TEncDM: Understanding the Properties of Diffusion Model in the Space of Language Model Encodings

Anonymous ACL submission

Abstract

Drawing inspiration from the success of diffusion models in various domains, numerous research papers proposed methods for adapting them to text data. Despite these efforts, none of them has managed to achieve the quality of the large language models. In this paper, we conduct a comprehensive analysis of key components of the text diffusion models and introduce a novel approach named *Text Encoding Diffusion Model* (**TEncDM**). Instead of the commonly used token embedding space, we train our model in the space of the language model encodings. Additionally, we propose to use a Transformer-based decoder that utilizes contextual information for text reconstruction. We also analyse self-conditioning and find that it increases the magnitude of the model outputs, allowing the reduction of the number of denoising steps at the inference stage. Evaluation of **TEncDM** on two downstream text generation tasks, QQP and XSum, demonstrates its superiority over existing non-autoregressive models.

1 Introduction

Autoregressive (AR) large language models such as GPT-4 (OpenAI, 2023) or Llama 2 (Touvron et al., 2023) are the current gold standard in the text generation problem. They are capable of creating high-quality and coherent texts that are practically indistinguishable the from human ones. However, the disadvantage of this approach is the inability of the model to correct its own mistakes made during generation. This may cause the text that follows such mistakes to be spoiled. In addition, the autoregressive method of token generation slows down the inference process as it requires performing a single model evaluation for each new token.

Diffusion model is currently the state-of-the-art approach for data generation in image (Rombach et al., 2022; Betker et al.), audio (Evans et al., 2024) and video (Blattmann et al., 2023) domains. They are a class of probabilistic generative models that are able to iteratively transfer noise to a representative sample of data. While some of the proposed text diffusion models are autoregressive (Lovelace et al., 2022; Zhang et al., 2023), the majority of them are not and, by the design, they have several advantages over AR language models. First, being non-autoregressive (NAR) models, they generate all the tokens simultaneously and can adjust any part of the sequence during the generation process. They also can be faster than AR models because the number of neural function evaluations for diffusion models depends on the number of denoising iterations rather than the length of the sequence. And given the possibility of distillation of diffusion models (Meng et al., 2023), the number of iterations can be greatly reduced.

To date, a number of text diffusion models have been proposed, each based on substantially new ideas with little overlap with other methods. Some works replace Gaussian noise with categorical noise (Hoogeboom et al., 2021; Austin et al., 2021), exploiting the discreteness of the text domain. Others train continuous diffusion on token embeddings (Li et al., 2022; Lin et al., 2023; Gong et al., 2023) or text latent representations reduced in size (Lovelace et al., 2022; Zhang et al., 2023). There are also differences in the way diffusion outputs are decoded back into text. Diffusion models trained on embeddings round their predictions to the nearest embeddings, while those that utilize small latent spaces decode the predictions with an AR model. This suggests that the scientific community has not found the most robust diffusion model design yet.

In this paper, we attempt to better understand the specifics of text distribution models and identify best practices for their development. We investigate each component in detail: text encoding and decoding methods, diffusion model architecture, noise schedule, and self-conditioning (Chen et al., 2023). As a result, we combine all our findings in a method called **Text Encoding Diffusion Model (TEncDM)**. It constructs the diffusion model, which operates in the latent space of the language model encodings (e.g. BERT (Devlin et al., 2019)). It also utilize the Transformer-based (Vaswani et al., 2017a) decoder, which is able not only to decode the latents but also to improve the text quality. We do not use an AR decoder on purpose so as not to transfer the limitation of AR language models to the diffusion.

We compare our approach with other works on two conditional text generation problems: **paraphrasing** and **summarization**, on which our method surpasses all on-autoregressive models. The main contributions of this work are as follows:

- We propose a new text diffusion framework **TEncDM**, which trains the diffusion model in the latent space constructed by the outputs of pre-trained Transformer-based encoder.
- We evaluate the importance of the decoder and conclude that its robustness to inaccuracies in the generated latents directly affects the generation quality. We then propose a decoder architecture and its training method that boosts the model performance.
- We analyse in detail the effect of self-conditioning on the denoising process and show that self-conditioning increases the magnitude of model's predictions, which in turn allows us to reduce the number of denoising steps during inference.
- Through a thorough ablation study, we reveal that commonly used *cosine* and *sqrt* noise schedules do not introduce enough difficulty to the denoising task during training. We show that the addition of more noise significantly increases the model quality.

2 Problem Statement and Background

Text generation problem. In the field of natural language processing, unconditional text generation is a task of sampling y from the unknown distribution $p(y)$, where $y = [y_1, \dots, y_n]$ is a sequence of tokens with variable length n. In conditional text generation the distribution of texts changes to $p(y|x)$, where x is a condition variable. The goal is to generate a text, that satisfies this condition.

Autoregressive language models. The most common approach for text generation is autoregressive (AR) left-to-right sampling of words. The idea is to approximate the factorised distribution $p(y) = \prod_{i=1}^{n} p(y_i|y_{<i})$ by learning a neural network $p_\theta(y_i|y_{<i})$. During generation, tokens are sampled sequentially with conditioning on the already generated ones.

Gaussian diffusion models. The standard diffusion models (Ho et al., 2020; Song et al., 2021) learn to sample data from an unknown distribution by gradually denoising random Gaussian noise. The train procedure is defined through a forward diffusion process that satisfies $q(z_t|z_0) = \mathcal{N}(\sqrt{\alpha_t} z_0, (1-\alpha_t)\mathbf{I})$, where $\alpha_t \in [0, 1]$ is a predefined noise schedule, $t \in [0, 1]$. The denoising network (parameterized by θ) is trained to reconstruct the original latent z_0 given the noisy latent z_t, as expressed in equation 1

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\varepsilon \sim \mathcal{N}(0,\mathbf{I}), t \sim U[0;1]} [||z_0 - \hat{z}_\theta(z_t, t)||^2] \quad (1)$$

Sampling procedure starts from a pure Gaussian noise $z_T \sim \mathcal{N}(0, \mathbf{I})$ and utilizes the denoising network to iteratively generate latents $z_{t_{T-1}}, ..., z_{t_1}$, where $1 = t_T > t_{T-1} > ... > t_1 = 0$.

