure noise (right) to samples (left). Each snapshot was taken at regularly spaced intervals, in other words every 150 denoising steps. The bottom rows are the model predictions, and the top rows are the graphs being denoised.

Chapter 5

Conclusion and Future Work

In this work, we present the first application of GNNs and diffusion to the task of CAD sketch generation, addressing limitations in existing methods that hinder their robustness and effectiveness in assisting the iterative CAD design process. To advance the state-of-the-art in this domain, we proposed a novel discrete diffusion strategy based on the Gaussian-Softmax distribution for simplex-constrained diffusion. This approach accommodates superposition and is presented as an alternative to the widely adopted categorical diffusion framework. Additionally, we introduce a variance schedule augmentation to ensure a gradual and stable noising process for our proposed discrete diffusion framework.

Empirical evaluation demonstrates that our diffusion-based approach achieves state-of-the-art (SOA) performance in terms of Negative Log-Likelihood (NLL) and Fréchet Inception Distance (FID) metrics. We observe that integrating GNNs with our novel diffusion strategy significantly outperforms the previous SOA model, Vitruvion, highlighting the effectiveness of our method in generating accurate and expressive CAD sketches.

A promising direction for future research lies in exploring conditional text-to-CAD sketch generation. Given that conditional image diffusion models have demonstrated significant improvements in sample quality over unconditional approaches, a similar benefit may be realized in the context of CAD sketch generation. Extending this concept would provide users with more precise control over generated sketches based on textual input, thereby increasing the applicability of CAD generation in the design iteration cycle. Another important avenue involves applying our proposed discrete diffusion strategy to domains beyond CAD, such as image generation and natural language synthesis, where the ability to handle simplex-constrained diffusion could offer novel modeling capabilities.

Finally, integrating 3D operations for volumetric generation represents a natural extension of this work, moving beyond 2D sketch generation to encompass full three-dimensional design capabilities. Such advancements could open pathways for generating comprehensive CAD models suitable for complex engineering and manufacturing tasks.

Appendix A

The Gaussian-Softmax distribution ($\mathcal{GS}$), introduced as the Logistic-Normal distribution by Aitchison [32], is the distribution of a Gaussian vector that has undergone the softmax transformation. The probability density is:

$$p(\vec{y}|\vec{\mu},\sigma^2) = D^{-\frac{1}{2}}(2\pi\sigma^2)^{\frac{1-D}{2}}\left(\prod_{i=1}^{D} y_i\right)^{-1} \times \exp\left\{-\frac{1}{2\sigma^2}\left[\sum_{i\neq D}\left(\log\frac{y_i}{y_D}-\mu_i+\mu_D\right)^2-\frac{1}{D}\left(\sum_{i\neq D}\log\frac{y_i}{y_D}-\mu_i+\mu_D\right)^2\right]\right\}$$

and the derivation of the density is provided in A.0.2.

A.0.1 Derivation of Cumulative Forward Transition

A simple derivation for the cumulative transition is:

$$p\left(\vec{x}_{t+2} \mid \vec{x}_t\right) = \text{softmax}\left\{\sqrt{a_{t+2}}\log\left[\text{softmax}\left\{\sqrt{a_{t+1}}\log\left(\vec{x}_t\right)+\sqrt{1-a_{t+1}}\vec{\epsilon}\right\}\right]+\sqrt{1-a_{t+2}}\vec{\epsilon}\right\}$$

Expand terms:

$$= \text{softmax}\left\{\sqrt{a_{t+2}}\left(\sqrt{a_{t+1}}\log\left(\vec{x}_t\right)+\sqrt{1-a_{t+1}}\vec{\epsilon}+C\right)+\sqrt{1-a_{t+2}}\vec{\epsilon}\right\}$$

Reduce terms:

$$= \text{softmax}\left\{\sqrt{a_{t+2}a_{t+1}}\log\left(\vec{x}_t\right)+\sqrt{a_{t+2}(1-a_{t+1})}\vec{\epsilon}+\sqrt{a_{t+2}}C+\sqrt{1-a_{t+2}}\vec{\epsilon}\right\}$$

Constants disappear due to shift invariance of softmax:

$$= \text{softmax}\left\{\sqrt{a_{t+2}a_{t+1}}\log\left(\vec{x}_t\right)+\sqrt{a_{t+2}(1-a_{t+1})}\vec{\epsilon}+\sqrt{1-a_{t+2}}\vec{\epsilon}\right\}$$

Sum of Gaussians is another Gaussian with summed variances:

$$= \text{softmax}\left\{\sqrt{a_{t+2}a_{t+1}}\log\left(\vec{x}_t\right)+\sqrt{a_{t+2}(1-a_{t+1})+1-a_{t+2}}\vec{\epsilon}\right\}$$

Merging terms accumulates the a terms:

$$= \text{softmax}\left\{\sqrt{a_{t+2}a_{t+1}}\log\left(\vec{x}_t\right) + \sqrt{1 - a_{t+2}a_{t+1}}\vec{\epsilon}\right\}$$

therefore iteratively applying the forward transition will simply accumulate the variance schedule terms.

A.0.2 Derivation of Gaussian Softmax Density

Our strategy is to use the change of variables formula:

$$p'(\vec{y}) = p\left(h^{-1}\left(\vec{y}\right)\right)\text{Det}\left[J\left(\vec{h}\left(\vec{y}\right)\right)\right]$$

where $\vec{h}(\vec{y})$ is some invertible function and $\text{Det}(J(\vec{h}(\vec{y}))$ is the determinant of the jacobian. More specifically,

$$\text{softmax}\{\vec{y}\} = \text{softmax}\{\vec{y} - \vec{1}\cdot y_D\}$$

holds due to the shift invariance of softmax, thus our strategy is to first find the density of $\vec{y'} = [y_1 - y_D, y_2 - y_D, ..., 0]$, as this "centered" form turns the softmax into an invertible function $\vec{h}(\vec{y'}) = \text{softmax}\{\vec{y'}\}$ where the inverse is $\vec{h}^{-1}(\vec{x}) = \log(\vec{x}/x_D)$. The derivation is as follows for $\vec{y} \sim \mathcal{N}(\vec{\mu}, \vec{\sigma}^2\mathbf{I})$, additionally for brevity we aggregate all factors into a single variable C:

