VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2024-25
School Of Computer Science and Engineering
(SCOPE)

Review of Generative Techniques in Deep Learning

Apurva Mishra: 22BCE2791

Contents

1 Abstract

Generative models have become central to advancements in deep learning, particularly with the rise of Large Language Models (LLMs) significantly influenced by the Transformer architecture. This paper presents a review of two dominant generative paradigms stemming from this architecture: autoregressive models and diffusion models. It begins by dissecting the foundational vanilla Transformer, detailing its encoder-decoder structure, attention mechanism, and layer normalization. The review then explores the principles of autoregressive generation, common in text-based LLMs, highlighting its sequential nature and associated challenges such as effective context length limitations and quadratic computational complexity. Subsequently, the paper introduces diffusion models, outlining the forward noising process and the learned backward denoising process, including considerations for training loss parameterization. Recognizing the inherent mismatch between continuous diffusion processes and discrete data like text, the review briefly surveys emerging techniques for discrete diffusion modeling, such as Mean Prediction, Ratio Matching, and Concrete Score Matching, culminating in the introduction of Score Entropy as a promising state-of-the-art approach. This review aims to provide a concise overview of the mechanisms, strengths, and challenges of these key generative techniques in modern deep learning.

2 Intro

Large Language Models have been instrumental in recent advances in the field of deep learning. This is in large part fueled through the introduction of new transformer architecture for sequence modelling [1].

Using this transformer architecture, two major paradigms have become popular: Auto-regressive and diffusion based.

3 Transformer

3.1 Vanilla Transformer

The vanilla transformer [1] is a sequence to sequence model, composed of a **encoder**, **stack of layers** and **decoder**.

Encoder: The encoder is responsible for converting the input sequence into representation in a continuous latent space. It contains multi-head self attention mechanism and position-wise feed-forward network (FFN). Finally each block is normalized [2].

```
Algorithm: Token Embedding
Input: $v \in\cong [N_V]$, a token ID.
Output: $e \in R^{d_e}$ , the vector representation of the token.
Parameters: $W_e \in R^{d_e * N_V}$, the token embedding matrix.
return $e = W_e[:, v]$
```

Table 1: Algorithm: Representing input tokens into latent space

Decoder: The decoder is responsible for converting the new transformation of the input sequence in the latent space through the stack of layers back into text. The decoder is similar to encoder in design. However it also includes a additional multi-head attention layer between the self attention layer and the feed-forward network (FFN) layer. Again this is also normalized.

```
Algorithm: Un-Embedding
Input: $e \in R^{d_e}$, a token encoding.
Output: $p \in \Delta(V)$ , a probability distribution over the vocabulary
Parameters: $W_e \in R^{d_e * N_V}$, the unembedding matrix.
return $\mathrm{softmax}(W_u e)$
```

Table 2: Algorithm: Converting vector from latent space to vocabulary

Layer Normalization For all: encoder, sub-layers and decoder layer normalization is performed for the final output. This is essential as transformer have shown to become unstable due to changes in activations from varying dataset. [3] Here layer normalization ensures that each output is normalized to a consistent distribution. This ensures a consistent distribution.

```
Algorithm: Layer Normalization
Input: $e \in R^{d_e}$, neural network activations.
Output: $e \in R^{d_e}$, normalized activations.
Parameters: $y, \beta R^{d_e}$, element-wise scale and offset

1. $m \leftarrow \sum_{i=1}^{d_e} \frac{e[i]}{d_e}$
2. $v \leftarrow \frac{\sum_{i=1}^{d_e} \left(\frac{e[i]}{d_e} - m\right)^2}{d_e}$

return $e = \frac{e-m}{\sqrt{v}} \odot y + \beta$, where $\odot$ denotes element-wise multiplication.
```

Table 3: Algorithm: Algorithm for layer normalization

3.2 Attention

The state of the art performance of vanilla transformer is obtained using the attention mechanism. It works based on three inputs: *Query-Key-Value (QKV)*. The function can be represented as:

