CSC 412:
Probabilistic Learning and Reasoning
Week 12: Speculative Decoding & Diffusion Models

Denys Linkov

University of Toronto

Sampling for LLMs Part 2

- LLM Sampling
  - Learned about general LLM sampling (greedy, beam, top-k, top-p)
  - We applied this to a constrained decoding problem of limiting our tokens
- Sampling from before
  - Rejection sampling
  - Using another distribution
  - Can we use simpler models to generate some tokens?

LLM Inference

- LLM inference is an important task, we can write it as trying to return the most likely token(s) given some inputs. $p(y|x, \theta)$
- We have a dependence on the previous token in an autoregressive model $p(x_t|x_{<t})$
- "Unfortunately, a single decode step from these larger models is significantly slower than a step from their smaller counterparts, and making things worse, these steps are done serially - decoding K tokens takes K serial runs of the model." (Leviathan, 2022)

Speculative Decoding

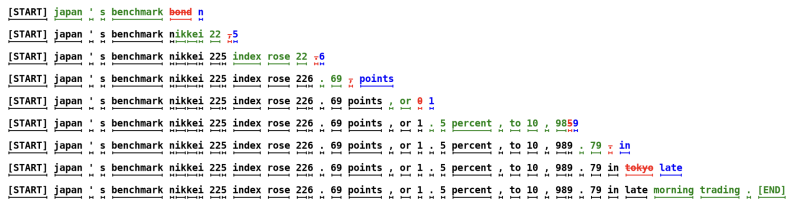
- Some tokens are easier to generate then otherwise
- Two parts in LLM generation, Fill the KV cache and Decode to generate Tokens
- Idea: Use a cheaper model to sample from when generating tokens
- Example: Output your favourite food with the following format "I like to eat"
- Accept or reject those tokens based on some probabilities
- Similar approach to rejection sampling based on a $q(x)$ and $p(x)$

LLM Inference

"Unfortunately, a single decode step from these larger models is significantly slower than a step from their smaller counterparts, and making things worse, these steps are done serially - decoding K tokens takes K serial runs of the model."

Speculative Decoding - (Leviathan, 2022)

- Define x as a token vector, M_p as the model and $p(x)$ as the distribution we are trying to sample from
- M_q and $q(x)$ are from the simpler model.
- If we are using a GPT style model, we are trying to predict $p(x_t|x_{<t})$
- Problem: Sequential generation is expensive with $p(x)$
- Idea: Generate γ tokens from M_q, use M_p to accept/reject
- Idea: Evaluate candidates in parrallel



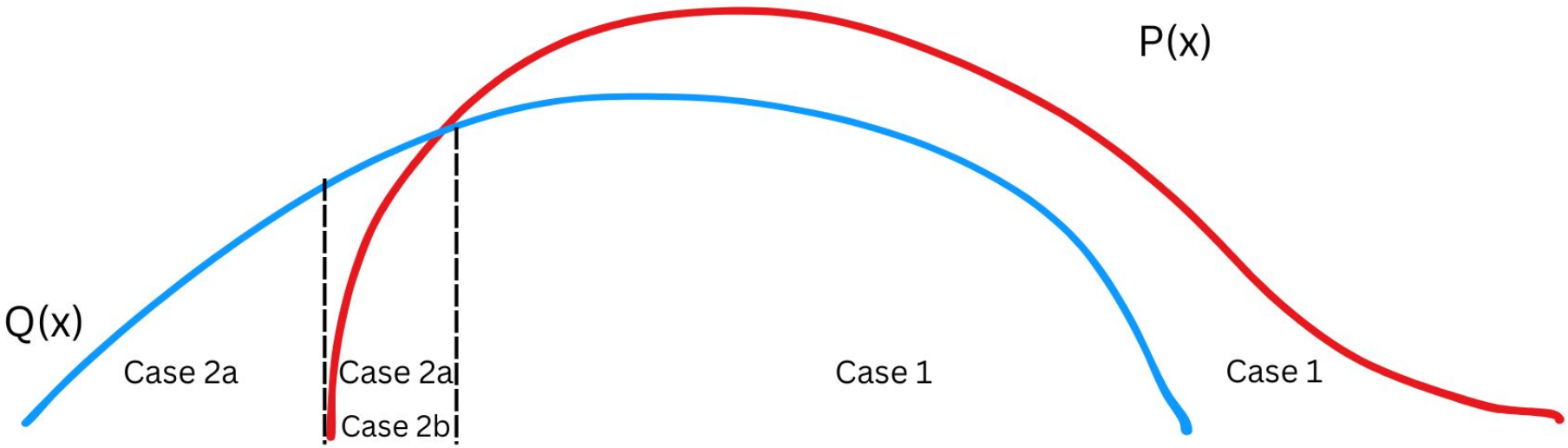
Speculative Decoding - (Leviathan, 2022)

- Use the more efficient model M_q to generate γ completions
- Use the target model M_p to evaluate all of the guesses and their respective probabilities from M_q in parallel, accepting all those that can lead to an identical distribution
- Sampling an additional token from an adjusted distribution to fix the first one that was rejected, or to add an additional one if they are all accepted.

Speculative Decoding - Sampling Approach

- Idea: use probabilities in both models to accept/reject.
- Sample $x \sim q(x)$
- Case 1 - $q(x) \leq p(x)$ - Keep
- Case 2a - $q(x) > p(x)$ - Keep with probability $\frac{p(x)}{q(x)}$
- Case 2b - $q(x) > p(x)$ - Sample $x \sim norm(max(0, p(x) - q(x)))$
- Equivalent to $p(x)$

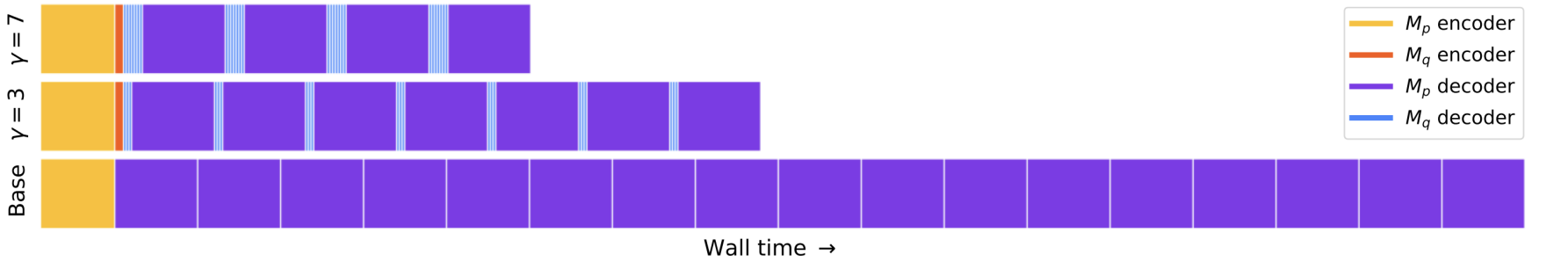
Speculative Decoding - Visual of the cases