Marginalizing over y_D yields the density of the "centered" density:

$$\begin{aligned}
p(\vec{y'}) &= \int_{-\infty}^{\infty} p(\vec{y'} \mid y_D)p(y_D)\,dy_D \\
&= \int_{-\infty}^{\infty}\left[\prod_{i\neq D}\left(2\pi\sigma^2\right)^{-\frac{1}{2}}\exp\left\{-\frac{1}{2\sigma^2}(y_i - y_D - \mu_i)^2\right\}\right]\left[\left(2\pi\sigma^2\right)^{-\frac{1}{2}}\exp\left\{-\frac{1}{2\sigma^2}(y_D - \mu_D)^2\right\}\right]dy_D
\end{aligned}$$

Expand terms:

$$\begin{aligned}
&= (2\pi\sigma^2)^{-\frac{D}{2}}\int_{-\infty}^{\infty}\left[\prod_{i\neq D}\exp\left\{-\frac{1}{2\sigma^2}(y_i^2 - 2y_i\mu_i + \mu_i^2)\right\}\right] \\
&\quad\times\exp\left\{-\frac{1}{2\sigma^2}\left(y_D^2 - 2y_D\mu_D + \mu_D^2 + (D-1)y_D^2 + 2y_D\sum_{i\neq D}\mu_i - y_i\right)\right\}dy_D
\end{aligned}$$

Reduce y_D terms:

$$= C \left[\prod_{i \neq D} \exp \left\{ -\frac{1}{2\sigma^2} (y_i - \mu_i)^2 \right\} \right]$$
$$\times \int_{-\infty}^{\infty} \exp \left\{ -\frac{1}{2\sigma^2} \left(y_D^2 - 2y_D\mu_D + \mu_D^2 + (D-1)y_D^2 + 2y_D \sum_{i \neq D} \mu_i - y_i \right) \right\} dy_D$$

Reduce terms:

$$= C \int_{-\infty}^{\infty} \exp \left\{ -\frac{1}{2\sigma^2} \left(Dy_D^2 - 2y_D \left(\mu_D + \sum_{i \neq D} y_i - \mu_i \right) + \mu_D^2 \right) \right\} dy_D$$

Complete the square:

$$= C \exp \left\{ -\frac{1}{2\sigma^2} \mu_D^2 \right\} \int_{-\infty}^{\infty} \exp \left\{ -\frac{1}{2\sigma^2} \left(Dy_D^2 - 2y_D \left(\mu_D + \sum_{i \neq D} y_i - \mu_i \right) \right) \right\} dy_D$$
$$= C \int_{-\infty}^{\infty} \exp \left\{ -\frac{1}{2\sigma^2} \left(\sqrt{D} y_D - \frac{\left(\mu_D + \sum_{i \neq D} y_i - \mu_i \right)}{\sqrt{D}} \right)^2 \right\} dy_D$$

The integrand is a Gaussian density in terms of y_D:

$$= (2\pi\sigma^2)^{-\frac{D}{2}} \left[\prod_{i \neq D} \exp \left\{ -\frac{1}{2\sigma^2} (y_i - \mu_i)^2 \right\} \right] \exp \left\{ -\frac{1}{2\sigma^2} \mu_D^2 \right\}$$
$$\times \exp \left\{ -\frac{1}{2\sigma^2} \frac{\left(\mu_D + \sum_{i \neq D} y_i - \mu_i \right)^2}{D} \right\} \sqrt{\frac{2\pi\sigma^2}{D}}$$

We can simplify this further using the fact that shifting the mean by any constant scalar does not affect the density due to the shift invariance of the softmax operation i.e.,

$p(\vec{y}|\vec{u}, \sigma^2) = p(\vec{y}|\vec{u} + c\vec{1}, \sigma^2)$, thus if we use $c = -\mu_D$ the density becomes:

$$= D^{-\frac{1}{2}} \left(2\pi\sigma^2\right)^{\frac{1-D}{2}} \exp\left\{-\frac{1}{2\sigma^2}\left[\sum_{i\neq D} (y_i - \mu_i + \mu_D)^2 - \frac{1}{D}\left(\sum_{i\neq D} y_i - \mu_i + \mu_D\right)^2\right]\right\}$$

Now that we have the density of $\vec{y'}$ we can use a straightforward application of the change of variables formula, with the known result that the determinant of the Jacobian of the softmax is $(\prod_i^D y_i)^{-1}$ [37, 38] to obtain:

$$p(\vec{y}|\vec{\mu}, \sigma^2) = D^{-\frac{1}{2}}(2\pi\sigma^2)^{\frac{1-D}{2}} \left(\prod_{i=1}^{D} y_i\right)^{-1}$$
$$\times \exp\left\{-\frac{1}{2\sigma^2}\left[\sum_{i\neq D}\left(\log\frac{y_i}{y_D} - \mu_i + \mu_D\right)^2 - \frac{1}{D}\left(\sum_{i\neq D}\log\frac{y_i}{y_D} - \mu_i + \mu_D\right)^2\right]\right\}$$

A.0.3 Derivation of Posterior for Guassian-Softmax

For the reverse process we sample from the posterior distribution $p(\vec{x}_{t-1}|\vec{x}_t, \vec{x}_0)$, using the same setup as in DDPM [23]:

$$p(\vec{x}_{t-1}|\vec{x}_t, \vec{x}_0) = \frac{p(\vec{x}_{t-1}, \vec{x}_t, \vec{x}_0)}{p(\vec{x}_t, \vec{x}_0)} = \frac{p(\vec{x}_t|\vec{x}_{t-1}, \vec{x}_0)p(\vec{x}_{t-1}, \vec{x}_0)}{p(\vec{x}_t, \vec{x}_0)} = \frac{p(\vec{x}_t|\vec{x}_{t-1}, \vec{x}_0)p(\vec{x}_{t-1}|\vec{x}_0)}{p(\vec{x}_t|\vec{x}_0)}$$

and due to the Markov property $p(\vec{x}_t|\vec{x}_{t-1}, \vec{x}_0) = p(\vec{x}_t|\vec{x}_{t-1})$ the posterior simplifies to:

$$p(\vec{x}_{t-1}|\vec{x}_t, \vec{x}_0) = \frac{p(\vec{x}_t|\vec{x}_{t-1})p(\vec{x}_{t-1}|\vec{x}_0)}{p(\vec{x}_t|\vec{x}_0)} \propto p(\vec{x}_t|\vec{x}_{t-1})p(\vec{x}_{t-1}|\vec{x}_0)$$

fortunately we have access to the necessary densities which are simply:

