Diffusion Modelling for Stochastic Dependence

December 5, 2024

Acknowledgement

Firstly, I would like to thank Dr Marius Hofert, who guided me in finishing this daunting challenge with his insightful advice. Without his continued support, I would not be able to complete such a monumental task. I would like to express my heartfelt gratitude to other lecturers in HKU who have taught me. Without your excellent lectures and well-thought explanations, I would not be able to gain such insight into statistics.

I would also like to take this chance to thank all my friends for their moral support and ideas for my project, inspiring me to achieve greater heights in my budding career as an actuary. Thanks for continuing to send job descriptions for me to apply and giving me the strength to fail again and again, without your help I would not have gotten a job.

I intend to use this chance to thank my family too as I could never thank them enough, without their sacrifices and encouragement throughout the years I would have failed countless times. I am grateful for your unconditional support in all my decisions and also your unwavering faith in me.

Contents

1 Introduction

With increasing access to high-dimensional data due to technological and recent algorithmic advances, the ability to model and study the dependence between different variables is of utmost importance. To study the dependence between random variables, copulas are often the tool of choice due to their ease of interpretation and flexibility.

However, copulas are often not flexible enough to model real-life data, making it necessary to use other methods to model dependence. With the advent of deep learning, many have proposed using neural networks to learn the dependence between random variables. For example, a recent breakthrough by Hofert, Prasad, and Zhu [HPZ21] discusses the use of neural networks to learn the dependence between two random vectors.

Although the use of such deep generative models to generate samples from a target distribution has been widely studied from the perspective of generating new photos and videos, less work has been done on the applications of such models to replace traditional statistical modelling techniques. In this report, we will extend Hofert, Prasad, and Zhu [HPZ21]'s work and explore the use of diffusion models as generative models to simulate samples of random variables.

1.1 Existing Difficulties

Previous attempts at training neural networks did not yield encouraging results, especially when relying on the mean squared error (MSE) loss function. The MSE, while being computationally efficient with a complexity of $O(n)$, is unable to learn the underlying characteristics of complex distributions. For instance, the neural network fails to capture the general dependence structure and also the tail structure which is vital for copula modelling.

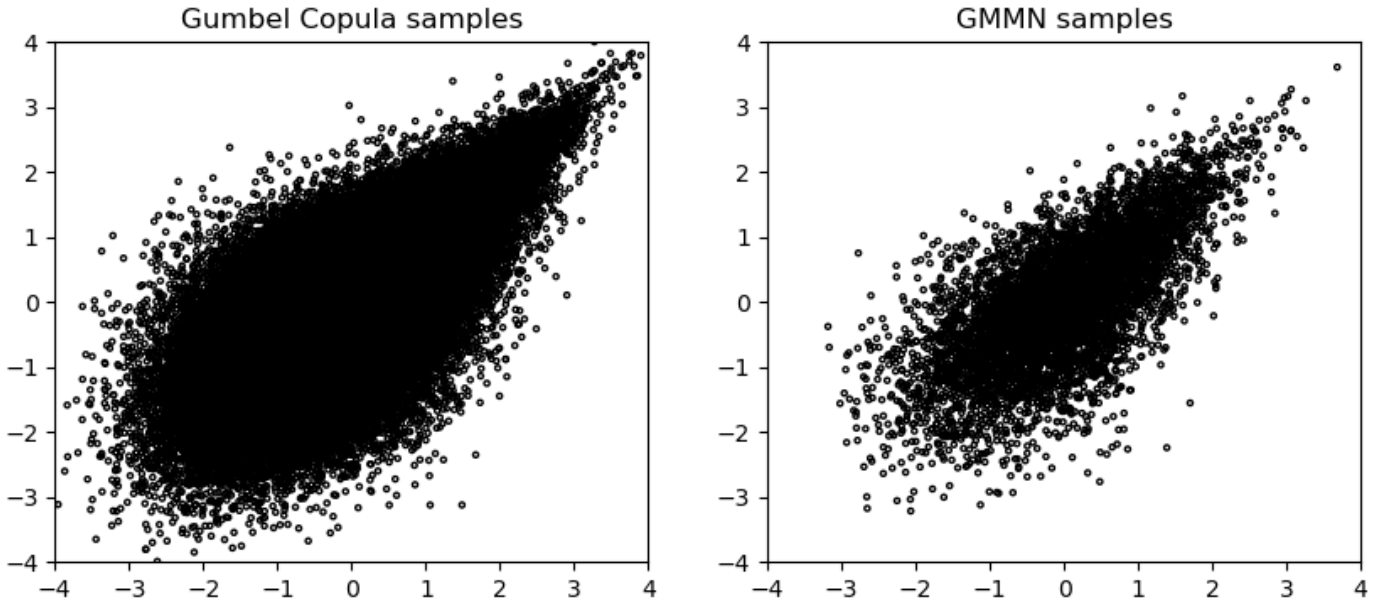


Figure 1: Comparison of samples simulated from Gumbel copula with $\theta = 2$ and GMMN trained with MSE

As observed in Figure 1, the Generative Moment Matching Network (GMMN) trained with the MSE loss function simulated samples clustered around the mean and did not exhibit similar diversity in simulated samples compared to a copula.

We suspect that this issue arises due to the MSE's tendency to minimize the distance between predicted and true values. As a result, it tends to generate samples close to the mean of the target

distribution. This behavior of the neural network is undesirable as copulas are used for modelling dependence structure for extreme events, which are vital for applications like risk management and modelling complex real-life phenomena.

Although this problem has been resolved in Hofert, Prasad, and Zhu [HPZ21] by using the maximum mean discrepancy (MMD) as a loss function to learn distributions, the MMD has a running time of $O(N^2)$ compared to $O(N)$ for MSE, making it difficult to scale up training efficiently. The MMD works by matching the moments of the distributions through the reproducing kernel Hilbert space (RKHS). This helps to learn the distribution of the copula properly.

1.2 Our Contribution

We investigate an alternative to GMMN with MMD to learn copulas by leveraging the strength of diffusion models. We propose a procedure inspired by diffusion models to learn the dependence structure by using the MSE. Using the MSE is less computationally intensive due to its complexity of $O(n)$. This allows us to use more data in training and scale up to higher dimensions as we do not need to use the MMD.

We also show that our model has the ability to generate quasi-random samples from the distribution the neural network is trained on. After training the neural network on losses from dependent factors, we can then generate quasi-random samples from the underlying loss distribution for VaR or expected shortfall calculations. This has the potential to improve existing risk management practices as we are able to build better models to model dependent risks.

2 Preliminaries

2.1 Copulas

Using a multivariate distribution to model high-dimensional data can be restrictive and cumbersome since it requires us to specify a joint density that accurately reflects the dependence and margins. Consequently, copulas have been a popular alternative to model multivariate data.

As stated in Joe [Joe14], copulas can be used in different settings where the multivariate data has a dependence structure. For instance, insurance claims with 2 types of losses, financial returns, item response in a survey, etc.

Copulas can be defined as a multivariate distribution with dependent $U(0,1)$ random margins. They map arbitrary random variables to uniform margins, simplifying dependence modelling to the modelling of uniform margins. To explain how copulas work, we start by transforming variables into uniform margins through the probability integral transform:

Theorem 2.1 (Probability Integral Transform)
Suppose a random variable X has a continuous distribution with cumulative distribution function F_X. Then the random variable U, defined by $U := F_X(X)$ follows a standard uniform distribution.

In the following section, we present the relationship between joint distributions and copulas.

Theorem 2.2 (Sklar's Theorem)
As mentioned in Joe [Joe14], multivariate distributions can be characterized by copulas and its univariate margins which is shown in the following theorem:

$$F(x_1, x_2, \ldots, x_d) = C(u_1, u_2, \ldots u_d) \tag{1}$$

where $x_1, x_2, \ldots, x_d \in \mathbb{R}$, $u_1, u_2 \ldots u_d \in [0,1]$, and $u_i = F_{X_i}(x_i)$
Proof:

$$\begin{aligned} F(x_1, x_2, \ldots, x_d) &= P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_d \leq x_d) \\ &= P(F(X_1) \leq F(x_1), F(X_2) \leq F(x_2), \ldots F(X_d) \leq F(x_d)), \quad x_1, x_2, \ldots, x_d \in \mathbb{R} \\ &= P(U_1 \leq u_1, U_2 \leq u_2, \ldots U_d \leq u_d) \\ &= C(u_1, u_2, \ldots u_d), \quad u_1, u_2 \ldots u_d \in [0,1] \end{aligned} \tag{2}$$

Armed with Sklar's theorem, we can separate the modeling of the dependence and marginals, thus making the modelling more flexible.

Algorithm 2.1 Modelling Multivariate Random Variables with Copulas

Consider a random vector $X = (X_1, X_2, \ldots, X_d)$

1: Model the individual X_i seperately as univariate distributions, we get $F_{X_1}, F_{X_2}, \ldots, F_{X_d}$
2: Transform the individual X_i's in uniform margins with their cdf, $U_i = F_{X_i}(X_i)$
3: Choose a copula $C(U_1, U_2, \ldots, U_d)$ to model $U = (U_1, U_2, \ldots, U_d)$

This report will mainly focus on the more commonly used copulas, such as the Gaussian and Gumbel, and will explore how to learn such copulas with neural networks. To investigate the limits of diffusion models in learning distributions, we also consider the Marshall-Olkin copula, which has a singular component and is much more challenging to model.

2.2 Diffusion Models

Diffusion models are a subset of deep generative models consisting of different neural network based models, such as generative adversarial network (GAN), autoregressive models, and variational autoencoders (VAE) used for image and audio synthesis. Bishop and Bishop [BB23] describes deep generative models as models that utilize deep learning to mimic the distribution of the training data and then simulate new samples from the underlying distribution. As stated in Ho, Jain, and Abbeel [HJA20] diffusion models perform surprisingly well compared to GANs which are significantly harder to train. This section will briefly introduce diffusion models, from training to sampling.

2.2.1 Forward Encoder

Consider an image, x from the training set and corrupt it by independent Gaussian noise to create a noisy representation

$$z_1 = \sqrt{1-\beta_1}x + \sqrt{\beta_1}\epsilon_1 \tag{3}$$

where $\epsilon_1 \sim \mathcal{N}(\epsilon|0, I)$ and $\beta < 1$ is the variance of the noise. Therefore, z_1 is distributed with

$$q(z_1|x) = \mathcal{N}(z_1|\sqrt{1-\beta_1}x, \beta_1 I) \tag{4}$$

The process is then repeated by adding independent Gaussian noise at each step $t = 1, 2, \ldots, T$, thus creating noisy images $z_1, z_2, \ldots, z_T$ with a relationship between successive pair of images of

$$\begin{aligned} z_t &= \sqrt{1-\beta_1}z_{t-1} + \sqrt{\beta_1}\epsilon_t \\ q(z_t|z_{t-1}) &= \mathcal{N}(z_t|\sqrt{1-\beta_t}z_{t-1}, \beta_t I) \end{aligned} \tag{5}$$

The variance parameters β_t are usually chosen such that $\beta_1 < \beta_1 < \ldots < \beta_T$.

After specifying the forward process for each timestep, we investigate the diffusion kernel $q(z_t|x)$ for the entire forward process. The joint distribution of the noisy latent variables given the data can be written as

$$q(z_1, \ldots, z_T|x) = q(z_1|x) \prod_{t=2}^{T} q(z_t|z_{t-1}) \tag{6}$$

To obtain the diffusion kernel, $q(z_T|x)$, we start with $z_2|x$ and generalise the pattern to T. This can be greatly simplified by the property that the normal distribution is a conjugate prior to another normal as shown below

$$\begin{aligned} P(z_2|x) &= \int P(z_2|z_1)P(z_1|x)dz_1 \\ &\propto e^{-\frac{(z_2-\sqrt{1-\beta_2}z_1)^2}{2\beta_2}} e^{-\frac{(z_1-\sqrt{1-\beta_1}x)^2}{2\beta_1}} \\ &\text{After integrating out } z_1\text{, we obtain} \\ &\propto e^{-\frac{1}{2\beta_2}(z_2^2 - 2\sqrt{1-\beta_2}z_1 z_2 + \ldots)} \end{aligned} \tag{7}$$

We can then observe $z_2|x$ follows a normal distribution with mean and variance stated below

$$\begin{aligned} \text{with } z_2|z_1, x \sim z_2|x &\sim \mathcal{N}(\sqrt{1-\beta_2}z_1, \beta_2 I) \\ E(z_2|x) &= E[E(z_2|x, z_1)] \\ &= \sqrt{1-\beta_2}\sqrt{1-\beta_1}x \\ Var(z_2|x) &= E[Var(z_2|z_1, x)] + Var[E(z_2|z_1, x)] \\ &= E(\beta_2) + Var(\sqrt{1-\beta_2}z_1) \\ &= 1 - (1-\beta_1)(1-\beta_2) \end{aligned} \tag{8}$$

Making use of the conjugate prior property and by repeating the previous process of marginalizing over all latent variables, we obtain the diffusion kernel, $q(z_t|x)$

$$q(z_t|x) = \mathcal{N}(z_t|\sqrt{\alpha_t}x, (1-\alpha_t)I) \tag{9}$$

where $\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$. Hence, we are able to write $z_t = \sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon_t$, allowing us to perform the diffusion with only one step.

As many steps are taken, the information in the original data is completely corrupted, and as $T \to \infty$ z_T is indiscernible from Gaussian noise

$$\begin{aligned} q(z_T|x) &= q(z_T) \\ &\sim \mathcal{N}(z_T|0, I) \end{aligned} \tag{10}$$

2.2.2 Distribution of the Reverse Process

After observing the forward process consisting of sequentially adding noise to corrupt data, it would be natural to ask about how we can invert the process to reconstruct data. From Bishop and Bishop [BB23], we can consider the reverse conditional distribution $q(z_{t-1}|z_t)$ which can be expressed as

$$q(z_{t-1}|z_t) = \frac{q(z_t|z_{t-1})q(z_{t-1})}{q(z_t)} \tag{11}$$

where $q(z_{t-1})$ can be written in the form

$$q(z_{t-1}) = \int q(z_{t-1}|x)p(x)dx \tag{12}$$

However, $p(x)$ is unknown to us. Therefore, we instead consider the reverse distribution conditional on x, allowing us to derive the reverse distribution as shown below

$$\begin{aligned} q(z_{t-1}|z_t, x) &= \frac{q(z_{t-1}|z_t, x)q(z_{t-1}|x)}{q(z_t|x)} \\ &= q(z_t|z_{t-1}) \text{ from the Markov property} \\ q(z_t|z_{t-1}) &\propto e^{-\frac{(z_t-\sqrt{1-\beta_t}z_{t-1})^2}{2\beta_t}} e^{-\frac{(z_{t-1}-\sqrt{1-\alpha_t}x)^2}{2(1-\alpha_{t-1})}} \\ &\propto e^{-\frac{1}{2}(\frac{1-\beta_t}{\beta_t}+\frac{1}{1-\alpha_{t-1}})z_{t-1}^2-\frac{1}{2}z_{t-1}\left[\frac{-2\sqrt{1-\beta_t}z_t}{\beta_t}\frac{-2\sqrt{1-\alpha_t}x}{1-\alpha_{t-1}}\right]+\ldots} \end{aligned} \tag{13}$$

which implies it is normally distributed with $N(z_{t-1}|\mu(x, z_t), {\sigma_t}^2 I)$, where

$$\begin{aligned} \mu(x, z_t) &= \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}z_t + \sqrt{\alpha_{t-1}}\beta_t x}{1-\alpha_t} \\ \sigma_t &= \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t} \end{aligned} \tag{14}$$

2.2.3 Reverse Decoder

As shown above, the reverse conditional distribution $q(z_{t-1}|z_t)$ is intractable since it requires us to integrate over the unknown $q(x)$ as shown in Bishop and Bishop [BB23]. Therefore, we use a deep neural network to model the reverse process $z_{t-1}|z_t$. With the neural network, we can sample from the diffusion model by sampling Gaussian noise, then passing the noise through the neural network to produce samples in the from the target distribution $p(x)$.

While setting up the forward process, we needed to choose the variance parameter, β_t beforehand. The choice of β_t is important as a small value makes the process easier to invert but a large value corrupts the data quicker and less steps are needed.

In practice, a smaller β_t is used due to its ease of inverting the noise process. An intuitive way to think about why smaller values of β_t are chosen is by inverting Equation 5

$$z_{t-1} = \frac{1}{\sqrt{1-\beta_t}}z_t - \sqrt{\frac{\beta_t}{1-\beta_t}}\epsilon_t \tag{15}$$

From this equation we observe that with small β_t, $Var(z_{t-1}|z_t) \approx \beta_t$, making it possible to model the reverse process with a Gaussian distribution

$$p(z_{t-1}|z_t, w) = \mathcal{N}(z_{t-1}|\mu(z_t, w, t), \beta_t I) \tag{16}$$

where $\mu(z_t, w, t)$ is a neural network parameterized with z_t and t. This allows us to account for the variances β_t and use the same neural network for all steps. The neural network used is usually a U-net but this our case we decided to use an MLP instead due to better interpretability.

Bishop and Bishop [BB23] also presents an alternate way of understanding why small values of β_t from the perspective of prior distributions.From Equation (11), we observe that

$$q(z_{t-1}|z_t) \propto q(z_t|z_{t-1})q(z_{t-1}) \tag{17}$$

If β_t is large, the density of $q(z_t|z_{t-1})$ will be spread over a large area, causing the density $q(z_{t-1}|z_t)$ to be similar to the prior $q(z_{t-1})$. However, if the density of $q(z_t|z_{t-1})$ is peaked, the density of $q(z_{t-1}|z_t)$ is close to a Gaussian instead. A detailed explanation for small values of β_t can be found in Bishop and Bishop [BB23].

2.2.4 Training with the Evidence Lower Bound

We can write the log-likelihood of x, $\ln p(x|w)$ as the sum of the ELBO, $\mathcal{L}(w)$ and Kullback-Leibner divergence

$$\ln p(x|w) = \mathcal{L}(w) + KL(q(z)||p(z|x,w)) \tag{18}$$

where the evidence lower bound or variational lower bound is defined by, $\mathcal{L}$

$$\mathcal{L}(w) = \int q(z) \ln \frac{p(x,z|w)}{q(z)} dz \tag{19}$$

and the Kullback-Leibner divergence $KL(f||g)$ between pdf $f(z)$ and $g(z)$ defined by

$$KL(f(z)||g(z)) = -\int f(z) \ln \frac{g(z)}{f(z)} dz \tag{20}$$

The ELBO can be interpreted as a lower bound of the log-likelihood of the data. Since the KL divergence is always > 0, we have

$$\ln p(x|w) \geq \mathcal{L}(w) \tag{21}$$

Therefore, maximizing the ELBO implies that we maximize the log-likelihood and also the KL divergence.This way, we can improve the overall fit (log-likelihood) and also the fit between internal components of the model (KL divergence).

The KL divergence can be interpreted as the difference between the cross entropy between $f(z)$ and $g(z)$ and also entropy of $f(z)$, acting as a measure of "distance" between two distributions. We can also understand it as the information lost if we use $g(z)$ to approximate $f(z)$.