Speculative Decoding - Sequence

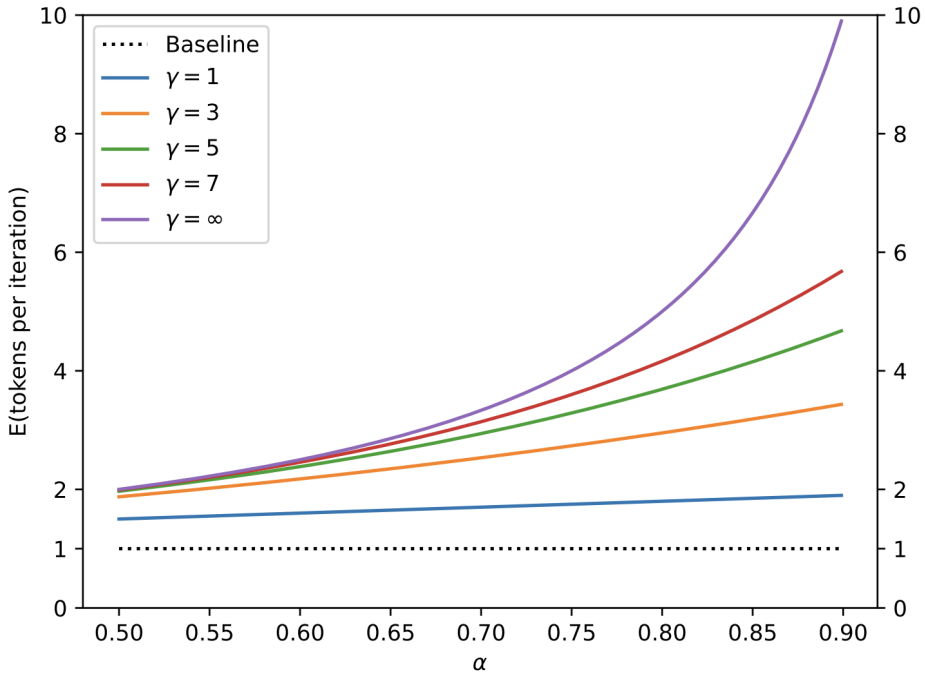
Token	X1	X2	X3	X4
	Japan	‘	s	benchmark
q(x)	0.3	0.1	0.5	0.2
p(x)	0.4	0.2	0.3	0.1

Accept Accept Reject 40% of the time Reject 50% of the time



Speculative Decoding - Expected Token generation

- $E(T) = \frac{1-\alpha^{\gamma+1}}{1-\alpha}$ where T is the number of generated tokens
- $D_{LK}(p,q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|$
- where $M(x) = \frac{p(x)+q(x)}{2}$.
- $D_{LK}(p,q) = 1 - \sum_x \min(p(x), q(x))$
- $\alpha = 1 - E(DLK(p,q)) = E(min(p,q))$



Speculative Decoding - Sampling Algorithm

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.

▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$

$M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$

▷ Determine the number of accepted guesses n.

$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

$p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$

end if

▷ Return one token from M_p, and n tokens from M_q.

