# Classifier-free Diffusion Models for Machine Translation

Bachelor's thesis

Yunus Demirag

Department for Computer Science
Karlsruher Institut für Technologie
Germany
01.08.2023.

# Contents


**1 Introduction** **3**
- 1.1 Motivation . . . 4
- 1.2 Thesis Statement . . . 4

**2 Background** **5**
- 2.1 Variational diffusion models . . . 6
  - 2.1.1 Variational autoencoders . . . 6
  - 2.1.2 Markovian hierarchical variational autoencoders . . . 8
  - 2.1.3 Variational diffusion models with fixed noise schedule . . . 10
  - 2.1.4 Two alternative ways to parameterize the estimated mean . . . 15
  - 2.1.5 Noise schedules . . . 16
  - 2.1.6 Classifier-free guidance . . . 16
- 2.2 Diffusion language models . . . 17
  - 2.2.1 Diffusion-LM . . . 18
  - 2.2.2 DiffuSeq . . . 19
- 2.3 Neural Machine Translation . . . 20

**3 Classifier-free Diffusion Models for Machine Translation** **21**
- 3.1 Diffusion-LM-based Model . . . 21
- 3.2 DiffuSeq-based Models . . . 21
  - 3.2.1 DiffuSeq with Autoregressive Sampling . . . 22
  - 3.2.2 DiffuSeq trained for word generation . . . 22
  - 3.2.3 DiffuSeq with Knowledge Distillation . . . 23
  - 3.2.4 Alternate Diffusion Kernel . . . 24
  - 3.2.5 Position-dependent noise scheduling . . . 25

**4 Model Evaluation** **26**
- 4.1 Experimental Setup . . . 26
  - 4.1.1 Dataset and data preparation . . . 26
  - 4.1.2 The evaluated implementations . . . 27
  - 4.1.3 Baselines . . . 28
- 4.2 Results . . . 28
  - 4.2.1 Analysis of the results over differing lengths of the source sequence . . . 29
  - 4.2.2 Models which converged to trivial distributions . . . 31
- 4.3 Training and decoding times of the models . . . 33
- 4.4 Conclusion . . . 34

**5 References** **35**

## Abstract

In the domain of natural language processing, specifically, the task of neural machine translation, non-autoregressive models (NAR) have struggled to achieve results comparable to state-of-the-art autoregressive (AR) models (Xiao et al., 2023). First attempts at applying the idea of Denoising Diffusion Probabilistic Models Ho et al., 2020; Sohl-Dickstein et al., 2015, a type of model from the family of diffusion models to language modeling tasks, have shown promising results Gong et al., 2023; Li et al., 2022. Diffusion models have previously been successfully applied to various continuous domain problems such as image generation (Sohl-Dickstein et al., 2015; Song & Ermon, 2019). For conditioning the generation process diffusion models may rely on classifier guidance (Li et al., 2022; Luo, 2022; Song & Ermon, 2019; Song et al., 2020) by external models, or alternatively methods of classifier-free guidance (Ho & Salimans, 2022). This thesis explores the potential of diffusion models with classifier-free guidance for the task of neural machine translation. In the experiments performed the evaluated models did not match the performance of the state-of-the-art transformer model (Vaswani et al., 2017). Further evaluations indicate that the evaluated diffusion models performed relatively worse than the transformer model on longer sequences.

# Introduction

For readers who are new to the field of deep generative modeling, I want to start by shallowly introducing a few concepts that will be needed in this thesis. In deep generative modeling, one generally aims to model an intractable distribution $\mathbf{D}$, by assuming a tractable underlying, unobserved distribution $\mathbf{Z}$. The **generative model** is then defined as a neural network $g_\theta$ with parameters $\theta \in \Theta$, which given $Z \sim \mathbf{Z}$, describes the distribution $g_\theta(Z) \sim \mathbf{D'}$ (Ruthotto & Haber, 2021). We will refer to the random variable $Z$ as a latent variable, meaning that it is not observed in samples drawn from $\mathbf{D}$. Training may happen by a maximum likelihood method, with the goal of maximizing the likelihood of observed samples under $\mathbf{D'}$. Deep generative modeling is often applied to image and video synthesis. However, more recently there has been an increasing number of studies regarding the applicability in other fields. Common approaches to deep generative modeling also include variational autoencoders (VAE) (Kingma & Welling, 2019, 2022), generative adversarial networks (GAN) (Goodfellow et al., 2014) and normalizing flows (NF) (Papamakarios et al., 2021; Ruthotto & Haber, 2021). Further approaches such as hierarchical variational autoencoders may assume a hierarchy of latent variable spaces and their respective generator functions (Wu et al., 2021). In a way, generative modeling can be thought of as implicitly modeling probability distributions.

**Language modeling** is the task of assigning probabilities to sequences of words $\mathbf{P}(w_{1:n})$ and is one common task in natural language processing (NLP). **Autoregressive language models** approach this task by evaluating the next tokens probability conditioned on the left context i.e. the words previously generated. Such models include the state-of-the-art transformer model (Vaswani et al., 2017), recurrent neural networks (RNN) (Vennerød et al., 2021), but also statistical language models such as n-Grams (Yoav, 2017). By design, autoregressive models can be used to predict likely candidates for the continuation of a given sequence of words, by applying a search algorithm such as beam search. In contrast, **Non-autoregressive language models** attempt to generate whole sequences in parallel, through methods such as iterative refinement (Lee et al., 2018), iterative insertion and deletion of tokens (Gu et al., 2019) or latent-variable-models (Gu et al., 2017a) like the diffusion language models we are focusing on (Li et al., 2022).

Diffusion models are a family of deep generative models and may refer to two common approaches to generative modeling, which in many aspects can be viewed as equivalent. Variational diffusion models introduced by Sohl-Dickstein et al. (2015) can be viewed as a certain class of hierarchical variational autoencoders, while score-based diffusion models arise from score-matching combined with the idea of sampling from intractable distributions by reversing diffusion processes (Song & Ermon, 2019). In the field of machine learning, diffusion models are generally understood to be models which gradually remove noise from noisy data, ultimately enabling one to sample from intractable distributions, by starting with pure noise (Ho et al., 2020; Luo, 2022; Song & Ermon, 2019). The theory underlying these models will be explored in the first section of the thesis. While variational diffusion models were initially proposed for both continuous and discrete data problems (Sohl-Dickstein et al., 2015), they have most prominently been applied to continuous domain problems, with image and video generation standing out as good examples. However, by viewing language modeling as a continuous domain problem, Diffusion-LM (Li et al., 2022), a pioneer diffusion language model, has recently shown some success at Plug-And-Play controllable text generation (Dathathri et al., 2019; K. Yang & Klein, 2021), a task many autoregressive models struggle with (Li et al., 2022), introducing

diffusion models to Natural Language Processing (NLP).

Diffusion-LM achieves this with the help of classifier models. Such models produce a probability vector over a set of labels (classes) given an input sequence. Given a classifier $C$ with $K$ classes, where for any intermediate latent variable $Z_t$ of the denoising process, $C(Z_t)$ is a probability vector, to produce a sample of a certain class $c \in \{1, \ldots, K\}$, Diffusion-LM modifies its intermediate latent variable as $\tilde{Z}_t = Z_t + \nabla_{Z_t} \log C(Z_t)_c$ at every step in the denoising process. In the relevant paper, this method is justified by Bayes rule as for a probability density $p : \nabla_x \log p(x|c) = \nabla_x \log p(x) + \nabla_x \log p(c|x)$ holds. In Diffusion-LM adding the gradient is replaced by multiple steps of the Adagrad (Duchi et al., 2011) optimization algorithm (Li et al., 2022).

In opposition to that, classifier-free guidance is an approach, where the model itself should learn to model conditional data distributions. Similar to Diffusion-LM, DiffuSeq (Gong et al., 2023) is a non-autoregressive sequence-to-sequence diffusion model, implementing classifier-free guidance, relying on the infilling capability proven by Diffusion-LM (Li et al., 2022). For that, DiffuSeq introduces changes to the training and sampling processes, which improve sample quality on sequence-to-sequence tasks, when compared with Diffusion-LMs infilling. Gong et al. (2023) further provide data on the diversity of text generated by DiffuSeq, showing that the generated results are more diverse than those of transformer or baseline non-autoregressive models. Notably, diversity is an important metric for tasks such as paraphrasing or open-domain dialogue, with models with higher generation diversity giving more possible paraphrased alternatives and seeming more alive in open-domain dialogue.

## 1.1 Motivation

Non-autoregressive models have not been able to match the autoregressive transformer models' performance on the task of neural machine translation so far (Ren et al., 2020; Xiao et al., 2023). Diffusion models, having shown their modeling strengths for other domains, might serve to narrow the gap. This thesis will explore the potential of diffusion language models for machine translation, with this task being a prominent example of sequence-to-sequence tasks, and will try to give the reader an understanding of the fundamentals of diffusion models. Due to the sequential nature of text, it seems reasonable to assume that modeling text autoregressively provides strong results and that longer sequences might prove more problematic for diffusion language models as presented by Diffusion-LM and DiffuSeq.

When implementing machine translation by diffusion models through an approach using external guidance as in classifier-guidance, the model's performance on a task will be bounded by the module rating how well this task is performed. So in a domain like machine translation, where one expects accurate translations, and where objectives such as answer diversity or methods of controlling the text generation are secondary, it seems more interesting to me to study the performance of classifier-free diffusion translation models.

## 1.2 Thesis Statement

Diffusion language models currently struggle with training and sampling speed (Li et al., 2022), so one aim of this thesis is to provide some insight into why these models train so slowly. To understand the impact that parallel generation of the whole sequence has on the quality of the results, a further goal is to evaluate whether non-autoregressive models perform relatively worse on longer sequences, compared to autoregressive models. To improve the performance of diffusion language models on long source sequences, this thesis proposes and evaluates a method of autoregressive sampling for diffusion language models and a diffusion language model utilizing a new position-dependent noise schedule. To that end, this thesis should provide insights on how these compare to baseline models for different lengths of the source sequence. Lastly, this thesis should evaluate how training diffusion models with knowledge distillation impacts their performance.

# Background

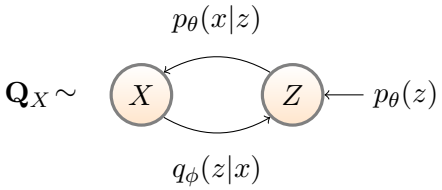
Before looking at the more specific implementations of diffusion models (i.e. Diffusion-LM & DiffuSeq), this chapter will serve as a short introduction to one common type of diffusion model. The great success diffusion models had in the domain of image and video generation (L. Yang et al., 2023) prompted the family to branch out into a vast ecosystem of different models, sampling methods, and approaches (L. Yang et al., 2023), so this introduction will only scratch the surface of the field of diffusion models. In Section 2.1 we will look at the class of variational diffusion models which the most prominent example of diffusion models, Denoising Diffusion Probabilistic Model (DDPM) by Ho et al. (2020), falls under. Section 2.2 will elaborate on how this concept is adapted by diffusion language models, and in Section 2.3 I want to provide a short introduction to neural machine translation, the task at hand.

### Notation

| | |
|---|---|
| $\lambda_d$ | The Lebesgue measure on $(\mathbb{R}^d, \mathcal{B}_d)$ |
| $\mathbf{P}, \mathbf{Q}$ | Probability measures |
| $p, q$ | Probability densities, where $\mathbf{P} = p\lambda_d, \mathbf{Q} = q\lambda_d$ for some $d \in \mathbb{N}$ |
| $X, x$ | Random variable and a realization thereof |
| $\mathbf{P}[\cdot\|\mathcal{A}]$ | Conditional probability with regards to a $\sigma$-algebra $\mathcal{A}$ |
| $\sigma(X)$ | Sigma algebra generated by $X$ |
| $\mathbf{P}[\cdot\|X] := \mathbf{P}[\cdot\|\sigma(\mathbf{X})]$ | Conditional probability with regards to random variable $X$ |
| $\mathbf{P}[\cdot\|X = x]$ | Conditional probability with regards to a realization $x$ |
| $\mathbf{P}^X$ | The image measure $\mathbf{P} \circ X^{-1}$ |
| $p^X$ | The marginal density of $X$ |
| $p^{\|Z}(\cdot\|z)$ | The conditional density where $Z = z$ |
| $p^{X\|z}$ | Shorthand for $p^{X\|Z}(\cdot\|z)$, used where interpretation is unambiguous |
| $p(x\|z)$ | Shorthand for $p^{X\|Z}(x\|z)$, used where interpretation is unambiguous |

## 2.1 Variational diffusion models

Figure 2.1: Latent variables and encoder-decoder transitions in a variational autoencoder




To properly define variational diffusion models, I quickly want to go over the definitions of **variational autoencoders (VAE)** and **hierarchical variational autoencoders (HVAE)**. A variational autoencoder is a deep generative model, that tries to model the distribution $\mathbf{Q}_X$ of an observed variable $X$ by assuming that there exists an unobserved variable $Z$ (**latent variable**) in some kind of dependency with $X$. It then models three densities, a **prior density** $p_\theta(z)$, the **decoder** $p_\theta(x|z)$, and the **encoder** $q_\phi(z|x)$ (Figure 2.1), with $\phi \in \Phi$ being the parameters of the encoder and $\theta \in \Theta$ the parameters of the decoder. The parameters $\theta, \phi$ are then jointly optimized, such that the encoder encodes observations with as little loss of information as possible, the prior density approximately samples from the latent encoding space, and lastly such that the decoder decodes the encodings given by the encoder with as little loss of information as possible.

### 2.1.1 Variational autoencoders

The following presentation of variational autoencoders is based on the work by Kingma and Welling (2019).

First let $(\Omega_X, \mathcal{A}_X, \mu_X)$ and $(\Omega_Z, \mathcal{A}_Z, \mu_Z)$ be two measure spaces, and $\mathbf{Q}_X \ll \mu_X$ with $\mathbf{Q}_X = q_X \mu_X$ be a probability measure. To ensure that all conditional densities exist, we assume that both $\Omega_X$ and $\Omega_Z$ are Polish spaces and that the $\sigma$-algebras are the Borel-$\sigma$-algebras generated by their respective open sets. Let the random variable $X$ be the identity $\mathrm{id} : \Omega_X \to \Omega_X; x \mapsto x$ on the measure space $(\Omega_X, \mathcal{A}_X, \mathbf{Q}_X)$. Now we choose a Markov kernel $\mathcal{Q}_{\phi;Z|X} : \Omega_X \times \mathcal{A}_Z \to [0,1]$, which for every $x \in \Omega_X$ has a parametric density $\mathcal{Q}_{\phi;Z|X}(x,\cdot) = q_\phi(\cdot|x)\mu_Z$. In this $q_\phi(\cdot|x)$ should not be read as a conditional density yet, but more as a parametric mapping from $\Omega_X$ to the space of density functions on the measure space $(\Omega_Z, \mathcal{A}_Z, \mu_Z)$.

Then by $\mathbf{Q}_\phi = \mathbf{Q}_X \otimes \mathcal{Q}_{\phi;Z|X}$ we define a (parametric) probability measure on the measurable space $(\Omega_X \times \Omega_Z, \mathcal{A}_X \otimes \mathcal{A}_Z)$ with:

$$\begin{aligned}\mathbf{Q}_\phi(A_X \times A_Z) &= \int_{\Omega_X \times \Omega_Z} \mathbb{1}_{A_X \times A_Z} \,\mathrm{d}\mathbf{Q}_\phi \underset{\text{by definition}}{=} \int_{A_X} \mathcal{Q}_{\phi;Z|X}(x, A_Z)\mathbf{Q}_X(\mathrm{d}x) \\ &= \int_{A_X} q_X(x) \int_{A_Z} q_\phi(z|x)\mu_Z(\mathrm{d}z)\mu_X(\mathrm{d}x) = \int_{A_X \times A_Z} q_\phi \,\mathrm{d}\mu_X \otimes \mu_Z.\end{aligned}$$

Using the additivity of both the integral and the probability measure, this can be extended to the countable unions of products from $\mathcal{A}_X$ and $\mathcal{A}_Z$, and by the good sets principle then to the product $\sigma$-algebra. Therefore $q_\phi(x,z) := q_\phi(z|x)q_X(x)$ is the density of $\mathbf{Q}_\phi$ with regards to the product measure $\mu_X \otimes \mu_Z$ by

Furthermore, the Markov kernel is the regular version of the conditional distribution by

$$\forall A_X \in \sigma(X) : \mathbb{E}_{\mathbf{Q}_\phi}\left[\mathbb{1}_{Z \in A_Z}\mathbb{1}_{A_X}\right] = \int_{A_X}\int_{\Omega_Z} \mathbb{1}_{z \in A_Z}\mathcal{Q}_{\phi:Z|X}(x,\mathrm{d}z)\mathbf{Q}_X(\mathrm{d}x) = \int_{A_X} \mathcal{Q}_{\phi;Z|X}(x, A_Z)\mathbf{Q}(\mathrm{d}x)$$

for $A_Z \in \mathcal{A}_Z$ so $\mathbf{Q}_\phi(Z \in A_Z|X) = \mathcal{Q}_{\phi;Z|X}$ and we can now speak of $q_\phi(z|x)$ as the conditional density.