In this setup, we are trying to use $q(z)$ to approximate $p(z|x,w)$. Since the log-likelihood is intractable, we take inspiration from variational autoencoders and maximize the evidence lower bound (ELBO) to train the neural network. Substituting $z = (z_1, \ldots, z_T)$ into z, we obtain

$$\begin{aligned}
\mathcal{L}(w) &= E_q \left[\ln \frac{p(z_T, \ldots, z_1, x|w)}{q(z_T, \ldots, z_1|x)} \right] \\
&= E_q \left[\ln \frac{p(z_T)\{\prod_{t=2}^{T} p(z_{t-1}|z_t, w)\} p(x|z_1, w)}{q(z_1|x) \prod_{t=2}^{T} q(z_t|z_{t-1}, x)} \right] \\
&= E_q \left[\ln p(z_T) + \sum_{t=2}^{T} \ln \frac{p(z_{t-1}|z_t, w)}{q(z_t|z_{t-1}, x)} - \ln q(z_1|x) + \ln p(x|z_1, w) \right]
\end{aligned} \tag{22}$$

where $q = q(z_1|x) \prod_{t=2}^{T} q(z_t|z_{t-1})$. The first term $\ln p(z_T)$ and third term $\ln q(z_1|x)$ is independent of w and therefore can be omitted. The fourth term $\ln p(x|z_1, w)$ is similar to the reconstruction term in the variational autoencoder and can be approximated by using Monte Carlo by simulating from $z_1 \sim N(z_1|\sqrt{1-\beta_1}x, \beta_1 I)$. The second term can also be approximated using Monte Carlo by simulating z_t from $z_1|x, \ldots, z_T|z_{T-1}$. However, using the pairs of sampled values to simulate variables will create noisy estimates and requires more samples for an accurate estimate.

2.2.5 Alternate Form of the ELBO for Efficient Training

Continuing the discussion above, we need to rewrite the ELBO in terms of KL divergences which have a closed form. After removing the irrelevant terms from Equation (22), we obtain

$$\mathcal{L}(w) = E_q \left[\sum_{t=2}^{T} \ln \frac{p(z_{t-1}|z_t, w)}{q(z_t|z_{t-1}, x)} + \ln p(x|z_1, w) \right] \tag{23}$$

As presented in Bishop and Bishop [BB23], $\mathcal{L}(w)$ can be further written as

$$\mathcal{L}(w) = \int q(z_1|x) \ln p(x|z_1, w) dz_1 - \sum_{t=2}^{T} \int KL(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t, w)) q(z_t|x) dz_t \tag{24}$$

where the first term in the equation is the reconstruction term and the second term acts as the consistency term.

From Bishop and Bishop [BB23], after substituting the distributions of $q(z_{t-1}|z_t, x)$ from Equation (14), the KL divergence can be written as

$$KL(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t, w)) = \frac{1}{2\beta_t}||m_t(x, z_t) - \mu(z_t, w, t)||^2 + \text{ constant} \tag{25}$$

where $m_t(x, z_t)$ is mean of reverse process distribution $q(z_{t-1}|z_t, x)$ and $\mu(z_t, w, t)$ is the predicted value from the neural network. Suprisingly, we can use the MSE between the neural network prediction and the mean of the reverse process to train the neural network.

However, better reconstruction results can be obtained by slightly changing the parametrization of the neural network. From Ho, Jain, and Abbeel [HJA20], it was observed that changing the neural network to predict the noise component at each step of the Markov chain performs better than predicting the denoised data. Since we have

$$x = \frac{1}{\sqrt{\alpha_t}} z_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \epsilon_t$$

we can rewrite the means in terms of z_t, ϵ_t and the neural network predictor for ϵ_t, $g(z_t, w, t)$

$$\begin{aligned} m_t(x, z_t) &= \frac{1}{\sqrt{1-\beta_t}} \left[z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \epsilon_t \right] \\ \mu(z_t, w, t) &= \frac{1}{\sqrt{1-\beta_t}} \left[z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(z_t, w, t) \right] \end{aligned} \tag{26}$$

After rewriting the loss in terms of $g(z_t, w, t)$ and ϵ_t, the KL divergence (consistency term) can be written as

$$KL(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t, w)) = \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}||g(z_t, w, t) - \epsilon_t||^2 + constant \tag{27}$$

and $\ln p(x|z_1, w)$ (reconstruction term) as

$$\ln p(x|z_1, w) = -\frac{1}{2(1-\beta_1)}||g(z_1, w, 1) - \epsilon_1||^2 + constant \tag{28}$$

We notice that $\ln p(x|z_1, w)$ is of the same form as the KL divergence and therefore we can combine both the reconstruction and consistency terms.

As stated by Ho, Jain, and Abbeel [HJA20], weighting all squared errors similarly by removing the factor $\frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}$ improves the empirical performance. In this case, $\mathcal{L}(w)$ is given by

$$\mathcal{L}(w) = -\sum_{t=1}^{T} ||g(\sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon_t, w, t) - \epsilon_t||^2 \tag{29}$$

2.2.6 Algorithms for Diffusion Models

Algorithm 2.2 Training a Diffusion Model

Input: Training data $D = \{x_1, x_2, \ldots, x_n\}$ where $x_i = (xi1, xi2, \ldots, x_{id})$
Noise Schedule: $\{\beta_1, \beta_2, \ldots, \beta_T\}$
Output: Network parameters: w

for $t \in \{1, 2 \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$
end for
repeat
  x $\sim D$ # **sample a datapoint**
  (used bootstrapped sample of entire dataset for faster convergence)
  t $\sim \{1, \ldots, T\}$ # **randomly sample a timestep**
  $\epsilon \sim \mathcal{N}(\epsilon|0, I)$ # **sample a noise vector**
  $z_t \leftarrow \sqrt{\alpha_t}x + \sqrt{1 - \alpha_t}\epsilon$ # **evaluate noisy latent variable**
  $\mathcal{L}(w) \leftarrow ||g(z_t, w, t) - \epsilon||^2$ # **evaluate loss**
  take optimizer step # **perform backward pass for neural network**
until neural network converges
return w

Algorithm 2.3 Sampling from a Diffusion Model

Input: Trained denoising neural network $g(z_t, w, t)$
Noise Schedule: $\{\beta_1, \beta_2, \ldots, \beta_T\}$
Output: Sample x in the original data space

$z_T \sim \mathcal{N}(z_T|0, I)$ # **sample from final latent distribution**
for $t \in \{T \ldots, 2\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$
  $\mu(z_t, w, t) \leftarrow \frac{1}{1-\beta_t}[z_t - \frac{\beta_t}{1-\alpha_t}g(z_t, w, t)]$ # **denoise** z_t **with neural network**
  $\epsilon \sim \mathcal{N}(\epsilon|0, I)$ # **sample noise**
  $z_{t-1} \leftarrow \mu(z_t, w, t) + \sqrt{\beta_t}\epsilon$ # **evaluate noisy latent variable at** $t - 1$
end for
$x = \frac{1}{\sqrt{1-\beta_1}}[z_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}g(z_1, w, 1)]$ # **final denoising step**
return x

2.2.7 Improvements Over the Original Diffusion Model

We will discuss different methods in the literature to enhance the performance of diffusion models in this section by referring to methods introduced in Yang et al. [Yan+24]

One possible way to improve diffusion model is to accelerate the sampling process to make the diffusion model more efficient. For example, the reverse process can be formulated as a diffusion and then discretized. This allows the sampling step size can be optimized to produce better samples. Instead of using just the sample of the last step to make predictions, we can use more intermediate timesteps to generate predictions and thus improve the sample quality.

Other than optimizing the step size, we can also learn the optimal variance for the forward diffusion process to improve the reverse decoding process. We can also adjust the likelihood in the last timestep

in the reverse decoder to better fit the data.

3 Pseudo-Inverse Rosenblatt Transform Inspired by Diffusion Models

While replicating the work of Hofert, Prasad, and Zhu [HPZ21], we wanted to try whether the neural network could be learnt without using the MMD as loss function for better efficiency.

Inspired by diffusion models, we tried to add Gaussian noise to our inputs before training. Surprisingly, just by this setup, we were successful in simulating copula samples without using the MMD.

3.1 Inverse Rosenblatt Transform

Theorem 3.1 (Rosenblatt Transform)
Let $X = (X_1, X_2, \ldots, X_d)$ be a random vector with cdf $F(x_1, x_2, \ldots, x_d)$. Then let $u = (u_1, u_2, \ldots, u_d) = T(x) = T(x_1, x_2, \ldots, x_d)$, where T is the transformation from the random vector from a copula, $X = (X_1, X_2, \ldots, X_d)$ to a vector of uniform margins, $U = (U_1, U_2, \ldots, U_d)$. T is given by

$$\begin{aligned} u_1 &= F(x_1) \\ u_2 &= F(x_2|x_1) \\ &\ldots \\ u_d &= F(x_d|x_{d-1}, \ldots, x_1) \end{aligned}$$

Proof:

$$\begin{aligned} &P(U_1 \leq u_1, U_2 \leq u_2, \ldots, U_d \leq u_d) \\ &= P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_d \leq x_d) \\ &= \int_{1\{X_{d-1} \leq x_{d-1}, \ldots, X_1 \leq x_1\}} F(x_d|x_{d-1}, \ldots, x_1) dF(x_{d-1}, \ldots, x_1) \end{aligned}$$

Note that

$$F(x_d|x_{d-1}, \ldots, x_1) = \int_{1\{X_d \leq x_d\}} dF_d(x_d|x_{d-1}, \ldots)$$

After repeated conditioning and rewriting in terms of T, we get

$$\begin{aligned} &= \int_{1\{U_1 \leq u_1\}} \ldots \int dF_d(x_d|x_{d-1}, \ldots, x_1) \ldots dF(x_1) \\ &= \int_0^{u_d} \ldots \int_0^{u_1} du_d \ldots du_1 \\ &= \prod_{i=1}^{d} u_i \end{aligned}$$

Which is joint cdf of d independent $U(0, 1)$

The inverse Rosenblatt transform T^{-1}, is the map from the vector $U = (U_1, U_2, \ldots, U_d)$ of independent uniform margins to $X = (X_1, X_2 \ldots, X_d)$. T can be written as

$$\begin{aligned} x_1 &= F^{-1}(u_1) \\ x_2 &= F^{-1}(u_2|u_1) \\ &\ldots \\ x_d &= F^{-1}(u_d|u_{d-1}, \ldots, u_1) \end{aligned}$$

The inverse Rosenblatt transform allows us to sample from a copula easily as we can simulate independent uniforms, then apply the transform to get uniform samples from a copula as stated in Rosenblatt [Ros52]. However, the inverse Rosenblatt transform is not available in closed form for most copulas. Therefore, we explore alternative ways to simulate from copulas using neural networks. Inspired by diffusion models, we investigate whether a pseudo-inverse Rosenblatt transform could be learnt.

3.2 Algorithms for the Pseudo-Inverse Rosenblatt Transform

Algorithm 3.4 Learning the Pseudo-Inverse Rosenblatt Transform

Input: Training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
where $x_i = ((x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \ldots, (x_{id}, y_{id}))$
x_i: independent samples from applying Rosenblatt Transform to y_i,
then apply Gaussian quantile function
y_i: dependent samples drawn from copula
Noise Schedule: $\{\beta_1, \beta_2, \ldots, \beta_T\}$
Output: Network parameters: w

for $t \in \{1, 2 \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$
end for
repeat
  $(x_1, y_1), \ldots, (x_n, y_n) \sim D$ # **draw batch of bootstrapped samples from dataset**
  t $\sim \{1, \ldots, T\}$ # **randomly sample a timestep**
  $\epsilon \sim \mathcal{N}(\epsilon|0, I)$ # **sample a noise vector**
  $z_t \leftarrow \sqrt{\alpha_t}x + \sqrt{1 - \alpha_t}\epsilon$ # **evaluate noisy latent variable**
  $\mathcal{L}(w) \leftarrow ||g(z_t, w, t) - y||^2$ # **evaluate loss**
  take optimizer step # **perform backward pass for neural network**
until neural network converges
return w

Algorithm 3.5 Sampling from the Pseudo-Inverse Rosenblatt Transform

Input: Trained denoising neural network $g(z_t, w, t)$
Noise Schedule: $\{\beta_1, \beta_2, \ldots, \beta_T\}$
Output: Sample y_i in the original data space

$z_T \sim \mathcal{N}(z_T|0, I)$ # **sample from a Gaussian**
return $y = g(z_t, w, t)$ # **undo simulated noise by passing it through the trained neural network**

After learning the Pseudo-Inverse Rosenblatt Transform, we can pass Gaussian noise through it and use the diffusion model as a pseudo-copula to simulate copula samples.

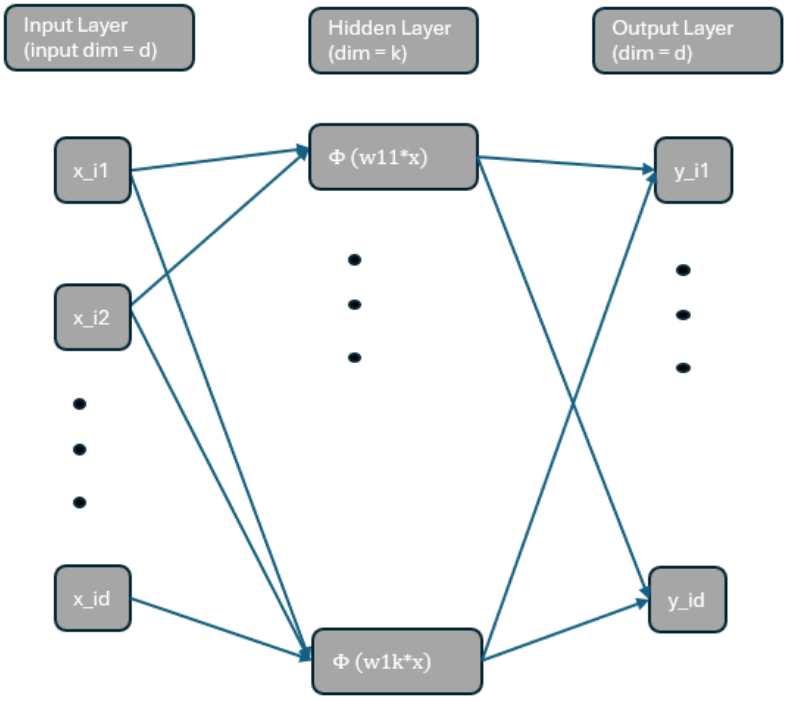


Figure 2: Structure of NN with input $x = (x_1, \ldots, x_d)$, 1 hidden layer, with activation function ϕ, output $y = (y_1, \ldots, y_d)$

Layers	Layer Type	Input Dimension	Output Dimension
1	Linear	d	300
2	ReLU	300	300
3	BatchNorm1d	300	300
4	Linear	300	d
5	Sigmoid	d	d

Table 1: Setup for Multilayer Perceptron

3.3 Specific Setup for Neural Network

For this project, we used R and Python together as the functionalities we needed could not be provided by either one programming language only. We used the R copula package to generate the copula samples and also to Rosenblatt transform the copula samples to independent uniforms. The Python pacakges Pytorch, Numpy and Scipy were used to build the neural network and perform scientific computing calculations.

To train the neural network for the 2-dimensional copula, 60000 samples are generated from a 2D copula so the input dimension will be (60000, 2). Then, we convert the copula samples into samples from $U[0, 1]$ with the Rosenblatt transform.

The Gaussian quantile function is then applied to the independent samples to help the neural network learn the distribution better as the input is unbounded on $(-\infty, \infty)$ after the transformation. Based on experience from Hofert, Prasad, and Zhu [HPZ21], this helps us to better learn the distribution for observations around the edges of the uniform margins as the inputs are now unbounded.

As opposed to the convolutional neural networks (CNN) commonly used in diffusion models, a multilayer perceptron similar to Hofert, Prasad, and Zhu [HPZ21] is used to due its simplicity and efficiency

in training. To capture the dependence for high dimensional dependent data, CNNs may need to use a larger kernel which will increase computational and memory cost.

After each linear layer, we use ReLU to allow our neural network to learn non-linear targets. ReLU is used due to its simplicity in calculation as compared to other activation function like sigmoid. We also use batch norm to prevent gradients from blowing up and to quicken the training process. Batch norm is also able to avoid the problem of vanishing gradients as ReLU only has a gradient when the weighted inputs are positive, hence normalizing the outputs from the first ReLU will help to maintain the distribution of the inputs for the following Linear layer. Before evaluating the final output, we use the sigmoid activation function to constrain the output to be between $[0, 1]$. This allows us to compare the diffusion model outputs directly with the copula outputs.

For our optimization algorithm, we choose to use the Adam optimizer. We use a learning rate of 0.0005 to avoid our gradient from blowing up and making the layer outputs unstable. As explained in Kingma and Ba [KB17], Adam uses the concept of momentum in its update, which means it takes a weighted average of previous values and current value, making the gradient updates for each step more stable. It also makes use of the sum of squares of the past gradients to reduce the magnitude of the update, decreasing the size of the updates for further iterations to prevent the update from overshooting the global minimum.

We also choose to use gradient clipping to limit our gradients to a norm of 0.5 to stabilize the training and avoid them from blowing up. This helps to stabilize our loss and also quicken our training process as the gradient updates do not cause the value of the layers to change around too much.

For our experiment, we decided to train our diffusion model to learn the Gaussian, Gumbel and Marshall-Olkin copulas. We choose the Gaussian as it is one of the most commonly used copulas. For the Gumbel, we wanted to check our diffusion model's ability to learn distributions with a stronger right-tail dependence as such copulas are commonly used for dependent losses in the financial industry and also for natural phenomenon. The Marshall-Olkin copula is also chosen due to its singularity component and we wanted to test our diffusion model's capability in learning such features in a copula.

We vary the number of epochs and samples utilized to train the our model based on the type of copula and dimension. For 2 dimensional copulas we use 60000 samples and bootstrap 60000 samples for each epoch. For the Gaussian and Gumbel, we train for 300 epochs as they are easier to learn. However, we train the diffusion model for 1000 epochs for the Marshall-Olkin copula as it can be harder to learn than the Gaussian and Gumbel copulas due to their singularity.

For scaling up to 50 dimensions, we decide to use a sample size of 100000 to enable the diffusion model to learn the distribution more easily. We also train the diffusion model for 1000 epochs to enable our neural network to learn the distribution more accurately.

3.4 Motivation

3.4.1 Noise as A Form of Regularization for Neural Networks

In this part, we explain why corrupting inputs with noise makes it possible to learn a better description of the training data as compared to just using uncorrupted data.

According to Bishop [Bis95], the error when training with noise can be written as

$$\tilde{E} = \int \int \int \sum_k [y_k(x+\epsilon) - t_k]^2 p(t_k|x)p(x)p(\epsilon)dxdt_k d\epsilon \tag{30}$$

We assume that the noise amplitude, ϵ is small. With Taylor's theorem, we can write $y_k(x+\epsilon)$ as

$$y_k(x+\xi) = y_k(x) + \sum_i \xi_i \frac{\partial y_k}{\partial x_i}\bigg|_{\xi=0} + \frac{1}{2}\sum_{i,j} \xi_i \xi_j \frac{\partial^2 y_k}{\partial x_i \partial x_j}\bigg|_{\xi=0} + O(\xi^3) \tag{31}$$

If the noise distribution is chosen to have zero mean, and uncorrelated between different inputs, we have

$$\begin{aligned} \int \epsilon_i p(\epsilon) d\epsilon &= 0 \\ \int \epsilon_i \epsilon_j p(\epsilon) d\epsilon &= \eta^2 \delta_{ij} \end{aligned} \tag{32}$$

After substituting the Taylor's series expansion into $\tilde{E}$, we obtain $\tilde{E} = E + \eta^2 E^R$, where E is the standard sum-of-squares error

$$E = \frac{1}{2}\int\int \sum_k (y_k(x) - t_k)^2 p(t_k|x) p(x) dx dt_k \tag{33}$$

and E^R is given by

$$E^R = \frac{1}{2}\int\int \sum_k \sum_i \{(\frac{\partial y_k}{\partial x_i})^2 + \frac{1}{2}\{y_k(x) - t_k\}\frac{\partial^2 y_k}{\partial x_i^2}\} p(t_k|x) p(x) dx dt_k \tag{34}$$

E^R acts like a regularization term added to the sum-of-squares error, with the coefficient of the regularization determined by noise variance η^2.