$$\text{Attention}(QKV) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \tag{1}$$

```
Algorithm: Attention
Input: $v \in\approxeq [N_V]$, a token ID.
Output: $e \in R^{d_e}$ , the vector representation of the token.
Parameters: $W_e \in R^{d_e * N_V}$, the token embedding matrix.

return $e = W_e[:, v]$

Input: $X \in R^{d_x * l_x}, Z \in R^{d_z * l_z}$, vector representation of primary and context sequence
Output: $V \in R^{d_{\text{out}} * l_x}$,, updated represetations of tokens in X, folding in information from token Z

Parameter: $W_{qkv}$ consist of
$W_q \in R^{d_{\text{attn}} * d_x}, b_q \in R^{d_{\text{attn}}}$
$W_k \in R^{d_{\text{attn}} * d_x}, b_k \in R^{d_{\text{attn}}}$
$W_v \in R^{d_{\text{attn}} * d_x}, b_v \in R^{d_{\text{attn}}}$
Hyperparameters: Mask $\in \{0, 1\}^{l_z * l_x}$
1. $Q \leftarrow W_q X + b_q$
2. $K \leftarrow W_k X + b_k$
3. $Q \leftarrow W_v X + b_v$
4. $S \leftarrow K^T Q$

return $V \cdot \text{softmax}\left(\frac{S}{\sqrt{d_{\text{attn}}}}\right)$
```

Table 4: Algorithm: Attention

3.3 Autoregressive

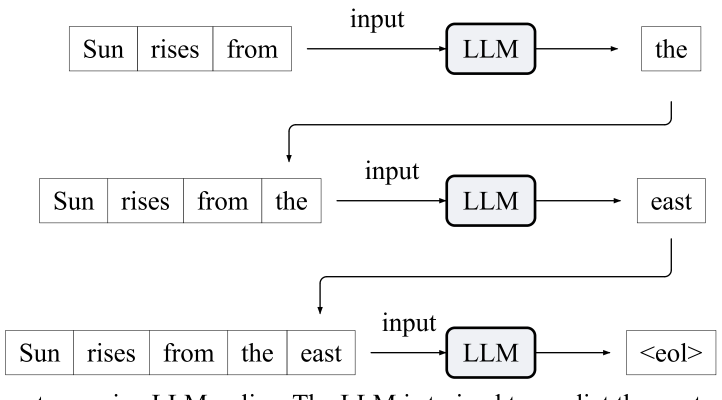


Figure 1: Example for an autogressive LLM policy. The LLM is trained to predict the next most probably word. Then this is appended to the original string and the process continues until the end of line special token is received.

For an input string $\{s_1..s_n\}$ and output string $\{s_{n+1}..s_{n+k+1}\}$, a model works on the policy of:

$$p(s_{n+1}..s_{n+k+1}|s_1..s_n) \\ p(s_{n+1}|\ s_1..s_n)p(s_{n+2}|\ s_1..s_n+1)..p(s_{n+k}|\ s_1..s_{n+k-1}) \tag{2}$$

Thus each successive string is sampled based all the previous set of strings in sequence. This policy can be generalized to simply predict the next word in the string using the chain rule and then repeated.

3.4 Context Length

The total length of the input tokens is called the context length. The context length can be considered as the short term memory of the llm and the conditional tokens against which the llm predicts the next most likely word.

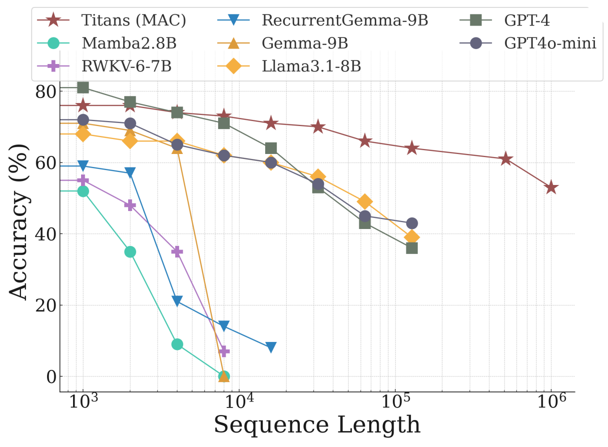
Therefore a high context length is topic of much discussion. Theoretically the length of input string can be infinite, however in practice its limited due to several factors. For example LLama 3.3 [4] family of models have an impressive context length of $128k$ tokens.

However the effective context length is often much shorter.