By the same method, we now describe the parametric prior distribution $\mathbf{P}_{\theta;Z} \ll \mu_Z$ with $p_\theta \mu_Z = \mathbf{P}_{\theta;Z}$ and a Markov kernel $\mathcal{P}_{\theta;X|Z}$ which for every $z \in \Omega_Z$ has a density $p_\theta(\cdot|z)\mu_X$. As before, we obtain a distribution $\mathbf{P}_\theta = \mathbf{P}_{\theta;Z} \otimes \mathcal{P}_{\theta;X|Z}$ for which $p_\theta(x,z) := p_\theta(z)p_\theta(x|z)$ is the density with regards to the product measure.

With this framework, we can derive a marginal density $p_\theta(x) := \int_{\Omega_Z} p_\theta(x,z)\mu_Z(\mathrm{d}z)$ and for a given sample $x$ from $\mathbf{Q}_X$ we want to maximize the log-likelihood of $x, \log p_\theta(x)$. To this end, we can derive a lower bound of the log-likelihood under the assumption that $p_\theta(x) > 0$.

$$
\begin{aligned}
\log p_\theta(x) &= \log p_\theta(x) \underbrace{\int_{\Omega_Z} q_\phi(z|x)\mu_Z(\mathrm{d}z)}_{=1} \\
&\underset{(I)}{=} \mathbb{E}_{\mathcal{Q}_{\phi;Z|X}(x,\cdot)}\left[\log \frac{p_\theta(x,Z)}{p_\theta(Z|x)}\right] \text{ with } p_\theta(z|x) := \frac{p_\theta(z,x)}{p_\theta(x)} \text{ and by } p_\theta(x,z) = p_\theta(x)p_\theta(z|x) \\
&= \mathbb{E}_{\mathcal{Q}_{\phi;Z|X}(x,\cdot)}\left[\log \frac{p_\theta(x,Z)}{q_\phi(Z|x)} \frac{q_\phi(Z|x)}{p_\theta(Z|x)}\right] \\
&= \underbrace{\mathbb{E}_{\mathcal{Q}_{\phi;Z|X}(x,\cdot)}\left[\log \frac{p_\theta(x,Z)}{q_\phi(Z|x)}\right]}_{\text{A}} + \underbrace{\mathbb{E}_{\mathcal{Q}_{\phi;Z|X}}\left[\frac{q_\phi(Z|x)}{p_\theta(Z|x)}\right]}_{\text{B}} \\
&\underset{\text{B is non-negative}}{\geq} \mathbb{E}_{\mathcal{Q}_{\phi;Z|X}(x,\cdot)}\left[\log \frac{p_\theta(x,Z)}{q_\phi(Z|x)}\right]
\end{aligned}
$$

The equation $(I)$ also assumes that $p(z|x) > 0$ $\mathcal{Q}_{\phi;Z|X}(x,\cdot)$-a.e. which I will denote as a constraint [VAE1]. Term A is called the Evidence Lower Bound, and commonly appears in latent variable models such as this one. Term B is a Kullback-Leibler divergence and we will show its non-negativity after a brief definition.

The **Kullback-Leibler divergence** of two densities $p, q$ given a measure $\mu$ on a measurable space $(O, \mathcal{A})$, where $p\mu \ll q\mu$, is defined as $D_{\mathrm{KL}}(p||q) := \int_O p(x) \log \frac{p(x)}{q(x)} \mu(\mathrm{d}x)$.

To show the non-negativity of the Kullback-Leibler divergence, we use the inequality $\log t \leq (t-1)$ for $t > 0$. Then $-D_{\mathrm{KL}}(p||q) \leq \int_O p(x)(\frac{q(x)}{p(x)} - 1)\mu(\mathrm{d}x) = \underbrace{\int_O q(x)\mu(\mathrm{d}x) - \int_O p(x)\mu(\mathrm{d}x)}_{=0}$, reminding ourselves that $p\mu \ll q\mu$ and therefore $\frac{q(x)}{p(x)} > 0$ $p\mu$-a.e..

So as the joint density $p_\theta(x,z)$ in the Evidence Lower Bound is the product of the prior density and the decoder conditional density, by maximizing the Evidence Lower Bound over both $\theta$ and $\phi$, we maximize the model's ability to both encode and decode the sample $x$ from $\mathbf{Q}_X$ with as little loss of information as possible, while also maximizing the log-likelihood of possible encodings under the prior density (Kingma & Welling, 2019).

*Example.* As a short example, let the observed distribution $\mathbf{Q}_X$ be a mixture distribution of $N(\mu_1, \Sigma), N(\mu_2, \Sigma), \Sigma \in \mathbb{R}^{2\times 2}$ and let the Gaussian encoder kernel be $\mathcal{Q}_\phi(x,\cdot) = N(\mu_\phi(x), \sigma_\phi(x))$. Then one could choose an appropriate prior distribution $\mathbf{P}_{\theta;Z}$ as a mixture distribution of $N(\mu_{1;\theta}, \sigma_{1;\theta}), N(\mu_{2;\theta}, \sigma_{2;\theta})$ and decoder $\mathcal{P}_{\theta;X|Z}(z,\cdot) = N(\mu_\theta(z), \Sigma_\theta(z))$. For this, $\mu_\phi$, $\sigma_\phi$, $\mu_\theta$ and $\Sigma_\theta$ need to be measurable for $\phi \in \Phi$ and $\theta \in \Theta$ so that we have well defined Markov kernels. Moreover, the marginal distribution $\mathbf{P}_\theta^X$ is absolutely continuous with regards to $\mathbf{Q}_X$, and the constraint [VAE1] is fulfilled.

More generally, due to normal distributions having positive density over all of $\mathbb{R}^d$ with the respective $d$, Gaussian transition kernels fulfill most of the constraints on such models by default, making them a common choice (Kingma & Welling, 2019).

### 2.1.2 Markovian hierarchical variational autoencoders

A Markovian hierarchical variational autoencoder extends this concept to multiple layers of latent variables $Z_1, \dots, Z_T$, in such a way that each layer $0 \leq i \leq T-1$ can be interpreted as a variational autoencoder, with the lower tier latent variable $X$ for $i = 0$ or $Z_i$ for $1 \leq i \leq T-1$ as the observed variable ($X$ in the description of the VAE) and $Z_{i+1}$ as the latent variable ($Z$ in the description of the VAE above). For that we again assume Polish spaces $\Omega_X, \Omega_{Z_1}, \dots, \Omega_{Z_T}$ their respective Borel-$\sigma$-algebras $\mathcal{A}_X, \mathcal{A}_{Z_1}, \dots, \mathcal{A}_{Z_T}$, and measures $\mu_X, \mu_{Z_1}, \dots, \mu_{Z_T}$.

*Note.* As by Klenke (2020), theorem 14.8, countable products of Polish spaces are Polish spaces and the product $\sigma$-Algebra is the Borel-$\sigma$-algebra on the product space.

As in the VAE we choose transition Markov kernels $\mathcal{Q}_{\phi;Z_i|Z_{i-1}}$ for $2 \leq i \leq T$, $\mathcal{Q}_{\phi;Z_1|X}$ and we also denote the observed distribution by $\mathbf{Q}_X$. Each kernel and the observed distribution has a conditional density denoted by $q(x)$,$q_\phi(z_1|x)$,$q_\phi(z_2|z_1), \dots, q_\phi(z_T|z_{T-1})$ respectively.
Similarly, we also choose the prior distribution as a parametric distribution $\mathbf{P}_{\theta;Z_T} \ll \mu_T$, the Markov kernels of the backward process $\mathcal{P}_{\theta;Z_i|Z_{i+1}}$ for $1 \leq i \leq T-1$ and $\mathcal{P}_{\theta;X|Z_1}$ which have densities with regards to the respective measures which will be denoted by $p_\theta(z_T)$,$p_\theta(z_i|z_{i+1})$,$p_\theta(x|z_1)$. Then at every step $1 \leq t \leq T-1$ this describes a VAE with the observed space $(\Omega_{Z_t}, \mathcal{A}_{Z_t})$ and the latent space $(\Omega_{Z_{t+1}}, \mathcal{A}_{Z_{t+1}})$. The distribution of the observed space and the prior distribution can then be obtained as follows (for reference see corollary 14.24 by Klenke (2020)):

$$\mathbf{Q}_{\phi;Z_t} = (\mathbf{Q}_X \otimes \bigotimes_{i=1}^{t} \mathcal{Q}_{\phi;Z_i|Z_{i-1}})^{\pi_t} \qquad \text{the observation distribution}$$

$$\text{where } \pi_t : \Omega_X \times \times_{i=1}^{t} \Omega_{Z_i} \to \Omega_{Z_t}, (x, z_1, \dots, z_t)) \mapsto z_t$$

$$\mathbf{P}_{\theta;Z_{t+1}} = (\mathbf{P}_{\theta;Z_T} \otimes \bigotimes_{i=T-1}^{t} \mathcal{P}_{\theta;Z_i|Z_{i+1}})^{\pi'_t} \qquad \text{the prior distribution}$$

$$\text{where } \pi'_t : \times_{i=t}^{T} \Omega_{Z_i} \to \Omega_{Z_t}, (z_t, \dots, z_T)) \mapsto z_t$$

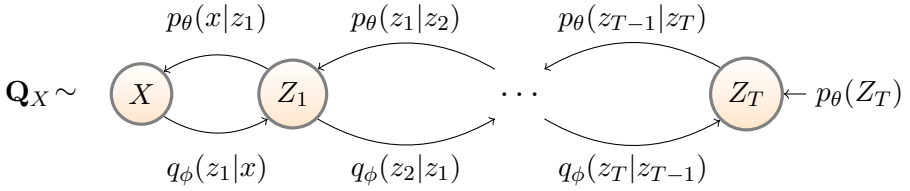
With the encoder kernel $\mathcal{Q}_{\phi;Z_{t+1}|Z_t}$ and the decoder kernel $\mathcal{P}_{\theta;Z_t|Z_{t+1}}$.

By the same method demonstrated above, for $1 \leq t \leq T$ the joint distribution $\mathbf{Q}_X \otimes \bigotimes_{i=1}^{t} \mathcal{Q}_{\phi;Z_i|Z_{i-1}}$ has the density $q(x)q_\phi(z_1|x)\prod_{i=2}^{t} q_\phi(z_i|z_{i-1})$ with regards to the product measure, and the marginal distribution has the marginal density accordingly. The same applies to the prior distribution and the decoder kernels.

Furthermore, using the marginal densities and the conditional densities from the joint density, we can derive:

$$\begin{aligned}
q_\phi(z_{i+1}|z_i, x) &= \frac{q_\phi(z_{i+1}, z_i, x)}{q_\phi(z_i, x)} \\
&\underset{\text{Fubini}}{=} \frac{\int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q_\phi(z_{i+1}, \dots, z_1, x) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots \mu_{Z_1}(\mathrm{d}z_1)}{\int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q_\phi(z_i, \dots, z_1, x) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots \mu_{Z_1}(\mathrm{d}z_1)} \\
&= \frac{\int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q(x) q_\phi(z_1|x) \prod_{j=2}^{i+1} q_\phi(z_j|z_{j-1}) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots \mu_{Z_1}(\mathrm{d}z_1)}{\int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q(x) q_\phi(z_1|x) \prod_{j=2}^{i} q_\phi(z_j|z_{j-1}) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots \mu_{Z_1}(\mathrm{d}z_1)} \\
&= \frac{q(x) q_\phi(z_{i+1}|z_i) \int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q_\phi(z_i|z_{i-1}) q_\phi(z_{i-1}|z_{i-2}) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots q_\phi(z_1|x) \mu_{Z_1}(\mathrm{d}z_1)}{q(x) \int_{\Omega_{Z_1}} \cdots \int_{\Omega_{Z_{i-1}}} q_\phi(z_i|z_{i-1}) q_\phi(z_{i-1}|z_i) \mu_{Z_{i-1}}(\mathrm{d}z_{i-1}) \dots q_\phi(z_1|x) \mu_{Z_1}(\mathrm{d}z_1)} \\
&= q_\phi(z_{i+1}|z_i)
\end{aligned}$$

Figure 2.2: Latent variables and encoder-decoder transitions in a Markovian hierarchical variational autoencoder




Given a sample $x$ from $\mathbf{Q}_X$ we can now attempt to maximize the log-likelihood of the marginal density $p_\theta(x) := \int_{\times_{i=1}^T \Omega_{Z_i}} p_\theta(x, z_{1:T}) \bigotimes_{i=1}^T \mu_{Z_i}(\mathrm{d}z_{1:T})$ by a maximizing a lower bound. This lower bound can be derived in a similar manner as for the VAE.

$$\log p_\theta(x) = \log p_\theta(x) \int_{\Omega_{Z_1}} q_\phi(z_1|x)\mu_{Z_1}(\mathrm{d}z_1) \tag{2.1a}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x, Z_1)}{p_\theta(Z_1|x)}\right] \tag{2.1b}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x, Z_1)}{q_\phi(Z_1|x)} \frac{q_\phi(Z_1|x)}{p_\theta(Z_1|x)}\right] \tag{2.1c}$$

$$= \underbrace{\mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x, Z_1)}{q_\phi(Z_1|x)}\right]}_{\mathrm{A}} + \underbrace{\mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}}\left[\frac{q_\phi(Z_1|x)}{p_\theta(Z_1|x)}\right]}_{\mathrm{B}} \tag{2.1d}$$

$$\underset{\text{B is non-negative}}{\geq} \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x, Z_1)}{q_\phi(Z_1|x)}\right] \tag{2.1e}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x|Z_1)}{q_\phi(Z_1|x)}\right] + \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log p_\theta(Z_1)\right] \tag{2.1f}$$

Moreover, for any $1 \leq i \leq T-1$ we can do the same calculation:

$$\mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i}|X}}(x,\cdot)\left[\log p_\theta(Z_i)\right] = \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i}|X}}(x,\cdot)\left[\log p_\theta(Z_i) \int_{\Omega_{Z_{i+1}}} q_\phi(z_{i+1}|Z_i)\mu_{Z_{i+1}}(\mathrm{d}z_{i+1})\right] \tag{2.2a}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i}|X}}(x,\cdot)\left[\mathbb{E}_{\mathcal{Q}_{\phi;Z_{i+1}|Z_i}}(Z_i,\cdot)\left[\log \frac{p_\theta(Z_i, Z_{i+1})}{p_\theta(Z_{i+1}|Z_i)}\right]\right] \tag{2.2b}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i+1}|X}}(x,\cdot)\left[\log \frac{p_\theta(Z_i, Z_{i+1})}{p_\theta(Z_{i+1}|Z_i)}\right] \tag{2.2c}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i+1}|X}}(x,\cdot)\left[\log \frac{p_\theta(Z_i, Z_{i+1})}{q_\phi(Z_{i+1}|Z_i)}\right] + \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i+1}|X}}(x,\cdot)\left[\log \frac{q_\phi(Z_{i+1}|Z_i)}{p_\theta(Z_{i+1}|Z_i)}\right] \tag{2.2d}$$