Diffusion models for text generation. The primary feature of the text domain is the discreteness of its samples. In order to train a diffusion model on them, they must first be translated into continuous space. Consequently, alongside the denoising model, the diffusion framework incorporates an *encoder* that maps tokens into the continuous latents and a *decoder* that performs the reverse operation, converting the generated latents into text.

3 Related Work

Embedding-based diffusion models. The majority of proposed text diffusion models use embeddings of tokens to construct the continuous latent space (Li et al., 2022; Lin et al., 2023; Strudel et al., 2022; Gong et al., 2023; Wu et al., 2023). At the inference stage, to convert the latent predictions into text, they map each latent vector to a token corresponding to the nearest embedding.

Latent diffusion models. Other studies suggest reducing the size of the latent space by training a diffusion model in the space of text autoencoder. PLANNER (Zhang et al., 2023) finetunes BERT (Devlin et al., 2019) to store all information in the first k hidden state vectors from its final layer and use them as a latent space. LD4LG (Lovelace et al., 2022) trains the compression network to reduce both the length and dimensionality of the latent space sample. Both methods utilize autoregressive language models to decode the latents into text.

Self-Conditioning. Self-conditioning is a technique that significantly increases the performance of the diffusion model (Chen et al., 2023; Strudel et al., 2022; Lovelace et al., 2022). Usually the model is conditioned only on the latent variable z_t and the current timestep t as $\hat{z}_0^t = \hat{z}_\theta(z_t, t)$. Self-conditioning proposes to also condition the model on the estimation of data sample from the previous timestep during generation in order to improve the prediction at the current timestep, $\hat{z}_0^t = \hat{z}_\theta(z_t, t, \hat{z}_0^{t-1})$.

Although widely used, no analysis has been conducted to determine why this method is effective or how it impacts the generation process.

Noise scheduler. Noise scheduler is a key component of a diffusion model that controls the amount of noise added on each timestep. Previous research (Li et al., 2022; Gao et al., 2023; Ye et al., 2023) has highlighted that the standard noise schedulers used for image diffusion models are unsuitable for the textual domain. Due to the discrete nature of the texts, it is unlikely that an addition of a small amount of noise to a latent will change its nearest text in the latent space. Therefore, to increase the difficulty of the denoising task for the model, the mentioned works recommend adding more noise on iterations that are close to 0.

4 Understanding Text Diffusion

In this section, we present our findings on the components of the diffusion model, discuss their weaknesses and propose ways to enhance them.

Encodings are better than embeddings. Most diffusion models utilize token embeddings to map text into a continuous latent space. However, this approach is not optimal because the embeddings do not convey contextual information. This requires the diffusion model to independently search for it to retrieve ambiguous tokens. To simplify the task, instead of embeddings, we can use the final layer outputs of a pre-trained language model (e.g. BERT). They contain this information and, thus, should be more suitable for training the diffusion model. We refer to these outputs as *encodings*.

Experimental results confirming our intuition are presented in Section 7.3. It is worth noting that the use of encodings does not slow down the generation process, as we need to compute them only during the training.

To improve the quality even further, it is possible to fine-tune the encoder, but we choose not to in order to avoid overcomplicating the approach. Investigation into fine-tuning is left for the future work.

Decoder is important. The purpose of the decoder in the diffusion model is to map the generated latents into text. Approaches that train diffusion in the space of token embeddings decode latents by rounding them to the nearest embeddings and selecting a corresponding token. However, the diffusion model may produce inaccurate latent samples due to accumulation of errors during the denoising process. Such inaccuracy might significantly spoil the text quality, so it would be wise to train a decoder that could improve it.

In the Section 7.4, we compare different decoder designs and conclude that an advanced decoder, which can consider the context for each token, indeed improves the generation quality.

Self-conditioning affects denoising dynamics. Self-conditioning improves sampling quality by conditioning the model on its previous prediction. However, the mechanics of self-conditioning are not fully understood yet. Our research demonstrates that the addition of self-conditioning increases the model's prediction confidence at each denoising timestep, resulting in a reduction in the required number of generation steps. Furthermore, the sample quality diminishes as the number of steps increases. We believe that a reason for this behaviour lies in a mismatch between the latents used at the training stage and those at the generation stage. We provide the evidence supporting our conclusions in Section 7.5, along with a comprehensive analysis of the model's behaviour with and without self-conditioning.

Diffusion needs even more noise. Following the recommendations of previous works (Li et al., 2022; Wu et al., 2023; Ye et al., 2023), we used *sqrt* noise scheduler that increases the amount of noise added to the diffusion model inputs during training beyond the amount of typically used *cosine* noise scheduler (Han et al., 2022; Lovelace et al., 2022; Strudel et al., 2022; Zhang et al., 2023). However, our experiments led us to conclusion that encoding-based diffusion model requires even more noise for successful training. We hypothesize that this

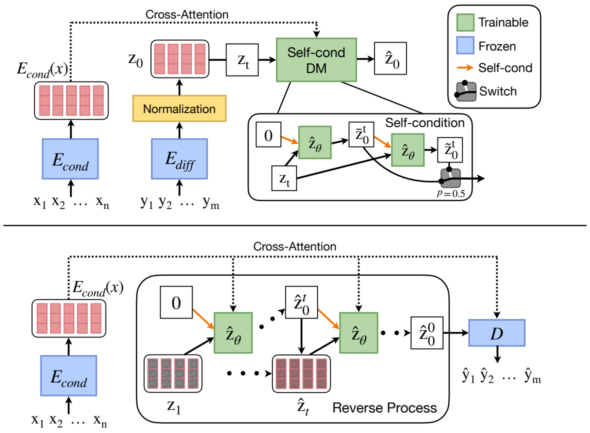

Figure 1: Overview of our framework design for conditional generation. Top is the training process, bottom is the generation process.

is due to the presence of contextual information in the encodings, which simplifies the denoising task.

In Section 7.6 of this study, we demonstrate that both commonly used *cosine* and *sqrt* noise schedules do not introduce a significant level of noise to the latent variables over a wide range of timesteps. As a result, the denoising task becomes too simple for the model, leading to a reduction in the effectiveness of the training signal.