1.
$$p(\vec{x}_{t-1}|\vec{x}_0) \propto \left(\prod_{i}^{D} x_{t-1,i}\right)^{-1} \exp\left[-\frac{1}{2(1-\bar{a}_{t-1})}\sum_{i\neq D} v_i^2\right] \exp\left[\frac{1}{2D(1-\bar{a}_{t-1})}\left(\sum_{i\neq D} v_i\right)^2\right]$$

   where $v_i = \log\frac{x_{t-1,i}}{x_{t-1,D}} - \sqrt{\bar{a}_{t-1}}\log\frac{x_{0,i}}{x_{0,D}}$

2.
$$p(\vec{x}_t|\vec{x}_{t-1}) \propto \exp\left[-\frac{1}{2(1-a_t)}\sum_{i\neq D} r_i^2\right] \exp\left[\frac{1}{2D(1-a_t)}\left(\sum_{i\neq D} r_i\right)^2\right]$$

   where $r_i = \log\frac{x_{t,i}}{x_{t,D}} - \sqrt{a_t}\log\frac{x_{t-1,i}}{x_{t-1,D}}$

Focusing on the first exponential terms with simplified notation where $z_i = \log \frac{x_{t,i}}{x_{t,D}}$, $y_i = \log \frac{x_{t-1,i}}{x_{t-1,D}}$, $x_i = \log \frac{x_{0,i}}{x_{0,D}}$, $\sigma_t^2 = 1 - a_t$, and $\overline{\sigma}_{t-1}^2 = 1 - \overline{a}_{t-1}$

$$\exp\left[-\frac{1}{2(1-\overline{a}_{t-1})}\sum_{i\neq D} v_i^2\right]\exp\left[-\frac{1}{2(1-a_t)}\sum_{i\neq D} r_i^2\right]$$

$$= \exp\left\{-\frac{1}{2\overline{\sigma}_{t-1}^2}\sum_{i\neq D}\left(y_i - \sqrt{\overline{a}_{t-1}}x_i\right)^2 - \frac{1}{2\sigma_t^2}\sum_{i\neq D}\left(z_i - \sqrt{a_t}y_i\right)^2\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\sum_{i\neq D}\frac{1}{\overline{\sigma}_{t-1}^2}\left(y_i - \sqrt{\overline{a}_{t-1}}x_i\right)^2 + \frac{1}{\sigma_t^2}\left(z_i - \sqrt{a_t}y_i\right)^2\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\sum_{i\neq D}\frac{1}{\overline{\sigma}_{t-1}^2}\left(y_i^2 - 2y_i\sqrt{\overline{a}_{t-1}}x_i + \overline{a}_{t-1}x_i^2\right) + \frac{1}{\sigma_t^2}\left(z_i^2 - 2\sqrt{a_t}y_iz_i + a_ty_i^2\right)\right]\right\}$$

$$\propto \exp\left\{-\frac{1}{2}\left[\sum_{i\neq D}\frac{y_i^2}{\overline{\sigma}_{t-1}^2} - \frac{2y_i\sqrt{\overline{a}_{t-1}}x_i}{\overline{\sigma}_{t-1}^2} - \frac{2\sqrt{a_t}y_iz_i}{\sigma_t^2} + \frac{a_ty_i^2}{\sigma_t^2}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\sum_{i\neq D}\left(\frac{1}{\overline{\sigma}_{t-1}^2} + \frac{a_t}{\sigma_t^2}\right)y_i^2 - 2\left(\frac{\sqrt{\overline{a}_{t-1}}x_i}{\overline{\sigma}_{t-1}^2} + \frac{\sqrt{a_t}z_i}{\sigma_t^2}\right)y_i\right]\right\}$$

observe that we can read the posterior mean and variance as:

$$\mu_{t-1,i} = \left(\frac{\sqrt{\overline{a}_{t-1}}x_i}{\overline{\sigma}_{t-1}^2} + \frac{\sqrt{a_t}z_i}{\sigma_t^2}\right)\sigma_{t-1}^2 = \frac{\sqrt{a_t}(1-\overline{a}_{t-1})z_i + \sqrt{\overline{a}_{t-1}}(1-a_t)x_i}{1-\overline{a}_t}$$

$$\sigma_{t-1}^2 = \left(\frac{1}{\overline{\sigma}_{t-1}^2} + \frac{a_t}{\sigma_t^2}\right)^{-1} = \frac{(1-a_t)(1-\overline{a}_{t-1})}{1-\overline{a}_t}$$

since the form has to be proportional to $\exp\{-\frac{1}{2\sigma^2}(y_i - \mu)^2\}$.

Similarly for the second exponential term:

$$\exp\left[\frac{1}{2D(1-\overline{a}_{t-1})}\left(\sum_{i\neq D} v_i\right)^2\right]\exp\left[\frac{1}{2D(1-a_t)}\left(\sum_{i\neq D} r_i\right)^2\right]$$

$$= \exp\left\{\frac{1}{2D}\left[\frac{1}{\overline{\sigma}_{t-1}^2}\left(\sum_{i\neq D} y_i - \sqrt{\overline{a}_{t-1}}x_i\right)^2 + \frac{1}{\sigma_t^2}\left(\sum_{i\neq D} z_i - \sqrt{a_t}y_i\right)^2\right]\right\}$$

$$= \exp\left\{\frac{1}{2D}\left[\frac{1}{\overline{\sigma}_{t-1}^2}\left(\sum_{\mathbf{i\neq D}}(\mathbf{y_i} - \sqrt{\mathbf{\overline{a}_{t-1}}}\mathbf{x_i})^{\mathbf{2}} + 2\sum_{j<i,i\neq D}(y_i - \sqrt{\overline{a}_{t-1}}x_i)(y_j - \sqrt{\overline{a}_{t-1}}x_j)\right)\right.\right.$$

$$\left.\left. + \frac{1}{\sigma_t^2}\left(\sum_{\mathbf{i\neq D}}(\mathbf{z_i} - \sqrt{\mathbf{a_t}}\mathbf{y_i})^{\mathbf{2}} + 2\sum_{j<i,i\neq D}(z_i - \sqrt{a_t}y_i)(z_j - \sqrt{a_t}y_j)\right)\right]\right\}$$