So by induction, we can see that:

$$\log p_\theta(x) \geq \mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}(x,\cdot)}\left[\log \frac{p_\theta(x|Z_1)}{q_\phi(Z_1|x)}\right] + \sum_{i=1}^{T-1} \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i+1}|X}(x,\cdot)}\left[\log \frac{p_\theta(Z_i|Z_{i+1})}{q_\phi(Z_{i+1}|Z_i)}\right] + \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:T}|X}(x,\cdot)}\left[\log p_\theta(Z_T)\right] \tag{2.3a}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:T}|X}(x,\cdot)}\left[\log\left(\frac{p_\theta(x|Z_1)}{q_\phi(Z_1|x)}\prod_{i=1}^{T-1}\frac{p_\theta(Z_i|Z_{i+1})}{q_\phi(Z_{i+1}|Z_i)}p_\theta(Z_T)\right)\right] \tag{2.3b}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:T}|X}(x,\cdot)}\left[\log\left(\frac{p_\theta(Z_T)p_\theta(x|Z_1)}{q_\phi(Z_1|x)}\prod_{i=1}^{T-1}\frac{p_\theta(Z_i|Z_{i+1})}{q_\phi(Z_{i+1}|Z_i,Z_0)}\right)\right] \tag{2.3c}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:T}|X}(x,\cdot)}\left[\log\left(\frac{p_\theta(Z_T)p_\theta(x|Z_1)}{q_\phi(Z_1|x)}\prod_{i=1}^{T-1}\frac{q_\phi(Z_i|x)p_\theta(Z_i|Z_{i+1})}{q_\phi(Z_{i+1}|x)q_\phi(Z_i|Z_{i+1},x)}\right)\right] \tag{2.3d}$$

$$= \mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:T}|X}(x,\cdot)}\left[\log\left(p_\theta(x|Z_1)\frac{p_\theta(Z_T)}{q_\phi(Z_T|x)}\prod_{i=1}^{T-1}\frac{p_\theta(Z_i|Z_{i+1})}{q_\phi(Z_i|Z_{i+1},x)}\right)\right] \tag{2.3e}$$

With that, we have described a lower bound for the log-likelihood of the Markovian HVEA for a sample $x$ from $\mathbf{Q}_X$, and although this can also be derived easily by Jensen's inequality, the calculation above also tells us that the gap in the inequality (2.3) is given by

$$\mathbb{E}_{\mathcal{Q}_{\phi;Z_1|X}}\left[\frac{q_\phi(Z_1|x)}{p_\theta(Z_1|x)}\right] + \sum_{i=2}^{T}\mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i}|X}}\left[\frac{q_\phi(Z_i|Z_{i-1})}{p_\theta(Z_i|Z_{i-1})}\right]$$
$$= D_{\mathrm{KL}}(q_\phi(\cdot|x)\|p_\theta(\cdot|x)) + \sum_{i=2}^{T}\mathbb{E}_{\mathcal{Q}_{\phi;Z_{1:i-1}}}\left[D_{\mathrm{KL}}(q_\phi(\cdot|Z_{i-1})\|p_\theta(\cdot|Z_{i-1})\right]$$

### 2.1.3 Variational diffusion models with fixed noise schedule

The following presentation of variational diffusion models is based on the model as described by (Luo, 2022).

Based off of the previous calculations, variational diffusion models arise from placing a number of constraints on Markovian hierarchical variational autoencoders. These constraints are:

[D1] $(\Omega_X, \mathcal{A}_X, \mu_X), (\Omega_{Z_1}, \mathcal{A}_{Z_1}, \mu_{Z_1}), \ldots, (\Omega_{Z_T}, \mathcal{A}_{Z_T}, \mu_{Z_T}) = (\mathbb{R}^d, \mathcal{B}^d, \lambda^d)$, meaning all random variables are of the same dimension. We will also denote $Z_0 = X, (\Omega_X, \mathcal{A}_X, \mu_X) = (\Omega_{Z_0}, \mathcal{A}_{Z_0}, \mu_{Z_0})$.

[D2] $\forall i \in \{1, \ldots, T\} : Z_i = \sqrt{\alpha_i}Z_{i-1} + \sqrt{1-\alpha_i}\varepsilon_i$ where $\varepsilon_i \sim N(0, I_d)$ for a **noise schedule** $(\alpha_i)_{i=1}^T$, so this corresponds to the choice of $\mathcal{Q}_{\phi;Z_i|Z_{i-1}}(z_{i-1}, \cdot) = N(\sqrt{\alpha_i}z_i, (1-\alpha_i)I_d)$. For a parametric noise schedule, this would mean $\phi = (\alpha_i)_{i=1}^T$ however we will constrain ourselves to considering models with a fixed noise schedule.

[D3] $\bar{\alpha}_T \simeq 0$, where $\forall i \in \{1, \ldots, T\}$: $\bar{\alpha}_i := \prod_{t=1}^{i}\alpha_t$ and $\alpha_i \in (0, 1)$

Due to [D2] I will also refer to $\mathbf{Q} = \mathbf{Q}_{\phi;X,Z_{1:T}}$ as the true distribution and in our further calculations, expectation values will implicitly mean expectation values in regards to $\mathbf{Q}$ unless specified otherwise.

The process described by this corresponds to a steady adding of noise coining the name diffusion model. Due to the last constraint, one finds that approximately $Z_T \sim N(0, 1)$ and therefore to sample from the model, one draws $z_T$ from $N(0, 1)$ and samples from $p(z_0, \ldots, z_{T-1}|z_T) = \prod_{i=0}^{T-1} p(z_i|z_{i+1})$. As with hierarchical variational autoencoders, we can derive a loss function from the Evidence Lower

Bound (ELBO), which can easily be derived by applying Jensen's inequality.

$$\log p_\theta(Z_0) \underset{\text{by definition}}{=} \log \int_{\times_{i=1}^T \mathbb{R}^d} p_\theta(Z_0, z_{1:T}) \bigotimes_{i=1}^T \lambda^d(\mathrm{d}z_{1:T}) \tag{2.4a}$$

$$= \log \int_{\times_{i=1}^T \mathbb{R}^d} p_\theta(Z_0, z_{1:T}) \frac{q(Z_{1:T}|Z_0)}{q(Z_{1:T}|Z_0)} \bigotimes_{i=1}^T \lambda^d(\mathrm{d}z_{1:T}) \tag{2.4b}$$

$$= \log \mathbb{E}_{\mathbf{Q}_\phi}\left[\frac{p_\theta(Z_{0:T})}{q(Z_{1:T}|Z_0)}|Z_0\right] \tag{2.4c}$$

$$\underset{\text{(Jensens Ineq.)}}{\geq} \mathbb{E}_{\mathbf{Q}_\phi}\left[\log \frac{p_\theta(Z_{0:T})}{q(Z_{1:T}|Z_0)}|Z_0\right] \tag{2.4d}$$

$$\underset{\text{(Markovian)}}{=} \mathbb{E}_{\mathbf{Q}_\phi}\left[\log \frac{p_\theta(Z_T)p_\theta(Z_0|Z_1)\prod_{t=2}^T p_\theta(Z_{t-1}|Z_t)}{q(Z_1|Z_0)\prod_{t=2}^T q(Z_t|Z_{t-1}, Z_0)}|Z_0\right] \tag{2.4e}$$

$$\underset{\text{(Bayes)}}{=} \mathbb{E}_{\mathbf{Q}_\phi}\left[\log \frac{p_\theta(Z_T)p_\theta(Z_0|Z_1)\prod_{t=2}^T q(Z_{t-1}|Z_0)\prod_{t=2}^T p_\theta(Z_{t-1}|Z_t)}{q(Z_1|Z_0)\prod_{t=2}^T q(Z_t|Z_0)\prod_{t=2}^T q(Z_{t-1}|Z_t, Z_0)}|Z_0\right] \tag{2.4f}$$

$$= \mathbb{E}_{\mathbf{Q}_\phi}\left[\log p_\theta(Z_0|Z_1) + \log \frac{p_\theta(Z_T)}{q(Z_T|Z_0)} + \sum_{t=2}^T \log \frac{p_\theta(Z_{t-1}|Z_t)}{q(Z_{t-1}|Z_t, Z_0)}|Z_0\right] \tag{2.4g}$$

$$= \underbrace{\mathbb{E}_{\mathbf{Q}_\phi}\left[\log p_\theta(Z_0|Z_1)|Z_0\right]}_{\text{[KL1]}} - \underbrace{D_{\mathrm{KL}}(q^{Z_T|Z_0}||p^{Z_T})}_{\text{[KL2]}} \tag{2.4h}$$

$$- \underbrace{\sum_{t=2}^T \mathbb{E}_{\mathbf{Q}_\phi}\left[D_{\mathrm{KL}}(q^{Z_{t-1}|Z_t,Z_0}||p_\theta^{Z_{t-1}|Z_t})|Z_0\right]}_{\text{[KL3]}} \tag{2.4i}$$

Thereby, the [KL3] term results from rewriting the expectation as an integral and by $\log \frac{p(z_{t-1}|t)}{q(z_{t-1}|z_t,z_0)} = -\log \frac{q(z_{t-1}|z_t,z_0)}{p(z_{t-1}|t)}$:

$$\mathbb{E}\left[\log \frac{p(Z_{t-1}|Z_t)}{q(Z_{t-1}|Z_t, Z_0)}|Z_0\right] = \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} q(z_t, z_{t-1}|Z_0) \log \frac{p(z_{t-1}|t)}{q(z_{t-1}|z_t, Z_0)} \lambda_d(\mathrm{d}z_t)\lambda_d(\mathrm{d}z_{t-1}) \tag{2.4j}$$

$$= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} q(z_{t-1}|z_t, Z_0)q(z_t|Z_0) \log \frac{p(z_{t-1}|t)}{q(z_{t-1}|z_t, Z_0)} \lambda_d(\mathrm{d}z_t)\lambda_d(\mathrm{d}z_{t-1}) \tag{2.4k}$$

Lastly, we can apply Jensen's inequality in the equation above by:

$$\mathbb{E}\left[|\frac{p_\theta(Z_{0:T})}{q(Z_{1:T}|Z_0)}|\right] = \int_{\times_{i=0}^T \mathbb{R}^d} |\frac{p_\theta(Z_{0:T})}{q(Z_{1:T}|Z_0)}| q(Z_{1:T}|Z_0) \bigotimes_{i=0}^T \lambda^d(\mathrm{d}z_{0:T}) \tag{2.4l}$$

$$= \int_{\times_{i=0}^T \mathbb{R}^d} p_\theta(Z_{0:T}) \bigotimes_{i=0}^T \lambda^d(\mathrm{d}z_{0:T}) = 1 < \infty \tag{2.4m}$$

The lower bound we derived by this is the same as we already derived in the inequality (2.3) and as noted there, the derivation above also tells us how tight this inequality is.

So equation (2.4i) defines a lower bound on the log-likelihood $p_\theta(Z_0)$ of the model for some $Z_0$ drawn from the intractable distribution $\mathbf{Q}_{Z_0}$. Training the variational diffusion model consists of maximizing the log-likelihood assigned to a set of samples $(Z_0^{(1)}, \ldots, Z_0^{(N)})$ drawn from $\mathbf{Q}_{Z_0}$, which is done by maximizing the lower bound we derived as equation (2.4i). Maximizing the lower bound can for instance be achieved by stochastic gradient descent (of the negative log likelihood).

Notably, under a static noise schedule, the term [KL2] has no trainable parameters (is independent of $\theta$), and can therefore be ignored in the optimization process.

*Note.* As the noise added in each step is independently drawn, using $\bar{\varepsilon}_i \sim N(0, I_d)$ we can derive $Z_i \sim \sqrt{\bar{\alpha}_i} Z_0 + \sqrt{1-\bar{\alpha}_i}\bar{\varepsilon}_i$ as a closed form equation for sampling $Z_i$. We will call this the $\alpha$ **reparameterization**.

*Proof.* Proof by induction: This holds trivially for $i = 1$, so for $i > 1$: $Z_i = \sqrt{\alpha_i} Z_{i-1} + \sqrt{1-\alpha_i}\varepsilon_i \sim \sqrt{\alpha_i\bar{\alpha}_{i-1}} Z_0 + \sqrt{\alpha_i(1-\bar{\alpha}_{i-1})}\bar{\varepsilon}_{i-1} + \sqrt{(1-\alpha_i)}\varepsilon_i \sim \sqrt{\bar{\alpha}_i} Z_0 + \sqrt{1-\bar{\alpha}_i}\bar{\varepsilon}_i$ as $\alpha_i(1-\bar{\alpha}_{i-1}) + (1-\alpha_i) = 1 - \bar{\alpha}_i$ □

So far we didn't need to define what the backward model $p_\theta$ should look like concretely and it is sensible to choose:

$$p_\theta(z_{t-1}|z_t) = N\left(z_{t-1}|\mu_\theta(z_t, t), \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} I_d\right) \tag{2.5a}$$

for $t \in \{1 \dots T\}$, so that $p_\theta$ resembles the true density conditioned on $z_0$ and $z_t$

$$q(z_{t-1}|z_t, z_0) = N\left(z_{t-1}|\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})z_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)z_0}{1-\bar{\alpha}_t}, \frac{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}{1-\bar{\alpha}_t} I_d\right). \tag{2.5b}$$

By choosing $p_\theta$ in this manner, the Kullback-Leibler divergence terms in equation (2.4i) reduce to a simple form by applying the following lemma.

**Lemma 1** (Kullback-Leibler Divergence of two multivariate normal Distributions)**:**
*Given two normal distributions on $\mathbb{R}^d$: $\mathbf{N}_i = N(\mu_i, \Sigma_i), i = 1, 2$ and their corresponding densities $n_1, n_2$ with respect to $\lambda_d$, the Kullback-Leibler divergence is given as:*

$$D_{\mathrm{KL}}(n_1||n_2) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) - d + (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1) + \log\left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)}\right)\right) \tag{2.6}$$

*Proof.* For $X \sim \mathbf{N}_1$:

$$\begin{aligned}
D_{\mathrm{KL}}(n_1||n_2) &= \int_{\mathbb{R}^d} n_1(x) \log\left(\frac{\sqrt{(2\pi)^k \det(\Sigma_2)}\exp(-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1))}{\sqrt{(2\pi)^k \det(\Sigma_1)}\exp(-\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2))}\right)\lambda_d(\mathrm{d}x) \\
&= \frac{1}{2}\int_{\mathbb{R}^d} n_1(x) \log\left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)}\frac{\exp((x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2))}{\exp((x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1))}\right)\lambda_d(\mathrm{d}x) \\
&= \frac{1}{2}\mathbb{E}_{\mathbf{N}_1}\left[\log\left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)}\frac{\exp((x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2))}{\exp((x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1))}\right)\right] \\
&= \frac{1}{2}\left(\log\left(\frac{\det(\Sigma_2)}{\det(\Sigma_1)}\right) + \mathbb{E}\left[(X-\mu_2)^T\Sigma_2^{-1}(X-\mu_2) - (X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1)\right]\right)
\end{aligned}$$

and as for any $\mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d\times d}$:

$$\begin{aligned}
\mathbb{E}\left[(X-\mu)^T\Sigma(X-\mu)\right] &= \mathbb{E}\left[X^T\Sigma X - X^T\Sigma\mu - \mu\Sigma X^T + \mu\Sigma\mu\right] \\
&= \mathbb{E}\left[X^T\Sigma x\right] - \mu_1^T\Sigma\mu - \mu\Sigma\mu_1^T + \mu\Sigma\mu \\
&= \mathbb{E}\left[X^T X \mathrm{tr}(\Sigma)\right] - \mu_1^T\Sigma\mu - \mu\Sigma\mu_1^T + \mu\Sigma\mu \\
&= \mathrm{tr}(\Sigma_1\Sigma) + (\mu_1-\mu)^T\Sigma(\mu_1-\mu)
\end{aligned}$$

Which, when combined with the equality proven above, proves the lemma. □

So now, by $\mu(z_t, z_0, t) := \mathbb{E}_{\mathbf{Q}}[Z_{t-1}|Z_t = z_t, Z_0 = z_0]$ as given by equation (2.5b), the loss term [KL3] in equation (2.4i) simplifies to

$$\sum_{t=2}^{T} \mathbb{E}\left[D_{\mathrm{KL}}(q^{Z_t|Z_{t-1},Z_0}||p^{Z_t|Z_{t-1}})\Big|Z_0\right] = \sum_{t=2}^{T} \frac{1}{2}\frac{1-\bar{\alpha}_t}{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}\mathbb{E}\left[\|\mu_\theta(Z_t,t) - \mu(Z_t,Z_0,t)\|^2\big|Z_0\right] \tag{2.7}$$

Thereby the problem essentially reduces to minimizing a weighted mean square error between the estimated mean of $Z_{t-1}$ and the true mean for which we have a closed form by the $\alpha$ reparameterization. This still leaves some freedom in how the estimated mean is parameterized, but before I introduce two other common methods besides estimating the mean directly, I want to wrap up the equation (2.5b) by a short proof.