5 Methodology

The design of **TEncDM** is depicted on Figure 1. It consists of three parts – diffusion encoder E_{diff}, diffusion model $\hat{z}_\theta$ and decoder D. For the conditional generation, we also add conditional encoder E_{cond}, which encodes an input text. Its output is provided to the diffusion model and decoder through cross-attention.

This section exclusively focuses on the topic of unconditional text generation. The details of the conditional model can be found in Section 5.5.

5.1 Diffusion encoder, E_{diff}

We use pre-trained Transformer-based (Vaswani et al., 2017a) language model E_{diff}, which we call diffusion encoder, to encode text y into the latent space z. Encoding of text does not change the length of the sequence. In order to align all texts in length, we add paddings to the end of short texts. After encoding the text, the encodings of all special tokens are replaced by their corresponding embeddings. This is necessary because diffusion model does not use an attention mask during training, which means that the reconstruction loss is calculated for both text and special tokens. However, special token encodings usually contain meaningless values, because encoder does not learn to store useful information in them. Therefore, minimization of reconstruction loss for these encodings only harms the diffusion training process. Embeddings of special tokens, on the other hand, only contain information about the token itself and the diffusion model recovers them much easier.

5.2 Decoder, D

The decoder D is required to convert latent variables generated by diffusion model into textual output. Although a basic linear decoder can effectively reconstruct tokens with high accuracy, we employ the BERT (Devlin et al., 2019) architecture for the decoder to provide it with the ability to capture context information and rectify potential errors originating from the diffusion model.

We train the decoder independently of the diffusion model using the following objective

$$-\mathbb{E}\log p_D(y \mid Cor(z_0)) \to \min_D, \qquad (2)$$

where $Cor(z_0)$ is a corrupted latent variable extracted from the diffusion encoder. Corruption is needed to expand the decoder training data domain and make it robust to distribution mismatch between text encodings z_0 and latents $\hat{z}_0$ generated by the diffusion model. This mismatch might arise due to the accumulation of errors during the denoising process. Its presence is especially evident for special tokens, which always have the same fixed representations in z_0. By default, we take $Cor(z_0)$ to be z_t with randomly sampled $t \in [0, 0.15]$. We use the diffusion's noise scheduler to calculate z_t.

5.3 Diffusion model, $\hat{z}_\theta$

The diffusion model consists of 12 BERT layers and it is trained to reconstruct the original latent z_0 given its noisy version z_t and a timestep t by minimizing the objective (1). We provide the model with information about the timestep by adding its embedding to the hidden state vectors of each layer.

We train the diffusion model using the variance preserving scheme, discussed in (Song et al., 2021). To achieve zero mean and unit variance we normalize the latent variables z_0 coordinate-wise, using the statistics from the training set.

Noise scheduler We adopt the noise scheduler from (Hoogeboom et al., 2023) and use the following equation for α_t:

$$\alpha_t = \frac{1}{1 + \tan(t\pi/2)^2 \cdot d^2}, \quad (3)$$

where d is a hyperparameter controlling the rate at which noise is introduced into the system. We set $d = 9$ by default, which corresponds to a significantly higher noise addition rate than what is used in all common noise schedulers. We further refer to our scheduler as *tan-d* noise scheduler.

Self-condition Following the previous approaches (Lovelace et al., 2022; Strudel et al., 2022) we incorporate self-conditioning into the diffusion model. In order to make the model utilize the data sample estimation from the previous generation step, we modify the training procedure.

According to (Chen et al., 2023) we design the training process to emulate the inference behavior. On each training iteration with the probability $p = 0.5$ the prediction is computed with the self-conditioning set to zero $\bar{z}_0^t = z_\theta(z_t, t, 0)$. And, with probability $(1 - p) = 0.5$ we first calculate $\bar{z}_0^t = z_\theta(z_t, t, 0)$ and then use it as an estimation of the data sample to obtain a second prediction $\tilde{z}_0^t = z_\theta(z_t, t, \text{SG}(\bar{z}_0^t))$, where SG is the stop-gradient function that does not allow the gradient to flow through $\bar{z}_0^t$. The diffusion model is optimized using the output $\bar{z}_0^t$ in the former scenario and $\tilde{z}_0^t$ in the latter. This training strategy allows the model to accurately approximate z_0 both with and without self-conditioning. We implement self-conditioning in a same manner as conditioning on timestep. For each diffusion model layer we pass the data estimation through a single linear layer and add it to the hidden state vectors.

5.4 Generation process

The generation process is illustrated on the Figure 1 (bottom). To generate text in the inference phase, we start with a random Gaussian sample and denoise it in T steps using the Euler solver. At each step, we apply self-conditioning and, because of it, use a small number of steps – 50 by default.

5.5 Conditional generation

For the conditional generation we keep the framework design similar to unconditional generation. The only difference is that we add conditional encoder to process the input text and provide both diffusion model and decoder with its output via cross-attention. Implementation details can be found in Appendix E.

6 Datasets

To evaluate the performance of our diffusion models we use three datasets in English language. The **ROCStories** (Mostafazadeh et al., 2016) dataset contains 98k five-sentence commonsense fictional stories, that capture causal and temporal relations between daily events. The subset of **QQP** (Chen et al., 2017) dataset, proposed in (Gong et al., 2023), consists of 144k question pairs from the Quora platform that are paraphrases of each other. The **XSum** (Narayan et al., 2018) dataset is used for summarization problem and it contains 204k BBC articles, which are provided as document and summary pairs[1]. The detailed statistics for each dataset can be found in Appendix F.

7 Empirical Analysis

In this section, we evaluate the components of our framework on the **ROCStories** dataset. To simplify the setup, we only consider unconditional generation. In Section 8, we demonstrate that our findings can be successfully transferred to the conditional generation problems. In this section, we do not compare our method with others. The comparison with the GPT2 is presented in Appendix G.

