1.1 Markov Decision Processes: a) Basic Concepts: *1, State Space(s)*: A finite set of states. *2, Action Space (A)*: A set of actions. *3, Transition Probability*: $P(s'|s,a)$. *4, Immediate Reward Function* $r(s,a,s')$: Defines the reward received after transitioning from state s to state s' due to action a. *5, Expected Immediate Reward:* $r(s,a) = \sum_{s'} r(s,a,s')P(s'|s,a)$: The expected reward for taking action a in state s, calculated as the sum of the rewards for each possible next state s', weighted by the probability of transitioning to s'. *6, Policy:* $a = \pi(s)$: A strategy or a rule that specifies which action a to take in each state s. *7. Discounted Total Reward under Policy* π: $R^\pi(s_0) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$ *8, Value Function of Policy* π: $V^\pi(s) = E[R^\pi(s)]$: The expected total reward for following policy π from a given state s. It quantifies the long-term benefit of being in state s under policy π. **b) Bellman Func** Pick $Q_0(s,a)$, k = 0; Repeat: $Q_{k+1}(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a)\max_{a'} Q_k(s',a')$; k = k+1; until $\max_s |\max_a Q_{k+1}(s,a) - \max_a Q_k(s,a)| \le \epsilon$; Also Got: $V^*(s) = \max_a\{r(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')\}$;$\pi^*(s) = \text{argmax}_a Q^*(s,a)$;$V^*(s) = \max_a Q^*(s,a)$**1.2 RL** Not have $P(s'|s,a)$and $r(s,a)$ **a) Q-Learning**: Take action a, observe r and s' , $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$; Exploitation: Explore new parts of state space so as to gain more experiences; Exploitation: maximize reward. ϵ-greedy policy: chose an action at random with prob of ϵ. on-policy: the agent learns the value of the policy that it's currently using to make decisions; Off-policy: learning the value of one policy, called the target policy, while following another policy. **b) Sarsa**: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$, where a' are chosen from ϵ-greedy policy. **1.3 DQN a) loss** $L(\theta) = \left([r(s,a) + \gamma \max_{a'} Q(s',a';\theta-)] - Q(s,a;\theta)\right)^2$, addressing the moving target problem with a separate target network θ^- updated every C steps to stabilize training. **b) Replay** enhances data efficiency by reusing each experience tuple multiple times, accelerates convergence through updates from diverse experiences, breaks correlation in sequential updates to reduce variance, and prevents oscillations or divergence in parameters. **c) Double DQN** $\theta \leftarrow \theta - \alpha\nabla_\theta\left([r(s,a) + \gamma Q(s',a^*;\theta^-)] - Q(s,a;\theta)\right)^2$ where $a^* = \text{argmax}_{a'} Q(s',a';\theta)$, Deals with the Overestimation Bias. **1.4 Policy Gradients a) basic concepts** deterministic policy: policy is deterministic in predicting; stochastic policy: the policy is a network, which gives the prob for actions. **b) REINFORCE algorithm**: 1, Sample a set of trajectories $\{\tau_i\}_{i=1}^N$ from $\pi_\theta(a_t|s_t)$ (run the current policy). 2, Estimate the gradient of the objective function $J(\theta)$ as follows:$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \sum_t \nabla_\theta \log\pi_\theta(a_t^i|s_t^i) r(\tau_i)$, 3, Update the policy parameters θ by taking a step in the direction of the gradient: $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$ **c) Physical Meaning** Optimization target $J(\theta)$: the expected total reward when following policy π_θ; $r(\tau)$ is the reward signal for a trajectory τ, $r(\tau) > 0$ means beneficial trajectory; The term $\nabla_\theta \log\pi_\theta(a_t^i|s_t^i)$ represents the gradient of the log-probability of taking action a_t^i in state s_t^i under the policy π_θ This gradient points in the direction of increasing the log-probability of the action. **d) Others** It is on-policy. And it got the issues of high