**Lemma 2** (Conditional Distributions of Multivariate Normal Distributions)**:**
*Given $X_i \sim N(\mu_i, \Sigma_i), i = 1, 2$, the joint distribution is again a multivariate normal distribution:*

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \Sigma_1 & \Sigma_{12} \\ \Sigma_{21} & \Sigma_2 \end{pmatrix}\right), \text{ where } \Sigma_{12} = \mathrm{Cov}(X_1, X_2), \Sigma_{21} = \Sigma_{12}^T.$$

*Then the conditional distribution $\mathbf{P}_{X_1|X_2}$ is also a multivariate normal distribution with mean $\mu_c = \mu_1 + \Sigma_{12}\Sigma_2^{-1}(X_2 - \mu_2)$ and covariance matrix $\Sigma_c = \Sigma_1 - \Sigma_{12}\Sigma_2^{-1}\Sigma_{21}$ where $\Sigma_2^{-1}$ is the generalized inverse.*

*Proof.* For better readability we will rename some matrices, namely $A = \Sigma_1, B = \Sigma_{12}, C = \Sigma_2$ and we denote the dimension of $X_1, X_2$ respectively by $k_1, k_2$. We will further denote $(x_1, x_2)^T = x, (\mu_1, \mu_2)^T = \mu, \mu_c = \mu_1 + \Sigma_{12}\Sigma_2^{-1}(x_2 - \mu_2)$. Notably, all covariance matrices are positive definite, and thereby the same holds for $\Sigma_c$. The inverse of the joint covariance matrix is given by:

$$\underbrace{\begin{pmatrix} A & B \\ B^T & C \end{pmatrix}}_{\text{further denoted as } \Sigma} \begin{pmatrix} \Sigma_c^{-1} & -\Sigma_c^{-1}BC^{-1} \\ -C^{-1}B^T\Sigma_c^{-1} & C^{-1} + C^{-1}B^T\Sigma_c^{-1}BC^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}$$

Thereby, we can calculate the conditional density directly by:

$$f(x_1|x_2) = \frac{f(x)}{f(x_2)} = \underbrace{\frac{\sqrt{2\pi}^{-k}}{\sqrt{2\pi}^{-k_2}}}_{\text{(I)}} \underbrace{\sqrt{\frac{\det\left(\begin{pmatrix} A & B \\ B^T & C \end{pmatrix}\right)}{\det(C)}}^{-1}}_{\text{(II)}} \underbrace{\frac{\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}{\exp\left(-\frac{1}{2}(x_2-\mu_2)^TC^{-1}(x_2-\mu_2)\right)}}_{\text{(III)}}$$

For the term (I) given that $k = k_1 + k_2$ we easily see that $\frac{\sqrt{2\pi}^{-k}}{\sqrt{2\pi}^{-k_2}} = \sqrt{2\pi}^{-k_1}$, for (II) we use that for a block matrix the determinant is given by:

$$\det\left(\begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix}\right) = \det(D')\det(A' - B'D'^{-1}C')$$

So it remains to show that term (III) is equal to the exponential term in the corresponding density. So by

$$\begin{aligned}(x-\mu)^T\Sigma^{-1}(x-\mu) = {} & (x_1-\mu_1)^T\Sigma_c^{-1}(x_1-\mu_1) - (x_1-\mu_1)^T\Sigma_c^{-1}BC^{-1}(x_2-\mu_2) \\ & - (x_2-\mu_2)^TC^{-1}B^T\Sigma_c^{-1}(x_1-\mu_1) + (x_2-\mu_2)^T(C^{-1} + C^{-1}B^T\Sigma_c^{-1}BC^{-1})(x_2-\mu_2)\end{aligned}$$

and

$$\begin{aligned}(x_1-\mu_c)^T\Sigma_c^{-1}(x_1-\mu_c) &= (x_1-\mu_1-BC^{-1}(x_2-\mu_2))^T\Sigma_c^{-1}(x_1-\mu_1-BC^{-1}(x_2-\mu_2)) \\ &= (x_1-\mu_1)^T\Sigma_c^{-1}(x_1-\mu_1) - (x_1-\mu_1)^T\Sigma_c^{-1}BC^{-1}(x_2-\mu_2) \\ &\quad - (x_2-\mu_2)^TC^{-1}B^T\Sigma_c^{-1}(x_1-\mu_1) + (x_2-\mu_2)^T(C^{-1}B^T\Sigma_c^{-1}BC^{-1})(x_2-\mu_2)\end{aligned}$$

due to the symmetry of $B, C^{-1}$, we observe that:

$$\begin{aligned}(x_1-\mu_c)^T\Sigma_c^{-1}(x_1-\mu_c) &= (x_1-\mu_1-BC^{-1}(x_2-\mu_2))^T\Sigma_c^{-1}(x_1-\mu_1-BC^{-1}(x_2-\mu_2)) \\ &= (x-\mu)^T\Sigma^{-1}(x-\mu)-(x_2-\mu_2)^TC^{-1}(x_2-\mu_2)\end{aligned}$$

Putting all of this together we calculated that:

$$f(x_1|x_2)=\frac{f(x)}{f(x_2)}=\underbrace{\sqrt{2\pi}^{-k_1}}_{\text{(I)}}\underbrace{\sqrt{\det(\Sigma_c^{-1})}}_{\text{(II)}}\underbrace{\exp\left(-\frac{1}{2}(x_1-\mu_c)^T\Sigma_c^{-1}(x_1-\mu_c)\right)}_{\text{(III)}} \tag{2.8}$$

This concludes the proof. □

**Theorem 3:** *The distribution of $Z_{t-1}$ conditioned on $Z_t$ and $Z_0$ is as given in equation (2.5b).*

*Proof.* By the $\alpha$ reparameterization we know that the conditional joint distribution of $Z_{t-1}, Z_t$ is

$$\begin{pmatrix}Z_{t-1}\\Z_t\end{pmatrix}\sim N\left(\begin{pmatrix}\sqrt{\bar\alpha_{t-1}}Z_0\\\sqrt{\bar\alpha_t}Z_0\end{pmatrix},\begin{pmatrix}\sqrt{(1-\bar\alpha_{t-1})}I_d & \zeta\\ \zeta^T & \sqrt{(1-\bar\alpha_t)}I_d\end{pmatrix}\right),\ \text{where } \zeta=\mathrm{Cov}(Z_{t-1},Z_t|Z_0) \tag{2.9}$$

where the covariance matrix $\zeta$ is also a diagonal matrix, due to the noise added at each step having a diagonal matrix as its covariance matrix, and with $\forall i \in \{1\dots d\}$ :

$$\begin{aligned}\zeta_{ii}=\mathrm{Cov}(Z_{t-1}^{(i)},Z_t^{(i)}|Z_0) &= \mathbb{E}\left[Z_{t-1}^{(i)}Z_t^{(i)}\right]-\mathbb{E}\left[Z_{t-1}^{(i)}|Z_0\right]\mathbb{E}\left[Z_t^{(i)}|Z_0\right] \\ &= \sqrt{\alpha_t}\mathrm{Var}(Z_{t-1}^{(i)}|Z_0)+\mathbb{E}\left[\sqrt{1-\alpha_t}\varepsilon_t Z_{t-1}^{(i)}|Z_0\right]=\sqrt{\alpha_t}(1-\bar\alpha_{t-1})\end{aligned}$$

By applying lemma 2, one arrives at equation (2.5b). □

By all of the above, we now know that with our choice of the transition density $p_\theta(z_{i-1}|z_i)$ the term

$$\mathbb{E}_{\mathbf{Q}_\phi}\left[\log p_\theta(Z_0|Z_1)|Z_0\right]-D_{\mathrm{KL}}(q^{Z_T|Z_0}||p^{Z_T})-\sum_{t=2}^{T}\frac{1}{2}\frac{1-\bar\alpha_t}{(1-\bar\alpha_{t-1})(1-\alpha_t)}\mathbb{E}_{\mathbf{Q}_\phi}\left[\|\mu_\theta(Z_t,t)-\mu(Z_t,Z_0,t)\|^2|Z_0\right] \tag{2.10}$$

is a lower bound of the log-likelihood $\log p_\theta(Z_0)$, $Z_0\sim\mathbf{Q}_{Z_0}$, and with this knowledge we can define a training algorithm.

**Algorithm 1** Training of a variational diffusion model

**while** Not converged **do**
  $z_0\sim\mathbf{Q}_{Z_0}$ ▷ Sample from dataset
  $t\sim U(\{2,\dots,T\})$ ▷ We're estimating the sum by drawing $t$ uniformly from $\{2,\dots,T\}$ (Luo, 2022)
  Draw $\varepsilon\sim N(0,I_d)$
  $z_t=\sqrt{\bar\alpha_t}z_0+\sqrt{1-\bar\alpha_t}\varepsilon$ ▷ Closed form sampling by the $\alpha$ reparameterization
  $z_1=\sqrt{\bar\alpha_1}z_0+\sqrt{1-\bar\alpha_1}\varepsilon$
  Gradient descent step over $\nabla_\theta(\|\mu_\theta(z_t,t)-\mu(z_t,z_0,t)\|^2-\log p_\theta(z_0|z_1))$
**end while**

**Algorithm 2** Sampling from a variational diffusion model

Draw $z_T\sim N(0,I_d)$
**for** t=T-1,...,0 **do**
  Draw $z_t\sim N(\mu_\theta(z_{t+1},t+1),\Sigma_{t+1})$ ▷ With $\Sigma_{t+1}$ as described in equation (2.5b).
**end for**
**return** $z_{0:T}$

### 2.1.4 Two alternative ways to parameterize the estimated mean

It seems intuitive to me that given some noisy data, the easiest way of making sense of this data is to take a guess at what could have been the initial data. This idea can motivate the first rather common way to parameterize the estimated mean. Equation (2.5b) suggests that

$$\mu_\theta(z_t, t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})z_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{z}_\theta(z_t, t)}{1-\bar{\alpha}_t} \tag{2.11a}$$

would be a sensible choice for this. The corresponding optimization problem [KL3] has a similarly simple form under this parameterization:

$$\sum_{t=2}^{T} \frac{1}{2} \frac{1-\bar{\alpha}_t}{(1-\bar{\alpha}_{t-1})(1-\alpha_t)} \mathbb{E}_{\mathbf{Q}_\phi}\left[||\mu_\theta(Z_t, t) - \mu(Z_t, Z_0, t)||^2 \middle| Z_0\right] \tag{2.11b}$$

$$= \sum_{t=2}^{T} \frac{1}{2} \frac{\bar{\alpha}_{t-1}(1-\alpha_t)}{(1-\bar{\alpha}_{t-1})(1-\bar{\alpha}_t)} \mathbb{E}_{\mathbf{Q}_\phi}\left[||\hat{z}_\theta(Z_t, t) - Z_0||^2 \middle| Z_0\right] \tag{2.11c}$$

which can be interpreted as training a model to predict the initial data from an arbitrarily noisy version of this data. However, while trying to make sense of noisy data by taking a guess at the initial data makes sense intuitively, this task is essentially equivalent to predicting what the added noise was, leading to the second way to parameterize the estimated mean. By combining the $\alpha$ reparameterization with equation (2.11a) we arrive at:

$$\mu_\theta(z_t, t) = \frac{1}{\sqrt{\alpha_t}} z_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \hat{\varepsilon}_\theta(z_t, t) \tag{2.12a}$$

and the corresponding optimization problem [KL3] turns out as:

$$\sum_{t=2}^{T} \frac{1}{2} \frac{1-\bar{\alpha}_t}{(1-\bar{\alpha}_{t-1})(1-\alpha_t)} \mathbb{E}_{\mathbf{Q}_\phi}\left[||\mu_\theta(Z_t, t) - \mu(Z_t, Z_0, t)||^2 \middle| Z_0\right] \tag{2.12b}$$

$$= \sum_{t=2}^{T} \frac{1}{2} \frac{(1-\alpha_t)}{(1-\bar{\alpha}_{t-1})\alpha_t} \mathbb{E}_{\mathbf{Q}_\phi}\left[||\hat{\varepsilon}_\theta(Z_t, t) - \bar{\varepsilon}_t||^2 \middle| Z_0\right] \tag{2.12c}$$

What makes this particularly interesting is that it reveals a connection to score matching by the method described by Vincent (2011):

$$\nabla_{z_t} \log q(z_t|z_0) = \frac{1}{1-\bar{\alpha}_t}(\sqrt{\bar{\alpha}_t} z_0 - z_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \bar{\varepsilon}_t \tag{2.12d}$$

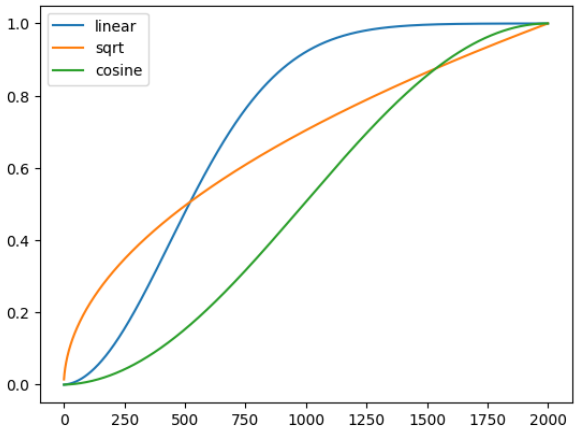
which also brings with it implications for conditioning variational diffusion models, as $\nabla_{z_t} \log q(z_t|c) = \nabla_{z_t} \log q(z_t) + \nabla_{z_t} \log q(c|z_t)$ holds trivially by Bayes rule.

With this, we have three common ways to parameterize the estimated mean, which for the most part can be viewed as equivalent. Estimating the mean directly, while being a straightforward method, receives little attention in relevant papers, and I haven't had the opportunity to test this method out. Ablation studies performed by Li et al. (2022) suggest that, under various noise schedules and prediction models, predicting the initial data is more robust than predicting the noise.

### 2.1.5 Noise schedules

The noise schedule $(\alpha)_{i=1}^{T}$ is not the same for all processes, in fact, there is a multitude of different proposed noise schedules (Li et al., 2022; Luo, 2022). The chosen noise schedule for a given model is not interchangeable, as it is a core part of the model itself. Some of the common noise schedules are visualized in figure 2.3. Notably, the "sqrt" noise schedule was introduced especially for diffusion language models by Li et al. (2022). $1-\bar{\alpha}_t$ is the variance of $Z_t$ given the initial data $Z_0$ under the forward process defined by $q^{Z_{0:T}}$. This noise schedule is either given by the individual values $(\alpha_t)_{t=1}^{T}$ at each step or given as the schedule $(\bar{\alpha}_t)_{t=1}^{T}$.

Figure 2.3: Three common noise schedules visualized by the derived variance at time $t$ by $1-\bar{\alpha}_t$.




### 2.1.6 Classifier-free guidance

As we have seen, variational diffusion models can be parameterized by score matching, which enables us to easily condition the generation process (Luo, 2022). So assuming we want our trained model $\mathbf{P}$ to model a conditional dependency, we can do so by estimating the conditional score $\nabla_{Z_t} \log q(Z_t|Z_0, c)$. By $\nabla_{z_t} \log q(z_t|c) = \nabla_{z_t} \log q(z_t) + \nabla_{z_t} \log q(c|z_t)$ this could, for instance, be estimated by the score predicted by an unconditional model plus the gradient of the probability of the condition being fulfilled under a guiding model. While in classifier-guidance, this is approached by training a separate model to predict if a condition is fulfilled (Li et al., 2022; Luo, 2022), the classifier-free approach chooses to train one model both on data with and without guidance (Ho & Salimans, 2022). The training and sampling algorithms for this approach as proposed by Ho and Salimans (2022) are shown below. For that approach, one interpolates between the guided and the unguided prediction by the guidance strength when sampling from the model.

The DiffuSeq-based models described below can thereby be interpreted as diffusion models with classifier-free guidance, by interpreting the tokens that are not corrupted throughout the diffusion process as the conditioning information. In this framework, DiffuSeq-based models have guidance strength 1 and a probability of unconditional training of 0.