3.4.1 Size of training dataset

The documents in training dataset have a finite length. Most documents in popular datasets often do not exceed 10k tokens. LLM(s) trained on this context length often do not generalize over longer contexts. Increasing the context length above this during inference causes the performance of the model to degrade rapidly. [5], [6]

Dataset Name	Average Length	Data Access
SumSurvey	>12,000 tokens	SumSurvey Dataset
NarrativeXL	>50,000 words	NarrativeXL Dataset
LongBench	10,000 tokens	LongBench Dataset
HiPool	4,034 tokens	HiPool Benchmark
Databricks DocsQA	2,856 tokens	Databricks Blog



,

Table 5: Left: Average sizes of documents in few popular LLM training datasets
Right: Accuracy of popular LLM with increasing context length. Signifying effective context length. [7]

3.4.2 Quadratic complexity

Transformer based attention architecture requires quadratic time complexity over the context length. Specifically QK^T multiplication requires $O(n^2)$ computation and memory. This is the vanilla attention and here the output token can attend to all the input tokens. This can be visualized using the attention mask.[8]

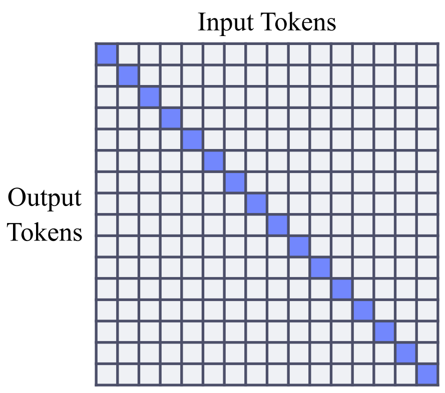


Figure 2: Attention Mask of Vanilla Transformer

3.5 Representation of Data

Let the vocabulary of Model be: V. The input text/data needs to be represented in this vocabulury on which the model would be trained. Empirical experiments have shown that correct scaling of vocabulary size in relation to parameter count of the model is very important for optimal performance. [9]

The process of representing input data as a sequence of vocabulury elements is called **tokenization**. Tokenization can be achieved in several manner. For a piece of text: "Sun rises from the east", it can be tokenized as:

1. **Character-level Tokenization**:

$$V \in [a, .., z] \tag{3}$$

   Here each vocabulary token corresponds to a alphabet: ['S', 'u', ..'s', 't']. These generalize well, however these are too small representation and very expensive for large context lengths due to quadratic complexity. In addition a big vocabulary causes the embedding matrix to huge causing increasing memory complexity.

2. **Word-level Tokenization**:

$$V \in \text{Words} \tag{4}$$

   Here each vocabulary token corresponds to a word: ['Sun', 'rises', 'from', 'the', 'east']. These require very large vocabulury and do not generalize well.

3. **Subword Tokenization**:

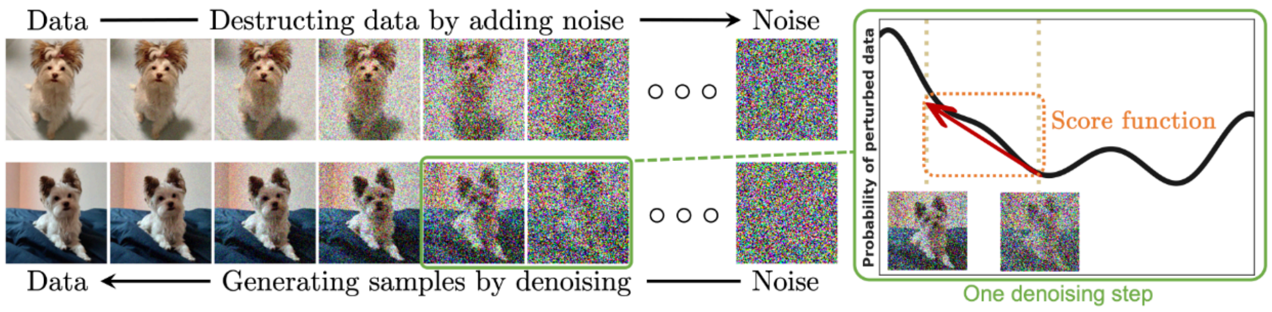
$$V \in \text{Commonly Occurring Sub Words} \tag{5}$$

   Here each vocabulary token corresponds to a commonly occurring word segments: ['ris', ses', 'the' ..] These offer the best middle ground between vocabulary size and generalization. Notably common words like *'the'* are tokenized as it is.

4. **Byte-Pair Encoding (BPE)**: This is the most popular type of sub-word tokenization scheme. It was introduced in the paper: Neural Machine Translation of Rare Words with Subword Units [10]

4 Diffusion

Diffusion models are based on non-equilibrium thermodynamics. Here we iteratively add noise to the original data (forward diffusion process) and then learn to iteratively get the data back from noise (backward diffusion process).