The terms in bold correspond exactly with the first exponential term, and imply the same posterior mean and variance, further more the remaining terms are proportional to:

$$\exp\frac{1}{2D}\left[2\sum_{j<i,i\neq D}\frac{1}{\overline{\sigma}_{t-1}^2}\left(y_iy_j - \sqrt{\overline{a}_{t-1}}x_jy_i - \sqrt{\overline{a}_{t-1}}x_iy_j\right) + \frac{1}{\sigma_t^2}\left(-\sqrt{a_t}z_iy_j - \sqrt{a_t}z_jy_i + a_ty_iy_j\right)\right]$$

$$= \exp\frac{1}{2D}\left[2\sum_{j<i,i\neq D}\frac{1}{\overline{\sigma}_{t-1}^2}\left(y_iy_j - \sqrt{\overline{a}_{t-1}}x_jy_i - \sqrt{\overline{a}_{t-1}}x_iy_j\right) + \frac{1}{\sigma_t^2}\left(-\sqrt{a_t}z_iy_j - \sqrt{a_t}z_jy_i + a_ty_iy_j\right)\right]$$

$$= \exp\frac{1}{2D}\left[2\sum_{j<i,i\neq D}\left(\frac{1}{\overline{\sigma}_{t-1}^2} + \frac{a_t}{\sigma_t^2}\right)y_iy_j - \left(\frac{\sqrt{\overline{a}_{t-1}}x_j}{\overline{\sigma}_{t-1}^2} + \frac{\sqrt{a_t}z_j}{\sigma_t^2}\right)y_i - \left(\frac{\sqrt{\overline{a}_{t-1}}x_i}{\overline{\sigma}_{t-1}^2} + \frac{\sqrt{a_t}z_i}{\sigma_t^2}\right)y_j\right]$$

which again imply the same posterior mean and variance since the form has to be proportional to:

$$\exp\left\{2\left[\sum_{j<i,i\neq D}\frac{y_iy_j}{\sigma^2} - \frac{\mu_j}{\sigma^2}y_i - \frac{\mu_i}{\sigma^2}y_j + \frac{\mu_i\mu_j}{\sigma^2}\right]\right\}$$

Thus all terms agree on the same posterior mean and variance of:

$$\mu_{t-1,i} = \left(\frac{\sqrt{\overline{a}_{t-1}}x_i}{\overline{\sigma}_{t-1}^2} + \frac{\sqrt{a_t}z_i}{\sigma_t^2}\right)\sigma_{t-1}^2 = \frac{\sqrt{a_t}(1-\overline{a}_{t-1})z_i + \sqrt{\overline{a}_{t-1}}(1-a_t)x_i}{1-\overline{a}_t}$$

$$\sigma_{t-1}^2 = \left(\frac{1}{\overline{\sigma}_{t-1}^2} + \frac{a_t}{\sigma_t^2}\right)^{-1} = \frac{(1-a_t)(1-\overline{a}_{t-1})}{1-\overline{a}_t}$$

We can simplify the posterior mean by utilizing the shift invariance property, where we observe that:

$$
\begin{aligned}
\mu_{t-1,i} &= \frac{\sqrt{a_t}(1-\overline{a}_{t-1})(\log x_{t,i} - \log x_{t,D}) + \sqrt{\overline{a}_{t-1}}(1-a_t)(\log x_{0,i} - \log x_{0,D})}{1-\overline{a}_t} \\
&= \frac{\sqrt{a_t}(1-\overline{a}_{t-1})\log x_{t,i} + \sqrt{\overline{a}_{t-1}}(1-a_t)\log x_{0,i}}{1-\overline{a}_t} - \frac{\sqrt{a_t}(1-\overline{a}_{t-1})\log x_{t,D} + \sqrt{\overline{a}_{t-1}}(1-a_t)\log x_{0,D}}{1-\overline{a}_t} \\
&= \frac{\sqrt{a_t}(1-\overline{a}_{t-1})\log x_{t,i} + \sqrt{\overline{a}_{t-1}}(1-a_t)\log x_{0,i}}{1-\overline{a}_t} + C
\end{aligned}
$$

so the posterior mean can be simplified as:

$$
\mu_{t-1,i} = \frac{\sqrt{a_t}(1-\overline{a}_{t-1})\log x_{t,i} + \sqrt{\overline{a}_{t-1}}(1-a_t)\log x_{0,i}}{1-\overline{a}_t}
$$

Since all the terms agree on the same posterior mean and variance, and furthermore the posterior density has the same form as the Gaussian-Softmax distribution, we can conclude that $p(\vec{x}_{t-1}|\vec{x}_t, \vec{x}_0) = p(\vec{x}_{t-1}|\vec{\mu}_{t-1}, \sigma^2_{t-1}\mathbf{I})$