**Algorithm 3** Classifier-free guidance training

**Input**
  $p_{\text{uncond}}$ Probability of unconditional training
**while** Not converged **do**
  $(x, c) \sim q(x, c)$ ▷ Sample from dataset
  $c \leftarrow \varnothing$ with probability $p_{\text{uncond}}$ ▷ Train without guidance with probability $p_{\text{uncond}}$
  $t \sim U(\{1, \dots, T\}$ ▷ Estimate sum by sampling t uniformly
  $\varepsilon \sim N(0, 1)$ ▷ Sample noise
  $z_t = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon$
  $z_1 = \sqrt{\bar{\alpha}_1} z_0 + \sqrt{1 - \bar{\alpha}_1}\varepsilon$
  Gradient descent step over $\nabla_\theta(\|\hat{\varepsilon}_\theta(z_t, c) - \varepsilon\|^2 - \log p_\theta(z_0|z_1))$
**end while**

Here the estimated mean is parameterized by the added noise. $\hat{\varepsilon}_\theta$ would then be a neural network estimating the added noise.

**Algorithm 4** Classifier-free guidance sampling

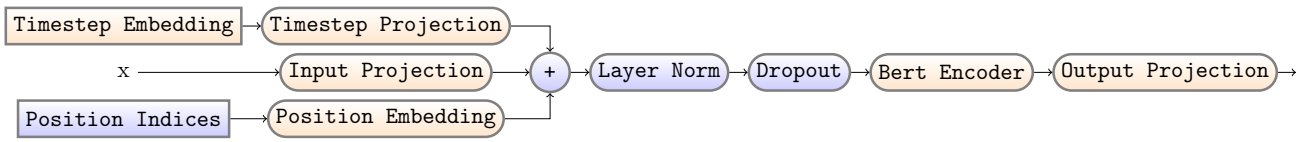
**Input**
  $\gamma$ Guidance strength
  $c$ Conditioning information
$z_T \sim N(0, I_d)$
**for** $t = T, \dots, 1$ **do**
  $\bar{\varepsilon}_t \leftarrow (1 - \gamma)\hat{\varepsilon}_\theta(z_t, t, c) - \gamma\hat{\varepsilon}_\theta(z_t, t)$ ▷ Interpolate between guided and unguided estimate
  $x \leftarrow (z_t - \sqrt{1 - \bar{\alpha}_t}\bar{\varepsilon}_t)/\sqrt{\alpha_t}$ ▷ Calculate an estimate of the initial data.
  $z_{t-1} \sim N(\mu(z_t, x, t), \Sigma_t)$ if $t > 1$ else $z_{t+1} = x_t$ ▷ With $\Sigma_t$ as in equation (2.5b)
**end for**

$\mu(z_t, x, t) = \mathbb{E}_{\mathbf{Q}}[Z_{t-1}|Z_0, Z_t]$ as defined above in equation (2.5b).

Training and sampling algorithm for classifier-free guidance models by Ho and Salimans (2022), matching the notation I used in this thesis. Statements regarding learned noise schedules are excluded.

## 2.2 Diffusion language models

Figure 2.4: A high-level overview of the diffusion-Kernel of Diffusion-LM and DiffuSeq




*Note:* Orange nodes are the target of training, and blue nodes are static parameters.

As briefly stated in the introduction, language modeling is commonly defined as the task of assigning probabilities to sequences of words $\mathbf{P}(w_{1:n})$ and diffusion models model such probability distributions implicitly, by approximately generating samples from them.[1] The diffusion language models covered here can be described as variational diffusion models, with the observed random variable $X$ being in the sequence space of fixed length sequences $V^S$, $V = \{1, \dots, \text{ Vocabulary Size}\}, S \in \mathbb{N}^+$. This variable is embedded into a continuous space by the trained embedding function $E_\theta(X) \in \mathbb{R}^{S \times d}$ and the first step of the fixed forwards process is $Z_1 = \sqrt{\alpha_1}E_\theta(X) + \sqrt{1 - \alpha_1}\varepsilon$, $\varepsilon$ being drawn from

[1] $\mathbf{P}$ is transformed into $\mathbf{L}$, a distribution over the sequence space $V^S$ by a tokenization algorithm.

$N(0, I_{S\times d})$. This slightly modifies the constriction [D1] in section 2.1, however when updating the definition this only affects the [KL1] loss term and therefore doesn't pose any problem. The [KL3] term is then a mean square error loss as derived in equation (2.11), and the estimated mean can be parameterized by various methods.[2] The [KL1] term can be viewed as the cross entropy, which is commonly used as a loss function in NLP.

**Definition 4** (Cross entropy)**.** Given two probability distributions $\mathbf{Q} \ll \mathbf{P}$ over $\mathbb{R}^d$ with densities $p, q$ in regards to $\mu$, we define the **cross entropy of Q under P** by:

$$CE(\mathbf{P}, \mathbf{Q}) := -\int_{\mathbb{R}^d} \log q \, \mathrm{d}\mathbf{P},$$

The last step in the backward process is described by another parametric function $R_\theta : \mathbb{R}^{S\times d} \to [0,1]^{S\times|V|}$, each column being a probability vector over $V$.

### 2.2.1 Diffusion-LM

This model builds upon the work of Ho et al. (2020), who studied diffusion models for image generation. Li et al. (2022) focus on controllable text generation, which is achieved by training classifiers to evaluate at every generation step how well a certain objective is fulfilled and shifting the result by the calculated gradient $\nabla_{X_t}\mathbf{P}(\text{Objective fullfilled}|X_t)$. The embedding function $E_\theta$ is simply a context-free token-wise embedding, and it should be noted that it is only used to obtain the training targets during training and when filling in masked data. Ablation studies performed on Diffusion-LM found that model performance suffered when using pretrained embeddings, so the model performs end-to-end training of embeddings and the diffusion kernel (Li et al., 2022). The recovery function $R_\theta$ is a linear layer followed by a softmax layer and is applied to compute probability vectors for each token. The recovery function is also trained jointly with the diffusion model. Diffusion-LM under its recommended settings parameterizes the estimated mean at step $t$ by the posterior given an estimated initial value, as described above in equation (2.11a) and derives its default loss term by the [KL3] term as:

$$\mathcal{L}^{e2e}_{x_0-simple}(x) = \sum_{t=2}^{T} \mathbb{E}\left[||\hat{z}_\theta(Z_t, t) - Z_1||^2 \big| X = x\right] \tag{2.13a}$$

and the [KL1] term as the cross entropy:

$$\mathcal{L}^{e2e}_{decoder\_nll}(x_0) = \frac{1}{M}\sum_{n=1}^{M} \mathbb{E}\left[-\log(R_\theta(Z_1)_{x_n})|X = x\right] \tag{2.13b}$$

$x_n$ being the target token id of the $n$th token.

Diffusion-LM treats the parameterization of the estimated mean at timestep $t$ as a hyperparameter, with the default recommended setting being parameterization by predicting the initial value. For this parameterization, the estimated mean at time $t$ given $z_t$ is $\mu_\theta(z_t, t) = \frac{\sqrt{\alpha_t}(q-\overline{\alpha}_{t-1})z_t + \sqrt{\overline{\alpha}_{t-1}}(1-\alpha_t)\hat{z}_\theta(z_t,t)}{1-\overline{\alpha}_t}$. The next timestep is then sampled as $z_{t-1} = \mu(z_t, t) + \sqrt{1-\overline{\alpha}_t}\epsilon_t$, $\epsilon_t \sim N(0, I_{S\times d})$.

For infilling of missing tokens, given a mask $M \subset \{1, \ldots, S\}$ and a masked sequence $v_{1:S} \in V^S$, Diffusion-LM has shown capable of filling in the missing data by overwriting the corresponding columns in the predicted initial sequence $\hat{z}_\theta(z_t, t)$ by $E_\theta(v_{1:S})$ at every step $t$, before deriving the estimated mean at time $t-1$ (Li et al., 2022).

$$\tilde{z}_\theta(z_t, t)_{ij} = \begin{cases} \hat{z}_\theta(z_t, t)_{ij} & \text{if } i \notin M \\ E_\theta(v_{1:S})_{ij} & \text{otherwise} \end{cases}$$

[2] See the section "Two alternative ways to parameterize the mean"

**Algorithm 5** Filling in missing data by Diffusion-LM

**Input**
  $\tilde{z} \in \mathbb{R}^d$ Data
  $m \in \{0,1\}^d$ Mask marking what data should be filled in
Draw $z_T \sim N(0, I_d)$
**for** $t = T-1, \ldots, 0$ **do**
  Draw $z_t \sim N(\mu_\theta(z_{t+1}, t+1), \Sigma_{t+1})$ ▷ With $\Sigma_{t+1}$ as described in equation (2.5b).
  $z_{t;i} \leftarrow \tilde{z}_i$ where $m_i = 0$ ▷ Overwrite where the data is given by $\tilde{z}$
**end for**
**return** $z_{0:T}$

### 2.2.2 DiffuSeq

Proposed by Gong et al. (2023), this model implements a sequence-to-sequence model by modeling the joint distribution of the input and output sequences.[3] To achieve that DiffuSeq only applies noise to the target sequence while leaving the source sequence fixed (**conditional noising**), both in training and when sampling. This corresponds to the modified training algorithm:

**Algorithm 6** Training of a DiffuSeq model

**while** Not converged **do**
  $(s, t) \sim q(s, t)$ ▷ Sample from dataset
  $m = (1)^{|s|+1} \oplus (0)^{S-|s|-1}$ ▷ Mask for conditional noising
  $j \leftarrow s \oplus (\hat{s}) \oplus t$ ▷ Concatenate the source and target sequences separated by $\hat{s}$
  Pad $j$ up to length $S$ or truncate to length $S$
  Draw $\varepsilon \sim N(0, I_{d \times S})$
  $z_0 = \sqrt{\bar{\alpha}_1} E_\theta(j) + \sqrt{1 - \bar{\alpha}_1}\varepsilon$ ▷ Embed the joint sequence and add starting noise
  $t \sim I(\{0, \ldots, T-1\}$ ▷ Sampling t by importance sampling (Gong et al., 2023)
  $\varepsilon' \sim N(0, I_{d \times S})$ ▷ Sample noise
  $z_t = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon'$
  $z_{t;i} \leftarrow z_{0;i}$ if $m_i = 1$, $1 \leq i \leq S$ ▷ Conditional noising
  **if** $t = 0$ **then**
    Gradient descent step over $\nabla_\theta(\|\hat{x}_\theta(z_t, t) - E_\theta(j)\|^2 - \sum_{i=|s|+2}^{S} \log(R_\theta(z_0)_{j_i})$
  **else**
    Gradient descent step over $\nabla_\theta(\|\hat{x}_\theta(z_t, t) - z_0\|^2 - \sum_{i=|s|+2}^{S} \log(R_\theta(z_0)_{j_i})$
  **end if**
**end while**

[3] Sequence to sequence tasks can be defined as in the section neural machine translation, so I will exclude the definition and refer the reader to that section.

## 2.3 Neural Machine Translation

Lastly, to wrap up the background information, I will quickly introduce what machine translation refers to and try to define the task more clearly. Machine translation is the problem of automatically translating text from one (source) language to another (target) language, and may be described as modeling the distribution over the target space, conditioned by a sequence from the source space. In neural machine translation, this is achieved by training specific neural networks on this task. (Yoav, 2017) Nowadays, the most common types of neural networks for machine translation are those based on the transformer architecture, however, neural machine translation also includes approaches using recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) (Vennerød et al., 2021).

To apply these methods one starts by applying a tokenization algorithm, with vocabularies $V_s = \{i \in \mathbb{N}_0 | i < \text{Source Vocabulary Size}\}$, $V_t = \{i \in \mathbb{N}_0 | i < \text{Target Vocabulary Size}\}$, resulting in an observed probability distribution $\mathbf{Q}^{X,Y} : \mathcal{P}(\bigcup_{l\in\mathbb{N}^+} V_s^l) \times \mathcal{P}(\bigcup_{l\in\mathbb{N}^+} V_t^l)$, where $X$ denotes random source sequences from $\bigcup_{l\in\mathbb{N}^+} V_s^l$ and $Y$ denotes target sequences from $\bigcup_{l\in\mathbb{N}^+} V_t^l$, with $\mathbf{Q}^{X,Y}$ describing their joint distribution. Then the task becomes modeling a Markov Kernel $\mathbf{T} : \mathcal{P}(\bigcup_{l\in\mathbb{N}^+} V_t^l) \times \bigcup_{l\in\mathbb{N}^+} V_s^l \to [0,1]$ which matches the "true" conditional probability distribution $\mathbf{Q}^{Y|X}$ arising from the tokenized sequences.

The state-of-the-art transformer model, introduced by Vaswani et al. (2017), achieves this by an encoder-decoder architecture, where an encoder module first encodes the source sequence, passing the encoding on to the decoder, which conditioned by the encoding of the source sequence generates an output sequence in an autoregressive manner.

To this end, formally we describe $\forall l_1, l_2 \in \mathbb{N}^+ : (X_1, \ldots, X_{l_1}) = X \in V_s^{l_1}, (Y_1, \ldots, Y_{l_2}) = Y \in V_t^{l_2}$ :

$$\mathbf{T}(Y_1, \ldots, Y_{l_2}; X_1, \ldots, X_{l_1}) = \prod_{i=1}^{l_2} \underbrace{\mathbf{T}(Y_i | Y_1, \ldots, Y_{i-1}; X_1, \ldots, X_{l_1})}_{\text{modelled by transformer}}$$

by Bayes rule. The Markov Kernel $\mathbf{T}$ is then optimized to approximate the conditional distribution $\mathbf{Q}^{X|Y}$.

# Classifier-free Diffusion Models for Machine Translation

With the task of neural machine translation defined above, we further constrain our problem to only consider sequences of a certain (maximum) length $S$, considering that the diffusion models described above generate sequences en bloc. Accordingly, sequences are padded or truncated to the length $S$.

## 3.1 Diffusion-LM-based Model

For the Diffusion-LM-based Model for machine translation, we use a shared dictionary $V = V_s, V_t$, and seek to model the joint distribution $\mathbf{J} : \mathcal{P}(V^S) \to [0, 1]$ of pairs of source and target sequences by training Diffusion-LM on this task. For this, given a number of source and target pairs $(s^{(1)}, t^{(1)}), \ldots, (s^{(n)}, t^{(n)})$ we use the concatenated sequences $j^{(n)} = s^{(1)} \oplus (\hat{s}) \oplus t^{(1)}, \ldots, j^{(n)} = s^{(n)} \oplus (\hat{s}) \oplus t^{(n)}$ separated by a separator token $\hat{s} \in V$ which does not occur in either source sequences or target sequences.

A Diffusion-LM model is trained to maximize the likelihood of the sequences $j^{(1)}, \ldots, j^{(n)}$. As described in Section 2.2.1, Diffusion-LM approximates conditional distributions by the infilling algorithm, which I will describe again here, adapted to the task of translation. The precise steps are given in Algorithm 7

**Algorithm 7** Translation by the Diffusion-LM-based model

**Input**
$s \in V^l, 1 \leq l \leq \frac{S}{2}$ The source sequence
Pad $s$ up to length $S$
$\tilde{z} \leftarrow E_\theta(s)$ ▷ Embed the source sequence as the partial data
$m_i = 1$ for $1 \leq i \leq l$, $m_i = 0$ for $l + 1 \leq i \leq S$ ▷ Calculate the mask
Draw $z_T \sim N(0, I_{d \times S})$
**for** $t = T - 1, \ldots, 0$ **do**
    Draw $z_t \sim N(\mu_\theta(z_{t+1}, t + 1), \Sigma_{t+1})$ ▷ With $\Sigma_{t+1}$ as described in equation (2.5b).
    $z_{t;i} \leftarrow \tilde{z}_i$ where $m_i = 1$ ▷ Overwrite where the data is given by $\tilde{z}$
**end for**
$r_i \leftarrow \underset{t \in V}{\text{argmax}} R_\theta(z_0)_{i,t}$ for $1 \leq i \leq S$ ▷ Recover the most likely token at each position
**return** $r_{l+1:S}$ ▷ Return the tokens after the source

This algorithm serves as an example of the more general infilling algorithm from Diffusion-LM, showing explicitly how translation is performed by infilling. For the more general algorithm see Algorithm 5.