$t \sim p'(x)$

return $prefix + [x_1, \ldots, x_n, t]$

Speculative Decoding - Review paper 2024

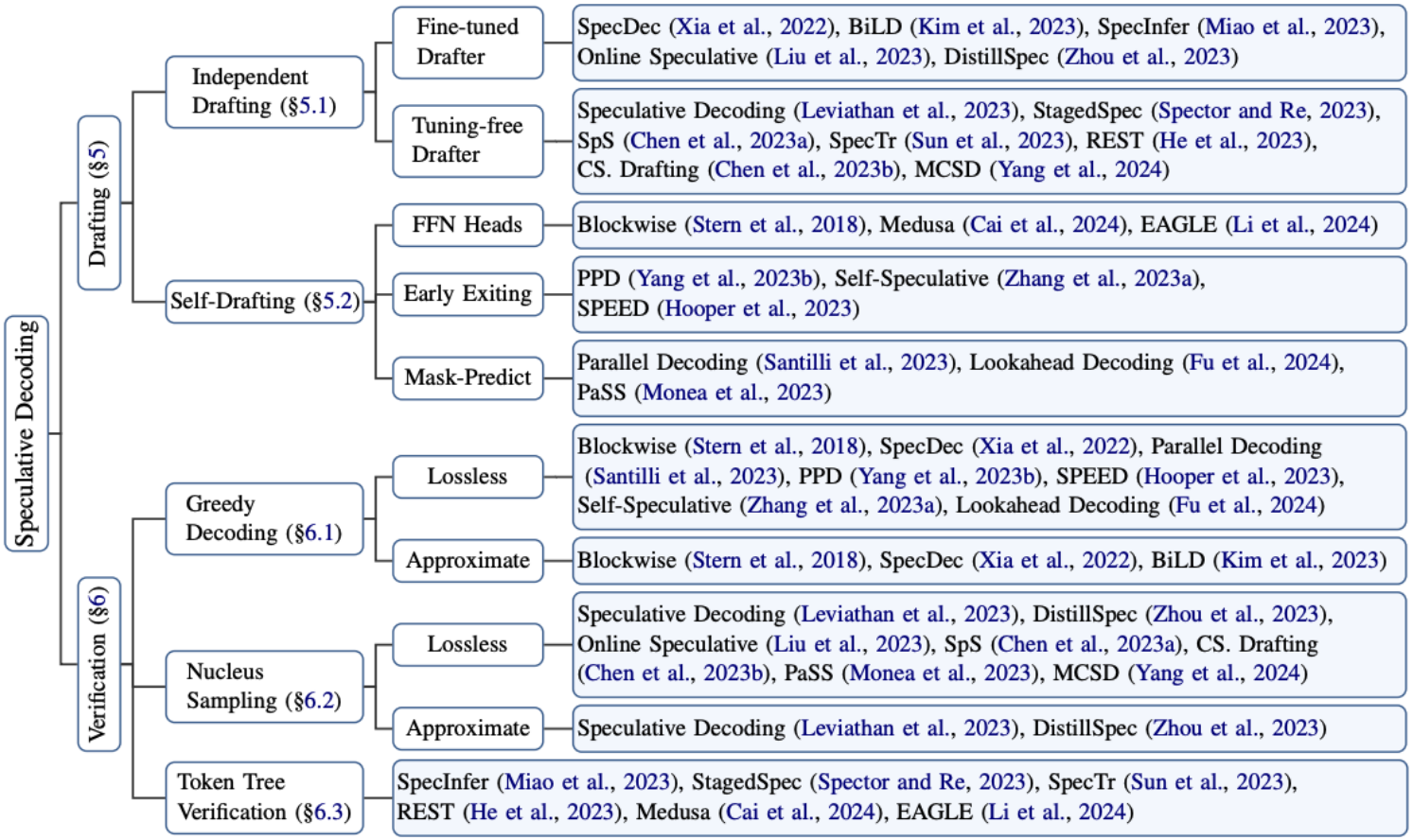
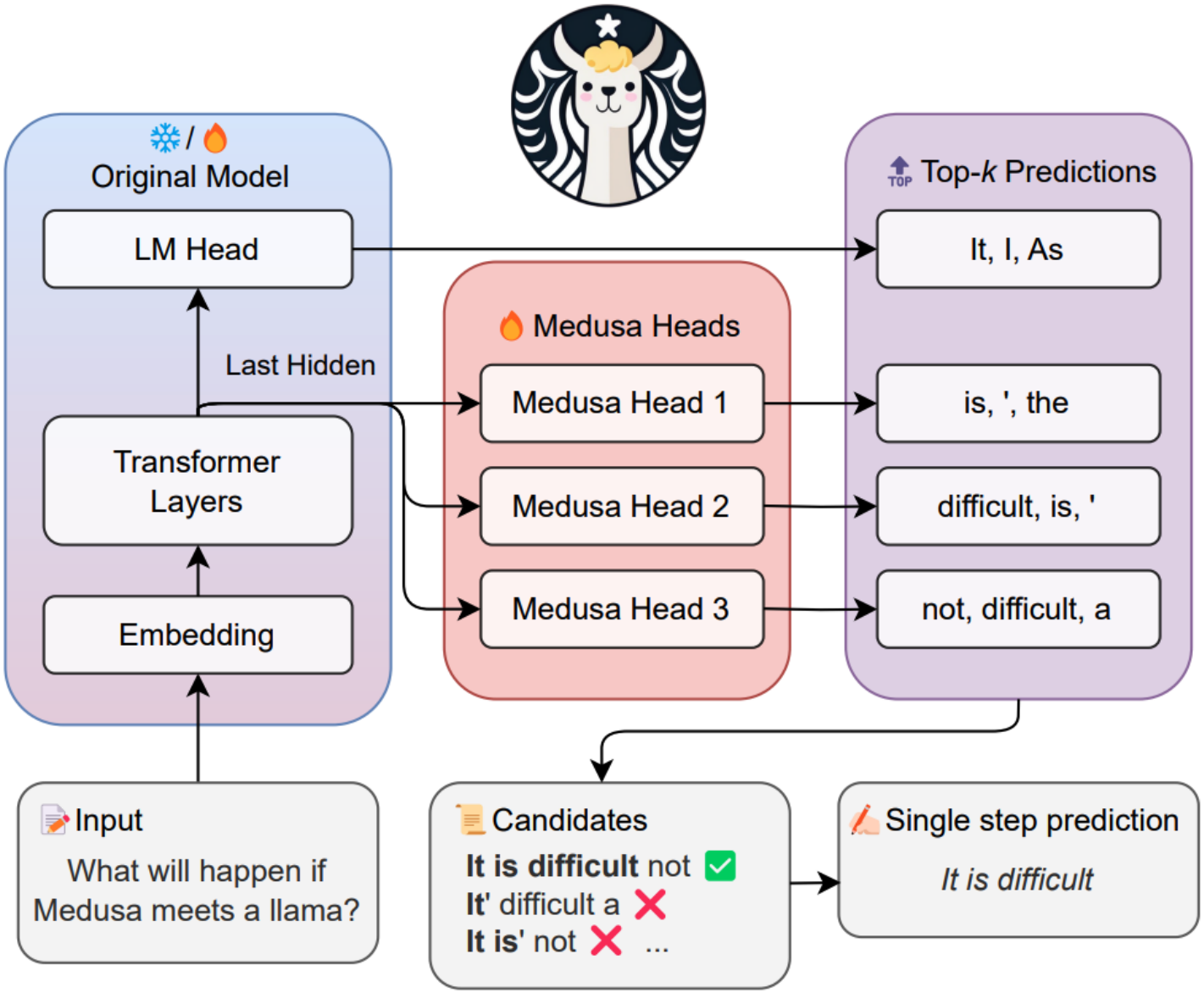


Figure 3: Taxonomy of Speculative Decoding.

Figure: Review paper by (Xia,2024)

Speculative Decoding - Multiple Inference Heads



Speculative Decoding - Multiple Inference Heads

- Choose a base model, it would predict the next token $p(x_{t+1}, x< t)$
- Add K decoding heads using the last hidden states h_t at position t,
- The k-th head is used to predict the token in the $(t + k + 1)$
- The prediction of the k-th head is denoted as $p_t^{(k)}$ the original model is denoted as $p_t^{(0)}$
- Use tree attention to sample multiple heads concurrently
- Accept with an entropy function

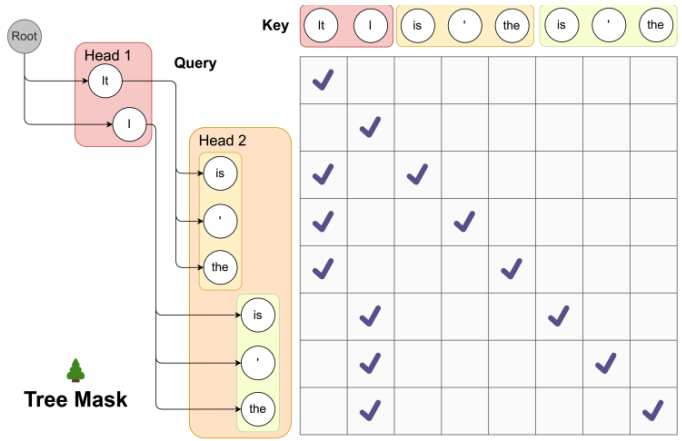


Figure: Tree Attention (Xia,2024)

Tree Attention

- Use tree attention to sample multiple heads concurrently
- "As exemplified, the top-2 predictions from the first Medusa head and the top-3 from the second result in 2*3=6 candidates. Each of these candidates corresponds to a distinct branch within the tree structure. To guarantee that each token only accesses its predecessors, we devise an attention mask that exclusively permits attention flow from the current token back to its antecedent tokens. The positional indices for positional encoding are adjusted in line with this structure."