In short, we conclude that adding noise into inputs forces the neural network to be less sensitive to small changes in input and therefore enables it to learn the "surface" of the distribution function.

3.4.2 Rationale for Small Values of $1 - \alpha_t$

Note that our diffusion model uses a small number of timesteps ($t \leq 10$), as opposed to the the commonly used $t > 1000$ in most diffusion model papers. While optimizing for the best t, we noticed that a smaller value of t (hence a smaller amount of corruption), produces samples that better fit the targeted copula.

Since we are learning a distribution, we are trying to estimate this quantity

$$P(f_\theta(X) \leq y) \tag{35}$$

where $X = (X_1, X_2, \ldots, X_d)$ represents a vector of input features, $y = (y_1, y_2, \ldots, y_d)$ is a specific value, and f_θ as neural network parameterized by θ.

However, since we may not have enough samples to learn the distribution (especially for high dimensions), we use the corrupted latent variable Z_t.

$$\begin{aligned} P(f_\theta(Z_t) \leq y) &= P(f_\theta(\sqrt{\alpha_t}X_0 + \sqrt{1-\alpha_t}\epsilon) \leq y) \\ &= \int P(f_\theta(\sqrt{\alpha_t}X_0 + \sqrt{1-\alpha_t}\epsilon) \leq y|\epsilon) f(\epsilon) d\epsilon \end{aligned} \tag{36}$$

With Taylor's theorem

$$\begin{aligned} &f(\sqrt{\alpha_t}X_0 + \sqrt{1-\alpha_t}\epsilon) \\ &= f(\sqrt{\alpha_t}X_0) + \sqrt{1-\alpha_t} Df(\sqrt{\alpha_t}X_0)^T \epsilon + \frac{1}{2}(1-\alpha_t)\epsilon D^2 f(\sqrt{\alpha_t}X_0)\epsilon^T + \ldots \text{ higher order terms} \end{aligned} \tag{37}$$

where $Df(X) = \left(\frac{df(x)}{dx_1}, \frac{df(x)}{dx_2}, \ldots, \frac{df(x)}{dx_d}\right)$ and $D^2 f(x) = \begin{pmatrix} \frac{d^2 f(x)}{dx_1^2} & \cdots & \cdots & \frac{d^2 f(x)}{dx_1 dx_d} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{d^2 f(x)}{dx_d dx_1} & \cdots & \cdots & \frac{d^2 f(x)}{dx_d dx_d} \end{pmatrix}$

Then we integrate the values with respect to ϵ, obtaining

$$\begin{aligned} P(f_\theta(Z_t) \leq y) &= \int_{-\infty}^{\infty} \int_{-\infty}^{y} [f(\sqrt{\alpha_t} X_0) + \sqrt{1-\alpha_t} Df(\sqrt{\alpha_t} X_0)^T \epsilon \\ &\quad + \frac{1}{2}(1-\alpha_t)\epsilon D^2 f(\sqrt{\alpha_t} X_0)\epsilon^T + \ldots] f(X_0) f(\epsilon) dX_0 d\epsilon \\ &= \int_{-\infty}^{y} \int_{-\infty}^{\infty} [f(\sqrt{\alpha_t} X_0) + \sqrt{1-\alpha_t} Df(\sqrt{\alpha_t} X_0)^T \epsilon \\ &\quad + \frac{1}{2}(1-\alpha_t)\epsilon D^2 f(\sqrt{\alpha_t} X_0)\epsilon^T + \ldots] f(\epsilon) d\epsilon f(X_0) dX_0 \\ &\text{Since } 1-\alpha_t \approx 0 \text{ and } \epsilon \sim N(0, I) \\ &\approx \int_{-\infty}^{y} \left[f_\theta(\sqrt{\alpha_t} X_0) + \sqrt{1-\alpha_t} Df(\sqrt{\alpha_t} X_0)^T (0) + \frac{1}{2}(1-\alpha_t) D^2 f(\sqrt{\alpha_t} X_0) + \ldots \right] dX_0 \\ &= P(f_\theta(\sqrt{\alpha_t} X_0) \leq y) \\ &\approx P(f_\theta(X_0) \leq y) \end{aligned} \tag{38}$$

Therefore, a small value of $1 - \alpha_t$ is required for this approximation to work. In the neural network, we use $t = 10$ to ensure that the corruption of the original data is weak.

3.5 Results and Analysis

This section will discuss the results from our experiments based on simulated data. We will also describe the setups and tests we used to check the validity of our results in detail.

3.5.1 Specific Setup for Analysis of Experiments

The MMD is used as a goodness-of-fit test for our pseudo-copulas due to its ease of computation. The MMD metric measures the distance between a higher dimensional projection of the summary statistics of two distributions, $\varphi(X)$ as shown below

$$\begin{aligned} &MMD(X, Y) \\ &= \left|\left| \frac{1}{n_{trn}} \sum_{t_1=1}^{n_{trn}} \varphi(X_{t_1}) - \frac{1}{n_{gen}} \sum_{t_2=1}^{n_{gen}} \varphi(Y_{t_2}) \right|\right|^{\frac{1}{2}} \end{aligned} \tag{39}$$

$$\begin{aligned} &MMD(X, Y) \\ &= \sqrt{\frac{1}{n_{trn}^2} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{trn}} \varphi(X_{t_1})^T \varphi(X_{t_2}) - \frac{2}{n_{trn} n_{gen}} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{gen}} \varphi(X_{t_1})^T \varphi(Y_{t_2}) - \frac{1}{n_{gen}^2} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{trn}} \varphi(Y_{t_1})^T \varphi(Y_{t_2})} \end{aligned} \tag{40}$$

where n_{trn} is the number of training samples and n_{gen} is the number of generated samples from the neural network. Substituting the kernel function, K into the product of the embedding, we obtain

$$\begin{aligned} &MMD(X, Y) \\ &= \sqrt{\frac{1}{n_{trn}^2} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{trn}} K(X_{t_1})^T K(X_{t_2}) - \frac{2}{n_{trn} n_{gen}} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{gen}} K(X_{t_1})^T K(Y_{t_2}) - \frac{1}{n_{gen}^2} \sum_{t_1=1}^{n_{trn}} \sum_{t_2=1}^{n_{trn}} K(Y_{t_1})^T K(Y_{t_2})} \end{aligned} \tag{41}$$

For our MMD, we use a similar setup to what Hofert, Prasad, and Zhu [HPZ21] used to train their GMMN. We used a sum of Gaussian kernels as the embedding to project the distance between the diffusion model and copula samples into higher dimensions and used bandwidths of [0.001,0.01,0.15,0.25,0.50,0.75], hence avoiding the need to perform grid search for the bandwidth parameter.

We will also present scatterplots of the copulas samples to show the relationship between different dimensions. This gives us a good visual representation of the generated samples.

The potential of neural networks as a quasi-random number generator will also be investigated too. We perform Monte Carlo simulation and evaluate the standard deviation from the simulation. We will then regress the log of the standard deviation against the number of iterations to investigate the rate of convergence.

3.5.2 Corruption Process by Timesteps

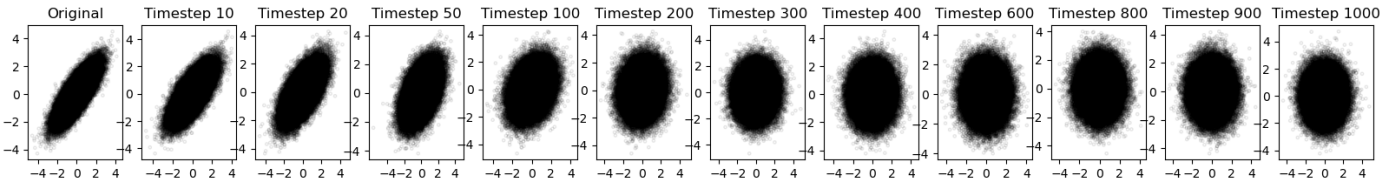


Figure 3: Corruption Process for Gaussian Copula

From Figure 3, the dependence is still observed for $t \leq 100$. After $t = 300$, the dependence is unnoticeable and the samples look like samples drawn from independent Gaussians. In our experiments, the denoising process works better with smaller values of t, thus supporting the previous argument in Section 3.4.2.

3.5.3 Generating samples from 2-dimensional copulas

By comparing the scatter plots of copula samples and the samples generated from our diffusion model, we observe that the diffusion model is able to generate samples similar to the true copula. From a visual inspection of the scatter plots, the intensity of the tails of the diffusion model-generated samples is similar to the tails of copula samples. The diffusion is able to learn the symmetry across the diagonal for the Gaussian and Gumbel copulas. Other than the overall density and tail intensity of the scatter plots, we also investigate whether the diffusion model is able to learn observable characteristics or patterns in the copula. Surprisingly, our setup managed to learn the distribution of the Marshall-Olkin copula properly. As mentioned previously in Hofert, Prasad, and Zhu [HPZ21], this is a remarkable

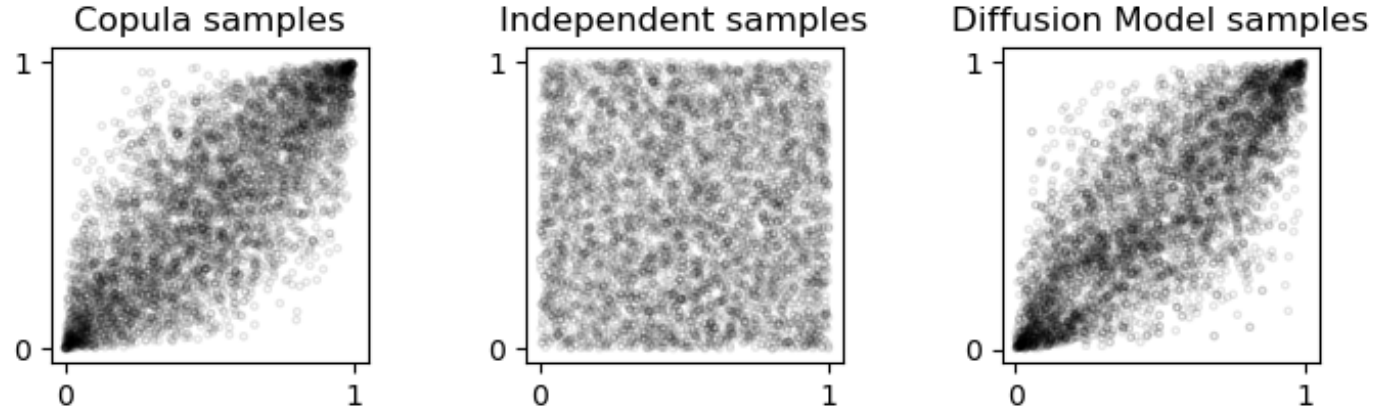


Figure 4: Scatter plot of 2-dimensional Gaussian Copula, Independent Uniforms, and Pseudo-Copula

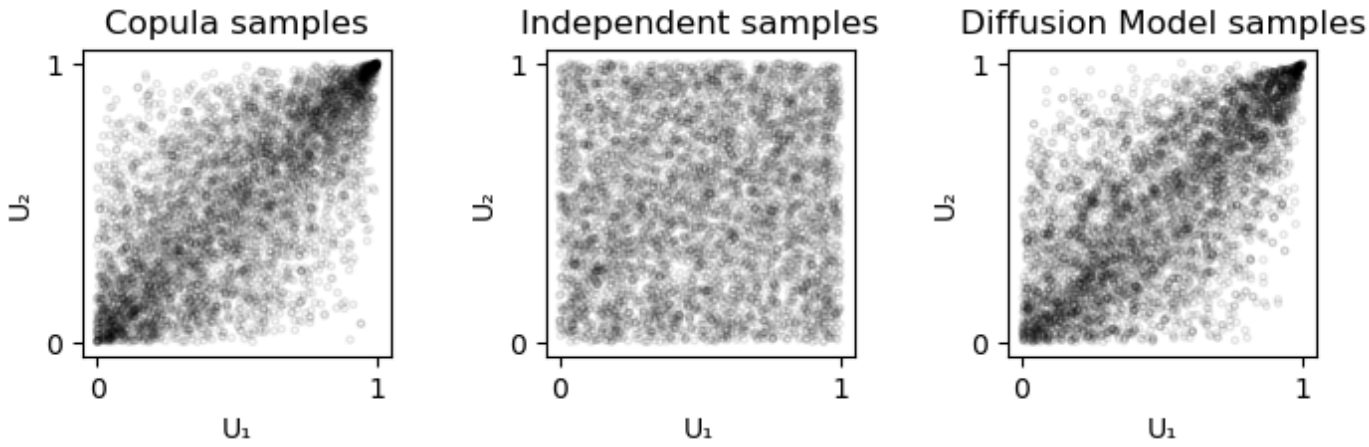


Figure 5: Scatter plot of 2-dimensional Gumbel Copula, Independent Uniforms, and Pseudo-Copula

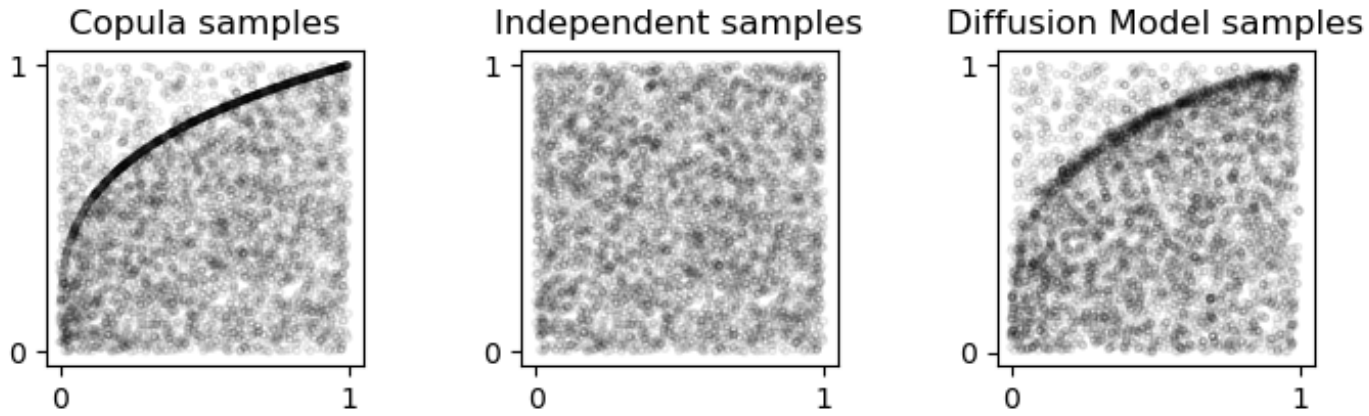


Figure 6: Scatter plot of 2-dimensional Marshall-Olkin Copula, Independent Uniforms, and Pseudo-Copula

feat as the Marshall-Olkin copula has a singularity that is hard to learn.

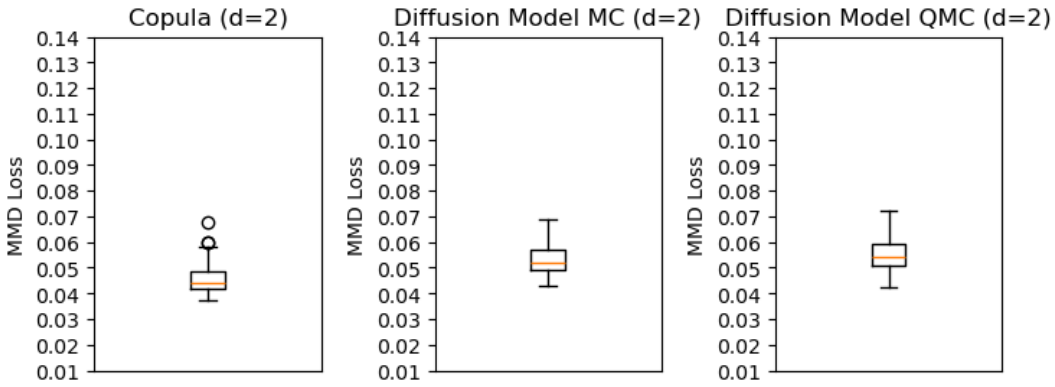


Figure 7: MMD of 2-dimensional Gaussian Copula and Diffusion Model trained with MC and QMC samples, $\rho = 0.8$, Copula MMD: 0.044, Diffusion Model MC: 0.053, Diffusion Model QMC: 0.054

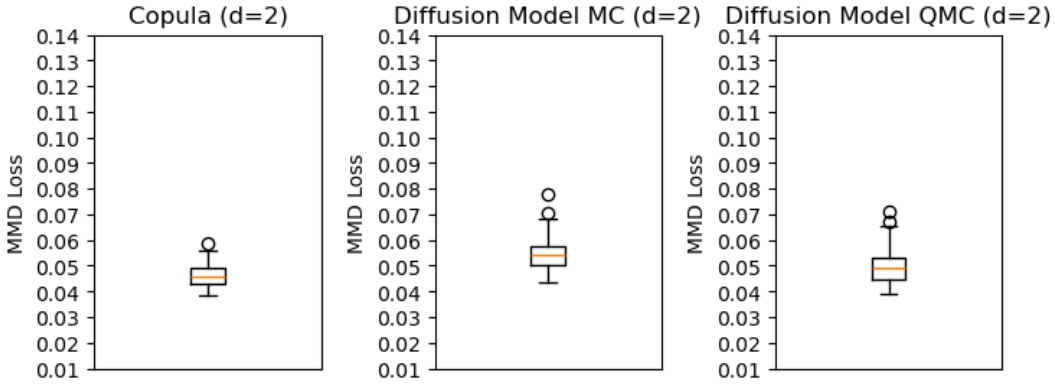


Figure 8: MMD of 2-dimensional Gumbel Copula and Diffusion Model trained with MC and QMC samples, $\theta = 2$, Copula MMD: 0.046, Diffusion Model MC MMD: 0.054, Diffusion Model QMC MMD: 0.049