This model is denoted as Diffusion-LM in Chapter 4.

## 3.2 DiffuSeq-based Models

Like the standard DiffuSeq and the Diffusion-LM-based model, the DiffuSeq-based models use shared vocabularies $V_s, V_t = V$. For the training algorithm, I refer back to Algorithm 6. The sampling algorithm is the same as described for the Diffusion-LM-based model (Gong et al., 2023). Building

upon that, we will look at four proposed models next to the standard DiffuSeq model and the method of autoregressive sampling for DiffuSeq models.

### 3.2.1 DiffuSeq with Autoregressive Sampling

The current state-of-the-art in natural language processing is the autoregressive transformer model, which generates a probability vector for the next token one token at a time. The sequential nature of text also suggests, that generating one token at a time is a good approach to text generation. By this method, we will try to understand whether diffusion language models benefit from fixing one generated token at a time and repeating the sampling process with the previously generated token as prior knowledge akin to the infilling method described for the Diffusion-LM-based model.

Algorithm 8 described the sampling algorithm in detail. For the naive implementation given there, this increases the time needed for decoding by a factor of $O(S)$. However, when detecting the end of the generation process, this factor is in $O(\text{Average generated sequence length})$.

**Algorithm 8** Autoregressive sampling from DiffuSeq models

**Input**
$s \in V^l, 1 \leq l \leq \frac{S}{2}$ The source sequence
$t \leftarrow \epsilon$ ▷ Initialize the translation as the empty word
**for** $k = l+2, \ldots, S$ **do**
  $j \leftarrow s \oplus (\hat{s}) \oplus t$ ▷ Concatenate the source and already generated tokens
  Pad $j$ up to length $S$
  $\tilde{z} \leftarrow E_\theta(j)$ ▷ Embed concatenated sequence as the partial data
  $m_i = 1$ for $1 \leq i \leq |j|$, $m_i = 0$ for $|j|+1 \leq i \leq S$ ▷ Calculate the mask
  Draw $z_T \sim N(0, I_{d \times S})$
  **for** $t = T-1, \ldots, 0$ **do**
    Draw $z_t \sim N(\mu_\theta(z_{t+1}, t+1), \Sigma_{t+1})$ ▷ With $\Sigma_{t+1}$ as described in equation (2.5b).
    $z_{t;i} \leftarrow \tilde{z}_i$ where $m_i = 1$ ▷ Overwrite where the data is given by $\tilde{z}$
  **end for**
  $t \leftarrow t \oplus \underset{j \in V}{\text{argmax}} R_\theta(z_0)_{k,j}$ ▷ Recover the most likely token for position $k$
**end for**
**return** $t$ ▷ Return the sequence of generated tokens $t$

For this algorithm, the inner loop corresponds to the infilling algorithm from Diffusion-LM with the outer loop sampling from the model in an autoregressive manner.

For brevity, results from the standard DiffuSeq model sampled by this method are denoted as DiffuSeqAR in Chapter 4.

### 3.2.2 DiffuSeq trained for word generation

As described above, the diffusion language models we are looking at here, just like the majority of autoregressive models, make use of a tokenization algorithm for converting text into a sequence of tokens from a vocabulary of fixed size. For this, words are commonly split into multiple tokens. This corresponds well with natural language, where words are often comprised of a word stem and then modified by some prefix or suffix. One example of this would be English verbs like for example walk, walked, walking, and so forth. Moreover, the concept also corresponds well with compound words, say herself, himself, itself, or also compound nouns.

In contrast to autoregressive models diffusion language models are able to generate multiple tokens at a time and this model tries to make use of this property by training a model to generate one word at a time. For this, given a pair of (untokenized) source and target sequences, the target sequence was split into words by whitespace characters. The model was then trained to predict the next word

(possibly comprised of multiple tokens) in the target sequence, given the source sequence and the words up to the current word. For this, the vocabulary $V$ also included a start token $\xi$, a separator token $\hat{s}$, and an end token $\hat{e}$.

**Algorithm 9** Training of DiffuSeq for word generation

**while** Not converged **do**
  $(s, t) \sim q(s, t)$ ▷ Sample from dataset
  $W \leftarrow$ Number of words in $t$
  $w_i \leftarrow$ Start index of i-th word $1 \leq i \leq W$, $w_{W+1} = |t| + 1$
  $i \sim U(\{1, \ldots, W\})$
  $s \leftarrow (\xi) \oplus s \oplus (\hat{s}) \oplus t_{1:w_i-1}$ ▷ Source concatenated with first i-1 words of the target sequence
  $t \leftarrow t_{w_i:w_{i+1}-1}$ ▷ Next word in target sequence
  $m = (1)^{|s|} \oplus (0)^{S-|s|}$ ▷ Mask for conditional noising
  $j \leftarrow s \oplus t$ ▷ Concatenate the source and target sequences separated by $\hat{s}$
  ▷ Rest is as in standard DiffuSeq training
  Pad $j$ up to length $S$ or truncate to length $S$
  Draw $\varepsilon \sim N(0, I_{d \times S})$
  $z_0 = \sqrt{\bar{\alpha}_1} E_\theta(j) + \sqrt{1 - \bar{\alpha}_1}\varepsilon$ ▷ Embed the joint sequence and add starting noise
  $t \sim U(\{0, \ldots, T-1\}$ ▷ Estimate sum by sampling t randomly
  $\varepsilon' \sim N(0, I_{d \times S})$ ▷ Sample noise
  $z_t = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon'$
  $z_{t;i} \leftarrow z_{0;i}$ if $m_i = 1$, $1 \leq i \leq S$ ▷ Conditional Noising
  **if** $t = 0$ **then**
    Gradient descent step over $\nabla_\theta(\|\hat{x}_\theta(z_t, t) - E_\theta(j)\|^2 - \sum_{i=|s|+2}^{S} \log(R_\theta(z_0)_{j_i})$
  **else**
    Gradient descent step over $\nabla_\theta(\|\hat{x}_\theta(z_t, t) - z_0\|^2 - \sum_{i=|s|+2}^{S} \log(R_\theta(z_0)_{j_i})$
  **end if**
**end while**

### 3.2.3 DiffuSeq with Knowledge Distillation

Non-autoregressive models can often profit from knowledge distillation as by training the model to imitate the results of a teacher model, the complexity of a dataset can be reduced (Gu et al., 2017b; Ren et al., 2020). Given a trained teacher model $\mathbf{T}_{\text{teacher}}$ this can be achieved in a simple manner by optimizing over

$$\underset{\theta \in \Theta}{\text{minimize:}}\ \mathbb{E}_{t \sim U(\{2,T\})}\left[\mathbb{E}\left[\|\hat{z}_\theta(Z_t, t) - (E(\hat{s}) + \sigma_0\varepsilon)\|^2 + \frac{1}{n}\sum_{n=1}^{M} -\log \text{Softmax}(R(Z_0))_{\hat{s}_n}\right]\right] \tag{3.1}$$

with $\hat{s} = \underset{s \in V^S}{\text{maximize}} : \mathbf{T}_{\text{teacher}}(s|s_{\text{source}})$ being the teacher models prediction. The maximizing sequence $\hat{s}$ can be approximated by the beam search algorithm. This approach is called sequence-level knowledge distillation and was proposed by Kim and Rush (2016).

Sequence-level knowledge distillation is also explored in Chapter 4, with the model using this technique being sometimes referred to as DiffuSeq-distilled.

While this approach presents a simple form of knowledge distillation, word-level knowledge distillation (Kim & Rush, 2016) could also be achieved by training on the cross entropy of the predicted tokens between the student model and the teacher model. The cross-entropy of the student model

under the teacher model would by this be:

$$CE(\mathbf{T}_t, \mathbf{P}^{Z_0}) = -\int_{\mathbb{R}^d} \log p_\theta(Z_0)\, \mathrm{d}\mathbf{T}_t \tag{3.2a}$$

$$\leq \mathbb{E}_{\mathbf{T}_t}\left[\mathbb{E}\left[-\log p_\theta(Z_0|Z_1) + D_{\mathrm{KL}}(q^{Z_T|Z_0} \| p_\theta^{Z_T}) + \sum_{t=2}^{T} D_{\mathrm{KL}}(q^{Z_{t-1}|Z_t, Z_0} \| p_\theta^{Z_{t-1}|Z_t}) \middle| Z_0\right]\right] \tag{3.2b}$$

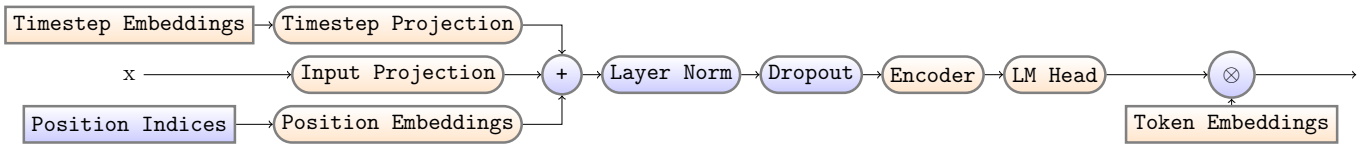
This, however, poses a problem given an autoregressive teacher model, as such a model does not in fact model the probability over an entire sequence, but instead on a token-to-token basis. One possible approach would be distilling knowledge from a masked language model, with the outer integral being a weighted sum and therefore giving a rather easy loss function. Still, due to diffusion language models struggling to commit to word embeddings (Li et al., 2022) this seems counterintuitive and this approach was not expanded upon due to time constraints.

### 3.2.4 Alternate Diffusion Kernel

Autoregressive models will commonly use a linear layer from the internal embedding dimension to the size of the vocabulary $|V|$ as a decoding layer. This relates well to the scalar product with $|V|$ different $d$ dimensional vectors, which we can intuitively understand by $x = \underset{y \in \mathbb{R}^d}{\operatorname{argmax}} < x, y >$. In machine translation, models using shared weights for the encoder and decoder embeddings and for the decoding layer may show better performance over models with separate weights for some pairs of languages. This alternate diffusion kernel attempts to build on that, by explicitly modeling $S$ probability vectors over $V$ and performing a matrix multiplication with the embedding matrix describing $E_\theta$. By this, this alternate diffusion kernel also models the estimated mean of the initial data explicitly and involves the input embeddings in decoding from the encoder at each time $t$.

In contrast to the standard DiffuSeq model the encoder in this diffusion, kernel is not a BERT model, but a transformer encoder as described by Vaswani et al. (2017). Like the BERT encoder in the usual architecture, this encoder is initialized with random weights.

Figure 3.1: A high-level overview of the alternate Diffusion Kernel modeling the mean by probability vectors for each token.




*Note:* The LM Head layer is comprised of a linear layer to the vocabulary size and a softmax layer, so given a vocabulary $V$ and a max sequence length $S$ the output of the LM Head layer is of shape $S \times |V|$, after multiplication with the embedding matrix of shape $|V| \times d$ we arrive back at data of shape $S \times d$. Yellow nodes indicate trained parameters.

### 3.2.5 Position-dependent noise scheduling

To approach the problem of diffusion language models' poor performance given long source sequences, I want to explore the position-dependent sigmoid noise schedule which will be described below. The idea motivating this noise schedule is that by lowering the noise faster in tokens early in the sentence this noise schedule might conform better to the sequential nature of language. The framework described in Chapter 2 can be trivially expanded to encompass sequences of diagonal matrices. So by using the shorthand notation:

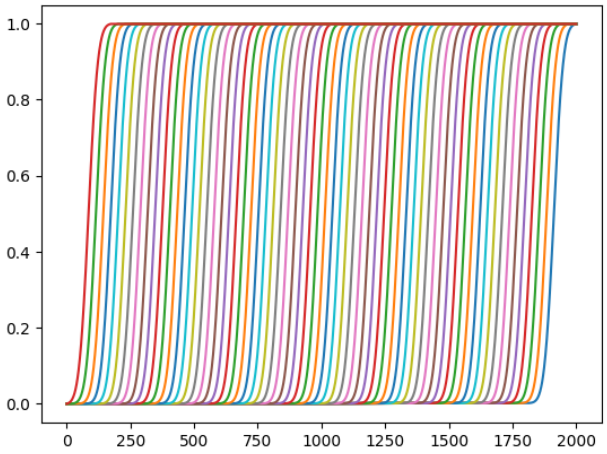
$$D(\lambda) := D(\lambda^{(1)}, \dots, \lambda^{(d)}) := \begin{pmatrix} \lambda^{(1)} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda^{(d)} \end{pmatrix}$$

for $\lambda = (\lambda_n)_{n=1}^d$ and

$$\sqrt{D(\lambda)} := D(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_d}),$$

we only need to ensure that restriction [D3] holds for each sequence $(\alpha_t^{(i)})_{t=1}^T, i = 1, \dots, d$. Then all results from Chapter 2 are still applicable by multiplying with $D((\alpha_t^{(i)})_{i=1}^d)$ at time $t$, where we previously multiplied by a scalar instead.

Figure 3.2: The proposed sigmoid noise schedule




*Note:* The proposed sigmoid noise schedule for a max sequence length of 64 tokens. Here, each line in the graphic represents the noise schedule for one token in the sequence. One can observe, that the tokens are corrupted by noise one after another, with some degree of overlap. Here, the first red line is the noise schedule of the 64th token, while the last blue line is the noise schedule of the first token in the sequence.

The sigmoid noise schedule chosen for this model is then derived as a series of $S$ cumulative distribution functions of normal variables, all with variance $\sigma$ and with different means $(\mu_k)_{k=1}^S$, evaluated in an interval $[m_{\min}, m_{\max}]$. To ensure a moderate degree of overlap in the generation of adjacent tokens, we choose $\sigma$ and the means $(\mu_k)_{k=1}^S$ as follows.

Given the three $\sigma$-rule shown in Table 3.1, under a latent space of dimension $M \times d$, a sensible choice would be $\sigma = \frac{1}{M+5}$, setting $\mu_k = (3+k)\sigma, k = 1, \dots, M$. The interval $[m_{\min}, m_{\max}]$ is then chosen as $[0, 1]$.

Table 3.1: Probabilities of deviations from the mean and certain quantiles of the normal distribution

| $k$ | $\mathbf{P}(\lVert X - \mu \rVert < k\sigma) \simeq$ | $\mathbf{P}(X < \mu + k\sigma) \simeq$ |
|---|---|---|
| 1 | 0.683 | 0.842 |
| 2 | 0.955 | 0.978 |
| 3 | 0.998 | 0.999 |

With this, we derive our position-dependent noise schedule as

$$\forall i \in \{1 \dots T\}, \text{ with } t = \frac{i}{T} : \bar{\alpha}_i = (\underbrace{\Phi(\frac{t-\mu_1}{\sigma}), \dots, \Phi(\frac{t-\mu_1}{\sigma})}_{d \text{ times}}, \dots, \underbrace{\Phi(\frac{t-\mu_M}{\sigma}), \dots, \Phi(\frac{t-\mu_M}{\sigma})}_{d \text{ times}}) \quad (3.3)$$

This choice of parameters would imply $\bar{\alpha}_1 \simeq 0$ and $\bar{\alpha}_T \simeq 1$. Due to variance levels getting very close to 0 this noise schedule needs further measures to ensure numerical stability.[1]

[1] In the model evaluated here, a cutoff for the variance was introduced, after which the variance decayed linearly instead of exponentially. This cutoff was at $1 - \bar{\alpha}_t \leq 0.02$

# Model Evaluation

## 4.1 Experimental Setup

To evaluate the models and methods described in section 3 we trained both the Diffusion-LM-based model and the DiffuSeq-based models (Gong et al., 2023; Li et al., 2022) on the bilingual corpus CoVoST (Wang et al., 2020). For the knowledge distillation model from 3.2.3 we used the `wmt19-en-de` model by Ng et al. (2019), as the teacher model. The detokenized results of all implementations and baselines were evaluated by BLEU-score as provided by SacreBLEU (Post, 2018) and by the `wmt20-comet-qe-da` model from "Unbabel's Participation in the WMT20 Metrics Shared Task" by Rei et al. (2020), which is the latest publicly available model for reference-free evaluation at the time of writing. The `wmt20-comet-qe-da` is a model trained to score machine translation systems on a scale of 0 to 1 (Rei et al., 2020) which in the tables given below is denoted as a percentage value.