A.0.4 Derivation of Variance Schedule Augmentation

As shown in Figure 3.6, we need to augment our chosen variance schedule to ensure that the class labels are gradually noised. Taking inspiration from Categorical diffusion, our desideratum is to smoothly noise the class label such that the argmax of $\mathbf{x}_t$ follows the distribution $\mathcal{C}(\overline{b_t}\mathbf{x}_0 + \frac{1}{D}(1-\overline{b_t})\mathbf{1})$ where $\mathcal{C}$ is the categorical distribution and $b_0, b_1, \ldots, b_T$ is a variance schedule of our choosing. Unfortunately there is no closed form formula to determine the argmax of a Gaussian vector, so we instead approximate a Gaussian vector with a Gumbel vector. A useful property of the Gumbel distribution is that it can be used to reparameterize the Categorical distribution where $\text{argmax}\{a \log \mathbf{p} + \mathbf{g}\} \sim \mathcal{C}(\text{softmax}\{a \log \mathbf{p}\})$, $\mathbf{g} \sim \mathcal{G}(0,1)$, and $\mathcal{G}$ is the Gumbel distribution [39]. Then considering the forward process in Gaussian-Softmax diffusion we can derive:

$$
\text{argmax}\{\mathbf{x}_t\} \approx \text{argmax}\left\{\text{softmax}\left(\sqrt{\overline{\alpha_t}}\log\left(k\mathbf{x}_0 + \frac{1-k}{D}\right) + \sqrt{1-\overline{\alpha_t}}\mathbf{g}\right)\right\}.
$$

Simplifying the expression inside the argmax:

$$
\begin{aligned}
&= \text{argmax}\left\{\sqrt{\overline{\alpha_t}}\log\left(k\mathbf{x}_0 + \frac{1-k}{D}\mathbf{1}\right) + \sqrt{1-\overline{\alpha_t}}\mathbf{g}\right\} \\
&= \text{argmax}\left\{\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\log\left(k\mathbf{x}_0 + \frac{1-k}{D}\mathbf{1}\right) + \mathbf{g}\right\} \sim \text{softmax}\left\{\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\log\left(k\mathbf{x}_0 + \frac{1-k}{D}\mathbf{1}\right)\right\}
\end{aligned}
$$

We aim to satisfy:

$$\text{softmax}\left\{\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\log\left(k\mathbf{x}_0+\frac{1-k}{D}\mathbf{1}\right)\right\}=\bar{b}_t\mathbf{x}_0+\frac{1-\bar{b}_t}{D}\mathbf{1}$$

Taking the logarithm of both sides gives:

$$\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\log\left(k\mathbf{x}_0+\frac{1-k}{D}\mathbf{1}\right)+\mathbf{c}=\log\left(\bar{b}_t\mathbf{x}_0+\frac{1-\bar{b}_t}{D}\mathbf{1}\right)$$

Assuming without loss of generality that $\mathbf{x}_0=[1,0,\ldots,0]$, we have:

$$\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\left[\log\left(k+\frac{1-k}{D}\right),\log\left(\frac{1-k}{D}\right),\ldots\right]+\mathbf{c}=\left[\log\left(\bar{b}_t+\frac{1-\bar{b}_t}{D}\right),\log\left(\frac{1-\bar{b}_t}{D}\right),\ldots\right]$$

Since $\mathbf{c}$ is a free parameter, we can set:

$$\mathbf{c}=\left[\log\left(\bar{b}_t+\frac{1-\bar{b}_t}{D}\right)-\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\log\left(k+\frac{1-k}{D}\right)\right]\mathbf{1}$$

This reduces the equation to:

$$\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}\left[0,\log\left(\frac{1-k}{(D-1)k+1}\right),\ldots\right]=\left[0,\log\left(\frac{1-\bar{b}_t}{(D-1)\bar{b}_t+1}\right),\ldots\right]$$

Thus, we deduce:

$$\sqrt{\frac{\overline{\alpha_t}}{1-\overline{\alpha_t}}}=\log\left(\frac{1-\bar{b}_t}{(D-1)\bar{b}_t+1}\right)/\log\left(\frac{1-k}{(D-1)k+1}\right)$$

Finally, isolating $\overline{\alpha_t}$ yields:

$$\overline{\alpha_t}=\frac{n^2}{n^2+m^2},\text{where}\quad n=\log\left(\frac{1-\bar{b}_t}{(D-1)\bar{b}_t+1}\right),\quad m=\log\left(\frac{1-k}{(D-1)k+1}\right).$$