Data
Destructing data by adding noise
Noise
Data
Generating samples by denoising
Noise
Probability of perturbed data
Score function
One denoising step

Figure 3: Noise is progressively added to an image and then a model is trained to progressively remove noise. [11]

4.1 Forward Diffusion Process

For data distribution: x_0 in $q(x)$ during diffusion process we add Gaussian noise to sample in T steps, producing a sequence of noisy samples $x_1, .., x_T$.

$$\begin{aligned} q\left(x_t \mid x_{\{t-1\}}\right) &= N\left(x_t; \sqrt{1 - b\eta_t} x_{\{t-1\}}, b\eta_t I\right) \\ q\left(x_{\{1:T\}} \mid x_0\right) &= \sum_{t=1}^{T} q(x_t \mid x_{t-1}) \end{aligned} \tag{6}$$

Here the step size is controlled using variance schedule: $\{\beta_t \in (0,1)\}_{t=1}^{T}$ hyperparameters. Key to the success of this sampling process is training the reverse Markov chain to match the actual time reversal of the forward Markov chain.

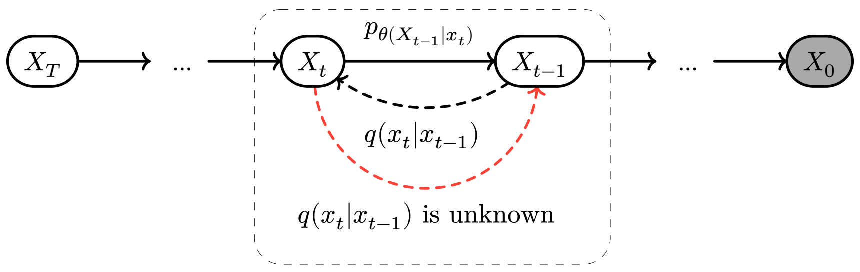


Figure 4: Forward Diffusion Process [12]

4.2 Backward Diffusion Process

If we can reverse the above process and sample from $q\left(x_{\{t-1\}} \mid x_t\right)$, we will be able to recreate the true sample from a Gaussian noise input, $x_T \approx N(0, I)$. Note that if β_t is small enough, $q(x_{t-1} \mid x_t)$ will also be Gaussian. Unfortunately, we cannot easily estimate $q(x_{t-1} \mid x_t)$ because it needs to use the entire dataset and therefore we need to learn a model p_θ to approximate these conditional probabilities in order to run the reverse diffusion process.

$$p_{\theta(x_{0:T})} = p(x_T) \sum_{t=1}^{T} p_{\theta(x_{t-1} \mid x_t)} p_{\theta(x_{t-1} \mid x_t)} = N\left(x_{t-1}; u_{\theta(x_t,t)}, \sum_{\theta(x_t,t)}\right) \tag{7}$$

4.3 Parameterization of L_t for Training Loss

Now for training a neural network which can learn to create an image from noise, we need to define a loss function, such that a model can learn to minimize it. The mathematical loss function is very complex, however empirically found simplified loss functions are used. [13]

$$\begin{aligned} L_t^{\text{simple}} &= E_{t\square[1,T],x_0,\varepsilon_t}\left[\left|\varepsilon_t - \varepsilon_{\theta(x_t,t)}\right|^2\right] \\ &= E_{t\square[1,T],x_0,\varepsilon_t}\left[\left|\varepsilon_t - \varepsilon_{\theta\left(\sqrt{\alpha_t}x_0+\sqrt{1-\alpha_t}\varepsilon_t,t\right)}\right|^2\right] \end{aligned} \tag{8}$$

Thus final objective becomes:

$$L_{\text{simple}} = L_t^{\text{simple}} + C \tag{9}$$

Algorithm: Diffusion Training

```
Require: Model $x_\theta(z_t)$ to be trained
Require: Data set $D$
Require: Loss weight function $w()$

while not converged do Sample Data : $x \approx D$
Sample Time : $t \approx U[0,1]$
Sample Noise: $e \approx N(0,I)$ Add Noise : $z_t = \alpha_t x + \sigma_t e$

Clean data is target for $x_t = x$
Log-SNR: $\lambda_t = \log\left[\frac{\alpha_t^2}{\sigma_t^2}\right]$
Loss: $L_\theta = w(\lambda_t) \mid \sim \{x\} - x_{\theta(z_t)}|_2^2$
Optimization: $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
end while
```