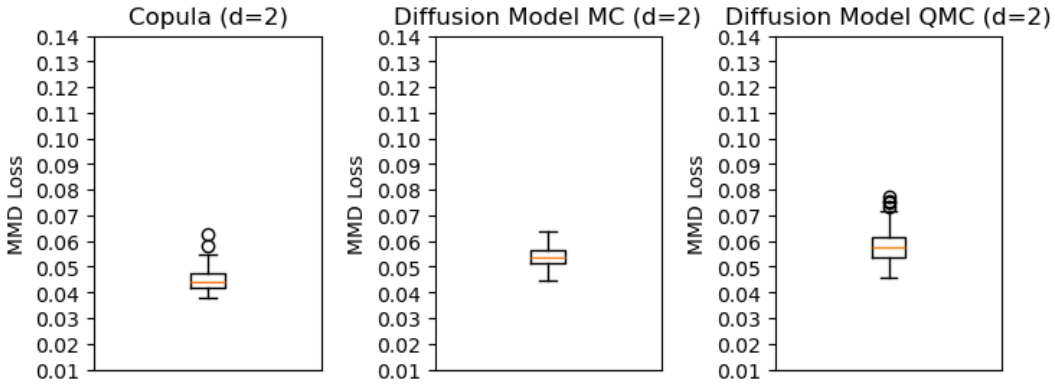


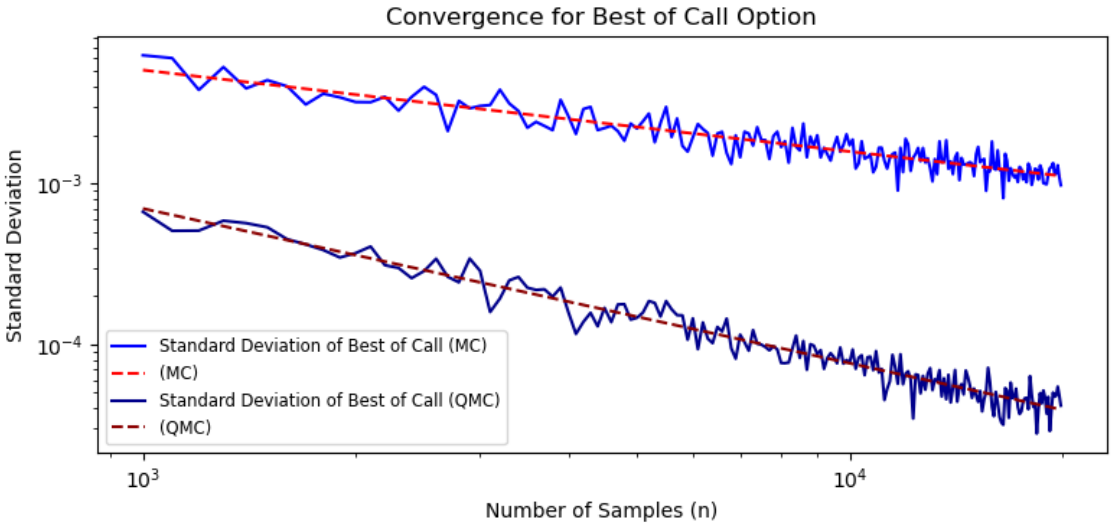
Figure 9: MMD of 2-dimensional Marshall-Olkin Copula and Pseudo-Copula trained with MC and QMC samples, $\theta = (0.2, 0.7)$, Copula MMD: 0.045, Diffusion Model MC MMD: 0.054, Diffusion Model QMC MMD: 0.058

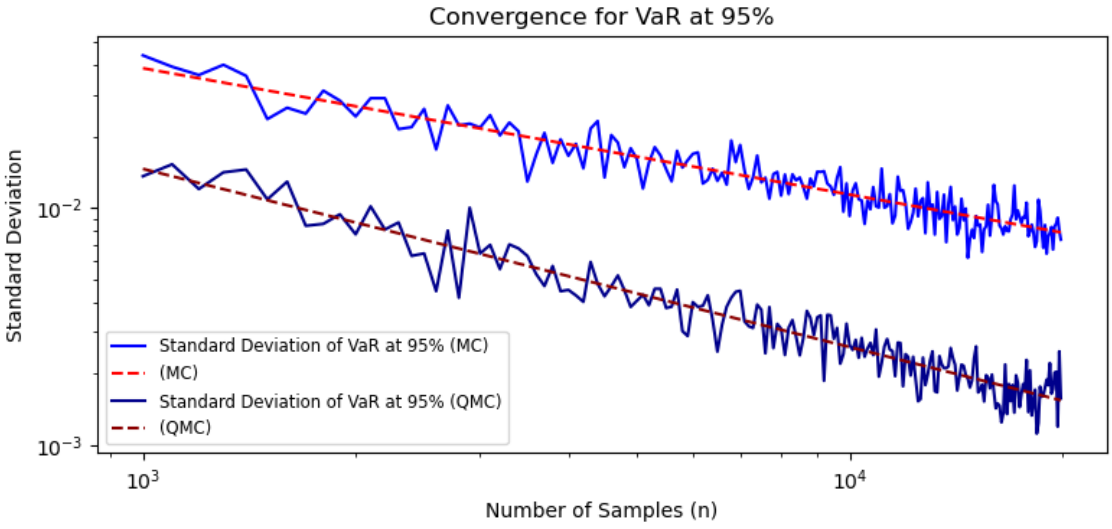
To further support our conclusion, the MMD loss for copula samples and diffusion model-generated samples are quite similar in terms of magnitude.

Since the low discrepancy property is quite hard to observe just by looking at the scatter plots, we compare the MMD and Monte Carlo convergence rates instead. However, the MMD loss of the neural

network trained with the Sobol sequence is higher as compared to the neural network trained with the pseudo-random sequence for both the Gaussian and Marshall-Olkin copula. This is unexpected as we expect the MMD for the diffusion model trained with quasi-random samples to be lower than the diffusion model trained with pseudo-random samples.

Therefore, we suggest that further investigations can focus on differentiating between copula samples and neural network samples. We can instead use the Cramer-von-Mises test for copulas as it checks the whether two distributions match as opposed to the MMD which instead compares the different moments.





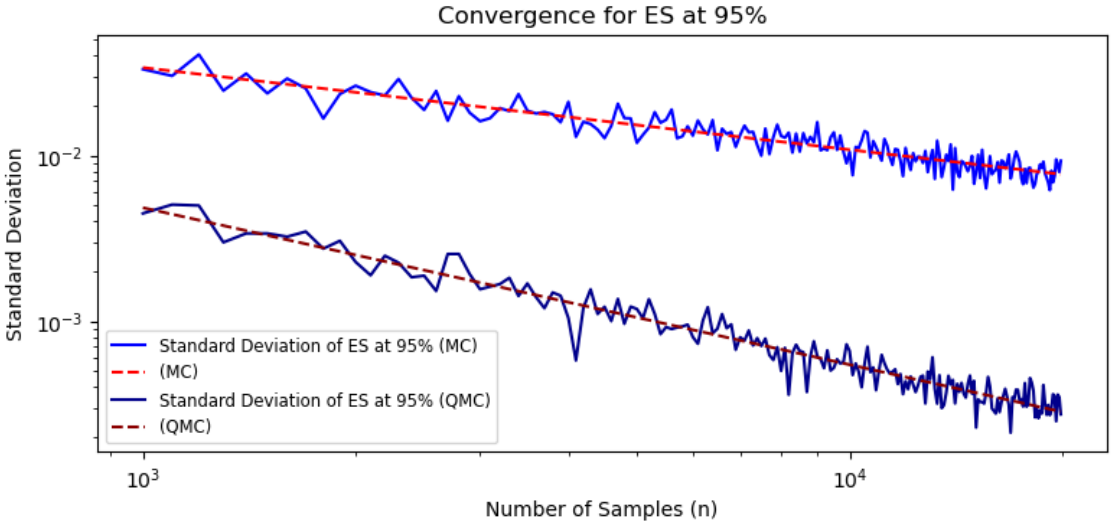
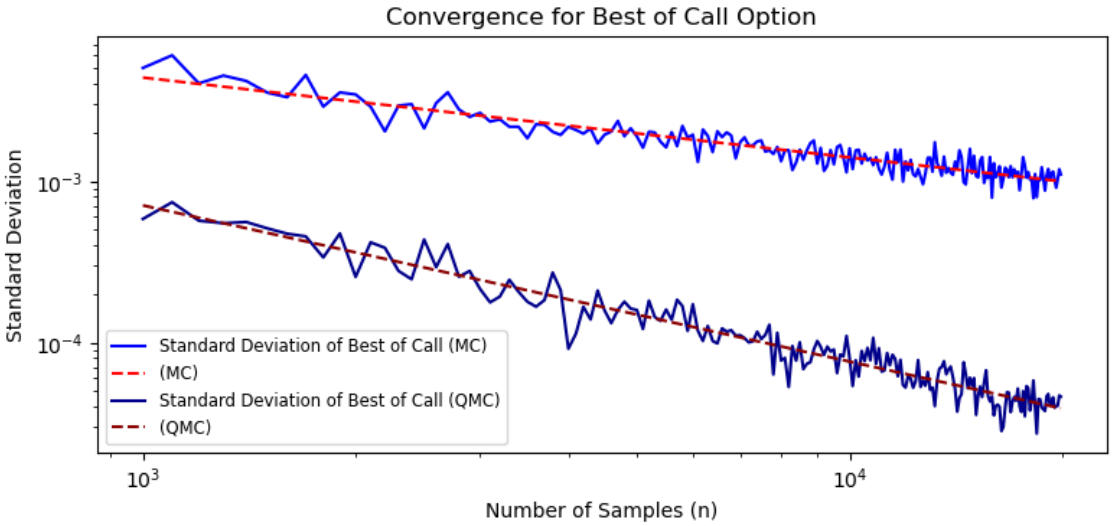
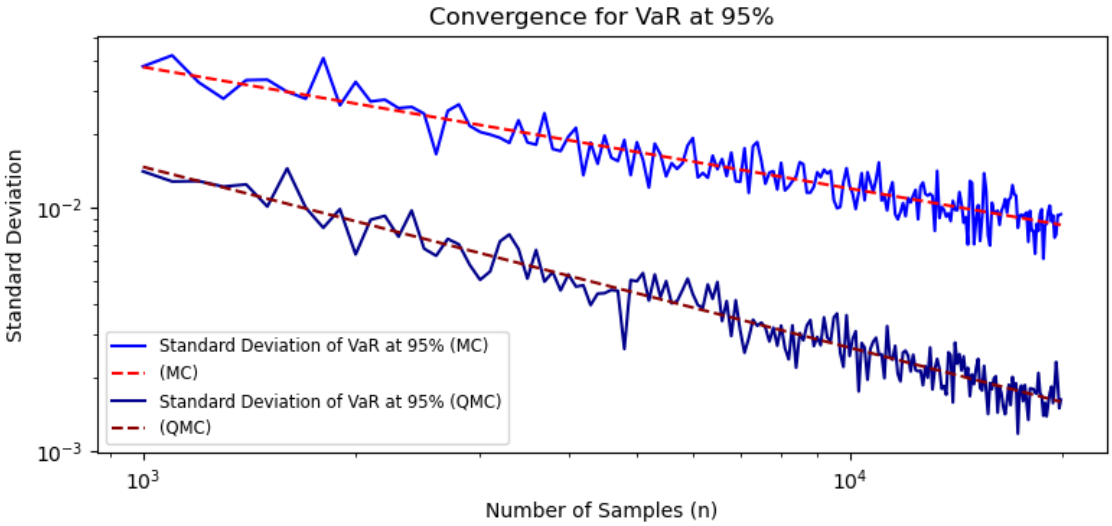


Figure 10: Regression coefficient for Best of Call (MC): -0.51, Regression coefficient for Best of Call (QMC): -0.96, Regression coefficient for VaR (MC): -0.53, Regression coefficient for VaR (QMC): -0.75, Regression coefficient for Expected Shortfall (MC): -0.50, Regression coefficient for Expected Shortfall (QMC): -0.95





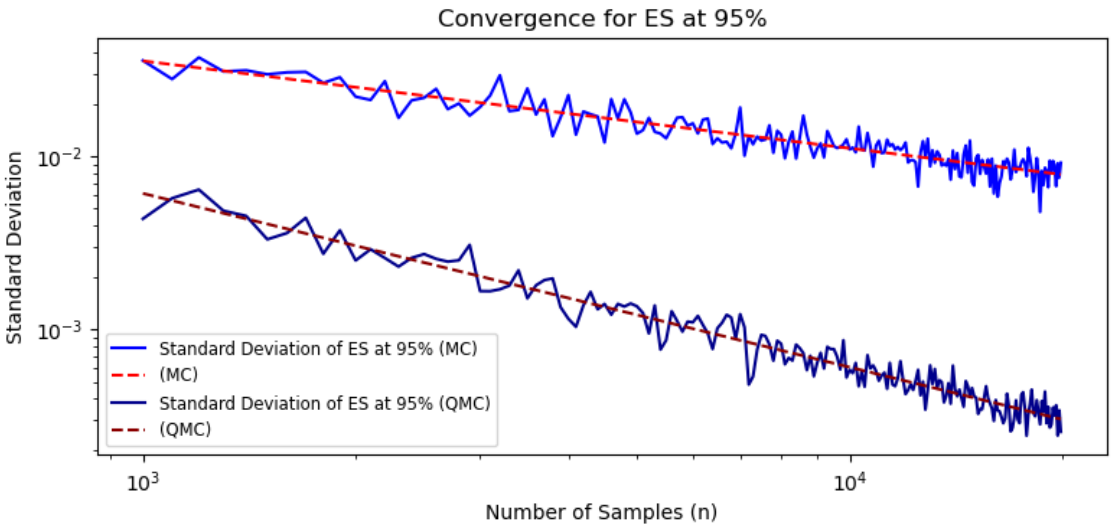
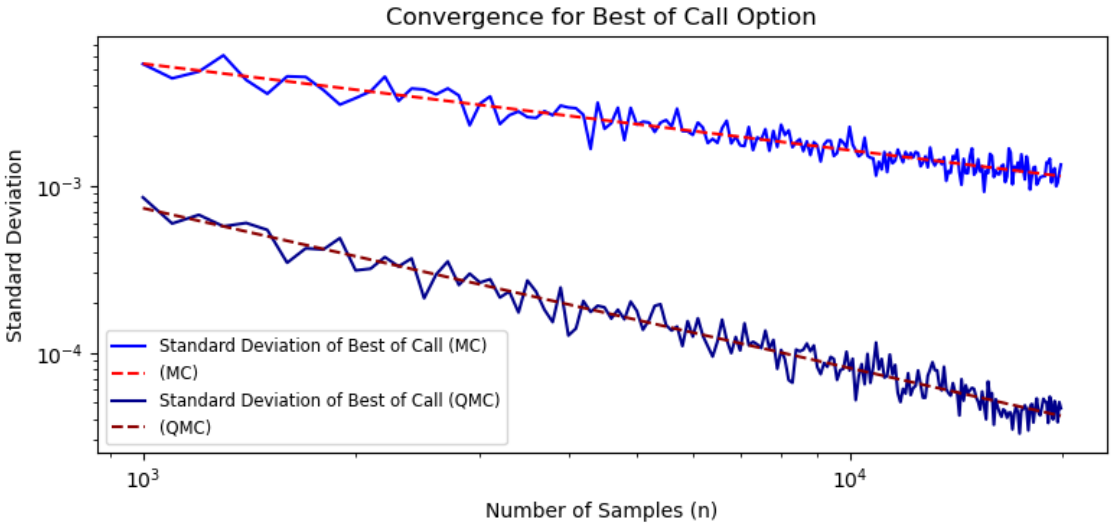
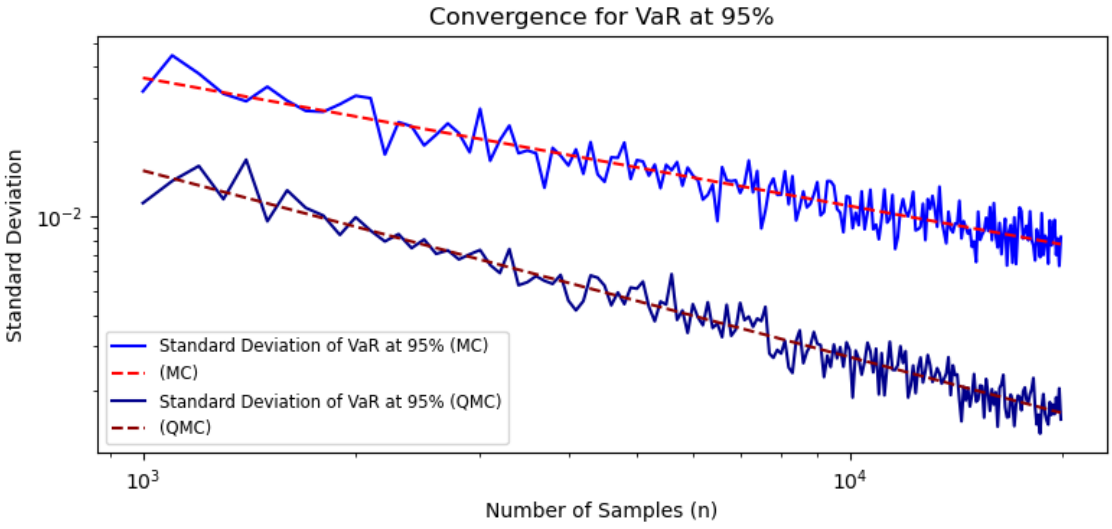


Figure 11: Regression coefficient for Best of Call (MC): -0.49, Regression coefficient for Best of Call (QMC): -0.97, Regression coefficient for VaR (MC): -0.49, Regression coefficient for VaR (QMC): -0.74, Regression coefficient for Expected Shortfall (MC): -0.50, Regression coefficient for Expected Shortfall (QMC): -1.00





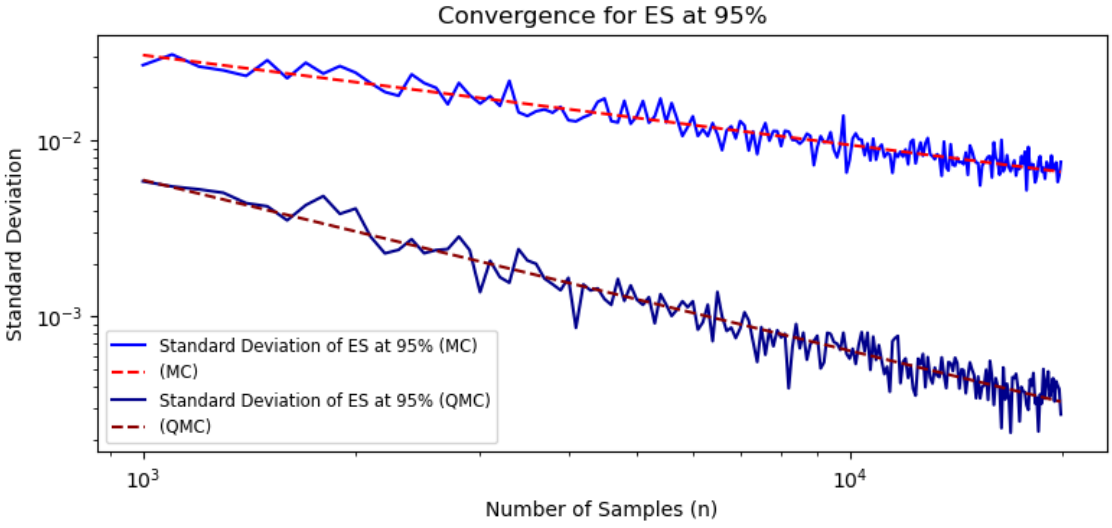


Figure 12: Regression coefficient for Best of Call (MC): -0.52, Regression coefficient for Best of Call (QMC): -0.96, Regression coefficient for VaR (MC): -0.52, Regression coefficient for VaR (QMC): -0.75, Regression coefficient for Expected Shortfall (MC): -0.51, Regression coefficient for Expected Shortfall (QMC): -0.97

The convergence rate of Monte Carlo simulation is of order $O(N^{-\frac{1}{2}})$, so if we plot the standard devi-

ation against the number of iterations in log-log scale, we will get a slope of $-\frac{1}{2}$. The graph shows that the convergence rate for Monte Carlo with diffusion model samples trained with is approximately $-\frac{1}{2}$ for the Gaussian, Gumbel and Marshall-Olkin copula. For quasi-Monte Carlo simulation, the convergence rate is $O((logN)^k N^{-1})$. The graph shows that the convergence rate for the neural network trained with the Sobol sequence is approximately -1 for all the copulas considered, which matches the rate of convergence for quasi-Monte Carlo simulations. This demonstrates that the diffusion model trained with quasi-random numbers to perform Quasi-Monte Carlo simulations have a faster convergence rate as compare to Monte Carlo.