A.0.5 Model Architecture

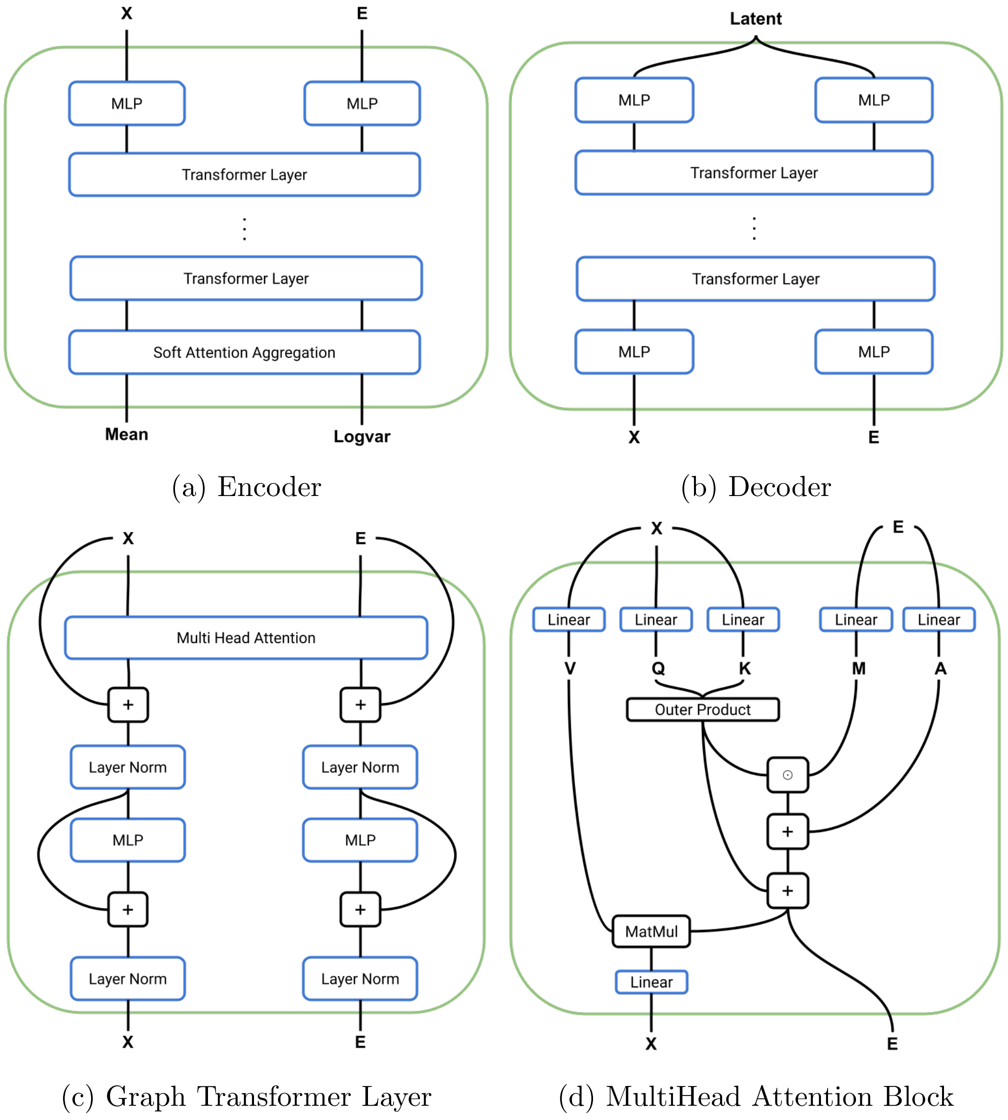


Figure A.1: Our VAE Architecture is heavily influenced off of Digress by Vignac et al., in particular we use the same multihead attention block and transformer layer.

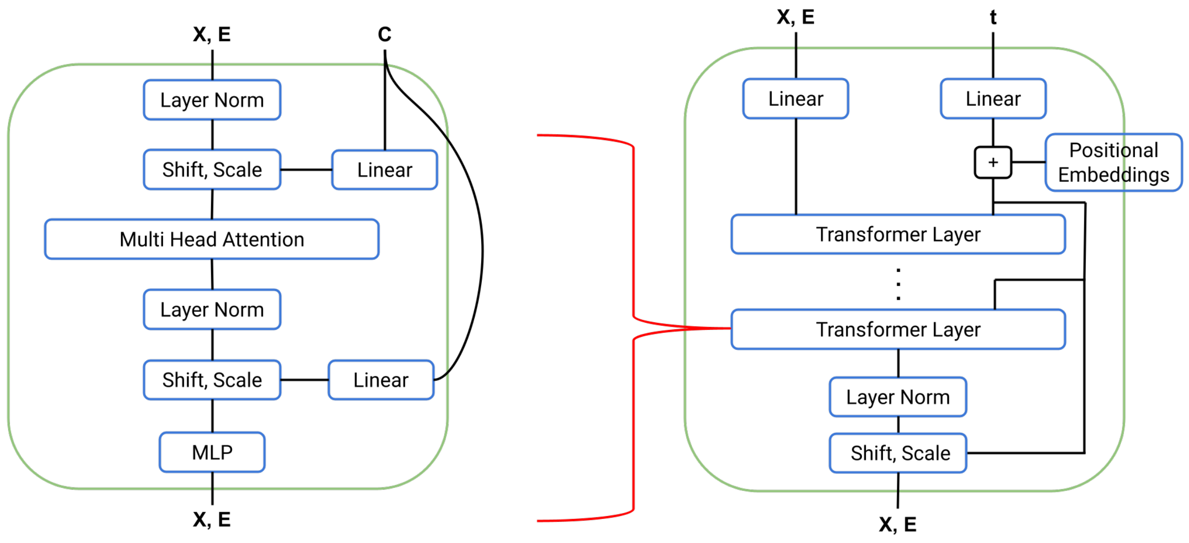
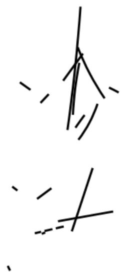
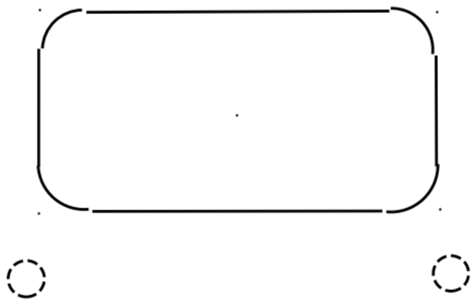


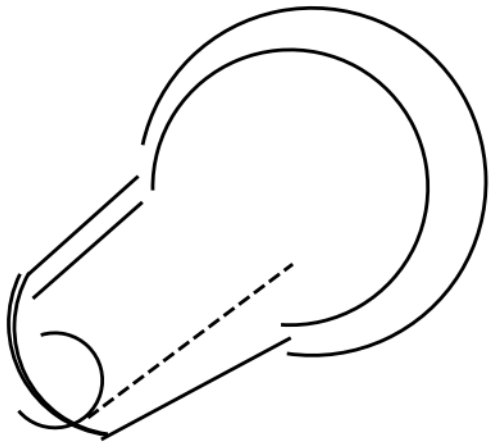
Figure A.2: The network architecture of diffusion denoiser network is a slight modification off of the DiT architecture introduced by Peebles & Xie, where we simply removed the gate scales.



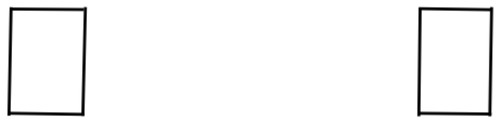
(a) Primitive generation has no discernible pattern.



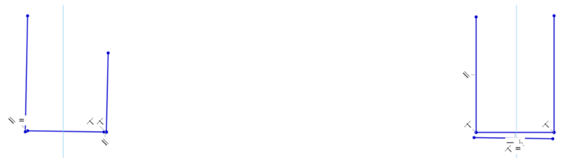
(b) Gaps exist between primitive terminations.



(c) Large gaps between termination along with extraneous primitives.



(d) Before constraint enforcement.



(e) After constraint enforcement in Onshape's constraint solver. Generated constraints damage sketch fidelity.

Figure A.3: We present the failure cases that we've experienced concerning our diffusion model. The most common of which are constraint failures and gaps between primitive terminations. We believe this to be an artifact of treating primitive parameters as continuous values.

Table A.1: A table of the most common primitives and a visualization to describe their semantics.