Algorithm: Sampling

```
1: $x_T \square N(0,I)$
2: for $t = T, ..., 1$ do
3: $z \square N(0,I)$ if $t > 1$, else $z = 0$
4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\varepsilon_{\theta(x_t,t)}\right) + \sigma_t z$
5: end for
6: return $x_0$
```

Table 6: Algorithm: Training and Sampling Diffusion Models [13]

5 Text Diffusion: Future Outlook

For text generation auto-regressive models are most widely used. However they can not be parallelized. This if fine for text generation which is fundamentally discrete in nature. However for data with high dimension and resolution such as images, auto-regressive paradigmn is very slow. Thus they use diffusion based generative models to make effective use of parallelism during training and inference.

However, with the rapid growth in the user of LLM and test time scaling [14] there ability to parallelise text generation is major consideration. Diffusion process is continuous in nature. For modelling text based discrete diffusion model, technique of score entropy is used.

5.1 Discrete Diffusion Modelling Techniques

Mean Prediction: Instead of directly parameterizing the ratios $\frac{p_{t(y)}}{p_{t(x)}}$,[15] instead follow a strategy of to learn the reverse density $p_0|t$. This actually recovers the ratios $\frac{p_{t(y)}}{p_{t(x)}}$ in a roundabout way, but comes with several drawbacks. Learning $p_0|t$ is inherently harder since it is a density (as opposed to a general value). Furthermore, the objective breaks down in continuous time and must be approximated [16]. As a result, this framework largely underperforms empirically.

Ratio Matching: Originally introduced in [17] , ratio matching learns the marginal probabilities of each dimension with maximum likelihood training. However, the resulting setup departs from standard score matching and requires specialized and expensive network architectures. As such, this tends to perform worse than mean prediction.

Concrete Score Matching: [18] generalizes the standard Fisher divergence in score matching, learning with concrete score matching: Unfortunately, the loss is incompatible with the fact that $\frac{p_{t(y)}}{p_{t(x)}}$ must be positive. In particular, this does not sufficiently penalize negative or zero values, leading to divergent behavior. Although theoretically promising, Concrete Score Matching struggles.

5.2 Score Entropy

Build on top of previous methods, score entropy aims to mitigate the problems faced in previous techniques. It is the current state of art method for discrete diffusion modelling, originally introduced in [19].

It has several favourable properties which can be used to derive loss function for discrete diffusion steps.

Properties:
1. **First**, score entropy is a suitable loss function that recovers the ground truth concrete score.
2. **Second**, score entropy directly improves upon concrete score matching by rescaling problematic gradients.
3. **Third**, similar to concrete score matching, score entropy can be made computationally tractable by removing the unknown $\frac{p(y)}{p(x)}$ term. There are two alternative forms, the first of which is analogous to the implicit score matching loss. [17]
4. **Fourth**, the score entropy can be used to define an ELBO for likelihood-based training and evaluation.
5. **Fifth**, score entropy can be scaled to high dimensional tasks.

Using these the denoising loss function becomes:

$$E_{\substack{x_0 s\,\mathrm{im}\, p_0 \\ t s\,\mathrm{im}\, U(0,1] \\ x \square p_{t(\cdot \mid x_0)}}} \left[\sum_{x=y} w_{xy} \left(s_{\theta(x)_y} - \frac{p(y \mid x_0)}{p(x \mid x_0)} \log s_{\theta(x)_y} \right) \right] \tag{10}$$

5.3 Comparing against Auto-regressive Model

1. **Unconditional generation quality**
   We compare SEDD and GPT-2 based on their zeroshot perplexity on test datasets and based on the likelihood of generated text using a larger model (GPT-2 large, 770M parameters), similar to Savinov et al. (2022); Strudel et al. (2022); Dieleman et al. (2022). As observed by Lou et al. (2023), SEDD produces text with lower perplexity than GPT-2 without annealing. When sampling with 1024 steps, SEDD produces text of similar likelihood as annealed sampling from GPT-2. See tables 1 and 3 for more details.
2. **Conditional generation quality**
   We evaluate the conditional generation quality using automated metrics and the lm-eval-harness suite of Gao et al. (2021).
   Automated metrics We compare the quality of text generated by SEDD and GPT-2 using the MAUVE metric (Pillutla et al., 2021) and the perplexity of sampled continuations using GPT-2 large. We compute those metrics using prefixes and continuations of 50 tokens each. We extract 1000 prompts from the test set of OpenAI's webtext dataset. While Lou et al. (2023) suggests that SEDD medium matches GPT-2 in quality, we observed that SEDD achieves slightly lower MAUVE score. Reproducing the MAUVE results of Lou et al. (2023) was non-trivial, and we observed that generating more tokens achieves better quality than generating 50 only, even if we use 100 tokens to compute MAUVE. Indeed, sampling the 50 tokens of the continuation only resulted in poorer performance. For more details on this surprising result and exact MAUVE values, refer to appendix A.4.
   Downstream performance We compare the accuracy of SEDD and GPT-2 on downstream tasks using the evaluation