variance: Policy gradients have high variance because they depend on limited sample trajectories, which are costly to obtain. This can cause unstable learning and slow convergence. **1.5 Actor-Critic a) Basic Concepts** Actor-Critic algorithms enhance policy gradients by using an advantage function $A^\pi(s_t,a_t)$ which indicates the relative quality of an action compared to the average. The policy gradient is estimated using:$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N \sum_{t=0}^T \gamma^t \nabla_\theta \log\pi_\theta(a_t^i|s_t^i)A^\pi(s_t^i,a_t^i)$. This method increases the probability of better-than-average actions and decreases the probability of worse-than-average actions. There are two networks in it: ϕ-teacher network; π-policy network **b) Variants** Online AC: 1, Take action $a \sim \pi_\theta(a|s)$, receive (s,a,s',r). 2, Update Critic: Minimize L2 loss with $\{\left(s, r + \gamma\hat{V}_\phi(s')\right)\}$. 3, Compute advantage: $\hat{A}^\pi(s,a) \leftarrow r + \gamma\hat{V}_\phi(s') - \hat{V}_\phi(s)$. 4, Update Actor: $\theta \leftarrow \theta + \alpha\nabla_\theta \log\pi_\theta(a|s)\hat{A}^\pi(s,a)$. Batch AC: 1, Sample experiences by following π_θ for multiple steps. 2-4. Same as steps 2-4 for Online AC. online version is typically faster to adapt but can be more susceptible to the variance in individual experiences, whereas the batch version is more stable due to averaging over multiple experiences but may be slower to adapt to new information. **1.6 Pros and Cons** 1, Value-Based RL: Pros: Off-policy, utilizes experience replay, sample efficient; Cons: Indirect method, accurate Q estimation is hard, may result in unstable policies. 2, Policy-Based RL: Pros: Direct policy optimization, typically more stable. Cons: On-policy, less sample efficient. 3, Actor-Critic Methods: Pros: Can leverage off-policy updates with experience replay, direct policy updates for stable learning. Cons: Complexity increases with the need to maintain and train two models, potential for one to outpace the other leading to suboptimal policies. **2.Information Theory 2.1 Jensen's inequality a) Basic concept**: Let $f: I \to \mathbb{R}$ be a convex function on an interval I. For any weights p_i, where $p_i \ge 0$ and $\sum_{i=1}^n p_i = 1$, and any points $x_i \in I$, the following inequality holds:$\sum_{i=1}^n p_i f(x_i) \le f(\sum_{i=1}^n p_i x_i)$ If f is strictly convex, then equality holds if and only if all the x_i are equal. **b) Logarithmic function** $E[logx] \le logE[x]$ **2.2 Entropy a) Def**: The entropy $H(X)$ of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ and probability mass function $P(X)$ is defined as: $H(X) = -\sum_{i=1}^n P(x_i)\log_b P(x_i)$**b) Properties of H(X)** : 1, $H(X) \ge 0$.; 2, $H(X) = 0$ if and only if the outcome is certain ($P(X = x_i) = 1$ for some i).; 3, For binary X, entropy is maximized when $P(X = 0) = P(X = 1) = 0.5$; 4, In general, $H(X)$ is maximized when X has a uniform distribution, $H(X) \le \log_b(|X|)$, with $|X|$ being the number of possible outcomes. **c) Notes** 1, The base b of the logarithm is often set to 2, which makes the unit of entropy bits; 2, For real-valued random variables, the sum is replaced by an integral. **d) Conditional Entropy**: $H(Y|X) = \sum_{x\in X} P(X = x)H(Y|X = x)$ Where: $H(Y|X = x) = -\sum_{y\in Y} P(Y = y|X = x)\log P(Y = y|X = x)$ It quantifies the average uncertainty remaining about a random variable Y given that the value of another random variable X is known. Also, we have $H(Y|X) = -E[\log P(Y|X)]$. It is possible that $H(Y|X = x) > H(Y)$ because x makes Y more uncertain. **2.3 Divergence a) Kullback-Leibler divergence**, or relative entropy, measures the difference between two probability distributions $P(X)$ and $Q(X)$:$KL(P| \quad |Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$ **b) Gibbs' inequality**: $KL(P,Q) \ge 0$, with equality if and only if P is identical to Q. **c) Cross Entropy** $H(P,Q) = \sum_x P(x)\log\frac{1}{Q(x)} = -E_P[\log Q(X)]$. It is a measure of the number of bits needed to represent or transmit an average event from one distribution using the optimal code for another **d) Cross Entropy & KL div**: $KL(P||Q) = H(P,Q) - H(P)$**2.4 Jensen-Shannon divergence a) Formula** $JS(P||Q) = \frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||M)$, where $M = \frac{P+Q}{2}$. It is a symmetrized and smoothed version of the Kullback-Leibler (KL) divergence. **b) Properties** 1, $0 \leq JS(P||Q) \leq \log 2$; 2, $JS(P||Q) = 0$ if $P = Q$; 3, $JS(P||Q) = \log 2$ if P and Q have disjoint support. **c) Equivalence** In unsupervised Learning: D is the dataset, p is the real distribution, $q_\theta(x)$ is our model. Log-likelihood:$\log L(\theta|D) = \sum_{i=1}^{N} \log q_\theta(x_i)$. We have: $max_\theta - \log L(\theta|D) = min_\theta - \sum_{i=1}^{N} \log q_\theta(x_i)$= $min_\theta - \frac{1}{N}\sum_{i=1}^{N} \log q_\theta(x_i) = min_\theta H(p,q) = min_\theta KL(p||q_\theta)$. In supervised learning, minimizing cross (conditional) entropy is equal to minimizing the negative loglikelihood(NLL, $min_\theta - \sum_{i=1}^{N} \log q_\theta(y_i|x_i)$). **2.5 Mutual Information a) Def** $I(X;Y) = H(X) - H(X|Y)$, being the average reduction in uncertainty about X after observing Y, or vice versa. **b) Properties** 1, $I(X;Y) = KL\big(P(X,Y)||P(X)P(Y)\big)$; 2, $I(X;Y) = I(Y;X)$; 3&4, $I(X;Y) \geq 0, H(X,Y) \leq H(X) + H(Y)$, with equality holding if and only if $X \perp Y$; **c) Venn Diagram** H(x,y): {(H(X|Y)[I(X;Y)]H(Y|X))}, where (]:H(X); [):H(Y) **3. Linear Regression 3.1 Optimization Target** $J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$; **3.2 Solution** $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, 1 is added to the head of every row of X: $n * (f+1)$ for the bias term. **3.3 Regularization** Ridge regression shrinks large regression coefficients; LASSO forces certain coefficients to zero. **4. Logistic Regression a) Loss** $J(\theta) = -\frac{1}{m}\sum\big[y_i \log\big(h(x_i)\big) + (1-y_i)\log\big(1-h(x_i)\big)\big]$ where $h(x_i) = 1/\{1+e^{-\theta^T x_i}\}$ Its Update Rule: $\theta := \theta - \alpha\frac{1}{m}\sum(h(x_i) - y_i)x_i$; **b) Newton's Method:** $\theta := \theta - H^{-1}\nabla J(\theta)$ where H is the Hessian matrix of second-order partial derivatives of the loss function $J(\theta)$, and $\nabla J(\theta)$ is the gradient. **c) Softmax Regression :** $J(\Theta) = -\sum_{k=1}^{K} y_k \log(\hat{p}_k)$, y_k is the binary indicator (0 or 1) if class label k is the correct classification for the observation, and $\hat{p}_k$ is the predicted probability that the observation is of class k. The predicted probability $\hat{p}_k$ is given by the softmax function: $\hat{p}_k = e^{\theta_k^T x}/\sum_{j=1}^{K} e^{\theta_j^T x}$; Update: $\nabla_{\theta_k} J(\Theta) = \sum_{i=1}^{m}\left(x_i\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\right)$, where x_i is the feature vector for the i^{th} training example, $\hat{p}_k^{(i)}$ is the predicted probability for the i^{th} example for class k, and $y_k^{(i)}$ is the true label of the i^{th} example for class k. **5. Generative Model a) Naive Bayes:** Probability of class C_k given features x is $P(C_k|x) = P(x|C_k)P(C_k)/P(x)$. Assumes feature independence, so $P(x|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$ for n features. Uses Laplace smoothing to handle zero probabilities: Adjusted $P(x_i|C_k) = \{N_{x_i,C_k} + \alpha\}/\{N_{C_k} + \alpha \times D\}$ where N_{x_i,C_k} is count of feature x_i in class C_k, N_{C_k} is total count of features in C_k, α (typically 1) is the smoothing parameter, and D is number of distinct feature values. **6. NN a) compute** In