To quantify the impact of masked noising during training (DiffuSeq) we generated a set of samples from Diffusion-LM without guidance and calculated a score by the `wmt20-comet-qe-da` model.

For providing insights into system performance over different lengths of the source sequence, the source and target pairs from the test set were split into buckets of roughly equal size. Then the respective BLEU score of each system in a certain bucket was calculated. On top of that, to test the hypothesis that the studied non-autoregressive systems show higher relative differences of performance on long sequences, the relative difference in scores between models was analyzed by linear regression, and by a t-test on whether the slope is positive. For this, we define the relative difference $d_r(a, b) := \frac{a-b}{\max(a,b)}$. For further information on the tests performed, the reader may refer to the source of the testing strategy (Fahrmeir, 2007).

### 4.1.1 Dataset and data preparation

Table 4.1: Key metrics of the dataset CoVoST, with the tokenizer used here.

| CoVoST Key metrics | # |
|---|---|
| Number of samples in train split | 127,638 |
| Mean length of source sequences in tokens | 12.6 |
| Mean length of target sequences in tokens | 12.4 |
| Number of samples in valid split | 13,510 |
| Number of samples in test split | 13,510 |
| Number of samples in reduced test split | 2010 |

The CoVoST dataset is a multilingual speech-to-text translation corpus, providing sentences in multiple languages as text and as spoken language (Wang et al., 2020). For the experiments conducted here, only the text data with sentences in German and English was used. For data preparation, the German characters ä, ü, ö, and ß were replaced by ae, ue, oe and ss respectively, accents were removed, and the set of characters was reduced to the alphabet, numerals, and punctuation marks (excluding brackets and parentheses).

All were trained from scratch on the corpus CoVoST, tokenized by BPE-tokenization with a vocabulary size of 30000. Byte-Pair Encoding is a subword-based algorithm that merges word pieces that frequently occur together in the training set until the maximum length of the vocabulary is

reached. In our case, this relates to 29,846 merges, which are applied greedily to new sequences. [1][2]

After applying a tokenization algorithm as described above, the diffusion language models further truncated some sequences longer than 64 tokens, accounting for less than 0.01% of the samples in the dataset. The dataset CoVoST was chosen due to its relatively small size, based on the datasets chosen in the relevant paper that introduced Diffusion-LM. The datasets chosen for evaluation there consisted of 50K and 98K samples respectively (Li et al., 2022). Some studies on Diffusion Language Models with comparable architectures have however achieved competitive results on bigger datasets such as IWSLT14 (Yuan et al., 2023). Due to the slow decoding process of the autoregressive sampling method, the test set was reduced to a subset of 2010 samples.[3]

### 4.1.2 The evaluated implementations

Next to the baseline models described below, the following models were evaluated here and are shown in the main results (hyperparameters marked in bold are non-defaults):

1. **Diffusion-LM-based**: This is the model based on Diffusion-LM with infilling as described in Section 3.1. Model hyperparameters:
   (a) **Embedding dimension: 256**
   (b) Diffusion steps: 4000
   (c) Noise schedule: "sqrt"
   (d) Estimated mean parameterized by estimating $z_0$
   (e) Batch size: 128
   (f) Maximum sequence length: 64
   (g) No gradient clipping
   (h) Fixed noise schedule
   (i) End-to-end training of diffusion kernel and embedding matrix
   (j) Decoding with the clamping trick applied [4]

   The encoder in the diffusion kernel was parameterized by a BERT model with the configuration `bert-base-uncased`. All weights were initialized randomly.

2. **DiffuSeq**: The standard DiffuSeq model. Model hyperparameters:
   (a) Embedding dimension: 128
   (b) Diffusion steps: 2000
   (c) Noise schedule: "sqrt"
   (d) Estimated mean parameterized by estimating $z_0$
   (e) Batch size: 2048 by gradient accumulation with microbatches of size 128
   (f) Maximum sequence length: 128
   (g) No gradient clipping
   (h) Fixed noise schedule
   (i) Decoding with clamping trick

[1] For further reference, the source of the implementation used for the diffusion language models is provided here.
[2] The tokenization algorithm used only subword-prefixes for the diffusion language models.
[3] The reduced test set is available under https://drive.google.com/file/d/1nj2S7dOLGBel7ZR4AWbVCxEFVcgWg_V3/view?usp=drive_link
[4] The paper introducing Diffusion-LM states that this empirically improves sample quality (Li et al., 2022), however, some more recent papers suggest that this might not consistently be the case (Yuan et al., 2023)

As before the encoder in the diffusion kernel was parameterized by a BERT model with the configuration `bert-base-uncased`. All weights were initialized randomly.

3. **DiffuSeqAR**: The standard DiffuSeq model with the method of autoregressive sampling as described in Section 3.2.1. As the model is the same as above I will refrain from listing the hyperparameters again.

4. **DiffuSeq-distilled**: The model utilizing sequence-level knowledge distillation as described in Section 3.2.3. This model also uses the same hyperparameters as the standard DiffuSeq model described above.

All models used a constant step size of 1 during the sampling process. This results in a very long decoding time, as the diffusion kernel needs to be applied 2000 (for DiffuSeq) or 4000 (for Diffusion-LM) times for translating a batch of samples. Using a lower number of diffusion steps during sampling can increase the speed of the sampling process, but generally leads to decreased performance (Gong et al., 2023; Li et al., 2022).

Some further models, which failed to produce non-trivial samples or generally showed decreased performance are also listed below in Section 4.2.2.

### 4.1.3 Baselines

Transformer encoder-decoder models are state-of-the-art in sequence-to-sequence tasks, so we will compare to a small transformer model with 6 feed-forward layers, embedding dimension 512, feed-forward layer embedding dimension 1024 and 4 attention heads in both encoder and decoder. The model uses shared weights for encoder and decoder embeds and for the language modeling head. Besides these changed parameters the model used was as described by the paper which introduced the transformer model (Vaswani et al., 2017). Decoding was performed with beam size of 10, length penalty of 1, temperature of 1, and no further modifications to the standard beam search. Given the non-autoregressive nature of Diffusion Models, you will also find the results of a non-autoregressive Levenshtein transformer with the parameters as given in the paper (Gu et al., 2019) in the results below. Decoding parameters were also chosen as presented by the paper. [5]

The baselines also used Byte-Pair Encoding as the tokenization algorithm with the same vocabulary size of 30,000. The implementation differed slightly, with this implementation occasionally producing a designated unknown token during decoding. [6]. In contrast to the other models, the algorithm used a subword-suffix in this case, slightly reducing the comparability.

## 4.2 Results

The results for this setup are presented in Table 4.2. First, we observe that the Diffusion-LM-based model performs poorly when faced with the problem of translating test data. When sampling from the Diffusion-LM model without the infilling algorithm, the model successfully generated pairs of German and English sentences. The data generated by this unguided approach, when evaluated by the reference-free COMET model `wmt20-comet-qe-da` achieved a score of 8.72%. However, when faced with the challenge of translating the test set, the score fell to 0.94%.[7] In terms of overall results, all Diffusion Language models underperformed compared to both the transformer model and the Levensthein-Transformer. Compared to the standard DiffuSeq, both the autoregressive sampling method and the model employing sequence-level knowledge distillation showed some improvements, with the difference being pronounced in the DiffuSeq-Distilled model. The Diffusion-LM-based model, modeling the joint distribution performed poorly when faced with the task of translating the test data.

[5] For details and reproducing the results, the implementation is given here.

[6] For reference the implementation is given here

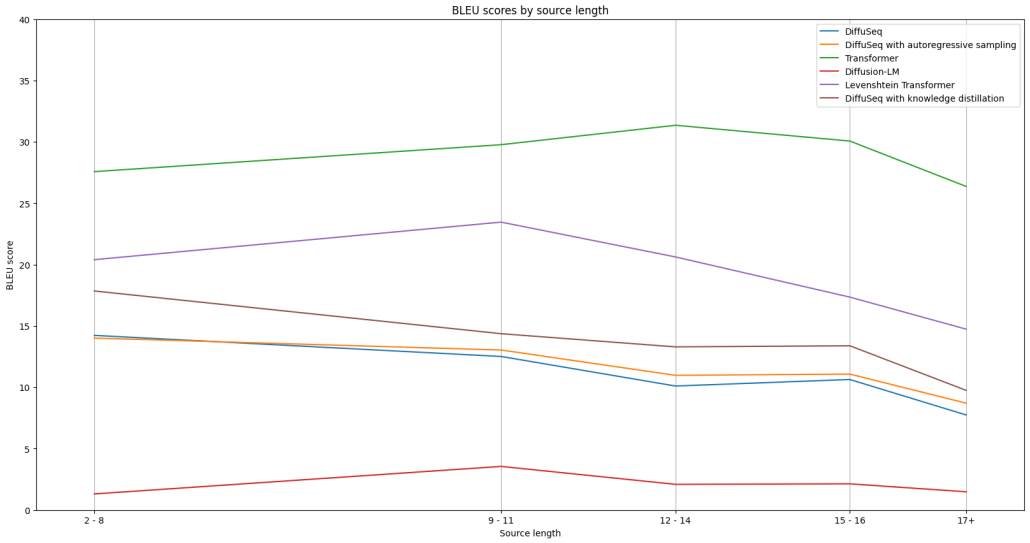
[7] This might indicate that during the generation process interdependencies within the German and English sentence are more generally more influential than the cross dependency between the sequence, which also provides an explanation for the improved performance of the DiffuSeq model. This hypothesis would need further testing however.

Table 4.2: Direct comparison of models by BLEU score and COMET score

| Model | BLEU↑ | COMET↑ |
|---|---|---|
| Diffusion-LM based model | 2.3% | 0.9% |
| DiffuSeq | 10.0% | 3.0% |
| DiffuSeq with autoregressive sampling | 10.7% | 2.5% |
| DiffuSeq-Distilled | 12.5% | 4.1% |
| Transformer | **28.8%** | **20.2%** |
| Levensthein-Transformer | 19.2% | 10.7% |

### 4.2.1 Analysis of the results over differing lengths of the source sequence

Figure 4.1: Comparison of BLEU score between all models, with the 2010 samples being split into buckets of approximately equal size by length.




When the translated samples are split into buckets of roughly equal size by the length of the source sequence, the diffusion language models fall off notably faster in BLEU score compared to the baseline transformer model, suggesting that long-range dependencies might be more problematic for these models to portray.

DiffuSeq with autoregressive sampling shows similar performance compared to the standard model, with the scores differing only marginally on shorter sequences and being slightly better on longer sequences. The model trained with knowledge distillation consistently outperformed both of the other DiffuSeq models.

Lastly, the non-autoregressive Levenshtein transformer, although it consistently outperforms all diffusion language models, also falls off faster than the autoregressive transformer model on longer sequences.

These results tie directly into the initial question, of whether parallel generation impacts the relative difference in performance over differing lengths of source sequences.

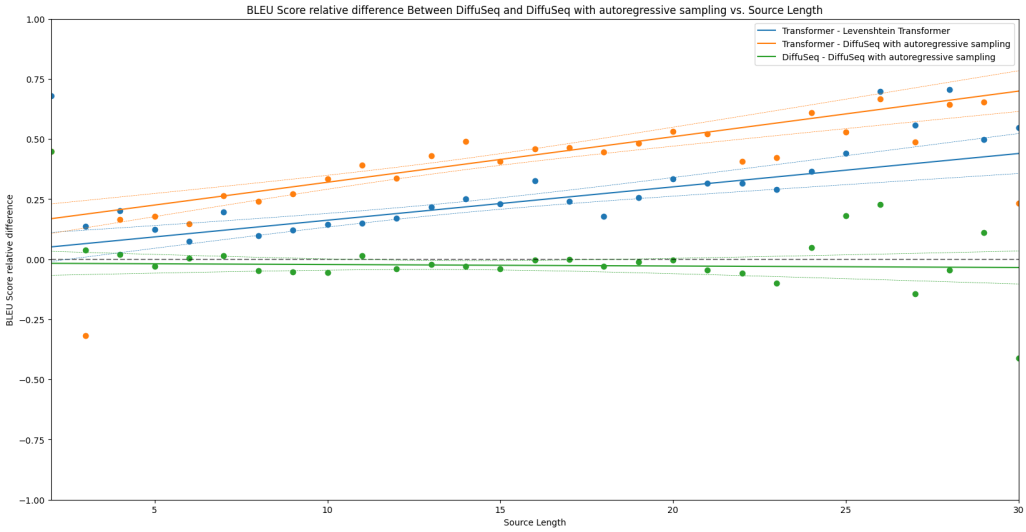
#### Linear regression analysis

Due to the tokenization algorithm of the baselines occasionally producing the designated unknown token during decoding, here a subset of 1987 samples was used where the source sequences could be clearly matched. Each sample was assigned a sentence-level BLEU score, and the length of the

corresponding source sequence was measured in tokens.

Given a sample $A \sim \mathcal{A}(\cdot, s_{1:x})$ from model $\mathcal{A}$ and sample $B \sim \mathcal{B}(\cdot, s_{1:x})$ from model $\mathcal{B}$ given the source sequence $s_{1:x}$ as defined in section 2.3, we assume a linear dependence of $d_r(\text{BLEU}(A), \text{BLEU}(B)) = \beta_0 + x \cdot \beta_1 + \varepsilon$. We additionally assume the error $\varepsilon$ to follow a normal distribution, with constant variance. Each of the 1987 paired samples was treated as one data point, meaning that while Figure 4.2 also shows a scatter-plot of the mean BLEU score for each length of the source sequence, the linear regression was performed over the set of all samples independently.

Figure 4.2: Linear regression over the relative difference between the BLEU scores of selected models over different lengths of the source sequences.




Note: 99% Confidence intervals for the mean relative difference are given by the dashed lines, while dots indicate the empirical mean relative difference for a certain length of source sequences. Still one should note that the sample sizes for very short and very long sequences are much smaller than for sequences of medium length.

As is visible in Figure 4.2, there clearly seems to be a steady increase in the relative difference in scores between the diffusion language models and the transformer model. The figure also shows the linear regression analysis of the relative difference in scores between the standard DiffuSeq model and the DiffuSeq model with autoregressive sampling. While the linear model has a slightly negative slope, the difference is marginal.

### t-Test of slopes

To test the hypothesis, that the relative difference in performance increases with an increase in the length of the source sequence, we can perform a t-Test on whether the slope in the corresponding linear regression is positive. For this, the t-Test assumed the null hypothesis "The length of the source sequence is uncorrelated to the relative difference in scores" for each pair of models.

The variance of the estimator for the slope is $\hat{\sigma}^2(D^T D)^{-1}_{11}$, $D$ being the designer matrix and $\hat{\sigma}^2$ the mean squared error between predicted and observed values. With the slope of the fitted linear model denoted as $\beta_1$, the test statistic $\frac{\beta_1}{\sqrt{\hat{\sigma}^2(D^T D)^{-1}_{11}}}$ follows a Student's t-distribution with 1987 degrees of freedom under the null hypothesis that $\beta_1 = 0$. The zero hypothesis can then be rejected on a 1% level if $|\frac{\beta_1}{\sqrt{\hat{\sigma}^2(D^T D)^{-1}_{11}}}| > t_{1987}(0.995)$ (Fahrmeir, 2007).

The test statistic for each pair of models is summarized in Table 4.3. Above all, all test statistics reported in the transformer column of the table are sufficient to reject the zero hypotheses, indicating

Table 4.3: Test statistics for the T-Test on whether the slope is greater than 0 under the linear model of the relative difference of the scores.

| Models | DiffuSeq | DiffuSeqAR | Transformer | Diffusion-LM | Lev-Transformer | DiffuSeq-distilled |
|---|---|---|---|---|---|---|
| DiffuSeq | N/A | 0.40 | **11.29** | -2.89 | **4.95** | -0.20 |
| DiffuSeqAR | -0.40 | N/A | **9.88** | -2.55 | **4.06** | -0.56 |
| Transformer | -11.29 | -9.88 | N/A | -10.42 | -7.36 | -10.97 |
| Diffusion-LM | **2.89** | 2.55 | **10.42** | N/A | **7.33** | **2.69** |
| Lev-Transformer | -4.95 | -4.06 | **7.36** | -7.33 | N/A | -4.97 |
| DiffuSeq-distilled | 0.20 | 0.56 | **10.97** | -2.69 | **4.97** | N/A |

*Note:* The critical value is $t_{1987}(0.995) \simeq 2.58$ for a 1% significance level. Pairs, where the null hypothesis "The relative difference of scores is uncorrelated to the length of the source sequence" can be rejected and where the slope is positive are marked in bold. By this, a positive test statistic indicates that there is a statistically significant impact of the length of the source sequence on the relative performance of the models, indicating that the model at the top of the column performs relatively better on longer sequences than the model at the start of the row.

that besides the transformer model performing better than all other models across all lengths of source sequences, the relative difference also increases with longer source sequences. The same can be said for the Levenshtein transformer regarding all diffusion models, although the slope is less steep in these combinations.