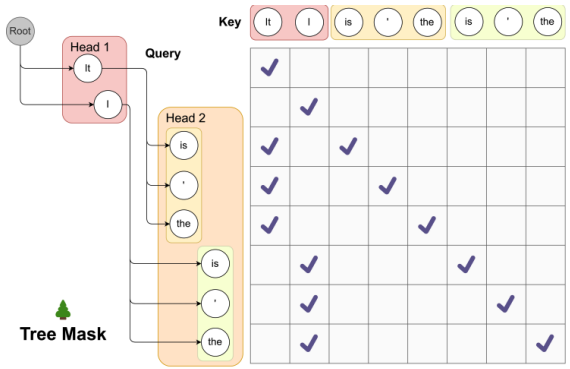


Figure: Tree Attention (Xia,2024)

Typical Acceptance Sampling

- Rather than mimicing the distribution, test empirically
- Use an entropy function with a delta-epsilon approach.
- ϵ sampling (Hewitt, 2022) allows us to sample from values above an absolute probability, working with top-k and top-p sampling
- $P_{\text{original}}(x_{n+k}|x_1, x_2, \ldots, x_{n+k-1}) > \min(\epsilon, \delta \exp(-H(P_{\text{original}}(x_1, x_2, \ldots, x_{n+k-1}))))$
- where $H(\cdot)$ denotes the entropy function, and ϵ, δ are the hard threshold and the entropy-dependent threshold respectively. This criterion is adapted from Hewitt et al. (2022).
- Lots of other interesting ideas when exploring entropy sampling

Speculative Decoding Overview

- We can use smaller models to help predict easier tokens
- Use rejection sampling or empircally show new distribution works well
- Many other techniques to draft and verify
- Lots of other ideas to explore in the entropy space

Diffusion Time

These slides are based on:

- CVPR 2022 Tutorial: Denoising Diffusion-based Generative Modeling: Foundations and Applications, by Kreis, Gao, and Vahdat
- Lilian Weng's blogpost: What are diffusion models?

Generative Modeling

Common methods:

- Variational Autoencoders
- Generative Adversarial Networks (GAN)
- Flow-based models
- Today: Diffusion Models

Diffusion Models: Text-to-Image Generation and More

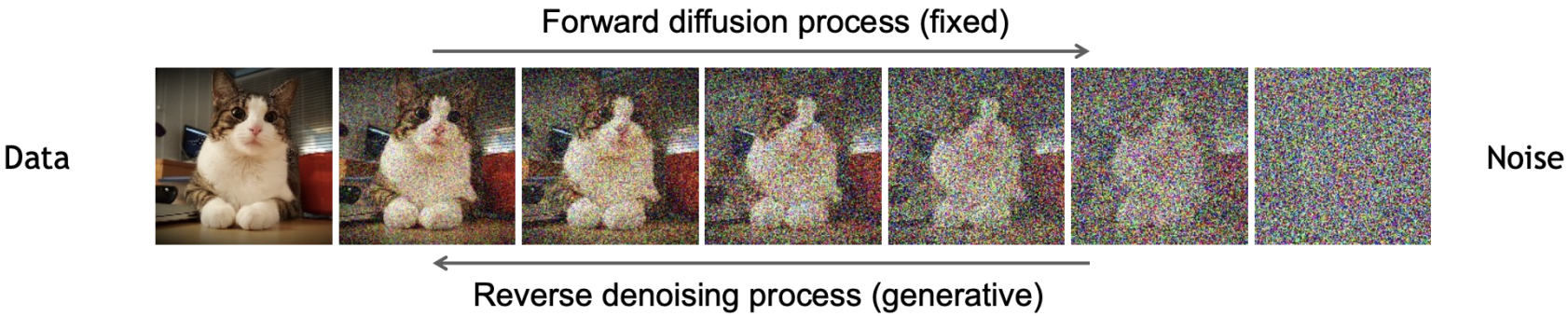
DALL·E 3, prompt: Tiny potato kings wearing majestic crowns, sitting on thrones, overseeing their vast potato kingdom filled with potato subjects and potato castles.

Stable Diffusion 3, prompt: Frog sitting in a 1950s diner wearing a leather jacket and a top hat. On the table is a giant burger and a small sign that says “froggy fridays”.

Diffusion Models

Diffusion models use two processes:

- A *forward process*, start from image and keep adding noise.
- A *reverse process*, start from noise and keep *denoising* it to recover an image.



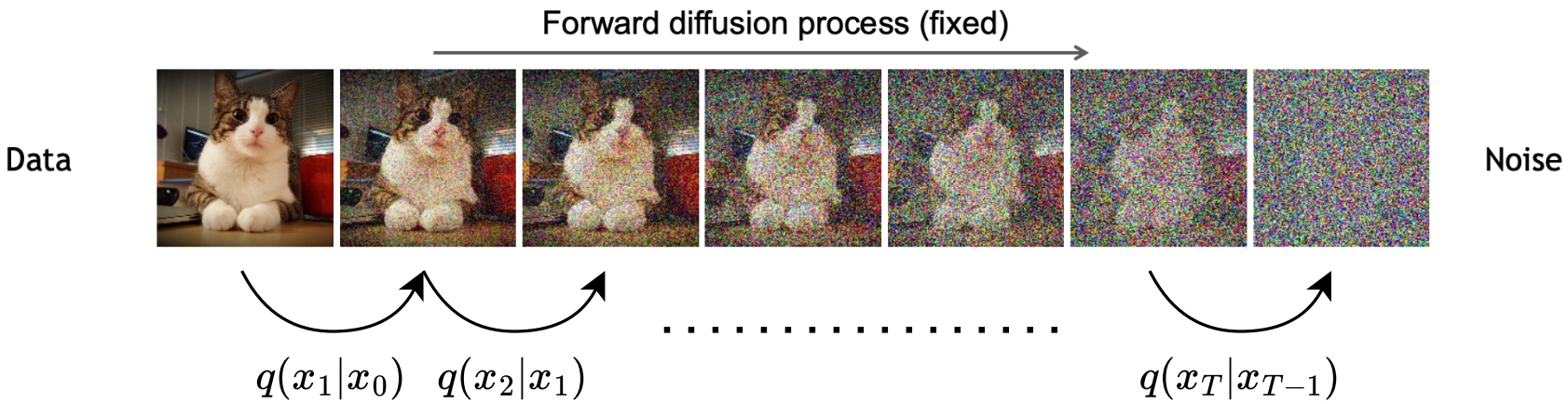
Forward Process

- The forward process is a Markov chain: $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$
- Each step adds Gaussian noise:

$$q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I),$$

Or equivalently,

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}, \quad \epsilon_{t-1} \sim \mathcal{N}(0, I).$$



Forward Process

Let $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i$.

$$\begin{aligned} x_t &= \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\ &\overset{(d)}{=} \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \end{aligned}$$