7.1 Evaluation Metrics

We follow the model evaluation scheme from the (Lovelace et al., 2022). To evaluate the quality of our model we use **Perplexity (ppl)**, calculated with GPT-2 Large (Radford et al., 2019). To measure the diversity of the generated text we utilize the diversity metric proposed in (Su et al., 2022). We calculate it as $\textbf{div}(y) = \prod_{n=2}^{4} \frac{|\text{\# of unique n-grams in } y|}{|\text{\# of n-grams in } y|}$, where y is a set of generated texts. To ensure that the model does not reproduce the training dataset during the generation we evaluate the **Memorization (mem)**. We calculate it as the proportion of generated 4-grams that are found in the training set. As Perplexity tends to be small for the texts with repetitions, we also use **MAUVE Score** (Pillutla et al., 2021) to estimate the quality of text. MAUVE is a language model-based metric that measures the distance between the distributions of generated and reference texts using divergence frontiers. We leave all MAUVE hyperparameters at the default values presented in the original paper.

[1]All the datasets we use in this work are publicly available under a creative commons or an open source license.

Encoder	**ppl** $\downarrow$	**mem** $\downarrow$	**div** $\uparrow$	**mauve** $\uparrow$
BERT emb	$48.9_{.36}$	$.371_{.003}$	$.324_{.002}$	$.600_{.016}$
BERT	$\mathbf{34.1}_{.66}$	$.412_{.005}$	$.304_{.006}$	$\mathbf{.707}_{.024}$
T5	$47.7_{.66}$	$.361_{.001}$	$.330_{.001}$	$.475_{.008}$

Table 1: Comparison of diffusion encoders.

To calculate all the metrics, we generate 1000 texts. For MAUVE, we sample 1000 reference texts from the test set. We repeat this procedure 5 times and report the mean and standard deviation of the obtained results in mean_{std} notation.

7.2 Model setup

The training of our diffusion model is conducted within the latent space of BERT encodings, as it has shown the best performance among all encoders. We employ a 3-layer transformer for the decoder and train it to reconstruct z_0 from z_t, where $t \in U[0, 0.15]$. A comprehensive analysis of various decoder modifications is presented in Section 7.4 and Appendix B. The diffusion model is the 12-layer transformer with dimensionality of 768. By default we train it with *tan-9* noise scheduler.

7.3 Effect of Diffusion Encoder

We compare latent spaces of BERT (Devlin et al., 2019) (`bert-base-cased`) and T5 (Raffel et al., 2020) (`t5-base`) encodings, as well as BERT embeddings, to ascertain the optimal choice for the diffusion model. In this experiment, we train diffusion models with the same set of hyperparameters across all diffusion encoders. We train the decoders according to the scheme described in Section 7.2. The results of this comparison are presented in Table 1 and they show a clear advantage of the latent space derived from BERT encodings. **div** and **mem** for T5 encoder and BERT embeddings are better, because their generated texts include words that do not fit the context. The text samples are presented Table 9 of Appendix H. This confirms our hypothesis that encodings are better suited for the training of a diffusion model.

7.4 Effect of Decoder

To confirm the hypothesis about the importance of the decoder architecture and its training scheme, we compare an MLP decoder consisting of two linear layers with a 3-layer transformer. We corrupt the decoder input z_0 by transforming it into z_t, using the diffusion forward process with $t \in U[0, 0.15]$. We choose this method, because it brings the decoder input closer to the diffusion output. A more detailed analysis of corruption techniques is presented in the Appendix B. To keep the experiment fair, we apply all decoders to the same generated latents. The results of the experiment are shown in Table 2. The MLP decoder achieves the worst text quality, because it overfits on the special token embeddings and fails to decode them from the generated latents. Examples of the generated samples are shown in Appendix H. Corruption of the input helps to avoid overfitting. At the same time, incorporating contextual information into the decoder increases the quality even more

Decoder	**ppl** $\downarrow$	**mem** $\downarrow$	**div** $\uparrow$	**mauve** $\uparrow$
MLP	$607.1_{15.6}$	$.332_{.003}$	$.400_{.004}$	$.004_{.00}$
$+ Cor(z_0)$	$36.2_{1.8}$	$.415_{.005}$	$.301_{.006}$	$.650_{.03}$
Transformer	$40.4_{.86}$	$.408_{.005}$	$.308_{.006}$	$.568_{.02}$
$+ Cor(z_0)$	$\mathbf{34.1}_{.66}$	$.412_{.005}$	$.304_{.006}$	$\mathbf{.707}_{.02}$

Table 2: Comparison of decoders for encoding-based diffusion model.

7.5 Effect of self-conditioning

We conduct a series of experiments to understand how self-conditioning affects the diffusion model. In Figure 2, we compare the quality of the models with and without self-conditioning for different number of denoising steps. The results show that while the quality of the model without self-conditioning increases as the number of steps increases, the quality of the model with self-conditioning reaches a maximum at a value of 50 steps in terms of MAUVE, after which it starts to drop. Nevertheless, at the highest point model with self-conditioning surpasses the model without it according to both MAUVE and perplexity.

We explain this drop in generation quality with mismatch between diffusion model inputs at train and inference stages. To confirm our hypothesis, we calculated the mean-squared norm (*magnitude*) of the values of each latent $\hat{z}_0^t$ in a mini-batch predicted by the diffusion model during generation (i.e. $\frac{1}{N \cdot d \cdot m} ||\hat{z}_0^t||_2^2$, where N is a batch size, d is a dimension and m is a sequence length). We plot this magnitude with respect to timestep for generations with different number of steps as well as for the predictions $\bar{z}_0^t$ from the training stage. The results are presented in Figure 3. They indicate that self-conditioning significantly increases the prediction magnitude as the number of steps increases. This can be explained by the following: during training, the model learns to use self-conditioning to approximate z_0 more accurately. Consequently,

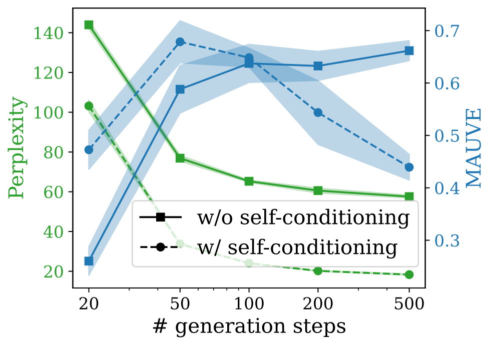

Figure 2: Comparison of models with and without self-conditioning

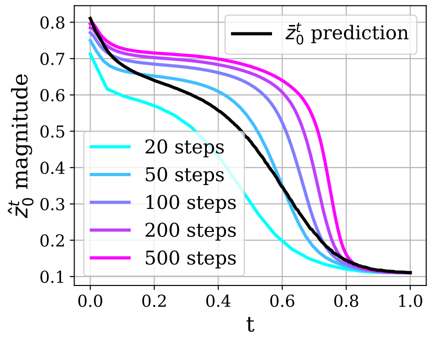

Figure 3: Comparison of magnitudes for generation processes with different amount of steps.