a feedforward neural network, forward propagation involves calculating neuron inputs with $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$ and outputs using activation functions like $\sigma(z) = \frac{1}{1+e^{-z}}$, $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, and $\mathrm{ReLU}(z) = \max(0,z)$; activations are $a^{[l]} = g^{[l]}\big(z^{[l]}\big)$. Backpropagation computes gradients: derivatives of Sigmoid are $g'(z) = \sigma(z)\big(1 - \sigma(z)\big)$, Tanh are $g'(z) = 1 - \tanh^2(z)$, and ReLU are $g'(z) = 1$ if $z > 0$ else 0. Error at the output layer is $\delta^{[L]} = \frac{\partial\mathcal{L}}{\partial a^{[L]}} \odot g'^{[L]}\big(z^{[L]}\big)$, propagating back via $\delta^{[l]} = \left(\left(W^{[l+1]}\right)^T\delta^{[l+1]}\right) \odot g'^{[l]}\big(z^{[l]}\big)$, and gradients are $\frac{\partial\mathcal{L}}{\partial W^{[l]}} = \delta^{[l]}\big(a^{[l-1]}\big)^T$, $\frac{\partial\mathcal{L}}{\partial b^{[l]}} = \delta^{[l]}$. **b) sigmoid func** Not recommended for hidden units due to saturation; its small derivative impedes error backpropagation. Suitable for output units when using cross-entropy loss, as the gradient involves $-\log\sigma\big((2y-1)z\big)$, which only saturates with correct classification. **c) Dropout:** A regularization technique in deep learning to prevent overfitting. It randomly deactivates units (by setting their outputs to 0) during training, reducing complex co-adaptations of parameters. **d) Adam Algorithm:** Combines momentum (to accelerate learning) and adaptive learning rates (slower updates for frequently changing parameters), with bias correction in moment estimates, enhancing convergence in neural network training. **7.CNN a) Basic** Denote D: depth, H: height, W: width. footmark of 1 and 2 mean the input and output. $W_2 = \lfloor\{W_1 - F + 2P\}/S + 1\rfloor$ for both width and height(change W to H), where F is the filter size, P is padding, and S is stride, with $D_2 = K$. The total parameters are calculated as $(F^2D_1 + 1)K$, adding 1 for bias per filter. FLOPs are given by $2 * F^2 D_1 W_2 H_2 K$, representing multiplications and additions per filter application across the output volume. Number of multiplication shall /2. **b) BN** Normalizes batches with $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$, and $\hat{x}_i = \{x_i - \mu_B\}/\sqrt{\sigma_B^2 + \epsilon}$; then scales and shifts with $\gamma\hat{x}_i + \beta$. Pros: Reduces internal covariate shift, faster convergence. Cons: Batch size dependent, not suitable for RNNs, requires memory for statistics. **c) Layer Normalization:** Normalizes across features rather than batches, allowing any batch size. Proven effective in RNNs and less dependent on batch statistics, making it more flexible in application. **d) Receptive Field:** $RF_{\text{layer}} = 1 + \sum_{l=1}^{\text{layer}}\left((F_l - 1) \times \prod_{k=1}^{l-1} S_k\right)$, where F_l is filter size and S_k is stride in layer l. **e) Convolution Formula:** For an input I and filter K, convolution is $(I * K)[i,j] = \sum_m \sum_n I[m,n] \cdot K[i-m, j-n]$, summing over spatial dimensions m, n. **f) CNN Effectiveness:** 1. Sparse Interactions through localized receptive fields. 2. Parameter Sharing for efficiency. 3. Equivariant Representations for consistent spatial feature detection. **g) Skip Connections** 1, Facilitates the training of deeper networks by addressing vanishing gradient problem; 2, Helps in propagating gradients through the network, improving learning; 3, Combines features from different layers, enriching the network's understanding. **8.RNN a) Formula:** $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$, $y_t = W_{hy}h_t + b_y$. Here, h_t is the hidden state at time t, h_{t-1} is the previous state, x_t is input at time t, and W and b are weights and biases. **b) Physical Meaning:** RNNs maintain a 'memory' (hidden state) of previous inputs in the sequence, allowing them to capture temporal dynamics and sequential dependencies in data. **c) Drawbacks** 1,Vanishing Gradient: Gradients diminish exponentially in long sequences due to repeated multiplication of small derivatives in backpropagation, hindering the learning of long-range dependencies. 