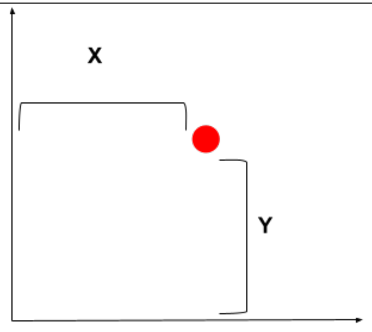
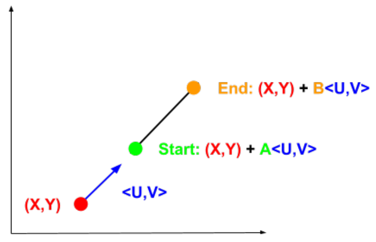
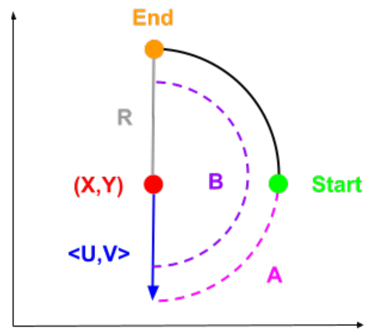
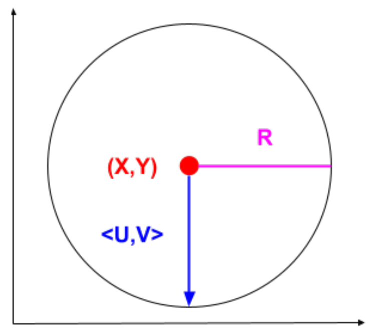
primitive	parameters	visualization
point	x(x-coordinate), y(y-coordinate)	
line	x, y, u, v, a, b	 End: (X,Y) + B<U,V> Start: (X,Y) + A<U,V> <U,V> (X,Y)
arc	x, y, u, v, r, c, a, b	
circle	x, y, u, v, r, c	

Table A.2: A table of the most common constraints and a visualization to describe their semantics.

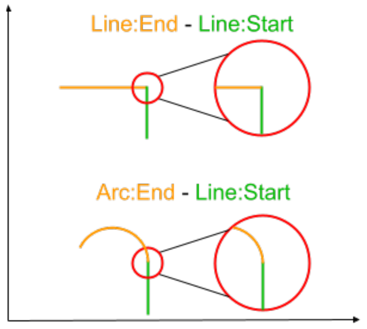
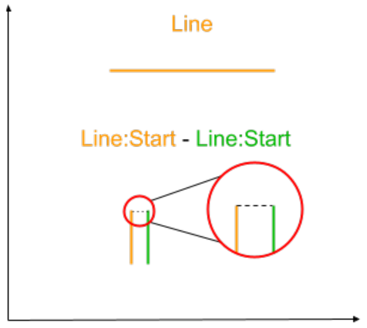
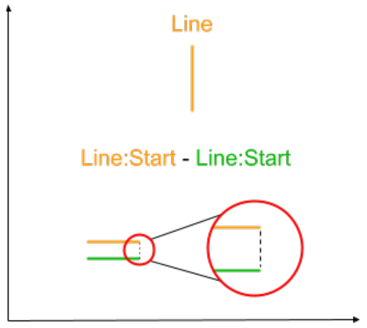
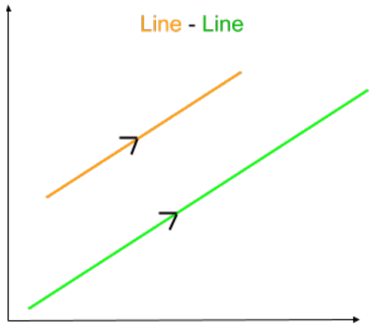
constraint	description	visualization
coincident	restricts two points to share the same position in space	
horizontal	either aligns a line to the x-axis or restricts two points to have the same y-coordinate	
vertical	either aligns a line to the y-axis or restricts two points to have the same x-coordinate	
parallel	aligns two lines to be parallel	

Table A.3: (Continued) A table of the most common constraints and a visualization to describe their semantics.

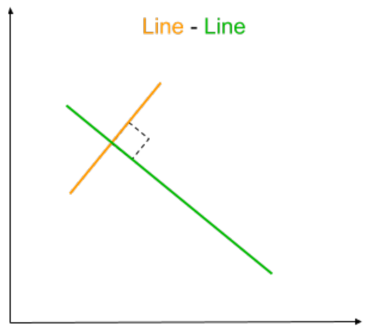
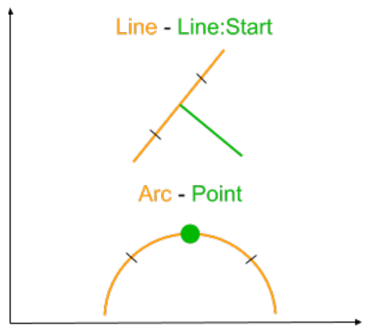
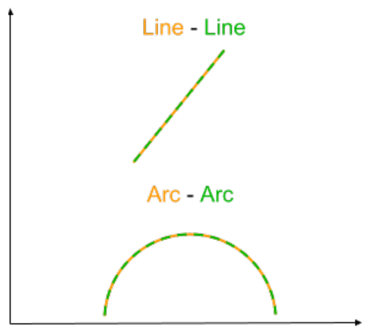
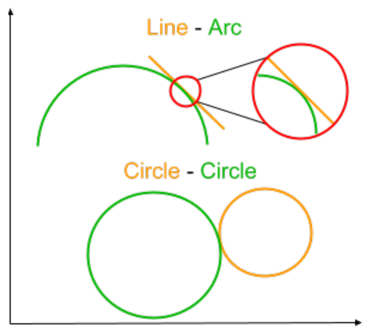
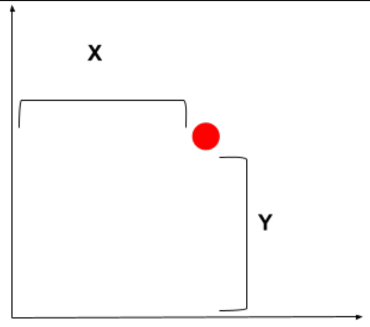
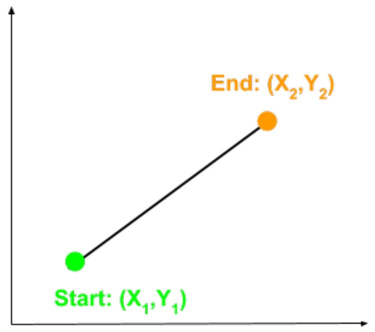
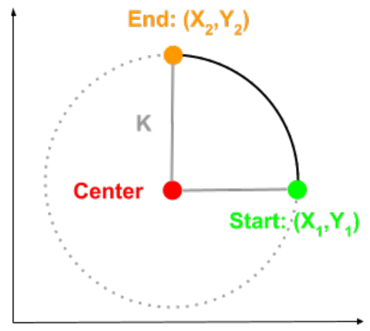
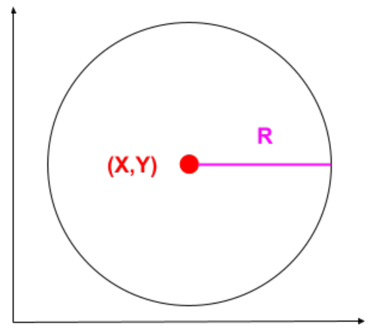
constraint	description	visualization
perpendicular	aligns two lines to be orthogonal	
midpoint	restricts a point to be the midpoint of an arc or line, or restricts a point to the center of a circle	
equal	restricts two primitives to have the same parameters	
tangent	restricts a line to be tangent to a circle/arc or two curves to be tangent to one another	