self-conditioning increases the model's confidence, which is directly related to prediction magnitude. During the generation process, the model takes its own prediction, which has an increased magnitude, as an input at each step and increases it further. Therefore, the increase in magnitude depends directly on the number of generation steps. Eventually, this leads to a mismatch between the predictions fed into the model during training and generation. In the Appendix C, we provide a more detailed discussion of this phenomenon. It is worth noting that the smallest mismatch is observed for the trajectory of 50 generation steps, which corresponds to the best quality.

7.6 Effect of Noise scheduler

We compare our noise scheduler *tan-d* with previously used *cosine* and *sqrt* (visualized in Appendix D) and present the quantitative results in Table 3. We use the same decoder and optimal amount of generation steps for each scheduler. In Figure 4, we evaluate the difficulty of recovering a data sample from noised latent z_t for diffusion model trained with different noise schedulers. We measure the reconstruction loss $\frac{1}{N \cdot d \cdot m}||z_0 - \bar{z}_0^t||_2^2$ and accuracy of token prediction for every timestep.

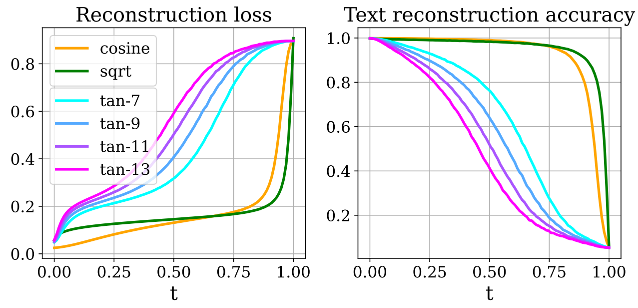

Figure 4: Comparison of noise schedulers.

Noise Scheduler	ppl ↓	mem ↓	div ↑	mauve ↑
cosine	$393.2_{127.6}$	$.262_{.004}$	$.474_{.006}$	$.098_{.011}$
sqrt	$127.2_{29.3}$	$.264_{.004}$	$.434_{.004}$	$.364_{.041}$
tan-7	$34.4_{.77}$	$.395_{.004}$	$.320_{.002}$	$.688_{.023}$
tan-9	$34.1_{.66}$	$.412_{.005}$	$.304_{.006}$	$\mathbf{.707}_{.024}$
tan-11	$31.9_{.31}$	$.428_{.004}$	$.288_{.003}$	$.694_{.026}$
tan-13	$35.5_{.62}$	$.406_{.003}$	$.298_{.002}$	$.676_{.031}$

Table 3: Comparison of noise schedulers.

While the *sqrt* noise scheduler adds significantly more amount of noise in the initial timesteps than *cosine* one, the rate of noise addition decreases for the subsequent timesteps. As a result, the denoising task becomes insufficiently hard for the timesteps $t \in [0, 0.5]$, which should lead to a decrease in their contribution to the generation process. This can be seen from the reconstruction accuracy. In contrast, *tan-d* noise scheduler adds more noise consistently across all timesteps, leading to a more challenging training task and improved generation performance.

Based on these observations, we conclude that in order to improve the efficiently of the denoising process, it is essential to increase the amount of added noise within all timesteps. However, it is important to strike a balance as adding excessive noise can negatively impact performance. In our experiments, *tan-9* produces the best result in terms of **mauve** keeping the **mem** and **div** reasonable.

As a rule of thumb, the noise schedule should be such that the diffusion model recovers approximately the same amount of information at each timestep. Otherwise, some of the them will not contribute to the denoising process enough.

8 Seq2Seq Experiments

We conduct experiments to validate the effectiveness of the proposed method on two different tasks, against ten AR ($\star$), non-diffusion NAR ($\circ$) and diffusion NAR ($\dagger$) baselines.

Metrics For evaluation of paraphrasing task, we adopt the setting of SeqDiffuSeq (Yuan et al., 2022)

Method	Sampling	R-L ↑	BS ↑	B-4 ↑
DiffuSeq†	Random	52.7	82.4	—
SeqDiffuSeq†		—	82.9	23.3
TEncDM† (BERT)		**56.4**	**83.0**	**30.4**
TEncDM† (T5)		**52.4**	**81.6**	**26.4**
DiffuSeq†	MBR-10	**58.8**	83.7	24.1
SeqDiffuSeq†		—	**84.0**	24.3
TEncDM (BERT)†		58.1	**84.0**	**31.8**
TEncDM (T5)†		53.5	82.3	27.4
GPT2-small FT⋆	Nucleus	52.1	82.5	19.8
Transformer-base⋆		57.5	83.8	27.2

Table 4: Seq2Seq evaluation results of AR and Diffusion methods on QQP. We calculate **ROUGE-L (R-L)**, **BERTScore (BS)** and **BLEU-4 (B-4)**.

Method	Sampling	ROUGE-1/2/L ↑	BS ↑
NAT◇	—	24.0 / 3.9 / 20.3	—
iNAT◇		24.0 / 4.0 / 20.3	
CMLM◇		23.8 / 3.6 / 20.2	
LevT◇		24.8 / 4.2 / 20.8	
DiffuSeq†	Random	18.9 / 1.3 / 13.6	46.8
TEncDM (BERT)†	Random	**32.2 / 10.8 / 25.7**	**69.5**
TEncDM (T5)†	Random	**32.4 / 10.9 / 25.7**	68.8
DiffuSeq†	MBR-5	19.3 / 1.7 / 14.1	46.9
TEncDM (BERT)†	MBR-5	**32.8 / 11.2 / 26.2**	**69.8**
TEncDM (T5)†	MBR-5	**32.9 / 11.4 / 26.5**	**69.2**
GENIE†	MBR-50	29.3 / 8.3 / 24.7	—
AR-Diffusion†	MBR-50	31.7 / 10.1 / 24.7	—
Transformer-base⋆	Nucleus	30.5 / 10.4 / 24.2	—

Table 5: Seq2Seq evaluation results of NAR, AR and Diffusion methods on XSum. **BS** is a **BERTScore**.

and calculate ROUGE-L (Lin, 2004), BERTScore (Zhang et al., 2019) and BLEU-4. In addition, we follow the approach of Wu et al. (2023) and report ROUGE-1/2 for summarization task.