2, Exploding Gradient: Occurs when large derivatives result in exponentially growing gradients, causing unstable weight updates and potentially divergent training behavior. 3, Sequential Computation: RNNs process data points in sequence, not in parallel, leading to slower training times, especially for long sequences. 4, Handling Long Sequences: Struggles with information retention over long sequences due to the fixed-length hidden state and compounded effects of vanishing gradients, limiting effectiveness in capturing long-term dependencies. **9. LSTM: a) Basic** Uses gates to manage memory: Forget Gate $f_t = \sigma\big(W_f[h_{t-1}, x_t] + b_f\big)$ determines which components of the previous state C_{t-1} and how much of them to remember/forget ; Input Gate $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ determines which components of the input from h^{t-1} and x^t and how much of them should go into the current state.. Cell State Update $\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$ generates candidate values for updating the cell state, influenced by h_{t-1} and x_t; Final Cell State $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$ combines old and new; Output Gate $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ decides which parts of the cell state C_t to output through the filter of h_{t-1} and x_t. Hidden State $h_t = o_t * \tanh(C_t)$ produce output, representing the filtered cell state. It enables long-term dependency learning, countering vanishing gradients. **b) Activation Functions:** Sigmoid for f_t and i_t: Produces values near 0 or 1, effectively controlling the gating (open/close) mechanism. Tanh for $\tilde{C}_t$ and h_t: Allows backpropagation of strong gradient signals, facilitating the flow of gradients and maintaining information across time steps. **10 Attention a) Basic concept** Inputs: Sequence S of length N with features F. 1, Transformations: a, $Q = SW^Q, K = SW^K, V = SW^V$; b, Dimensions: $Q, K, V \in \mathbb{R}^{N\times D}$; 2, Attention Scores: a, $A = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)$; b, Dimension: $A \in \mathbb{R}^{N\times N}$; 3, Context Vector: a, $C = AV$; b, Dimension: $C \in \mathbb{R}^{N\times D}$. Where: 1, N: Sequence Length. 2, F: Feature Size of Input. 3, D: Dimensionality for Queries, Keys, and Values. 4, W^Q, W^K, W^V: Weight Matrices for transforming input into Query, Key, and Value. 5, A: Attention Matrix, capturing relationships between different positions in the sequence. 6, C: Context Vectors, aggregated information with attention applied. The attention mechanism dynamically weighs different parts of the input sequence, allowing the model to focus on the most relevant parts for a given task. **b) Multi-Head Attention** Multi-head attention allows a model to simultaneously focus on different aspects of the input sequence, capturing various dimensions of the data's context and relationships. For example, in a language translation task, one attention head might concentrate on grammatical structure, while another might focus on semantic meaning, ensuring a more accurate and contextually rich translation. **c) BERT Pos Embedding** Essential because BERT processes all input tokens simultaneously, unlike AutoRegressive models. Without positional embeddings, the model lacks information about token order in a sentence. Adding positional embeddings allows each token to have a unique representation based on its position, maintaining