suite of Gao et al. (2021). The results are shown in table 2. We noticed the performance of SEDD and GPT-2 overall are very close1.

Diversity We evaluate diversity based on statistics computed on the training data and synthesized samples. See table 4 for all details. Surprisingly, the repetition rate of SEDD increases as the number of sampling steps increases. Similarly, the unigram entropy negatively correlates with the number of sampling steps. Importantly, SEDD appears less diverse than GPT-2 with and without annealing.

3. **Sampling speed**
   We compare the generation latency of SEDD and GPT-2 with KV-caching on a single NVIDIA A100SXM4-40GB GPU in fig. 1. See appendix A.3 for more details

Model	**params**	**LAMBADA**	**Wikitext2**	**PTB**	**WikiText103**	**1BW**
GPT-2 (small)	124M	**44.663**	39.576	129.977	39.576	**52.808**
SEDD (small)	170M	$\leq$ 51.146	**38.941**	**110.97**	**38.887**	$\leq$ 78.67
GPT-2 (medium)	255M	**35.879**	29.575	116.516	29.575	**43.627**
SEDD (medium)	424M	$\leq$ 42.385	**28.855**	**83.411**	**28.54**	$\leq$ 60.97
GPT-2 (large)	774M	33.969	26.664	139.427	26.664	44.783

Table 7: Zero-shot test perplexity

5.4 Industry Applications

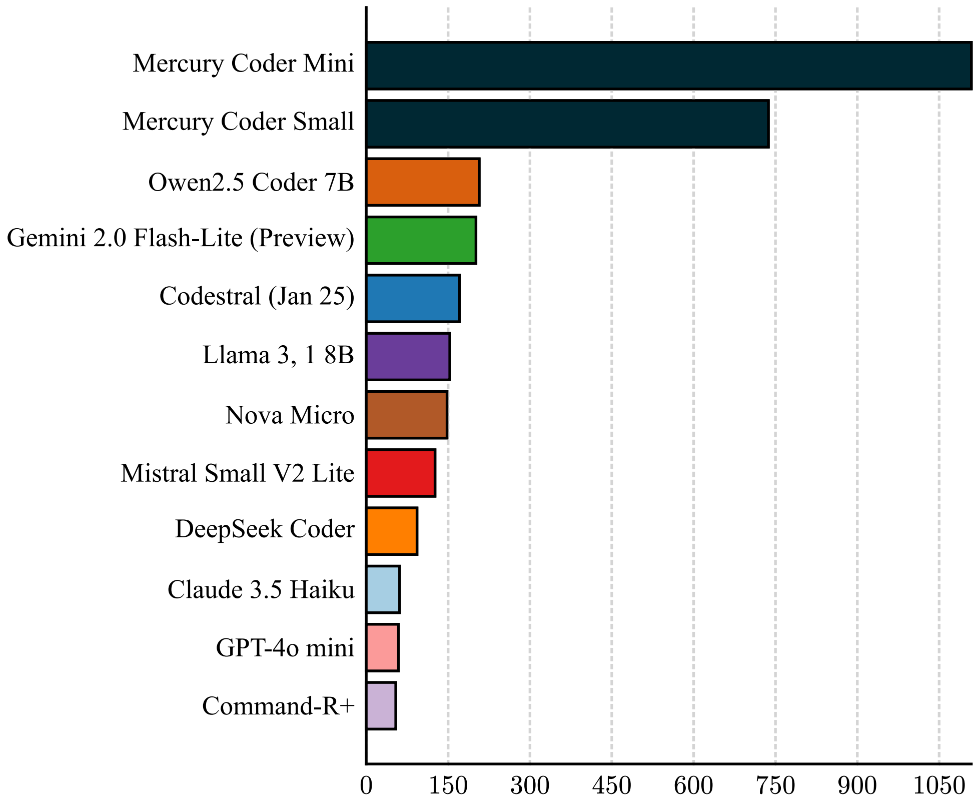


Figure 5: **Speed Comparison**: Output Tokens per Second

Mercury Coder which are first large scale discrete diffusion models are significantly faster than auto-regressive models.