Baselines We include three groups of baselines. The first group comprises of classical AR baselines: Transformer (Vaswani et al., 2017b) and finetuned GPT-2 (Radford et al., 2019). We also compare against NAR methods: NAT (Gu et al., 2017), iNAT (Lee et al., 2018), CMLM (Ghazvininejad et al., 2019), LevT (Gu et al., 2019). Besides, we compare the approach to other diffusion-based methods: DiffuSeq (Gong et al., 2023), SeqDiffuSeq (Yuan et al., 2022), GENIE (Lin et al., 2023), AR-diffusion (Wu et al., 2023).

Results We report our comparison on QQP and XSum in Table 4 and Table 5, respectively. We took the results of NAR and AR approaches from the corresponding papers (Qi et al., 2021; Wu et al., 2023; Yuan et al., 2022).

We use BERT as diffusion encoder and experiment with two conditional encoders: BERT and T5. We observe that both encoders are effective for XSum and QQP datasets, but using BERT leads to a better quality on QQP across all metrics and on XSum these encoders performs similarly.

The comparison with other methods clearly demonstrate that **TEncDM** outperforms the existing non-diffusion NAR approaches across all metrics. Furthermore, **TEncDM** surpasses diffusion and AR approaches by a large margin on summarization task. It also achieves consistent improvements over diffusion models on QQP with random candidate sampling.

Recent works (Li et al., 2022; Wu et al., 2023) utilize Minimum Bayes Risk (MBR) (Kumar and Byrne, 2004) decoding to select the best sample. For fair comparison, we also employ MBR decoding with the same number of candidates. As we can see from Table 5, TEncDM significantly outperforms diffusion baselines with even less number of candidates on XSum. At the same time, Table 4 shows that the results on QQP are comparable with other models.

9 Limitations

There are two limitations that warrant further investigation. First, while the quality of the model can be improved by training diffusion encoder, decoder and denoising model simultaneously, we avoid doing so in order to avoid overcomplicating the approach. Second, samples from the latent space have a high dimensionality that depends on the sequence length, making the training of our method significantly slower as the length increases. This problem can probably be eliminated by training the autoencoder, which is a great direction for the further research.

10 Conclusion

In this work, we explore key details of the diffusion pipeline for text generation. We propose **TEncDM** which trains the diffusion model inside the latent space of language encoder model. In order to improve text generation performance, we analyse the effect of self-conditioning and conclude that it increases the magnitudes of model's predictions and results in reducing of generation steps. We also propose an efficient decoder that boosts the diffusion model performance. The extensive ablation on ROCStories proves the impact of proposed design choices. **TEncDM** outperforms recent diffusion models, non-autoregressive and classical autoregressive methods thorough experiments on downstream tasks.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993. Curran Associates, Inc.

James Betker, Gabriel Goh, Li Jing, † TimBrooks, Jianfeng Wang, Linjie Li, † LongOuyang, † JuntangZhuang, † JoyceLee, † YufeiGuo, † WesamManassra, † PrafullaDhariwal, † CaseyChu, † YunxinJiao, and Aditya Ramesh. Improving image generation with better captions.

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, Varun Jampani, and Robin Rombach. 2023. Stable video diffusion: Scaling latent video diffusion models to large datasets.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. 2023. Analog bits: Generating discrete data using diffusion models with self-conditioning.

Zihang Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2017. Quora question pairs.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zach Evans, CJ Carr, Josiah Taylor, Scott H. Hawley, and Jordi Pons. 2024. Fast timing-conditioned latent audio diffusion.

Zhujin Gao, Junliang Guo, Xu Tan, Yongxin Zhu, Fang Zhang, Jiang Bian, and Linli Xu. 2023. Difformer: Empowering diffusion models on the embedding space for text generation.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. 2023. Diffuseq: Sequence to sequence text generation with diffusion models. In *The Eleventh International Conference on Learning Representations*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. *Advances in Neural Information Processing Systems*, 32.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. 2022. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. 2023. Simple diffusion: end-to-end diffusion for high resolution images. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. OpenReview.net.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems*, volume 34, pages 12454–12465. Curran Associates, Inc.

Shankar Kumar and Bill Byrne. 2004. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. 2022. Diffusion-lm improves controllable text generation. *ArXiv*, abs/2205.14217.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Zhenghao Lin, Yeyun Gong, Yelong Shen, Tong Wu, Zhihao Fan, Chen Lin, Nan Duan, and Weizhu Chen. 2023. Text generation with diffusion language models: a pre-training approach with continuous paragraph denoise. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Weinberger. 2022. Latent diffusion for language generation. *arXiv preprint arXiv:2212.09462*.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. 2023. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14297–14306.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems*, volume 34, pages 4816–4828. Curran Associates, Inc.

Weizhen Qi, Yeyun Gong, Jian Jiao, Yu Yan, Weizhu Chen, Dayiheng Liu, Kewen Tang, Houqiang Li, Jiusheng Chen, Ruofei Zhang, et al. 2021. Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. In *International Conference on Machine Learning*, pages 8630–8639. PMLR.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, and Rémi Leblond. 2022. Self-conditioned embedding diffusion for text generation.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. In *Advances in Neural Information Processing Systems*.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017b. Attention is all you need. *Advances in neural information processing systems*, 30.

Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, yelong shen, Jian Jiao, Juntao Li, zhongyu wei, Jian Guo, Nan Duan, and Weizhu Chen. 2023. AR-diffusion: Auto-regressive diffusion model for text generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Mingxuan Wang. 2023. Dinoiser: Diffused conditional sequence learning by manipulating noises. *arXiv preprint arXiv:2302.10025*.