crucial sequence information. **d) vit** The Vision Transformer (ViT) works by first dividing an input image into small, fixed-size patches. Each patch is then flattened and transformed into a vector (embedding) using a trainable linear projection. To maintain positional information, positional embeddings are added to these patch embeddings. These combined embeddings are processed by a transformer encoder, similar to those used in NLP, which uses self-attention mechanisms to capture complex relationships between all patches. The encoder's output is then used for tasks like image classification, where the model learns to interpret and classify the image based on the aggregated information from all patches. **11 VAE&&GAN a) VAE** 1,Encoder: $q_\phi(z|x)$ approximates true posterior $p(z|x)$, maps input x to latent space. 2, Decoder: $p_\theta(x|z)$ reconstructs x from latent variable z. 3, Loss Function: ELBO (Evidence Lower Bound) = $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL\left(q_\phi(z|x) \parallel p(z)\right)$, balancing reconstruction quality and latent space regularization. **b) Reparameterization** $aN(\mu, \sigma^2) + b = N(a\mu + b, a^2\sigma^2)$**c) VAE vs. GAN:** 1, VAE learns conditional distribution $p(x|z)$ with a generative model (decoder) and an encoder network. Minimizes KL divergence for parameter training, using encoder for tractable lower bound estimation. 2, GAN learns deterministic function $x = g(z)$ with a generative model and a discriminator. Minimizes Jensen-Shannon divergence, using discriminator for approximation; trains generator and discriminator alternately. 3, application diff: GANs are primarily used for high-quality image generation and style transfer, excelling in tasks where realistic and visually compelling outputs are crucial. VAEs, on the other hand, are better suited for tasks involving data reconstruction and generation, such as in collaborative filtering, due to their probabilistic framework and ability to create smoother latent spaces. **12 DDPM a) Basic Concepts** 1. Training-Repeat: a, Sample x_0 from a distribution $q(x_0)$; b, Sample t uniformly from the set {1, ..., T}; c, Sample ϵ from a normal distribution $\mathcal{N}(0, I)$; d, Perform a gradient descent step on the objective function with respect to ϵ given by: $\nabla_\epsilon \left\| \epsilon - \epsilon_\theta\left(\sqrt{\alpha_t} x_0 + \sqrt{1-\alpha_t}\epsilon, t\right) \right\|^2$. 2, Sampling - Sample x_T from a normal distribution $\mathcal{N}(0, I)$. Repeat: a, Sample z from a normal distribution $\mathcal{N}(0, I)$; b, Compute x_{t-1} using the following update rule:$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$. Finally, return x_0. **b) Diff from DDIM** DDIMs differ from DDPMs primarily in their sampling process, where DDIMs allow for a deterministic and faster sampling with fewer steps, whereas DDPMs involve a stochastic and slower process with many time steps. DDIMs can produce consistent results with the same initial conditions, beneficial for tasks requiring precise control, but may limit diversity in generated samples. On the other hand, DDPMs can generate a wide variety of samples due to their stochastic nature but at the cost of computational efficiency. Consequently, DDIMs are favored for speed and control, while DDPMs are chosen for their diversity and quality of the generated samples, despite being more resource-intensive. **c) LDM** Latent Diffusion Models (LDMs) improve upon Denoising Diffusion Probabilistic Models (DDPMs) by operating in a lower-dimensional latent space, which significantly enhances computational efficiency and reduces the time needed to generate high-quality samples. This approach maintains the generative performance of DDPMs while enabling faster and more resource-efficient model training and sampling processes.