Therefore, we can conclude that neural networks trained with the low-discrepancy Sobol sequence are able to generate quasi-random samples from a copula.

3.5.4 Performance on Limited Data

Since we expect to use deep learning techniques on financial data, we need to investigate whether how many samples our empirical copula needs to be able to properly learn a distribution. We used a bootstrap sample of $N = 5000$ and trained for 2000 epochs for the Gaussian and Gumbel copulas and 5000 epochs for the Marshall-Olkin copula.

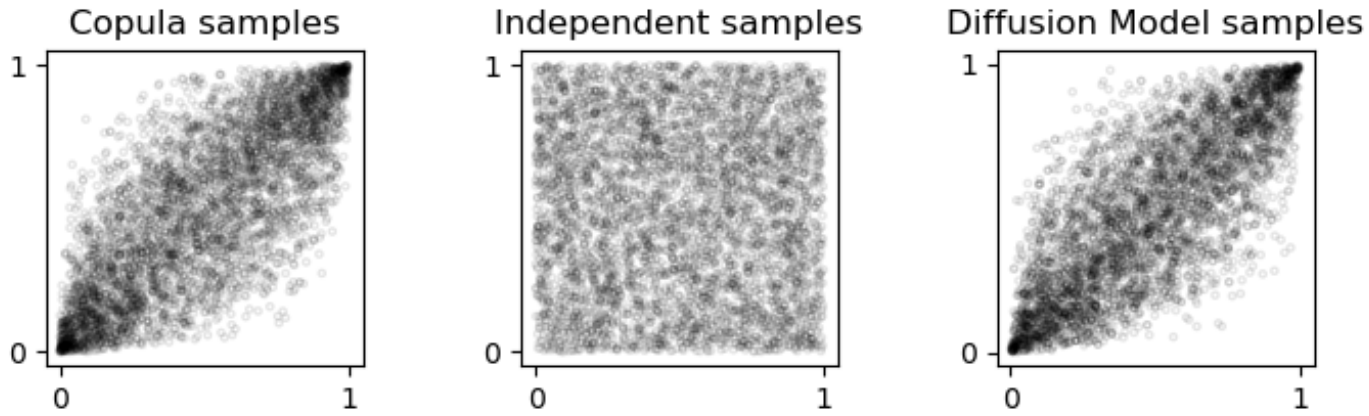


Figure 13: Scatter plot of 2-dimensional Gaussian Copula, Independent Uniforms, and Pseudo-Copula Trained with N=5000 samples

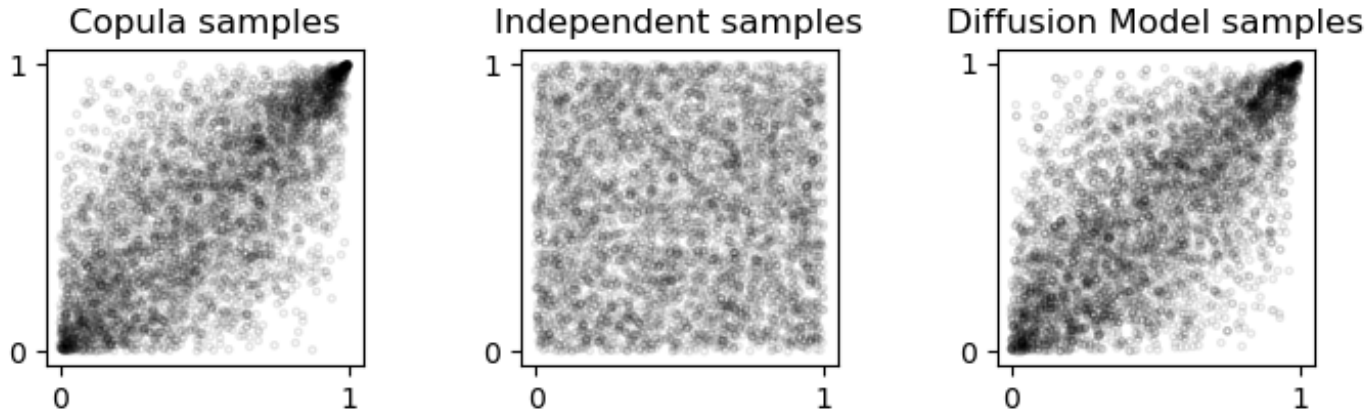


Figure 14: Scatter plot of 2-dimensional Gumbel Copula, Independent Uniforms, and Pseudo-Copula with N=5000 samples

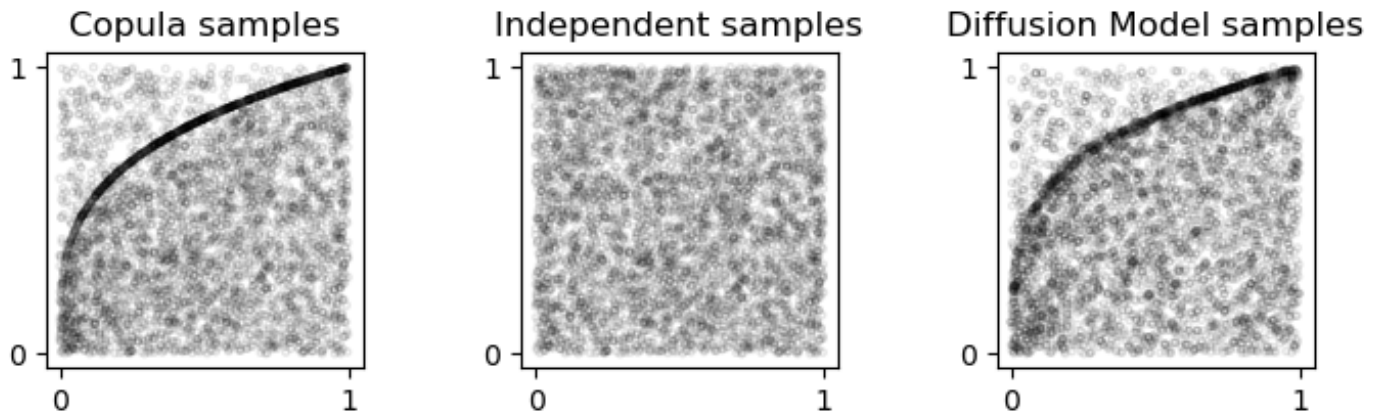


Figure 15: Scatter plot of 2-dimensional Marshall-Olkin Copula, Independent Uniforms, and Pseudo-Copula with N=5000 samples

From the scatter plots, we observe that the diffusion model is able to generate samples similar in distribution to the copula. Similar to the previous training procedure with the full dataset, our limited dataset was able to learn the symmetry across the diagonal for the Gaussian and Gumbel copulas. It also successfully captured the tail dependence for the Gaussian and Gumbel copulas. We were surprised that we could still learn the singularity of the Marshall-Olkin even with the reduced sample size of $N = 5000$.

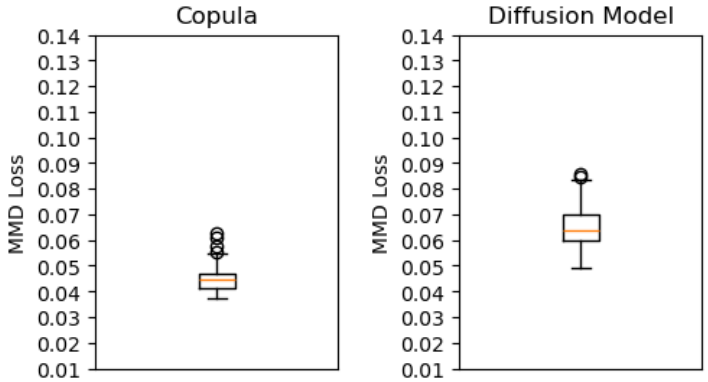


Figure 16: MMD of 2-dimensional Gaussian Copula and Diffusion Model trained with MC samples, $\rho = 0.8$, Copula MMD: 0.045, Diffusion Model MC: 0.053

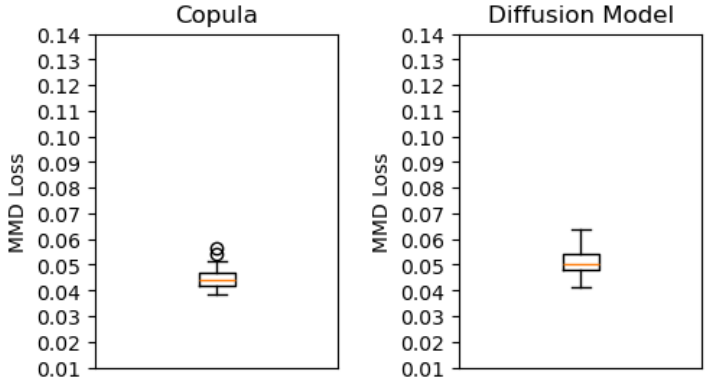


Figure 17: MMD of 2-dimensional Gumbel Copula and Diffusion Model trained with MC, $\theta = 2$, Copula MMD: 0.045, Diffusion Model MC MMD: 0.051

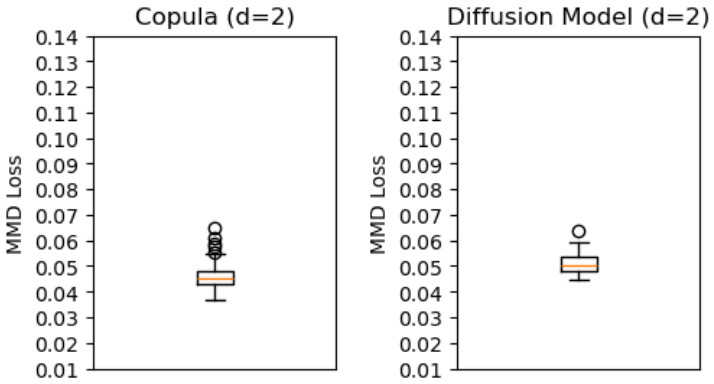


Figure 18: MMD of 2-dimensional Marshall-Olkin Copula and Pseudo-Copula trained with MC and QMC samples, $\theta = (0.2, 0.7)$, Copula MMD: 0.046, Diffusion Model MC MMD: 0.055

The MMD loss for copula samples and diffusion model generated samples are quite similar in terms of magnitude. This shows that even with training on a small sample with multiple epochs is sufficient to assist the neural network in learning the inverse Rosenblatt transform. Therefore, we suggest that further studies can focus on investigating the minimum number of samples needed to learn a copula properly.

3.5.5 Scaling Up to Higher Dimensions

In Hofert, Prasad, and Zhu [HPZ21], using MMD to train neural networks has a complexity of $O(N^2)$ and they could not scale up to larger dimensions due to the computational cost and resource constraints. Since we are able to use MSE to train our models, we attempt to scale the process up and learn copulas with 50 dimensions. Being able to calculate the MSE for a larger batch allows us to train on fewer epochs and is less memory intensive to calculate as compared to the MMD.

To learn the 50-dimensional copula, we used a larger sample of 100000 observations and increased the number of epochs of training to 1000 to ensure that the neural network is able to learn the 50-dimensional copula. This ensures that the diffusion models has fully captured the dependencies between the variables.

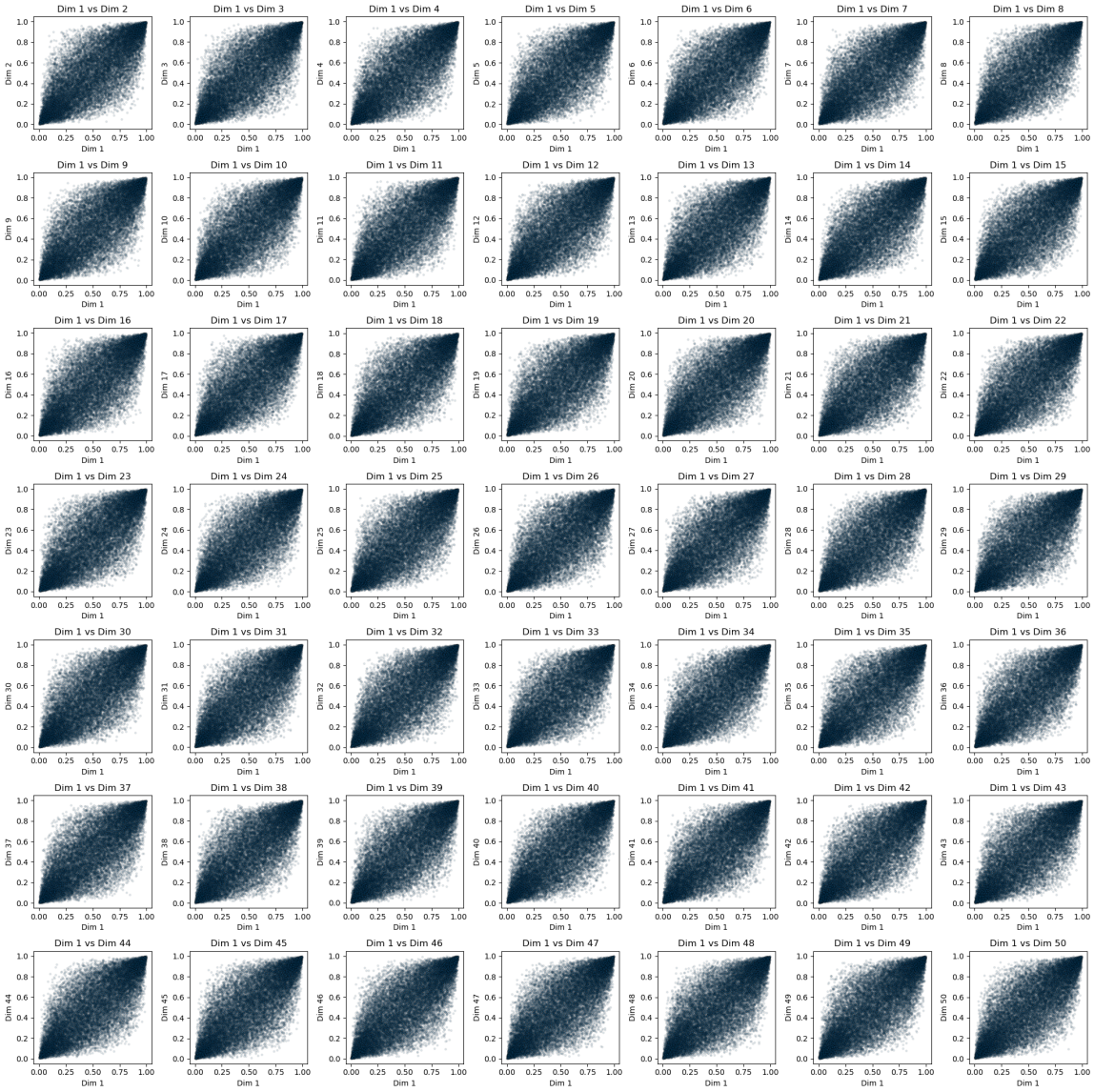


Figure 19: Pairwise scatter plot of 50-dimensional Gaussian for 1^{st} dimension against all other dimensions

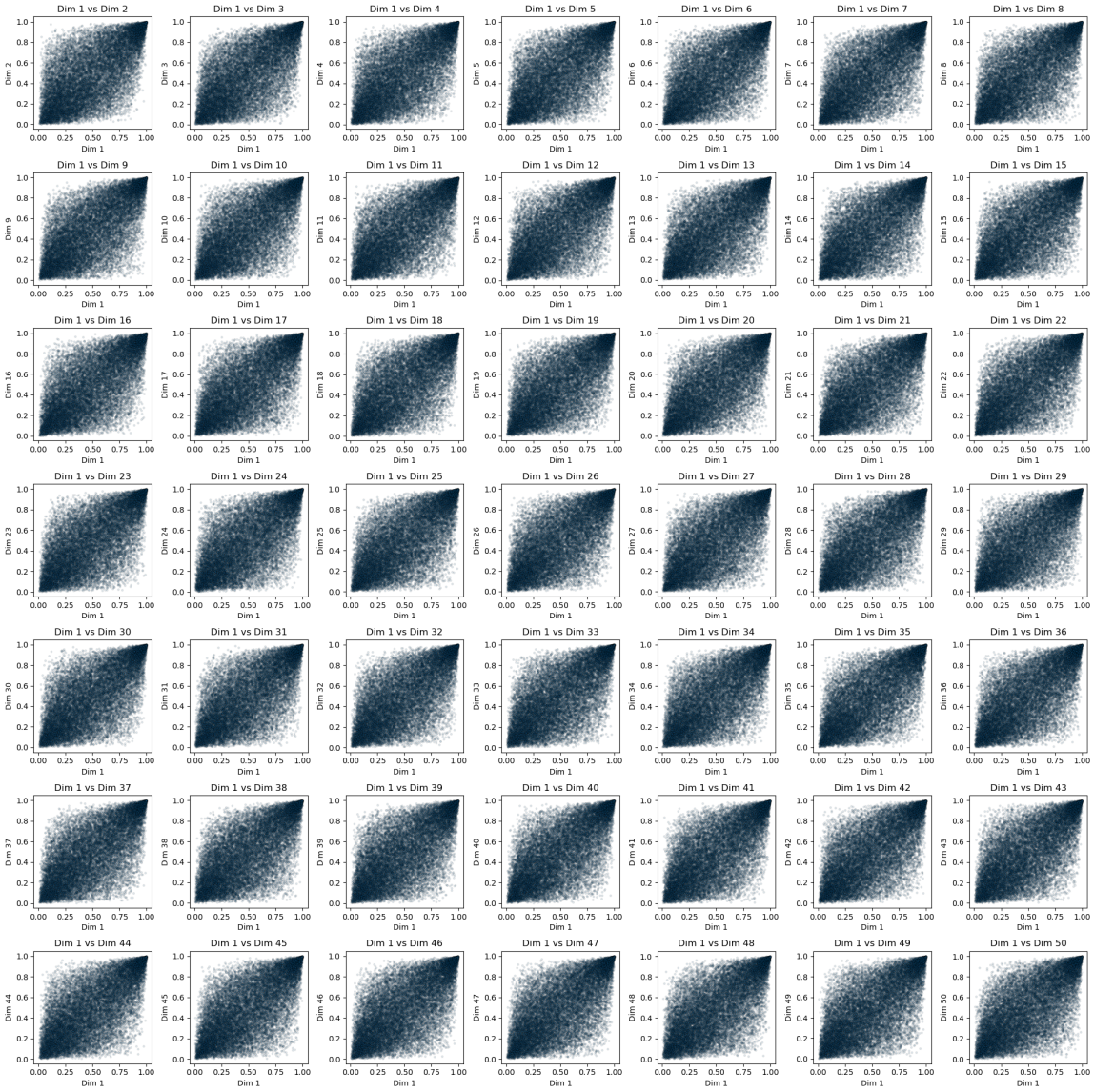


Figure 20: Pairwise scatter plot of 50-dimensional Gumbel for 1^{st} dimension against all other dimensions

The pairwise comparisons of the Gaussian and Gumbel are symmetric for the 1^{st} vs all other dimensions, implying that our neural network managed to learn the symmetry of the copulas correctly. This symmetry holds true as the Gaussian and Gumbel copulas are Archimedean copulas and they are symmetric in terms of the generator function.