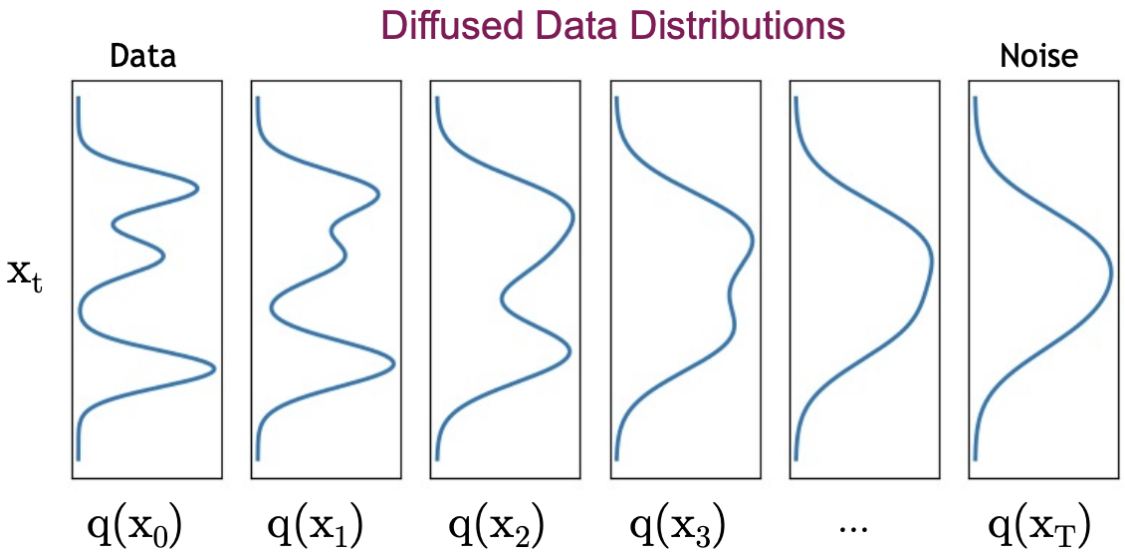
Therefore

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

Forward Process and White Noise

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

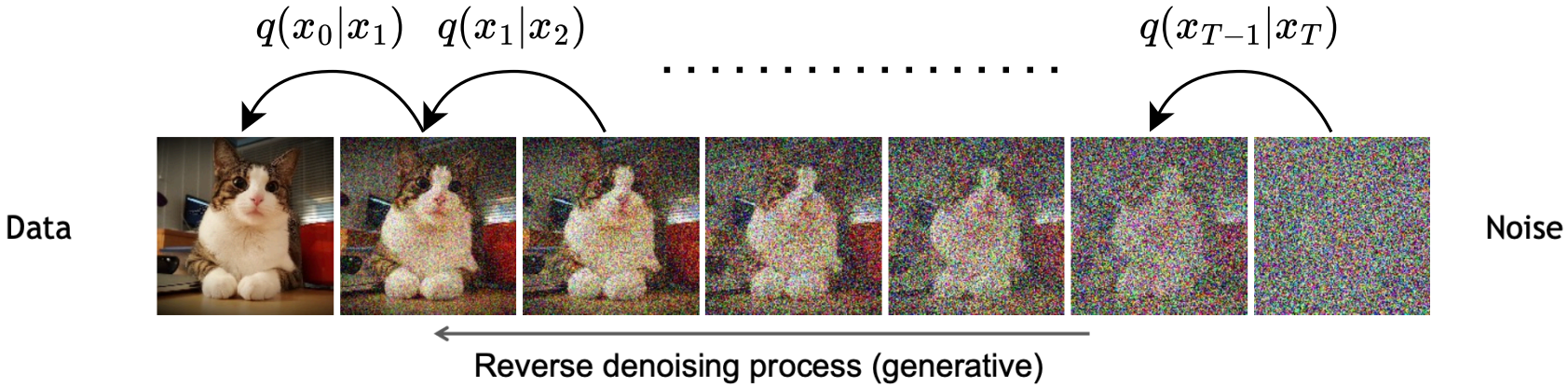
$(\beta_t)_{t=1}^T$ is chosen such that $\bar{\alpha}_T \to 0$, thus x_T converges to a standard normal random vector.



Connection to VAEs

Reverse Process

- Ideally, to generate a sample:
  1. $x_T \sim \mathcal{N}(0, I) \approx q(X_T)$.
  2. $x_{t-1} \sim q(x_{t-1}|x_t)$ for $t = T, \ldots, 1$.
- But $q(x_{t-1}|x_t)$ is intractable.



Reverse Process

- $q(x_{t-1}|x_t)$ is approximately normal if β_t is small.
- We approximate it with

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_t^2).$$

- μ_θ comes from a trainable architecture (e.g. neural network) with parameter θ.
- For the reverse process

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t).$$

- To make $p_\theta(x_{1:T}|x_0)$ close to $q(x_{1:T}|x_0)$, we will use ideas from Variational Inference.

Evidence Lower Bound: ELBO

- Recall the ELBO:

$$KL(q(x_{1:T}|x_0)||p_\theta(x_{1:T}|x_0)) + \mathbb{E}_{x_{1:T}\sim q}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] = \log p_\theta(x_0)$$

$$\implies \text{Minimize} \quad \mathbb{E}_{x_{0:T}\sim q}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] =: L.$$

- Our goal becomes minimizing L.

Variational Upper Bound

- While expressing $q(x_{t-1}|x_t)$ is difficult. By Bayes rule, expressing $q(x_{t-1}|x_t, x_0)$ is easy:

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

$$\implies q(x_t|x_{t-1}) = \frac{q(x_t|x_0)q(x_{t-1}|x_t, x_0)}{q(x_{t-1}|x_0)}. \qquad (*)$$

Variational Upper Bound

$$
\begin{aligned}
L &= \mathbb{E}_{x_{0:T}\sim q}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[\log \frac{\prod_{t=1}^{T} q(x_t|x_{t-1})}{p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[\log \frac{q(x_T|x_0)\prod_{t=2}^{T} q(x_{t-1}|x_t,x_0)}{p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}\right] \quad \text{By } (*) \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[\underbrace{KL(q(x_T|x_0)||p_\theta(x_T))}_{l_T} + \sum_{t=2}^{T}\underbrace{KL(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))}_{l_{t-1}}\right. \\
&\quad \left. + \underbrace{-\log p_\theta(x_0|x_1)}_{l_0}\right]
\end{aligned}
$$

Variational Upper Bound

- Let $L_t = \mathbb{E}_q[l_t]$ for $t = 0, \ldots, T$.
- L_T is constant because x_T is just the standard normal random vector.
- To compute L_t for $t = 1, \ldots, T-1$, we first show $q(x_{t-1}|x_t, x_0)$ is Gaussian.

$$
\begin{aligned}
q(x_{t-1}|x_t, x_0) &\propto q(x_t|x_{t-1})q(x_{t-1}|x_0) \\
&\propto \exp\left(-\frac{\|x_t - \sqrt{\alpha_t}x_{t-1}\|^2}{2\beta_t} - \frac{\|x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}x_0\|^2}{2(1-\bar{\alpha}_{t-1})}\right) \\
&\propto \exp\left(-\frac{\|x_{t-1} - \tilde{\mu}(x_t, x_0)\|^2}{2\tilde{\beta}_t}\right),
\end{aligned}
$$

Variational Upper Bound

- Therefore, $q(x_{t-1}|x_t, x_0) = \mathcal{N}(\tilde{\mu}(x_t, x_0), \tilde{\beta}_t I)$.
- Basic algebra shows

$$\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t.$$

- Computing KL between two Gaussians is straightforward:

$$\begin{aligned} L_{t-1} &= \mathbb{E}_q \left[KL(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) \right] \\ &= \mathbb{E}_q \left[\frac{\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|^2}{2\sigma_t^2} \right] + \text{ const.} \end{aligned}$$