A significant area of current development, discussed in Section 5, is the application of diffusion principles to discrete data such as text and code. While autoregressive models are established here, their sequential generation can be slow. Discrete diffusion techniques, including the promising Score Entropy method (Section 5.2), aim to overcome this limitation. As illustrated in Figure 5, recent discrete diffusion models like Mercury Coder demonstrate substantial improvements in generation speed (output tokens per second) for tasks like code generation compared to several autoregressive counterparts. This progress signifies the potential for diffusion-based approaches to offer faster and potentially more parallelizable alternatives in domains traditionally dominated by autoregressive generation, impacting future tools for software development, writing assistance, and more.

6 Conclusion

This review has examined two cornerstone generative techniques in modern deep learning: autoregressive models, largely enabled by the Transformer architecture, and diffusion models. We explored the foundational components of the vanilla Transformer, the principles of autoregressive sequence generation common in LLMs, and associated considerations like context length and data representation. Subsequently, the review detailed the iterative noising (forward) and learned denoising (backward) processes underpinning diffusion models, including loss parameterization.

In conclusion, both autoregressive and diffusion paradigms represent powerful approaches to generative modeling, each with distinct strengths and areas of active research. The successful application of diffusion to discrete data marks a significant development, opening new possibilities for efficient and high-quality generation beyond continuous domains. As these techniques continue to evolve and potentially converge, they will undoubtedly fuel further breakthroughs across a wide spectrum of AI-driven applications.

Bibliography

[1] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[3] L. Berlyand, P.-E. Jabin, and C. A. Safsten, "Stability for the training of deep neural networks and other classifiers," *Mathematical Models and Methods in Applied Sciences*, vol. 31, no. 11, pp. 2345–2390, 2021.

[4] A. Dubey *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[5] C. An *et al.*, "Why Does the Effective Context Length of LLMs Fall Short?," *arXiv preprint arXiv:2410.18745*, 2024.

[6] M. Ulčar and M. Robnik-Šikonja, "Training dataset and dictionary sizes matter in Bert models: the case of Baltic languages," in *International Conference on Analysis of Images, Social Networks and Texts*, 2021, pp. 162–172.

[7] A. Behrouz, P. Zhong, and V. Mirrokni, "Titans: Learning to memorize at test time," *arXiv preprint arXiv:2501.00663*, 2024.

[8] Q. Fournier, G. M. Caron, and D. Aloise, "A practical survey on faster and lighter transformers," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–40, 2023.

[9] S. Takase, R. Ri, S. Kiyono, and T. Kato, "Large Vocabulary Size Improves Large Language Models," *arXiv preprint arXiv:2406.16508*, 2024.

[10] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[11] L. Yang *et al.*, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[12] L. Weng, "What are diffusion models?," *lilianweng.github.io*, Jul. 2021, [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[13] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[14] N. Muennighoff *et al.*, "s1: Simple test-time scaling," *arXiv preprint arXiv:2501.19393*, 2025.

[15] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg, "Structured denoising diffusion models in discrete state-spaces," *Advances in neural information processing systems*, vol. 34, pp. 17981–17993, 2021.

[16] A. Campbell, J. Benton, V. De Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet, "A continuous time framework for discrete denoising models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28266–28279, 2022.

[17] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching.," *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[18] C. Meng, K. Choi, J. Song, and S. Ermon, "Concrete score matching: Generalized score matching for discrete data," *Advances in Neural Information Processing Systems*, vol. 35, pp. 34532–34545, 2022.

[19] A. Lou, C. Meng, and S. Ermon, "Discrete diffusion modeling by estimating the ratios of the data distribution," *arXiv preprint arXiv:2310.16834*, 2023.

[20] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” *arXiv preprint arXiv:2307.08691*, 2023.

[21] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, “Long-context llms struggle with long in-context learning,” *arXiv preprint arXiv:2404.02060*, 2024.

[22] A. Hyvärinen, “Some extensions of score matching,” *Computational statistics & data analysis*, vol. 51, no. 5, pp. 2499–2512, 2007.