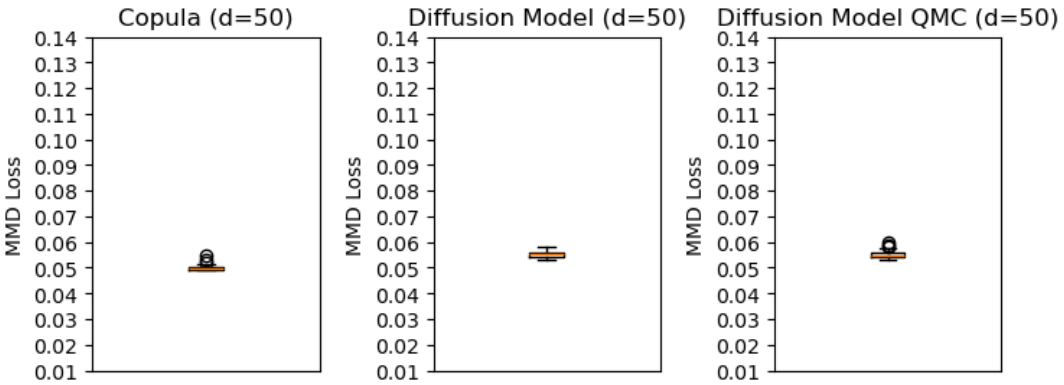


Figure 21: MMD of 50-dimensional Gaussian Copula and Diffusion Model trained with MC and QMC samples, $\rho = 0.8$, Copula MMD: 0.050, Diffusion Model MC: 0.055, Diffusion Model QMC: 0.055

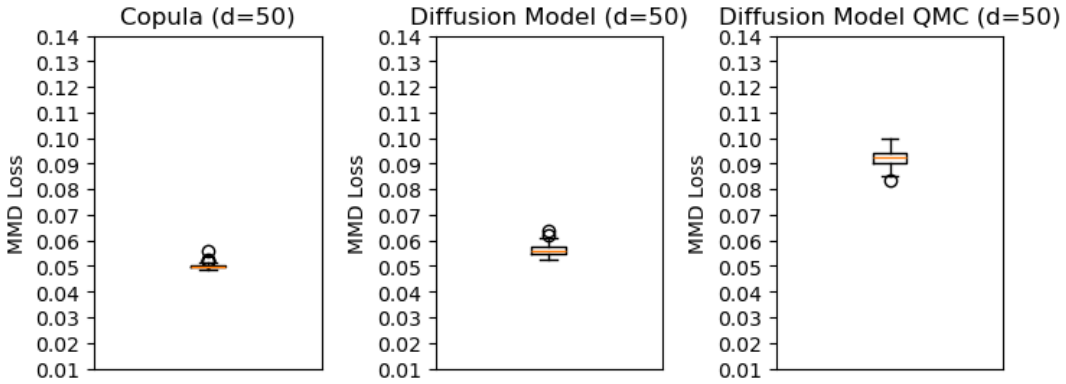
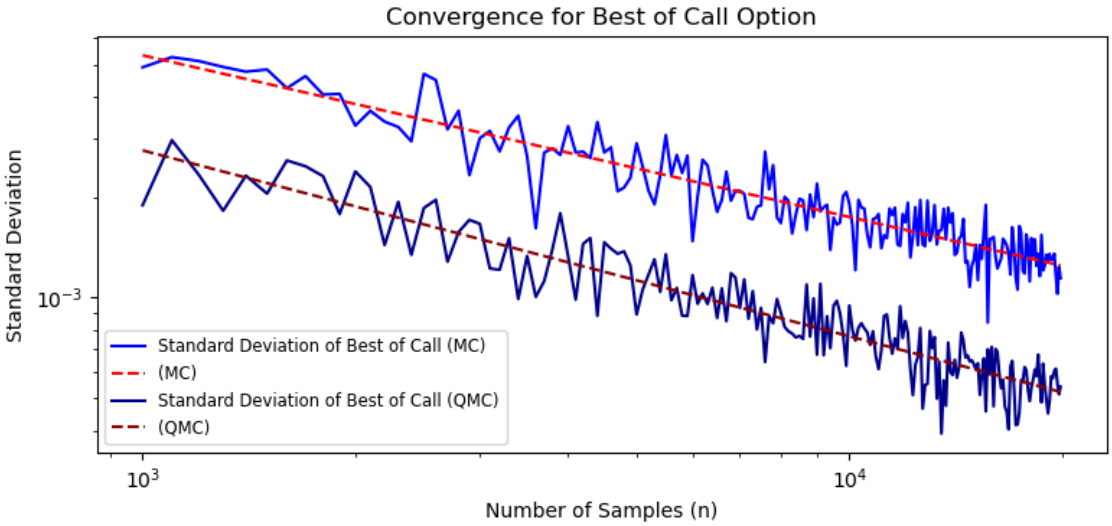
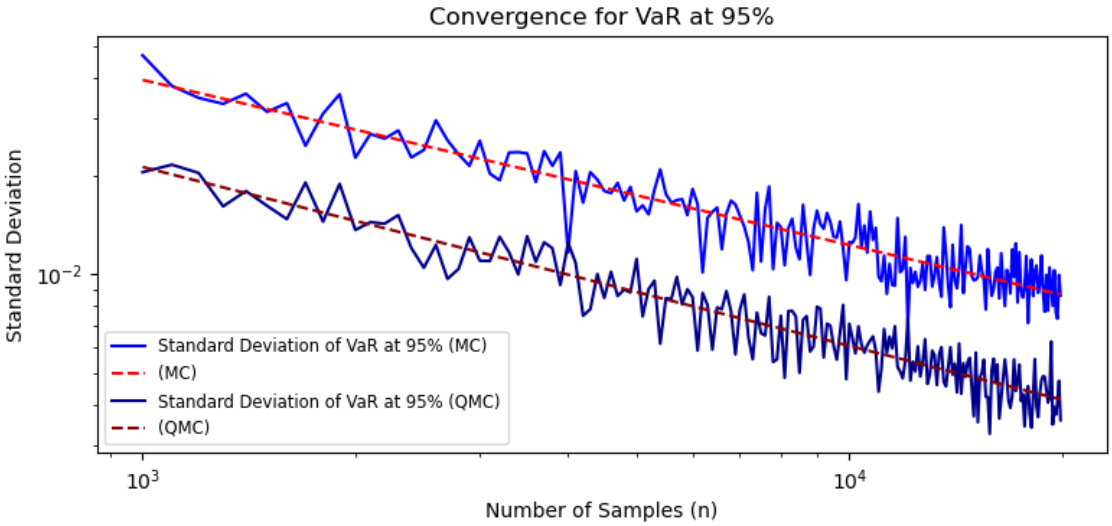


Figure 22: MMD of 50-dimensional Gumbel Copula and Diffusion Model trained with MC and QMC samples, $\theta = 2$, Copula MMD: 0.050, Diffusion Model MC: 0.056, Diffusion Model QMC: 0.092

For 50 dimensions, the MMD for the quasi-random samples is slightly higher than the diffusion model trained with random samples, with the MMD of the Diffusion model (QMC) reaching up to 0.089. We suggest that further investigation into this unexpected result to be done as quasi-random samples should have a lower discrepancy as compared to the pseudo-random samples.





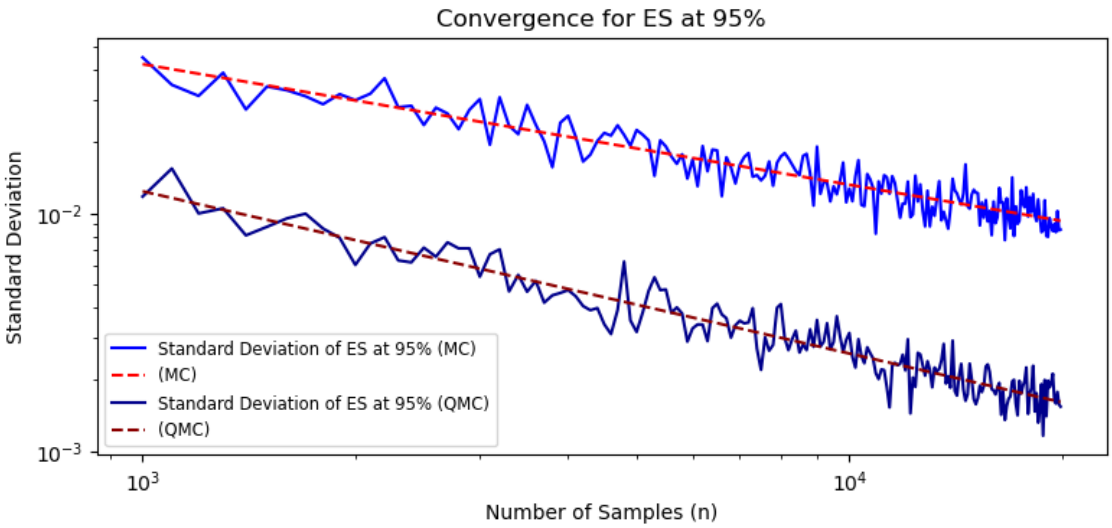


Figure 23: Regression coefficient for Best of Call (MC): -0.48, Regression coefficient for Best of Call (QMC): -0.56, Regression coefficient for VaR (MC): -0.50, Regression coefficient for VaR (QMC): -0.55, Regression coefficient for Expected Shortfall (MC): -0.51, Regression coefficient for Expected Shortfall (QMC): -0.68

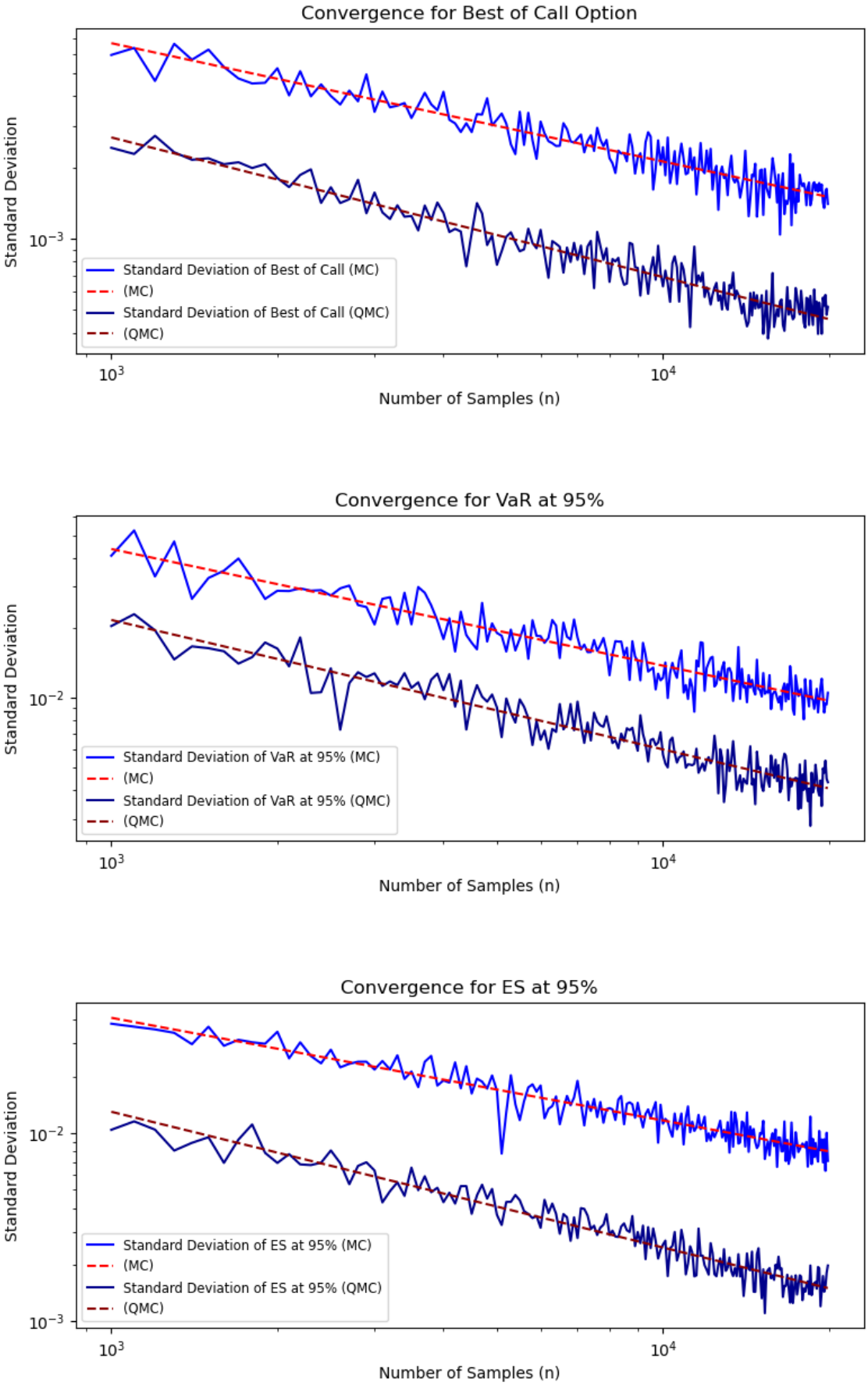


Figure 24: Regression coefficient for Best of Call (MC): -0.50, Regression coefficient for Best of Call (QMC): -0.59, Regression coefficient for VaR (MC): -0.50, Regression coefficient for VaR (QMC): -0.56, Regression coefficient for Expected Shortfall (MC): -0.55, Regression coefficient for Expected Shortfall (QMC): -0.72

For higher dimensions, the benefit from the low discrepancy sequence is less evident as the quasi-

Monte Carlo convergence rate is of $O((logN)^k N^{-1})$, so applying log to the convergence rate gives us a gradient of $klog(log(N)) - log(N)$, which at lower values of N will be larger than -1. Due to resource constraints, we did not run the simulation for larger samples on N. For further investigations, we suggest to run the experiment on larger values of N to make sure the Monte Carlo simulation benefits from fast convergence with using low discrepancy sequences.

3.5.6 Getting Low Discrepancy Properties As A Free Lunch

It was observed that diffusion models trained with Sobol sequences is able to retain the low discrepancy property in the generation of new samples. However, this defeats the purpose of training a quasi-random number generator if we already have a quasi-random number generator at hand. Therefore, this part will investigate an even more difficult task: whether a diffusion model trained with pseudo-random numbers can generate quasi-random samples from a copula by passing in quasi-random Gaussian noise into the neural network.

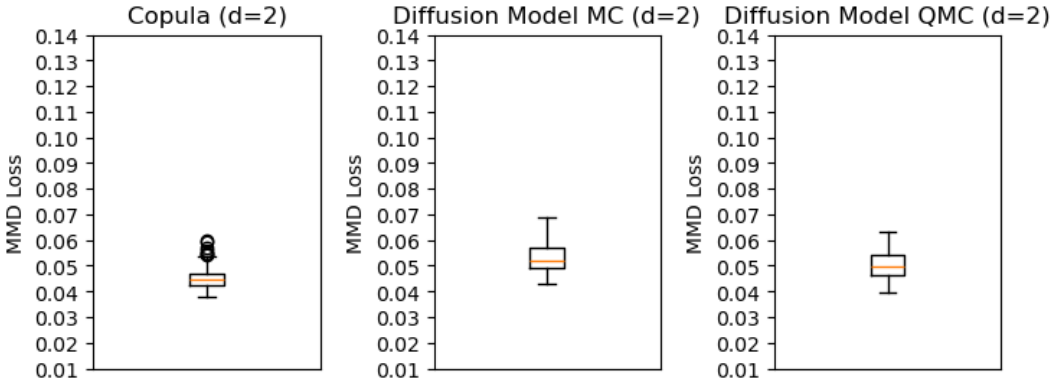


Figure 25: MMD of 2D Gaussian Copula and Diffusion Model trained with Pseudo Random Samples, $\rho = 0.8$, Copula MMD: 0.046, Diffusion Model MC MMD: 0.053, Diffusion Model QMC MMD: 0.052

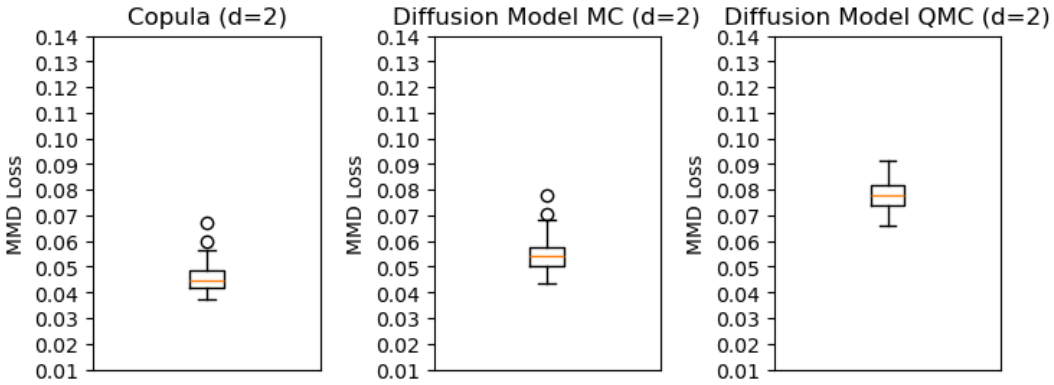


Figure 26: MMD of 2D Gumbel Copula and Diffusion Model trained with Pseudo Random Samples, $\theta = 2$, Copula MMD: 0.045, Diffusion Model MC MMD: 0.061, Diffusion Model QMC MMD: 0.084