Parameterizing the Mean

- Recall $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$.
- By plugging in x_0 in terms of x_t and ϵ:

$$\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right).$$

- As a result, we parameterize $\mu_\theta(x_t)$ to try to predict the noise using a neural network $\epsilon_\theta(x_t, t)$,

$$\mu_\theta(x_t, t) := \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$$

Training Objective

- The loss thus becomes

$$L_{t-1} = \underbrace{\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\alpha_t)}}_{\lambda_t} \mathbb{E}_{x_0\sim q,\epsilon\sim\mathcal{N}(0,I)}\left[\|\epsilon_\theta(x_t,t)-\epsilon\|^2\right],$$

  where $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$.
- Ho et al. [2020] observed that the performance improves if we simply choose $\lambda_t = 1$.
- The simplified loss is thus

$$L_{t-1}^{\text{simple}} = \mathbb{E}_{x_0\sim q,\epsilon\sim\mathcal{N}(0,I)}\left[\|\epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)-\epsilon\|^2\right].$$

Test-Time Sample Generation

- Start from $x_T \sim \mathcal{N}(0, I)$.
- For $t = T, \ldots, 1$, sample $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$
  - Recall $p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_t^2)$.
  - As a result

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z, \quad z \sim \mathcal{N}(0, I).$$

Summary

Algorithm 1 Training

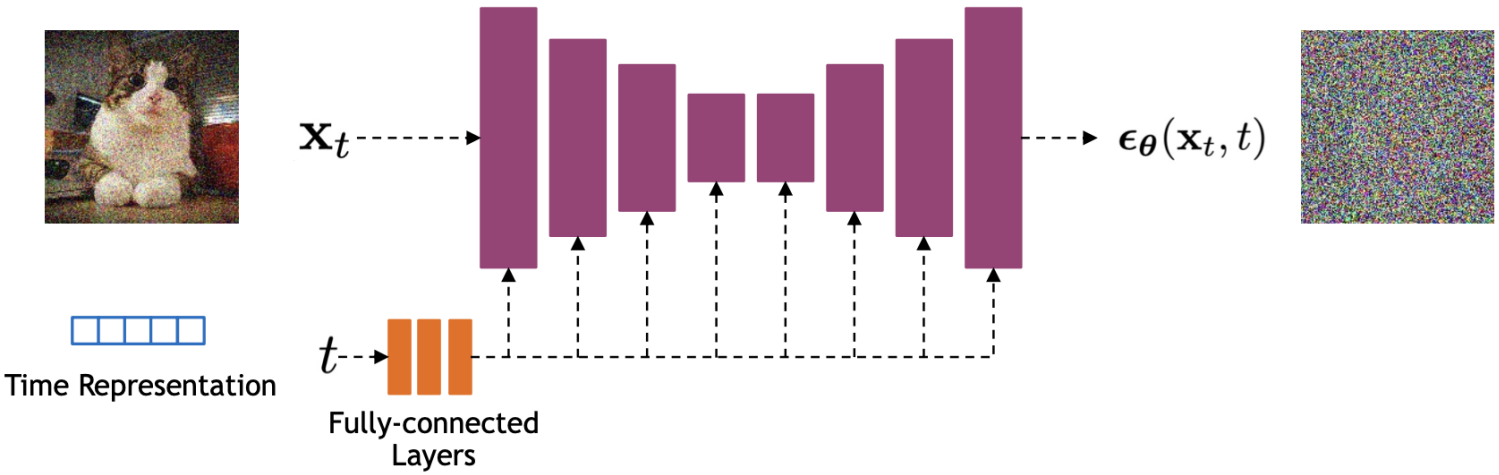
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Design Choices: Architecture

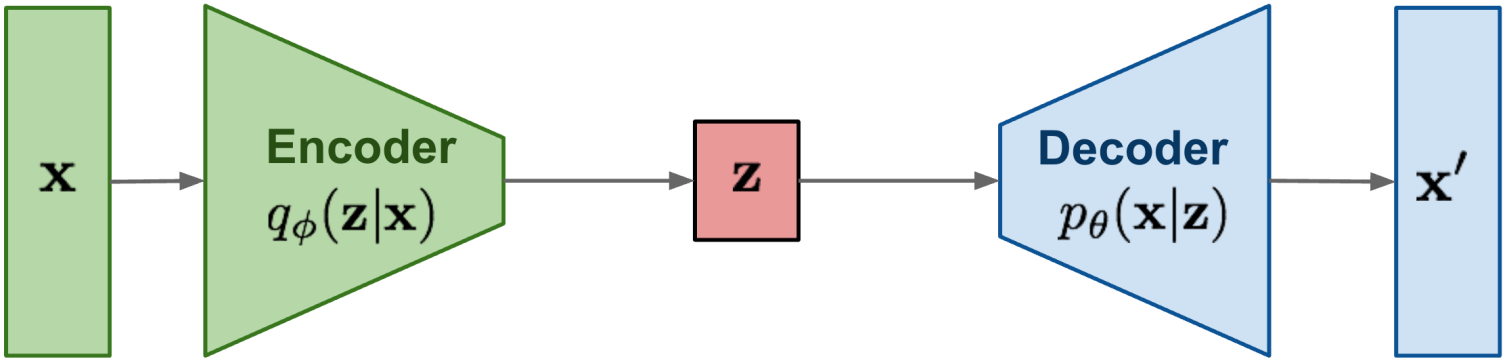
- $\epsilon_\theta(x_t, t)$ is implemented using a **U-Net** architecture, with **residual blocks** and **self-attention** layers.
- Weights are shared across time.



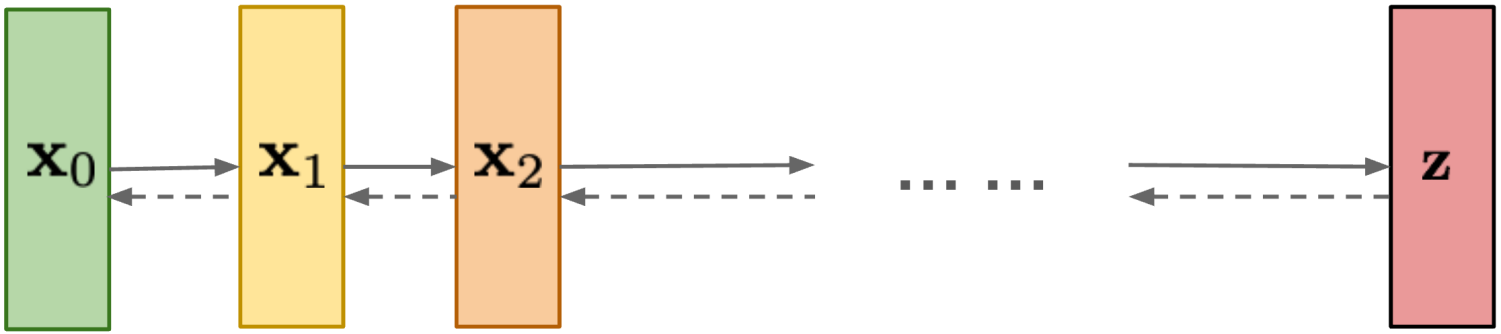
Design Choices: Hyperparameters

- β_t and σ_t control the variance of the forward and reverse process respectively.
- β_t is chosen linearly between $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$.
- $\sigma_t^2 = \beta_t$. We could instead consider a trainable full covariance matrix, i.e. $\Sigma_\theta(x_t, t)$.
- $T = 1000$ steps are taken.
- Fancier β_t schedules can further reduce loss [Nichol and Dhariwal, 2021].