Hongyi Yuan, Zheng Yuan, Chuanqi Tan, Fei Huang, and Songfang Huang. 2022. Seqdiffuseq: Text diffusion with encoder-decoder transformers. *arXiv preprint arXiv:2212.10325*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Yizhe Zhang, Jiatao Gu, Zhuofeng Wu, Shuangfei Zhai, Joshua M. Susskind, and Navdeep Jaitly. 2023. PLANNER: Generating diversified paragraph via latent language diffusion model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

A Decoder for embedding-based model

We show that our proposed decoder is robust not only for encoding-based diffusion model, but also for embedding-based one. In Table 6, we compare our decoder described in Section 7.2 with the commonly used rounding to the closest embedding. It is easy to see that our decoder improve the text quality according to MAUVE. Also, it hugely improves Memorization and Diversity. Low value of Perplexity for the rounding method comes from the low diversity and it does not imply the high quality of the generated samples.

Decoder	ppl ↓	mem ↓	div ↑	mauve ↑
Rounding	$32.4_{.41}$	$.437_{.007}$	$.252_{.005}$	$.421_{.043}$
Transformer + $Cor(z_0)$	$48.9_{.36}$	$.371_{.003}$	$.324_{.002}$	$.600_{.016}$

Table 6: Decoders for the BERT embedding-based model.

B Corruption for decoder training

Decoder is trained to map the latents $\hat{z}_0$ generated by the diffusion into text. These latents might be inaccurate and the decoder must take this into account in order to produce the best possible text. Therefore, we make the training task harder for the decoder by corrupting the input latents z_0 in order to mimic an imprecision of $\hat{z}_0$.

In this section, we experiment with two corruption techniques:

1. Replacing z_0 with z_t by the diffusion forward process, $Cor(z_0) = \sqrt{\alpha_t} z_0 + \sqrt{(1-\alpha_t)}\varepsilon = z_t$.

2. Adding a random Gaussian noise to decoder input, $Cor(z_0) = z_0 + \sigma\varepsilon$, where $\varepsilon \in \mathcal{N}(0, 1)$.

The both techniques introduce the random noise into the decoder input. However, the first one attempt to mimic the samples from the diffusion model denoising trajectory. We implement it by randomly sampling the timestep from the range $t \in [0, t_max]$ and calculating the corresponding z_t. In Figure 6, we show the text generation quality in terms of Perplexity and MAUVE Score with respect to t_max. In Figure 5, we present the similar result for the second decoder training technique with varying noise strength σ. To make the comparison fair we apply all decoders to the same latents produced be the diffusion model. Both plots suggest that there is an optimal amount of noise that should be added. However, the first technique results in a better performance.

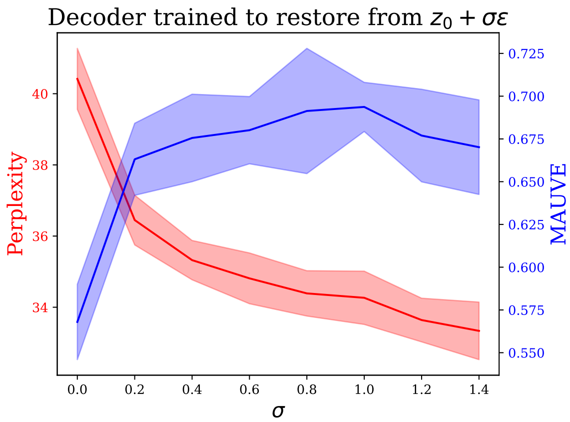

Figure 5: The dependence between the generation quality and the maximum amount of noise added to the latents during the decoder training.

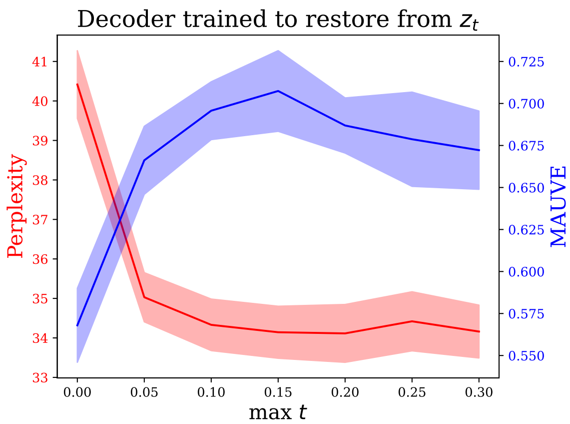

Figure 6: The dependence between the generation quality and the maximum amount of noise in z_t during the decoder training.

C Self-conditioning increases prediction magnitude

We show that self-conditioning tend to increase the magnitude of values of model's output by conducting the following experiment. We sample z_t

using the diffusion forward process and predict $\tilde{z}_0^t = \hat{z}_\theta(z_t, t, \tilde{z}_0^t)$ from it several times. Each time we feed the model its previous prediction and do not change z_t and timestep t. In Figure 7, we plot the trajectories of prediction magnitude obtained by this repeated prediction scheme for different timesteps t. The results show that the prediction magnitude grows at each step, even though we change only the sample, which we provide to a model using the self-conditioning. This allows us to conclude that self-conditioning is indeed responsible for the increase in prediction magnitude, which is reflected in the inference behaviour of the model.

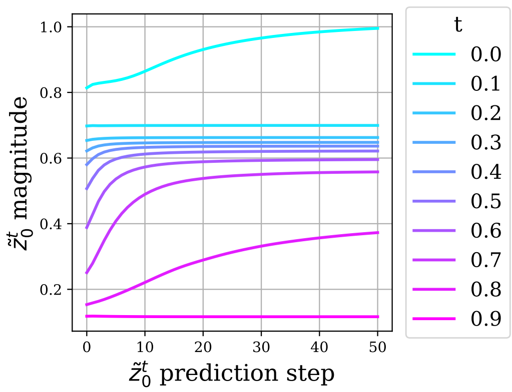

Figure 7: The effect of repeatedly predicting $\tilde{z}_0^t$ without deviating from the noisy latent z_t on the magnitude of that prediction.