Table A.4: A table of the primitive reparameterizations we employ and a visualization to describe their semantics. For arcs κ is the curvature of the osculating circle, negating κ reflects the arc center point across the line formed by the endpoints.

primitive	parameters	visualization
point	x, y	
line	x_1, y_1, x_2, y_2	
arc	x_1, y_1, x_2, y_2, κ	
circle	x, y, r	

References

[1] Monika Dyavenahalli Shivegowda, Pawinee Boonyasopon, Sanjay Mavinkere Rangappa, and Suchart Siengchin. A Review on Computer-Aided Design and Manufacturing Processes in Design and Architecture. *Archives of Computational Methods in Engineering*, 29(6):3973–3980, October 2022.

[2] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5885–5897. Curran Associates, Inc., 2021.

[3] Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. Sketchgen: generating constrained cad sketches. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2024. Curran Associates Inc.

[4] Autodesk. https://www.autodesk.com/products/autocad/overview, March 2023. Date Accessed: 2023-03-19.

[5] Solidworks. https://www.solidworks.com/domain/design-engineering, March 2023. Date Accessed: 2023-03-19.

[6] Onshape. https://www.onshape.com/en/, March 2023. Date Accessed: 2023-03-19.

[7] Tom Veuskens, Florian Heller, and Raf Ramakers. Coda: A design assistant to facilitate specifying constraints and parametric behavior in cad models. In *Graphics Interface*, 2021.

[8] Sofia Kyratzi and Philip Azariadis. A Constraint-based Framework to Recognize Design Intent during Sketching in Parametric Environments. *Computer-Aided Design and Applications*, 18:545–560, September 2020.

[9] Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. Sketchgraphs: A large-scale dataset for modeling relational geometry in computer-aided design. *ArXiv*, abs/2007.08506, 2020.

[10] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric CAD sketches. In *International Conference on Learning Representations*, 2022.

[11] Yang Jing and Yang Song. Application of 3d reality technology combined with cad in animation modeling design. *Computer-Aided Design and Applications*, 18(S3):164–175, 2020.

[12] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023.

[13] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 412–422, Cham, 2018. Springer International Publishing.

[14] Bohdan Macukow. Neural Networks – State of Art, Brief History, Basic Models and Architecture. In Khalid Saeed and Władysław Homenda, editors, *Computer Information Systems and Industrial Management*, Lecture Notes in Computer Science, pages 3–14, Cham, 2016. Springer International Publishing.

[15] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, January 1982.

[16] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. Place: US Publisher: American Psychological Association.

[17] Bernhard Bettig and Christoph M Hoffmann. Geometric constraint solving in parametric computer-aided design. *Journal of computing and information science in engineering*, 11(2), 2011.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, January 1991.

[19] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, June 1993.

[20] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[21] Stanley H. Chan. Tutorial on diffusion models for imaging and vision. *Found. Trends Comput. Graph. Vis.*, 16:322–471, 2024.

[22] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[24] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, 18–24 Jul 2021.

[25] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[26] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[27] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. Diffusion-LM improves controllable text generation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[28] Hongyi Yuan, Zheng Yuan, Chuanqi Tan, Fei Huang, and Songfang Huang. Seqdiffuseq: Text diffusion with encoder-decoder transformers. *ArXiv*, abs/2212.10325, 2022.

[29] Peiyu Yu, Sirui Xie, Xiaojian Ma, Baoxiong Jia, Bo Pang, Ruiqi Gao, Yixin Zhu, Song-Chun Zhu, and Ying Nian Wu. Latent diffusion energy-based model for interpretable text modelling. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 25702–25720. PMLR, 17–23 Jul 2022.

[30] Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Annual Meeting of the Association for Computational Linguistics*, 2022.

[31] Ting Chen, Ruixiang ZHANG, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *The Eleventh International Conference on Learning Representations*, 2023.

[32] J. Aitchison and S. M. Shen. Logistic-normal distributions: Some properties and uses. *Biometrika*, 67(2):261–272, 1980.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[34] William S. Peebles and Saining Xie. Scalable diffusion models with transformers. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4172–4182, 2022.

[35] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7441–7451, October 2023.

[36] Kai Wang, Yukun Zhou, Mingjia Shi, Zhihang Yuan, Yuzhang Shang, Xiaojiang Peng, Hanwang Zhang, and Yang You. A closer look at time steps is worthy of triple speed-up for diffusion model training. *ArXiv*, abs/2405.17403, 2024.

[37] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[38] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[39] Iris A. M. Huijben, Wouter Kool, Max B. Paulus, and Ruud J. G. van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1353–1371, 2023.