Comparison with Variational Autoencoders (VAE)



VAE



Diffusion Model

Comparison with VAEs

- Diffusion models and VAEs both map to isotropic Gaussian.
- The latent space has the same dimension as the input space in DMs. In VAEs, it is smaller dimensional.
- The forward process is the encoder, which is **fixed**. This is trained in VAEs.
- The reverse process is the decoder, which is **trained**, similar to the VAEs.

Conditional Generation

- The original examples we saw were images generated conditioned on a text caption. More examples:

Midjourney, prompt: A close-up profile of a cute green-eyed kitten with a black nose and light cheeks sitting on top of a wooden floor under bright daylight.

DALL·E 3, prompt: Illustration of a chic chair with a design reminiscent of a pumpkin's form, with deep orange cushioning, in a stylish loft setting.

- But the diffusion model we learned about can only generate unconditioned images.

Conditional Generation: General Formulation

- Suppose we want to condition on y (e.g. class label or describing caption).
- The training data are pairs of (x_0, y).
- Conditional reverse process:

$$p_\theta(x_{0:T}|y) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t, y)$$

- We still model the transition probabilities as Gaussian:

$$p_\theta(x_{t-1}|x_t, y) = \mathcal{N}(\mu_\theta(x_t, t, c), \Sigma_\theta(x_t, t, y)).$$

Conditional Generation

- The new loss:

$$L = \mathbb{E}_q \left[l_T + \sum_{t \geq 2} KL(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t, y)) - \log p_\theta(x_0|x_1, y) \right]$$

- How to incorporate y into the U-Net architecture?
  - Different techniques for different types of conditioning (labels, images, captions, ...)

Classifier Guidance

- To further strengthen conditioning, we can train a classifier $p_\phi(y|x)$ and incorporate its log-gradient into score with a scale s.
- [Nichol and Dhariwal, 2021]

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Classifier-Free Guidance

- Instead of training a separate classifier, simultaneously train a conditional and an unconditional diffusion model.
- We have an implicit classifier by Bayes rule,

$$p(y|x_t) \propto_y \frac{p(x_t|y)}{p(x_t)}.$$

- We can simply use

$$\nabla_{x_t} \log p(y|x_t) = \nabla_{x_t} \log p(x_t|y) - \nabla_{x_t} \log p(x_t).$$

Tradeoff: Sample Quality vs Diversity

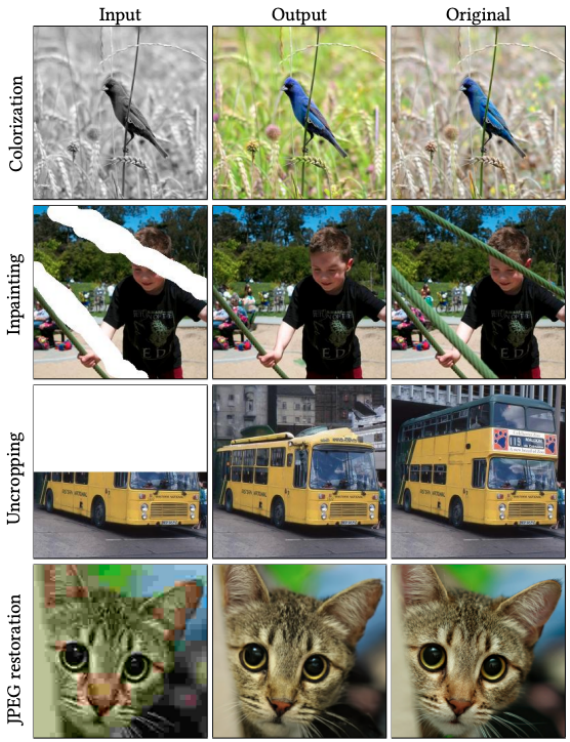
s = 0

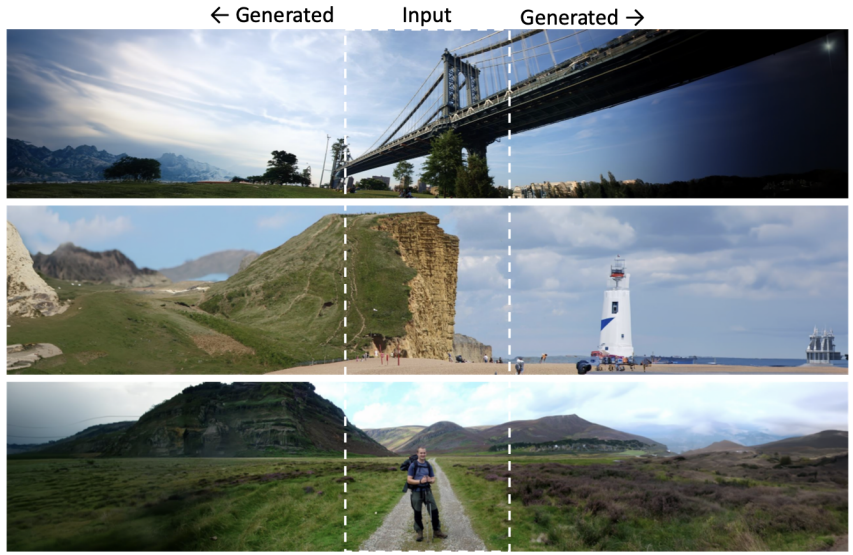
s = 1

s = 3

Conditioning Applications

Conditioning is not always on captions.