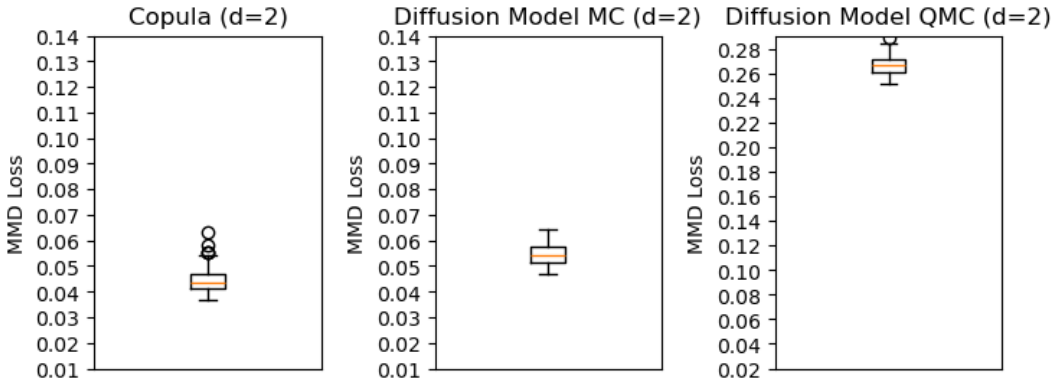
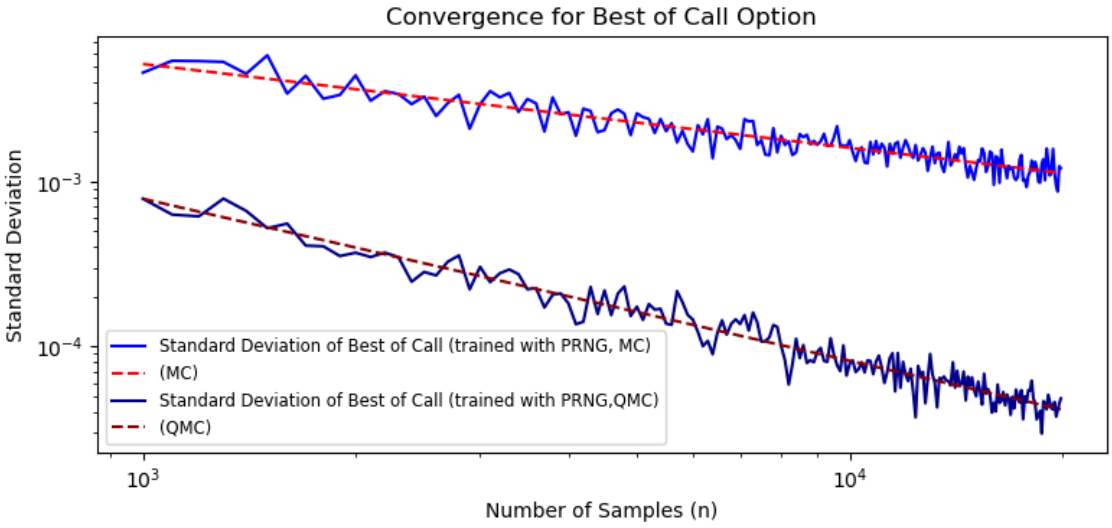
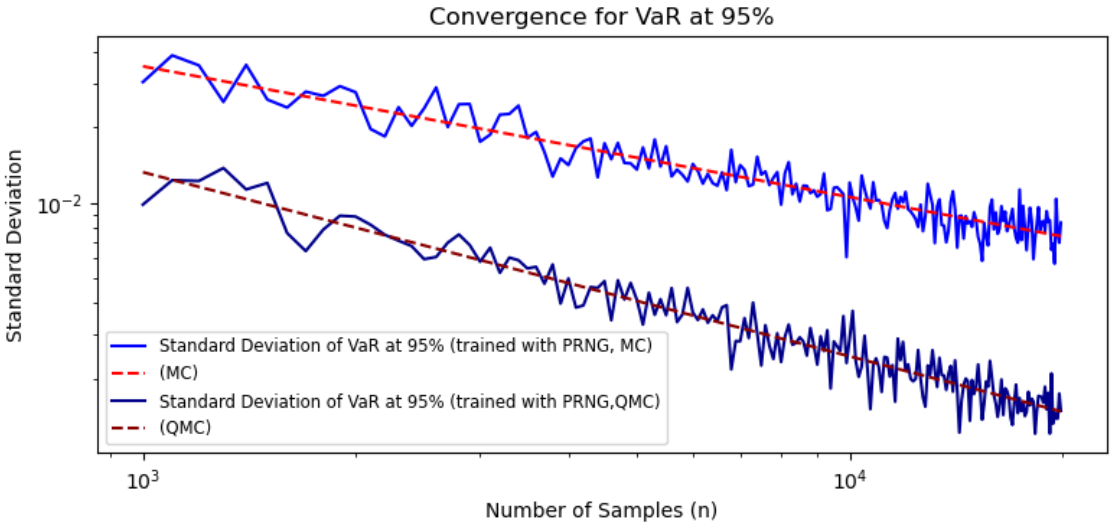


Figure 27: MMD of 2-dimensional Marshall-Olkin Copula and Pseudo-Copula trained with MC and QMC samples, $\theta = (0.2, 0.7)$, Copula MMD: 0.044, Diffusion Model MC MMD: 0.054, Diffusion Model QMC MMD: 0.27

From observing the plots, we notice that the MMD for the diffusion model with quasi-random noise is much higher than the MMD for the copula for the Gumbel and Marshall-Olkin copula. The MMD for the diffusion model for Marshall-Olkin copula is much more higher than the copula with a value of 0.27 vs 0.044 from the copula. This contradicts our expectation that the quasi-Monte Carlo sample will have a lower discrepancy against the copula as compared to the Monte Carlo samples. Similar to the previous sections, we suggest to perform further investigation into this phenomenon.





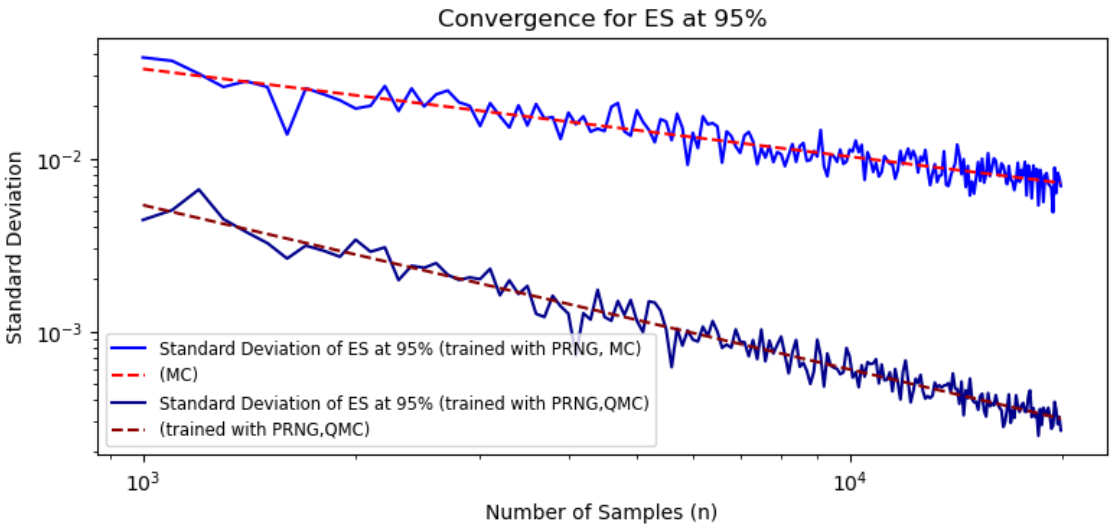
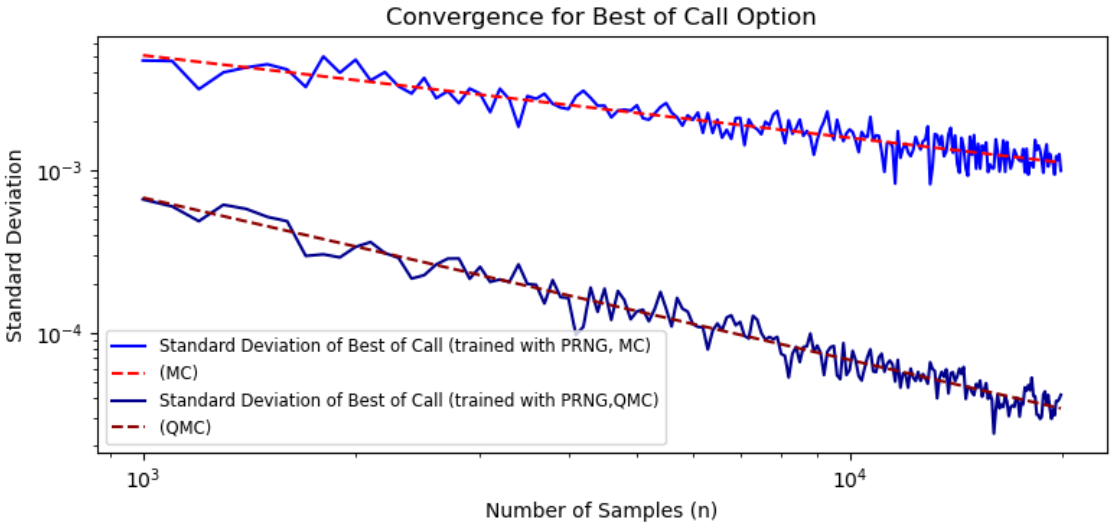
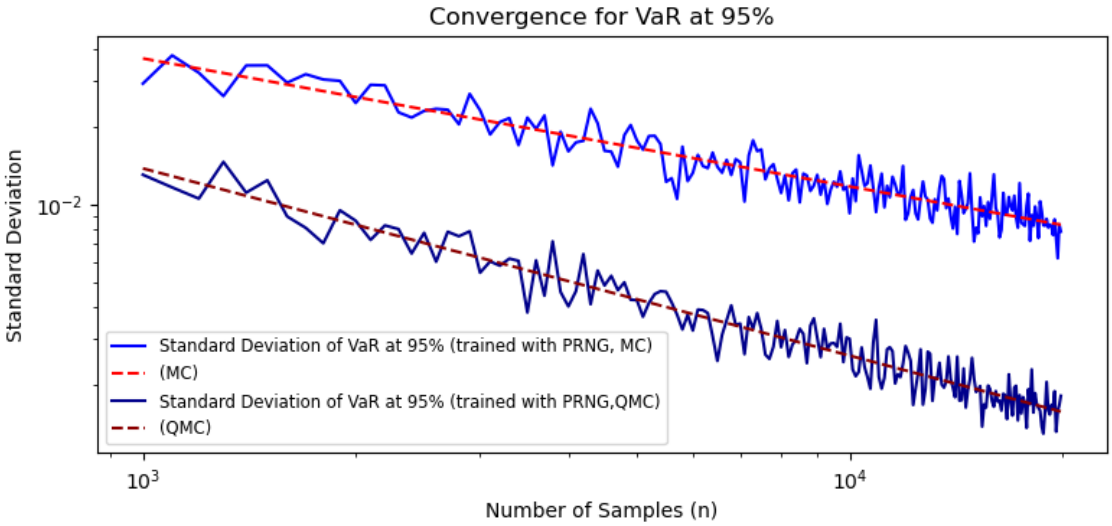


Figure 28: Regression coefficient for Best of Call (PRNG): -0.49, Regression coefficient for Best of Call (QRNG): -0.96, Regression coefficient for VaR (MC): -0.53, Regression coefficient for VaR (QMC): -0.73, Regression coefficient for Expected Shortfall (MC): -0.50, Regression coefficient for Expected Shortfall (QMC): -0.94





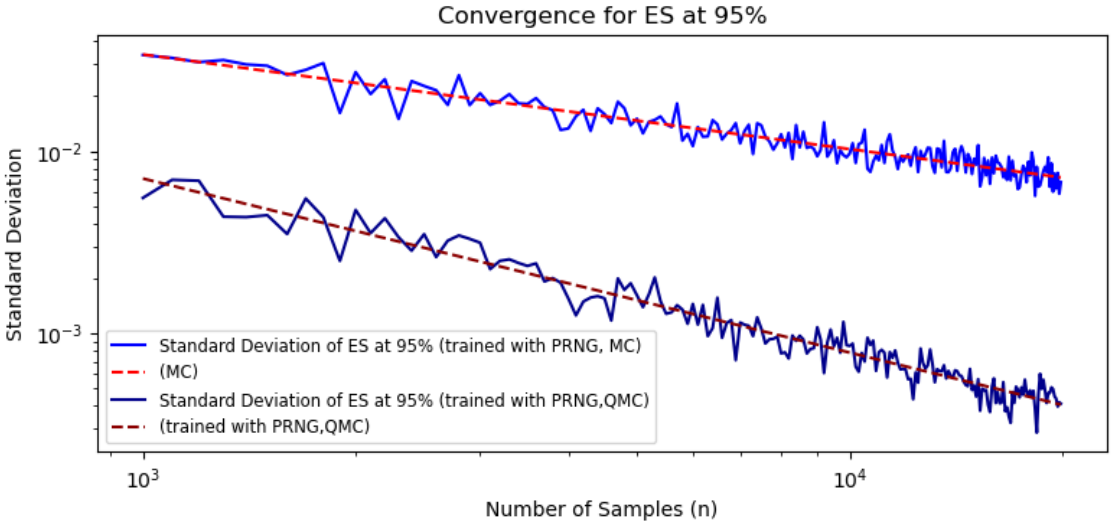


Figure 29: Regression coefficient for Best of Call (MC): -0.51, Regression coefficient for Best of Call (QMC): -1.00, Regression coefficient for VaR (MC): -0.50, Regression coefficient for Best of Call (QMC): -0.72, Regression coefficient for Expected Shortfall (MC): -0.52, Regression coefficient for Expected Shortfall (QMC): -0.96

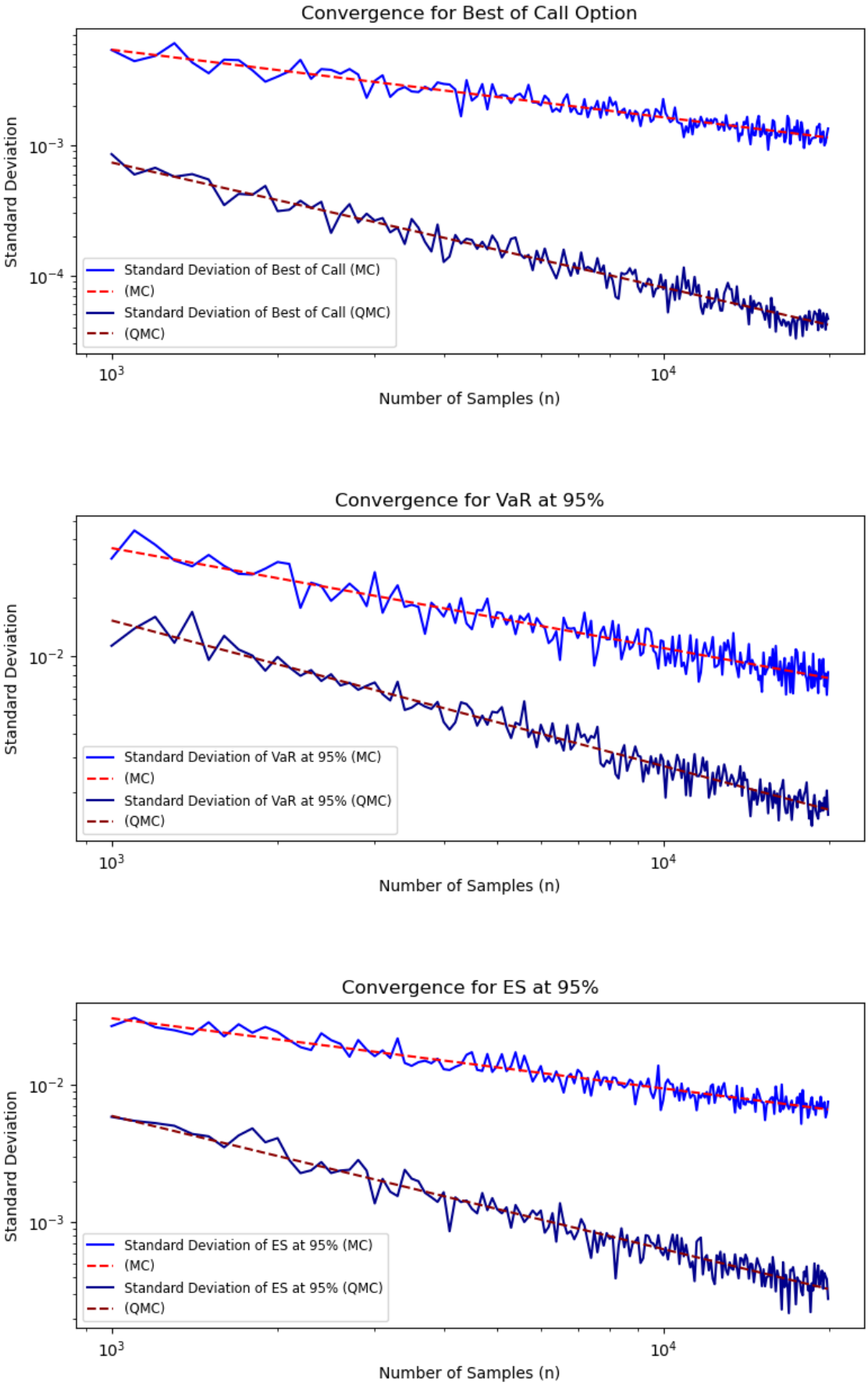


Figure 30: Regression coefficient for Best of Call (MC): -0.52, Regression coefficient for Best of Call (QMC): -0.96, Regression coefficient for VaR (MC): -0.52, Regression coefficient for Best of Call (QMC): -0.75, Regression coefficient for Expected Shortfall (MC): -0.51, Regression coefficient for Expected Shortfall (QMC): -0.97

From the graphs of the convergence rate, we notice that the convergence rate follows the Monte

Carlo and Quasi-Monte Carlo convergence rates. This shows that we can train our neural network using pseudo-random samples, then pass quasi-random samples into the diffusion model and use it as a quasi-random number generator. This will be useful for risk management as we can train the model on real-life data, then pass in quasi-random samples to perform quasi-Monte Carlo to improve the convergence rate of risk measure estimates.

After looking at 2-dimension copulas, we also scale our results up to a 50-dimension copula for both the Gumbel and Gaussian to investigate the ability of diffusion models to perform quasi-Monte Carlo simulation after being trained with pseudo-random numbers. The MMD for the Gaussian with quasi-random samples is quite similar to the Gaussian copula. However, for the Gumbel copula the quasi-Monte Carlo samples have a much larger MMD as compared to the Monte Carlo samples. We suggest further investigations into why the quasi-random property does not give us a lower MMD as we expected, especially for higher dimensions and also copulas with singularities.