For the various DiffuSeq-based models the null hypothesis could not be rejected for any pair of models, however, this does not mean that there is no impact of the length of the source sequence on the relative performance between these models, but just, that if there does exist one our data is not sufficient to show so. Notably, the slope of the linear model between the DiffuSeq model with autoregressive sampling and the standard DiffuSeq model is slightly positive, so there might still be a slightly positive impact and both the DiffuSeq model with autoregressive sampling and the standard DiffuSeq model show a slight improvement over length against the DiffuSeq model with sequence-level knowledge distillation.

### 4.2.2 Models which converged to trivial distributions

The following models were also evaluated in the same manner as the diffusion language models described above:

1. **DiffuSeq trained for word generation** as described in Section 3.2.2. Here the number of diffusion steps was reduced to 200, by the intuition that only very few tokens are generated at a time, so a shorter trajectory might suffice. This model is shown by the name WordGeneration in Figure 4.3.

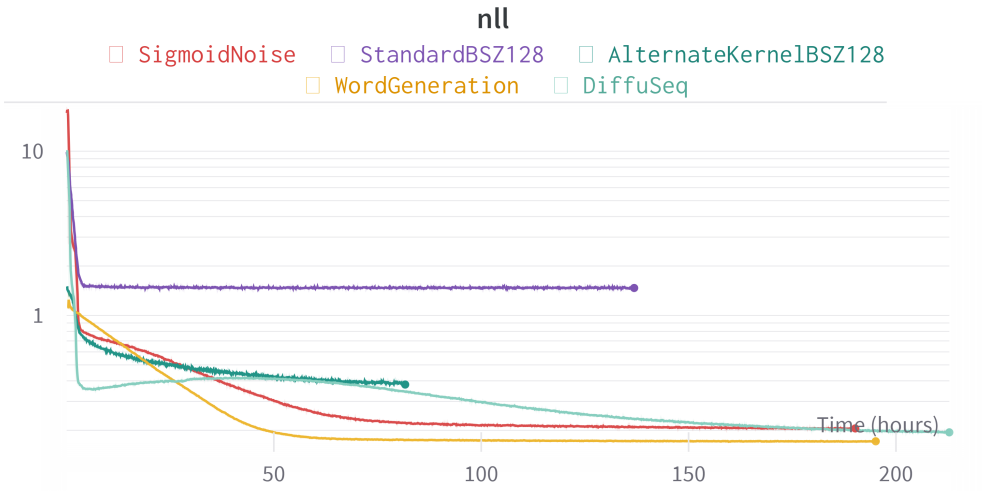
2. **DiffuSeq** as described above, but with a reduced batch size of 128, without gradient accumulation. Otherwise, the hyperparameters were set just as in the standard model. In Figure 4.3 this model is denoted by StandardBSZ128.

3. **DiffuSeq with an alternate diffusion kernel** as described in Section 3.2.4. This model was also trained with a batch size of 128 without gradient accumulation. The implementation of the transformer encoder in the alternate diffusion kernel described in Section 3.2.4 is publicly available in the fairseq repository.[8] In Figure 4.3 this model is denoted by AlternateKernelBSZ128.

4. **DiffuSeq using the sigmoid noise schedule** was trained with the position-dependent noise schedule proposed in Section 3.2.5. This model was trained with both the position-dependent noise schedule and trained both with and without guidance as shown in the background section on classifier-free guidance. The probability for unconditional training was 50%. Furthermore, in

[8]The exact configuration used was that of `transformer_iwslt_de_en`

Figure 4.3: Training process of different models which converged to a trivial distribution, compared to the standard model which is denoted by DiffuSeq.




*Note:* The metric "nll" shown in this figure is the negative log-likelihood, of the tokens recovered by the model. At timestep $t$ this can be expressed by $\frac{1}{M}\sum_{n=1}^{M} -\log(R_\theta(\hat{z}_\theta(z_t, t))_{s_n})$, $s_n$ being the n-th token of the target sequence. The models do not optimize this metric directly, it is reported for tracking the generation quality of a model during training (Gong et al., 2023).

contrast to the standard DiffuSeq model, this model used fixed positioning for embedding the guidance and target sequences.[9] In Figure 4.3 this model is denoted as SigmoidNoise.

5. **DiffuSeq using the sigmoid noise schedule with parameterization by score matching**: This model experimented with parameterization by score matching as described in the Chapter Backgrounds. Besides that, this model is the same as the other model using the proposed sigmoid noise schedule.

However, all of these models besides the model using the alternate diffusion kernel converged to produce trivial or meaningless samples. After convergence, models 1. and 2. produced only padding tokens when sampled from, while models 4. and 5. produced meaningless data. The model using the alternate diffusion kernel converged to produce non-trivial samples, yet is still listed here due to generally poor performance.

This offers one possible explanation for why the model trained for word generation failed, as the Diffusion Language Models studied operate on fixed-length sequences. With sequences being padded up to the maximum length, when under the tokens that the loss is calculated over, the number of padding tokens far outweighs the number of non-padding tokens, the batch size might need to be appropriately higher to prevent the model from converging to such a trivial distribution. The standard DiffuSeq model, has a maximum combined sequence length of 128 tokens under default parameters, while the Diffusion-LM model under standard parameters has a sequence length of 64 tokens. Given a mean source sequence length of 12.6, and a mean target sequence length of 12.4, this amounts to on average 66% padding tokens for Diffusion-LM, accounting for a separator token between source and target. For DiffuSeq, the average percentage of padding tokens would be 87% and for the model trained for word generation, this percentage is greater than 95%.

Consequently, a diffusion model trained for word generation might still be possible but would require a diffusion kernel or a loss function optimized toward that goal. The model utilizing the

[9]This corresponds to first padding/truncating the sequences up/down to a certain length before training the model on the joint sequences. The conditional noise from DiffuSeq was then not applied to padding in the guidance sequence.

alternate diffusion kernel proposed in 3.2.4 converged with a batch size of 128, yet performed poorly, with a BLEU score of 6.0% under SacreBLEU.

The low negative log-likelihood achieved by the model trained for word generation in Figure 4.3 can be explained by the fact that this metric is also computed over all padding tokens. For this model, padding tokens made up $> 95\%$ of the target tokens.

Finally, there were two tests done on the proposed sigmoid noise schedule, one with the estimated mean parameterized by score matching and one with the estimated mean parameterized by the estimate of the initial data. Neither of the tests produced noteworthy results, with the converged models producing meaningless data. Both models used a training procedure inspired by classifier-free guidance as described in 2.1.6, training on data with and without guidance, with a probability of unguided training set to 0.5 (50%).

## 4.3 Training and decoding times of the models

Table 4.4: Key metrics regarding training and decoding times of the different non-autoregressive models

| | Training Time | # of steps | Batch Size | Decoding time | GPUs |
|---|---|---|---|---|---|
| Lev-Transformer | 19h | 300,000 | 128 | 6s | 1 NVIDIA RTX 3070 |
| Diffusion-LM | 3d 6h | 600,000 | 128 | 1h 44m 46s | 1 NVIDIA TITAN RTX |
| DiffuSeq | 14d 9h | 80,000 | 2048 | 3h 23m 17s | 1 NVIDIA TITAN RTX |
| DiffuSeq AR | 14d 9h | 80,000 | 2048 | >30h | 1 NVIDIA TITAN RTX |
| DiffuSeq-distilled | 10d 12h | 60,000 | 2048 | 3h 23m | 1 NVIDIA TITAN RTX |

The DiffuSeq-based models still showed improvement even after extensive training times, so due to the very long time needed to train a diffusion language model, the DiffuSeq Standard model was trained for 80,000 steps, while the DiffuSeq-distilled model was trained for 60,000 steps. Key metrics on the training and decoding times of the various models are summarized in Table 4.4.

The previous section on models which converged to trivial distributions, suggests one explanation for the slow optimization of the DiffuSeq-based models. Due to the very high degree of gradient accumulation, DiffuSeq-based models compute the loss of one batch in 16 micro-batches of 128 samples. This might be important to stabilize the training process and prevent issues as described above under Models which converged to trivial distributions. On top of that, the constant sequence length of 128 tokens for the DiffuSeq-based models inflates the time needed to compute each step further. In comparison with Diffusion-LM, which performs the same basic calculations with a sequence length of 64 tokens, DiffuSeq takes much longer to calculate the loss over one step than Diffusion-LM takes for 16 steps with the same amount of samples overall.

## 4.4 Conclusion

All evaluated diffusion language models struggle to compete with both conventional transformer models and the non-autoregressive Levenshtein transformer model. While both the Levenshtein transformer model and the diffusion language model showed an increase in the relative difference of scores compared to the transformer model with increasing lengths of the source sequence, the scores of the Diffusion Language Models also decreased faster in relative terms than those of the Levenshtein transformer. Autoregressive sampling from the DiffuSeq model exhibits slightly increased scores over conventional sampling, yet there is not enough data to reject the null hypothesis "The mean relative difference in scores between DiffuSeq with conventional sampling and DiffuSeq with autoregressive sampling is constant over various lengths of the source sequence."

In line with findings on other non-autoregressive language models, diffusion language models seem to profit from sequence-level knowledge distillation, exhibiting faster convergence and improved results overall.

In terms of training and decoding times, Diffusion Language Models are still far slower than Transformer models and the Levenshtein transformer. Previous studies showed that decreasing the number of diffusion steps to down to 1000 had little impact on performance (Li et al., 2022). This would decrease the decoding time by roughly half, as the diffusion process dominates the model's time complexity. On top of that, with Diffusion Models having become the state of the art in image synthesis, much work is being done on alleviating the issues of slow training and decoding, with the possibility that these solutions might be applicable to Diffusion Language Models. Alternate diffusion kernels (to the one seen in Figure 2.4) might also optimize better with a lower degree of gradient accumulation.

# References


Dathathri, S., Madotto, A., Lan, J., Hung, J., Frank, E., Molino, P., Yosinski, J., & Liu, R. (2019). Plug and play language models: A simple approach to controlled text generation. *CoRR*, *abs/1912.02164*. http://arxiv.org/abs/1912.02164

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, *12*(null), 2121–2159.

Fahrmeir, L. (2007). *Regression : Modelle, methoden und anwendungen* (T. Kneib & S. Lang, Eds.). Springer. https://doi.org/10.1007/978-3-540-33933-5

Gong, S., Li, M., Feng, J., Wu, Z., & Kong, L. (2023). DiffuSeq: Sequence to sequence text generation with diffusion models. *International Conference on Learning Representations, ICLR*. https://doi.org/10.48550/arXiv.2210.08933

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. https://doi.org/10.48550/arXiv.1406.2661

Gu, J., Bradbury, J., Xiong, C., Li, V. O. K., & Socher, R. (2017a). Non-autoregressive neural machine translation. *CoRR*, *abs/1711.02281*. http://arxiv.org/abs/1711.02281

Gu, J., Bradbury, J., Xiong, C., Li, V. O. K., & Socher, R. (2017b). Non-autoregressive neural machine translation. *CoRR*, *abs/1711.02281*. http://arxiv.org/abs/1711.02281

Gu, J., Wang, C., & Zhao, J. (2019). Levenshtein transformer. *CoRR*, *abs/1905.11006*. http://arxiv.org/abs/1905.11006

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *CoRR*, *abs/2006.11239*. https://arxiv.org/abs/2006.11239

Ho, J., & Salimans, T. (2022). Classifier-free diffusion guidance. https://arxiv.org/abs/2207.12598

Kim, Y., & Rush, A. M. (2016). Sequence-level knowledge distillation. *CoRR*, *abs/1606.07947*. http://arxiv.org/abs/1606.07947

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *CoRR*, *abs/1906.02691*. http://arxiv.org/abs/1906.02691

Kingma, D. P., & Welling, M. (2022). Auto-encoding variational bayes. https://doi.org/10.48550/arXiv.1312.6114

Klenke, A. (2020). *Wahrscheinlichkeitstheorie* (4., überarbeitete und ergänzte Auflage). Springer Spektrum. https://doi.org/10.1007/978-3-662-62089-2

Lee, J., Mansimov, E., & Cho, K. (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. *CoRR*, *abs/1802.06901*. http://arxiv.org/abs/1802.06901

Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., & Hashimoto, T. B. (2022). Diffusion-lm improves controllable text generation. https://doi.org/10.48550/ARXIV.2205.14217

Luo, C. (2022). Understanding diffusion models: A unified perspective. *CoRR*, *abs/2208.11970*. https://doi.org/10.48550/arXiv.2208.11970

Ng, N., Yee, K., Baevski, A., Ott, M., Auli, M., & Edunov, S. (2019). Facebook fair's WMT19 news translation task submission. *CoRR*, *abs/1907.06616*. http://arxiv.org/abs/1907.06616

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. https://doi.org/10.48550/arXiv.1912.02762

Post, M. (2018). A call for clarity in reporting BLEU scores. *Proceedings of the Third Conference on Machine Translation: Research Papers*, 186–191. https://doi.org/10.18653/v1/W18-6319

Rei, R., Stewart, C., Farinha, A. C., & Lavie, A. (2020). Unbabel's participation in the WMT20 metrics shared task. *Proceedings of the Fifth Conference on Machine Translation*, 911–920. https://aclanthology.org/2020.wmt-1.101

Ren, Y., Liu, J., Tan, X., Zhao, S., Zhao, Z., & Liu, T. (2020). A study of non-autoregressive model for sequence generation. *CoRR*, *abs/2004.10454*. https://arxiv.org/abs/2004.10454

Ruthotto, L., & Haber, E. (2021). An introduction to deep generative modeling. *CoRR*, *abs/2103.05180*. https://arxiv.org/abs/2103.05180

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 2256–2265). PMLR. https://proceedings.mlr.press/v37/sohl-dickstein15.html

Song, Y., & Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. *CoRR*, *abs/1907.05600*. http://arxiv.org/abs/1907.05600

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *CoRR*, *abs/2011.13456*. https://arxiv.org/abs/2011.13456

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. http://arxiv.org/abs/1706.03762

Vennerød, C. B., Kjærran, A., & Bugge, E. S. (2021). Long short-term memory RNN. *CoRR*, *abs/2105.06756*. https://arxiv.org/abs/2105.06756

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Comput.*, *23*(7), 1661–1674. https://doi.org/10.1162/NECO_a_00142

Wang, C., Pino, J., Wu, A., & Gu, J. (2020). CoVoST: A diverse multilingual speech-to-text translation corpus. *Proceedings of The 12th Language Resources and Evaluation Conference*, 4197–4203. https://www.aclweb.org/anthology/2020.lrec-1.517

Wu, B., Nair, S., Martin-Martin, R., Fei-Fei, L., & Finn, C. (2021). Greedy hierarchical variational autoencoders for large-scale video prediction. *CoRR*, *abs/2103.04174*. https://arxiv.org/abs/2103.04174

Xiao, Y., Wu, L., Guo, J., Li, J., Zhang, M., Qin, T., & Liu, T.-y. (2023). A survey on non-autoregressive generation for neural machine translation and beyond. https://doi.org/10.48550/arXiv.2204.09269

Yang, K., & Klein, D. (2021). FUDGE: controlled text generation with future discriminators. *CoRR*, *abs/2104.05218*. https://arxiv.org/abs/2104.05218

Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., & Yang, M.-H. (2023). Diffusion models: A comprehensive survey of methods and applications. https://doi.org/10.48550/arXiv.2209.00796

Yoav, G. (2017). *Neural network methods for natural language processing*. Springer. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d1506512%26site%3dehost-live

Yuan, H., Yuan, Z., Tan, C., Huang, F., & Huang, S. (2023). Seqdiffuseq: Text diffusion with encoder-decoder transformers. https://doi.org/10.48550/arXiv.2212.10325