D Noise Schedulers

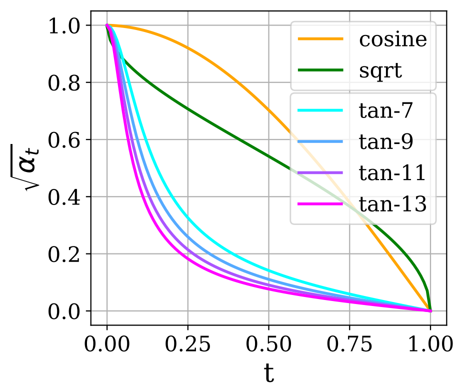

Figure 8: Visualizing different noise schedulers $\sqrt{\alpha_t}$.

E Implementation details

We train our models using 4 A100 GPUs. The training takes approximately 10 hours for **ROCStories**, 10 hours for **QQP** and 30 hours for **XSum**.

	ROCStories	**XSum**	**QQP**
Diffusion Trainable Params		101M	
Decoder Trainable Params		44M	
Transformer Layers		12	
Transformer Dim		768	
Self-Attention Heads		12	
Optimizer		AdamW	
Learning Rate	2e-4	4e-4	4e-4
(β_1, β_2)		(0.9, 0.980)	
Batch Size		512	
Warmup Steps		500	
Learning Rate Sch		Constant	
Weight Decay		0.01	
Gradient Clipping		1	
EMA Decay		0.9999	
Training Steps	100k	50k	100k
Max Seq Length	80	64	64
Max Context Length	–	256	32

Table 7: Training details for TEncDM across different datasets.

F Dataset Statistics

ROCStories The dataset consists of 98,161 instances. 93,161 instances are held out for training, 1,000 instances for validation, 4,000 instances for testing.

XSum The dataset is used for summarization task and it contains 204k BBC articles, which are provided as document and summary pairs and covered wide range of topics (Sports, Politics, etc.). It has 204,045 training instances, 11,332 validation instances, and 11,334 test instances.

QQP The subset of QQP dataset, proposed in (Gong et al., 2023), consists of 144k question pairs from the Quora platform that are paraphrases of each other. It has 144,715 training instances, 2,048 validation instances, and 2,500 test instances.

G Comparison with GPT2

We compare our diffusion model with fine-tuned GPT2-small (Radford et al., 2019) on an unconditional generation task using **ROCStories** (Mostafazadeh et al., 2016) dataset. We use the Nucleus sampling with $p = 0.9$ for the GPT generation, as is produced the best results. Both models have similar amount of parameters (124M for GPT2 and 145M for TEncDM). The result of the comparison is presented in Table 8 and it shows that GPT2 has a higher MAUVE, but it also tends to memorise the training data set more and has a lower diversity. The perplexity comparison is unfair as it is computed with the GPT2-large model, which behaves similarly to GPT2-small. Given that the GPT2 is pre-trained and TEncDM was trained from scratch, we can conclude that both models perform at about the same level.

Decoder	ppl ↓	mem ↓	div ↑	mauve ↑
GPT2-small FT	$15.5_{.11}$	$.519_{.004}$	$.269_{.003}$	$.739_{.031}$
TEncDM	$34.1_{.66}$	$.412_{.005}$	$.304_{.006}$	$.707_{.024}$

Table 8: Comparison on unconditional generation (ROC-Stories).

H Generation examples

BERT enc with MLP decoder	##ocks A man wanted to go swimming. They packed up the boat, and drove to the beach. They found a nice spot by the water. They swam for hours, remorying the scenery. At the end of the trip, they had to go home. chantingctic Widow leopard paranoidntialivatingolar chanting Xiaocticntlymurananteurboopectatorctictleshalanadonantnantnadonantavesmura ##mps Heather wanted to bake a cake. She grabbed some ingredients and put the cake in the oven. Her alarm rang, but didn't go off. To her dismay the cake was on fire! The cake was so mess that she forgot to turn off the oven. putmpsmuravatednantpectatoraves Wan emitted chantingmura leopardmura leopardave sputvatednantavesnant Widownantnantavesshing ##eur Rita was always bullied in school. But every time time she stood up, she was bullied. Rita was too young. But as the bully grew, she improved. After school, Rita was no longer bullied. fraction Signlatingbeknantbekaveslaxivatingmpsivatingmpspectatoromoomonadoavespectatoravessh ingavesmpsolarivatingutavatedivating Widowoveavesriotmpsmps
BERT emb with Transformer decoder	Last week my brother brought my skateboard with me. He started using the skateboard after half an hour long. I bit my leg and started to fall out of my foot. My brother got into the piece. He was able to scolded me and take me to the hospital. Liz was in the kitchen watching watching TV. She heard a sharp s Henk. She picked it up and ran downstairs to grab what her sandwich was. She quickly grabbed a hot cheese from her sandwich. She put the sandwich on the stove and turned it down the plate. Larry and his girlfriend were making family dinner last night. After a long time, they decided to make lasagna. They made the meat mix and tested the bread. They had to cut the meat off the pizza. It lit up as soon as it was done.
BERT enc with Transformer decoder	Emily wanted her nails become pink. She took some nailolish from a grocery store and thought it looked horrible. She tried everything to get rid of it. It ended up making a ton of mess. Emily had to throw the mess all out. Bianca was at a local tennis party. She was having a good time with her friends. Suddenly she realized that she had lost her wallet! She searched for an hour to no avail. Luckily she found it there and was glad that she didn't lose it. Ally wakes up one morning feeling very well. Ally realizes she has a pregnancy test. Ally decides she will go to the doctor to get her test. Ally is shocked when the results show that she is pregnant. Ally is very excited when her pregnancy test is confirmed.
T5 enc with Transformer decoder	Jack had a dog that he loved named Frankt. He was a big Shepherd who had lotss barkles and collar. One day, Jack left Fredt at his house and didn t find him. After three days, Ft's owner found out. bought a searches. The next day, his owner found Frankt in the house. The kids climbed outside with the gun. They wanted to shooting their neighbor' to gun. They fell on a higher of a mountain. mom tried to carry the rifle for them. It was too heavy to carry from the kids. Shera and her weddingrs were packing a box of pictures. Shera du searched through each box for the favorite picture. Finally it was time stamped the numbers. Shera put the pictures in the box in front of the machine..a is, it took of lot time to up the number right.

Table 9: Examples of generated texts for different models.