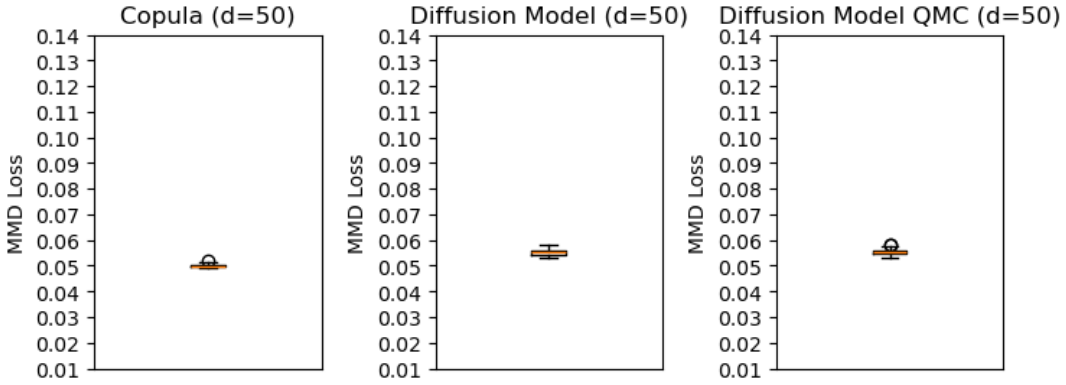


Figure 31: MMD of 50-dimensional Gaussian Copula and Diffusion Model trained with Pseudo Random Samples, $\rho = 0.8$, Copula MMD: 0.050, Diffusion Model MC MMD: 0.055, Diffusion Model QMC MMD: 0.055

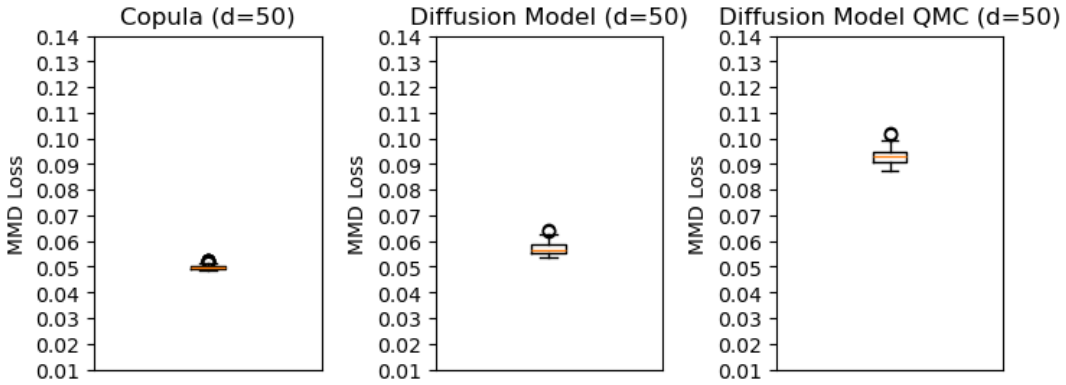


Figure 32: MMD of 50-dimensional Gumbel Copula and Diffusion Model trained with Pseudo Random Samples, $\theta = 2$, Copula MMD: 0.050, Diffusion Model MC MMD: 0.057, Diffusion Model QMC MMD: 0.093

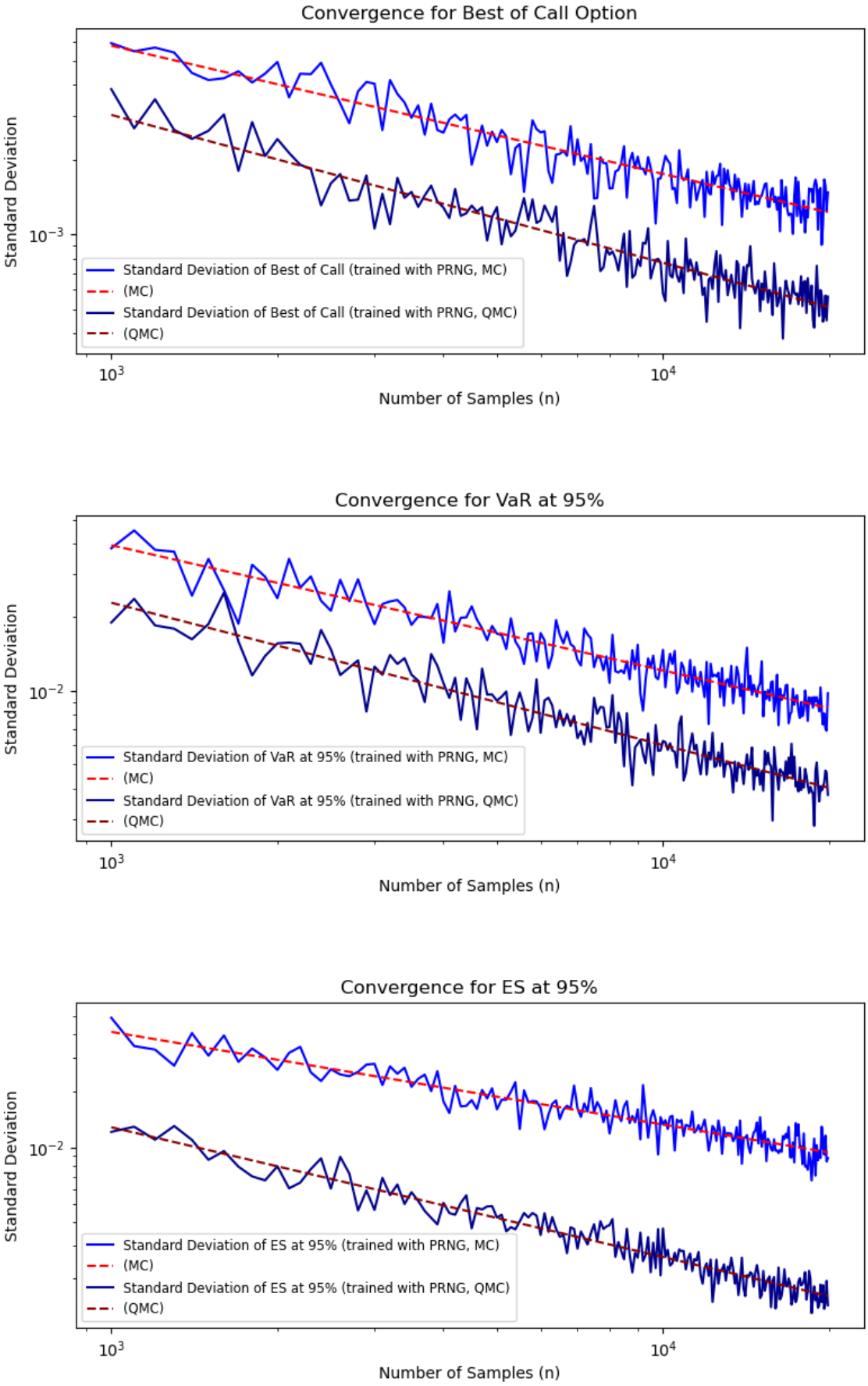


Figure 33: Regression coefficient for Best of Call (PRNG): -0.51, Regression coefficient for Best of Call (QRNG): -0.60, Regression coefficient for VaR (MC): -0.51, Regression coefficient for VaR (QMC): -0.59, Regression coefficient for Expected Shortfall (MC): -0.49, Regression coefficient for Expected Shortfall (QMC): -0.69

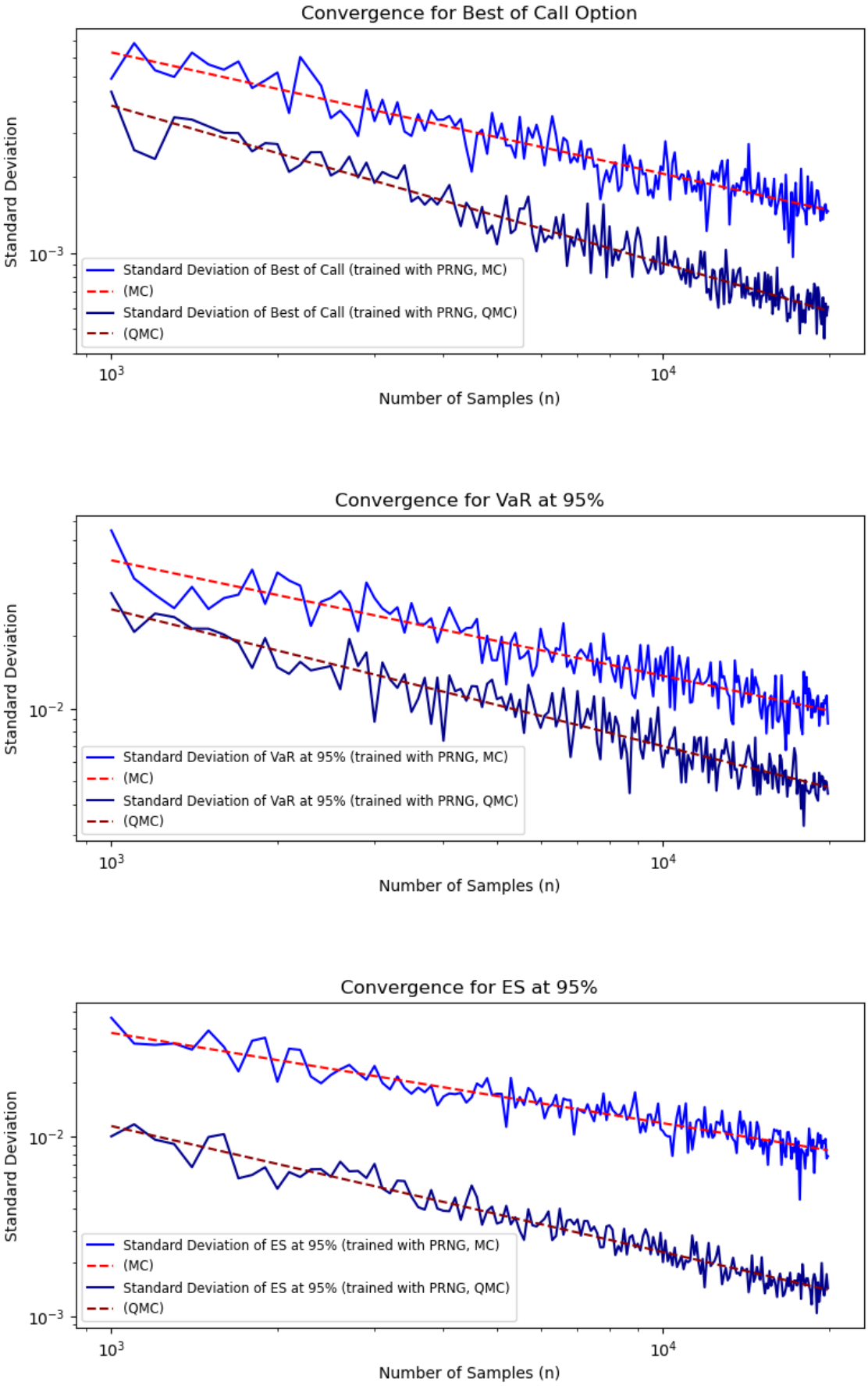


Figure 34: Regression coefficient for Best of Call (MC): -0.49, Regression coefficient for Best of Call (QMC): -0.63, Regression coefficient for VaR (MC): -0.49, Regression coefficient for VaR (QMC): -0.57, Regression coefficient for Expected Shortfall (MC): -0.50, Regression coefficient for Expected Shortfall (QMC): -0.70

The Quasi-Monte Carlo convergence for a 50-dimension diffusion model is much slower as compared

to the 2-dimension case. However, the quasi-Monte Carlo convergence is still quicker than the Monte Carlo simulation. Therefore, we suggest running the simulation for larger values of N to make sure the convergence rate is -1. This also implies that the benefits of using quasi-Monte Carlo simulations are diminished for higher dimensions.

4 Limitations in Our Investigation

4.1 Computational Power and Resource Constraints

While scaling up the diffusion model to 50 dimensions, we did not perform the Monte Carlo simulations for a larger number of iterations ($N > 20000$). Hence, we were unable to show that the convergence rate for quasi-Monte Carlo simulations equals to -1.

4.2 Limited Goodness-of Fit Tests

The goodness-of-fit tests used in this report were not comprehensive enough. We only used the MMD to compare the discrepancy between copula and diffusion model samples. Although it showed that the diffusion model with pseudo-random observations was able to learn the copula, the MMD for diffusion models with quasi-random noise was much larger than expected.

4.3 Limited Range of Copulas Tested

Due to time constraints, we only tested the ability of diffusion models to learn copulas for the Gaussian, Gumbel, and Marshall-Olkin copulas. We did not explore the ability of our model to learn more complex copulas like nested copulas and vine copulas.

4.4 Lack of Real Life Applications

In our report, we did not explore real-life applications of the diffusion model, primarily due to time constraints. While our report demonstrates our ability to learn copula structures, its practical applicability remains untested. Real-life datasets are often plagued with additional complexities, like noise, missing values, and non-stationarity, which may not be present in our experimental setups. This irregularity in real-life data causes copula modelling to be more difficult as most copulas will not be a good fit to real-life data due to such reasons.

4.5 Insufficient Justification and Proofs for the Underlying Theory

Due to time constraints, we were not able to prove why MSE is unable to learn a distribution. This will help us gain further understanding on how neural networks learn a distribution and what metrics can be used to describe a distribution.

We also did not manage to prove why the MSE with noisy inputs is superior to the MSE with uncorrupted inputs.

5 Topics for Further Investigation

5.1 Lower Bound on Sample Size Needed to Learn a Copula

Firstly, I would suggest investigating the minimum sample size needed to learn a copula. We generated 5000 pairs of copula and independent samples, then trained the 2-dimensional diffusion model for 2000 epochs with bootstrapped samples of size 5000. Therefore, we suggest further research on the lower bound for the number of samples needed to properly train the neural network to learn the copula. We

can check the goodness-of-fit test of the diffusion model and copula, or even compare the tail dependence of the diffusion model and copula as this is a much more important question to answer.

This will be extremely important for applications in the financial industry as the amount of data available for training is much more limited. Being able to train on the same data for multiple epochs without overfitting as opposed to other models will allow us to obtain a model that fits the data well.

5.2 Comparison Against Empirical Copulas

We can also compare our diffusion models against existing research on empirical copulas. For example, we can use diffusion models as empirical copulas to approximate other copulas. Further investigations can look into deep learning techniques that can approximate dependence better than existing empirical copula approaches. Another potential use of our diffusion model is to generate more samples to train empirical copulas so that we can get a smooth distribution.

5.3 Repeat Experiment for More Complex Copulas

In this report, we merely trained the diffusion model to learn the distributions of the Gaussian, Gumbel, and Marshall-Olkin copulas. We suggest exploring the ability of our diffusion model for different dependence structures, for example, changing the upper tail dependence for the copulas to confirm the diffusion model can learn the dependence for copulas of different tail dependence. We can also investigate whether diffusion models are able to learn nested copulas which are more complex than the elementary copulas presented in this report.

5.4 Repeat Experiment for Higher Dimensions to Confirm Quasi-Monte Carlo Convergence

To confirm that our procedure scales up to higher dimensions, further investigations can perform the Monte Carlo simulation for larger number of iterations of N to confirm that the quasi-Monte Carlo convergence is of value -1.

5.5 Alternative methods to Test for Goodness of Fit

While comparing the copula samples and diffusion model samples, due to resource constraints, we did not use the Cramer-von-Mises test for copula samples but instead used the MMD metric. This was because the code for the Cramer-von-Mises was in R and we had to use the rpy2 package in Python to call the test in R. Contrary to our expectations, the MMD for diffusion models (QMC) samples is higher than the MMD for diffusion models (MC), in which it should be lower. We suspect that the MMD metric is unable to show that the diffusion model generated samples through quasi-random noise are similar to copula. Hence, we propose to reimplement the Cramer-von-Mises test in Python and perform the goodness of fit test again.

Besides using the Cramer-von-Mises test, we can also use other tests to make sure the diffusion model is able to approximate the copula well. As copulas are usually used to model tail events, we should also look at the performance of the diffusion models on extreme events.

5.6 Investigate Original Diffusion Model Setup

We also propose investigating the original diffusion model setup as the generation process to generate new samples. However, preliminary results from diffusion models generated show that the samples tend to be squeezed to the edges of the uniform margin for large timesteps. As observed in our scatter plot of the generated samples, having a smaller timestep causes the observations to be more spread out.

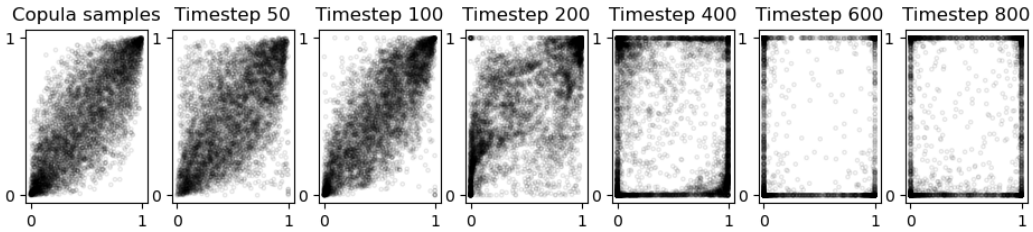


Figure 35: Scatter of Sampled Samples from the Original Diffusion Model Setup with U-Net

As none of the papers in our literature review mentioned that the diffusion model is able to generate samples from a distribution, we propose to investigate the ability of diffusion models to learn distributions.

5.7 Relationship with Maximum Mean Discrepancy (MMD)

We also suspect that adding noise to the inputs before training the neural network on the noisy inputs is somewhat equivalent to using the MMD to train the neural network. It may be possible that both of them are a special case of the integral probability metric. We will attempt to prove that training with noisy inputs is equivalent to minimizing the integral probability metric.

We also propose to investigate whether the MSE with noisy inputs is superior to the original MSE in terms of the loss between the simulated and original samples.

5.8 Application on Real Life Data

Other than merely focusing on theoretical aspects of diffusion models, we also propose to investigate the ability of diffusion models to model real-life data, especially financial data. We propose to use our diffusion model setup to build a pairs trading strategy as described in Hofert, Prasad, and Zhu [HPZ21]. After fitting an ARIMA-GARCH model on the data, and normalizing the data with the ARIMA-GARCH model, we will then use the diffusion model to model the dependence structure in the normalized returns.

We can also test our model's capability to model relationships for real-life phenomena. For instance, modelling the temperature or water level observations for different nearby weather stations and comparing their performance against existing copula methods. It will be interesting to investigate whether neural networks are able a better representation of the data as compared to a statistician using copula modelling. Real-life data is usually hard to model using copulas due to their irregularity and lack of a defined structure. Further experiments on real-life data will help test the ability of diffusion models to learn complex and irregular patterns and improve existing modelling practices.

List of Algorithms

References

[Ros52] Murray Rosenblatt, Remarks on a Multivariate Transformation, *The Annals of Mathematical Statistics*, 23 (3) (1952), 470–472, doi:10.1214/aoms/1177729394, https://doi.org/10.1214/aoms/1177729394.

[Bis95] Chris M. Bishop, Training with Noise is Equivalent to Tikhonov Regularization, *Neural Computation*, 7 (1) (Jan. 1995), 108–116, ISSN: 0899-7667, doi:10.1162/neco.1995.7.1.108, eprint: https://direct.mit.edu/neco/article-pdf/7/1/108/812990/neco.1995.7.1.108.pdf, https://doi.org/10.1162/neco.1995.7.1.108.

[Joe14] Harry Joe, Dependence Modelling with Copulas, 1st ed., Taylor Francis Group, 2014, XX, 479.

[KB17] Diederik P. Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimization, 2017, arXiv: 1412.6980 [cs.LG], https://arxiv.org/abs/1412.6980.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel, Denoising Diffusion Probabilistic Models, *CoRR*, abs/2006.11239 (2020), arXiv: 2006.11239, https://arxiv.org/abs/2006.11239.

[HPZ21] Marius Hofert, Avinash Prasad, and Mu Zhu, Multivariate time-series modeling with generative neural networks, 2021, arXiv: 2002.10645 [stat.ME], https://arxiv.org/abs/2002.10645.

[BB23] Christopher M. Bishop and Hugh Bishop, Deep Learning, Foundations and Concepts, 1st ed., Springer Cham, 2023, XX, 649, doi:https://doi.org/10.1007/978-3-031-45468-4.

[Yan+24] Ling Yang et al., Diffusion Models: A Comprehensive Survey of Methods and Applications, 2024, arXiv: 2209.00796 [cs.LG], https://arxiv.org/abs/2209.00796.