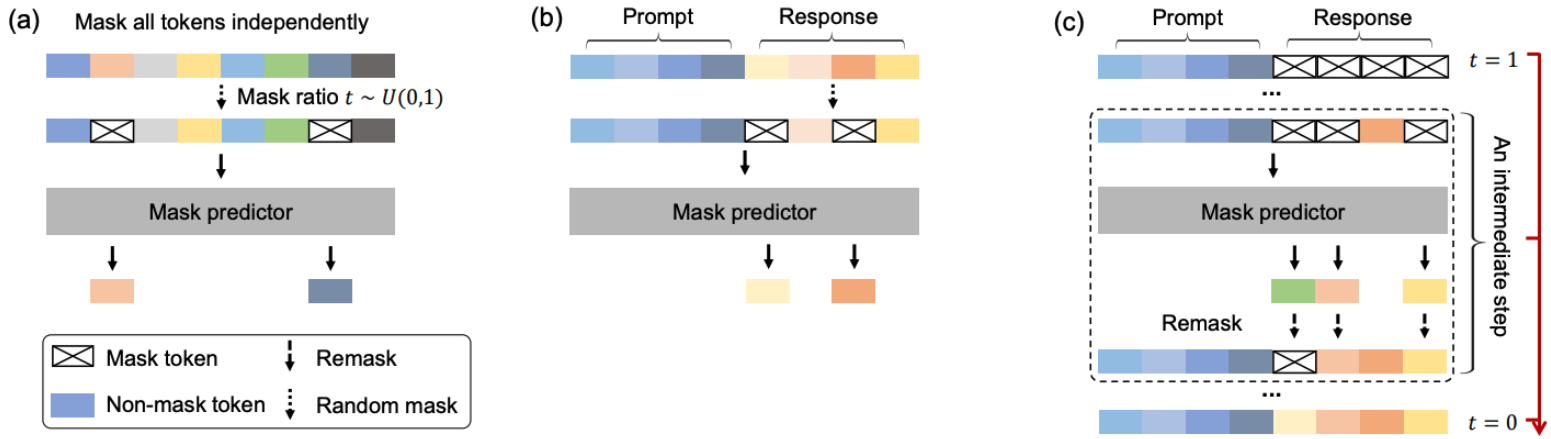




[Saharia et al., 2022]

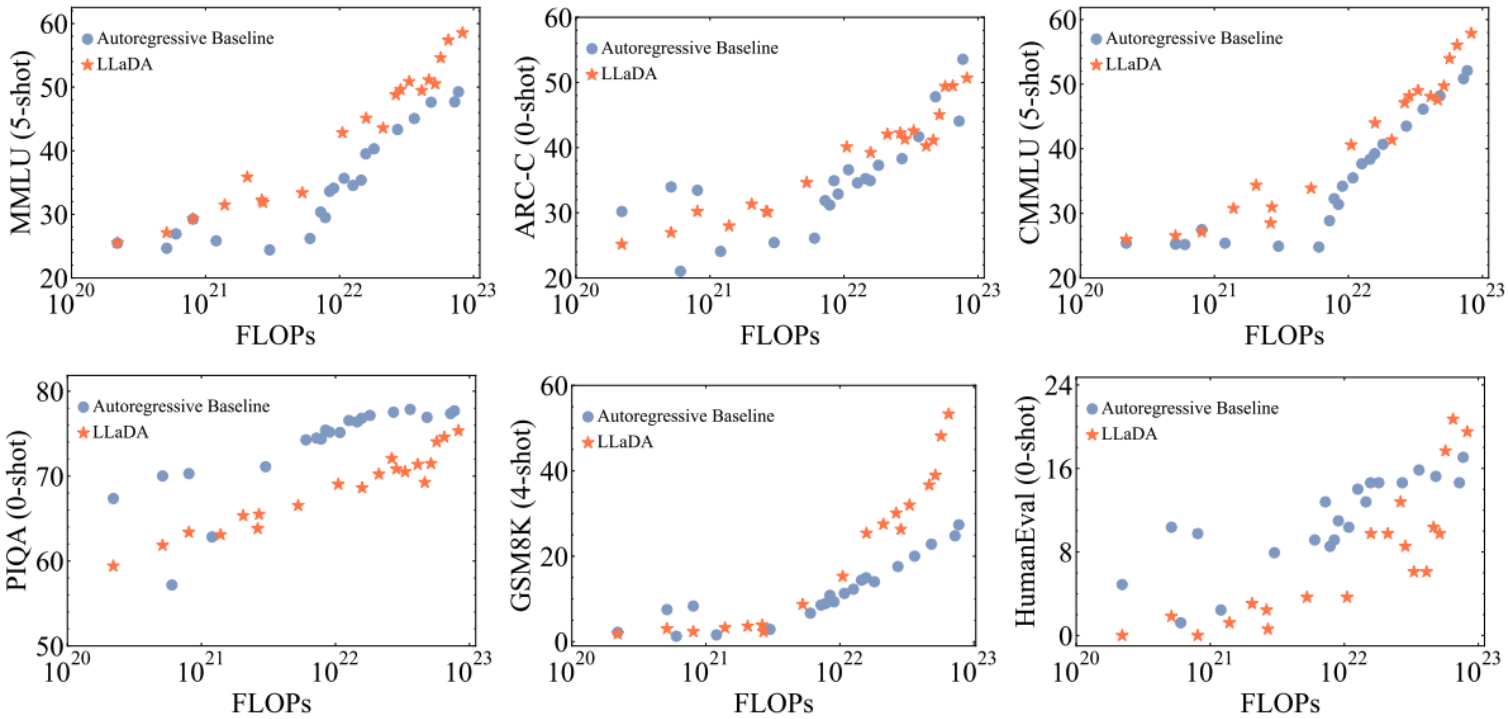
Diffusion Language Models

- What if we used diffusion for language objectives - Try to predict the answer all at once
- LLaDa model (Nie, 2025)
- For a training sequence x_0, we randomly sample $t \in [0, 1]$,mask each token independently with the same probability t to obtain x_t
- Estimate Eq. (3) via the Monte Carlo method for stochastic gradient descent training.

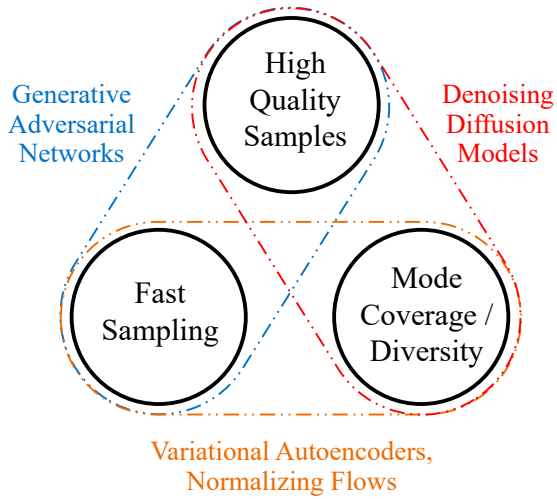


Diffusion Language Models

- Will cover equations more in tutorial, but results are promising



Tradeoffs in Generative Modeling



[Xiao et al., 2021]

- Accelerating diffusion models can overcome the above trilemma.

Summary

- Diffusion Models: Forward and Reverse Processes
- Training via Variational Upper Bounds
- Conditional Generation
- Tradeoffs

Topics for Further Reading

- Denoising Score Matching [Song et al., 2021]
- Probability Flow ODE [Song et al., 2021]

References I

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

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–10, 2022.

Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in neural information processing systems*, 34:1415–1428, 2021.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021.