Chương 1. RESEARCH METHOD

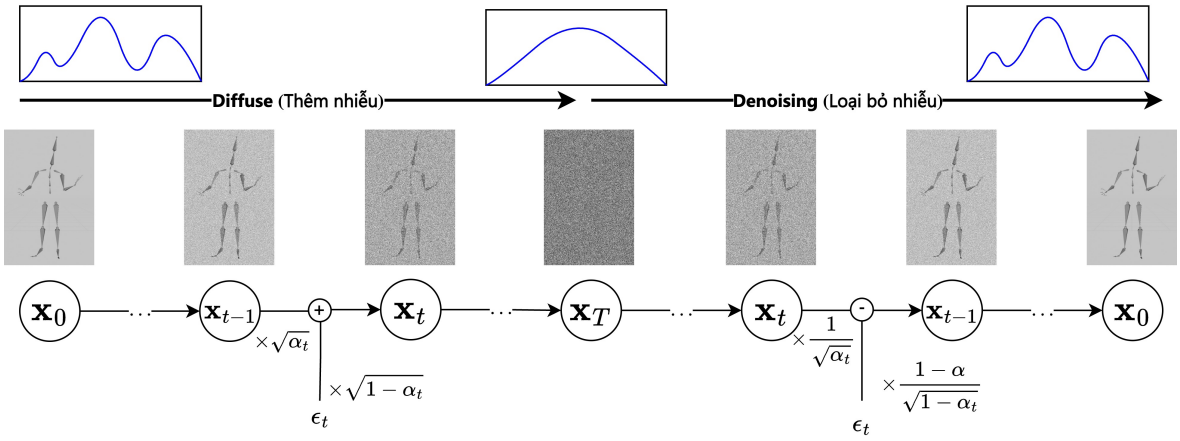
The nature of neural network-based methods is to estimate the probability density of data, which requires normalization. In contrast, Diffusion models learn to estimate the gradients of data distribution [**?**] and do not require normalization over the entire dataset, resulting in better performance than neural network methods that do not use diffusion. Diffusion models can approximate data distributions even in low-density regions and generate highly detailed results.

This thesis builds upon the DiffuseStyleGesture model [**?**], with the main improvement being the integration of speech. Speech is transcribed into text, and the text is embedded to obtain textual semantic feature vectors, which are then used as conditioning vectors in the conditional diffusion process, as described in **??**. In addition, in *Stage 7. Rendering* (**??**), Unity is used to visualize the generated gestures.

The main contributions of the thesis are presented in **??**.

The thesis first describes the diffusion model in Section 1.1, followed by the proposed OHGesture model in Section 1.2.

1.1 Basic Diffusion Model and Improvements



Hình 1.1: The **non-training** Diffusion and Denoising Process

In neural network methods like ResNet or InceptionNet, the goal is to learn the weight θ of a function $f_\theta(x)$ by minimizing a loss function $\mathcal{L}_{\text{loss}}$ between a label y and its prediction $\hat{y}$. Once training is complete, the learned weight θ' can be used to predict a new sample x' through $f_{\theta'}$ to obtain prediction y'.

Similarly to VAE, where the input matrix is encoded into a latent vector z and then decoded back into a matrix of the original size, the diffusion model breaks the learning process into T steps. At each step t, the forward diffusion process $q(\mathbf{x})$ from $1 \to T$ adds Gaussian noise $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ is the identity matrix of the standard Gaussian distribution with $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = 1$.

The process of adding noise from left to right is called $q(\mathbf{x})$, while the denoising process from right to left is $p(\mathbf{x})$. Note that ϵ_t is randomly sampled at each step t, but once sampled, it is fixed. As illustrated in Figure 1.1, if we add a noise ϵ_t during diffusion and subtract exactly the same noise during denoising, the result $\mathbf{x}_0$ is **identical** to the original input.

However, during denoising, instead of subtracting the actual noise ϵ_t, the model uses a function f_θ to **predict the added noise** during diffusion. Then, $\mathbf{x}_T$ is iteratively denoised by subtracting the predicted noise to obtain $\hat{\mathbf{x}}_{T-1}$, continuing until $\hat{\mathbf{x}}_0$ is reached.

In the basic Diffusion model or Denoising Diffusion Probabilistic Models (DDPM [**?**]), the denoising process goes from $T \to 1$, and the goal is to learn

the weights θ of the noise prediction function f_θ (also denoted as ϵ_θ). After training, the learned weights θ' are used to predict the noise $\hat{\epsilon}$. Then, the noise is subtracted from the noisy image $\mathbf{x}_t$ to get $\mathbf{x}_{t-1}$, with additional noise $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ added to maintain gesture diversity. This process is repeated until reaching the final prediction $\hat{\mathbf{x}}_0$.

1.1.1 Forward Diffusion Process

Given the input data $\mathbf{x}_0 \sim q(x)$ from the real dataset, at each step t, noise is added to $\mathbf{x}_0$ with the noise-to-signal ratio controlled by a factor β:

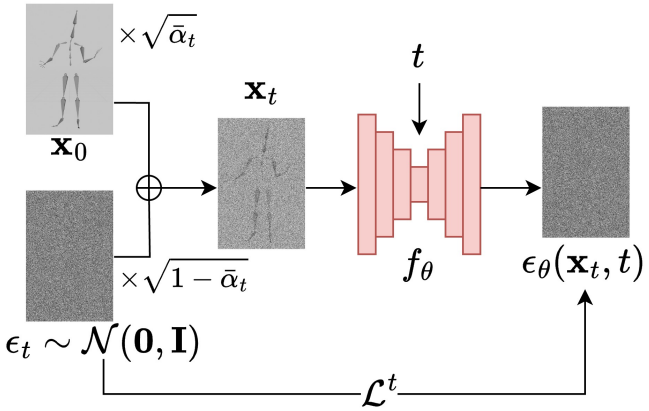
$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_{t-1} \tag{1.1}$$

The forward process runs from $1 \to T$, with $\beta_t \in (0, 1)$ for each t.

Since a function of the form $f(x) = ax + b\epsilon$, with $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, implies $f(x) \sim \mathcal{N}(ax, b^2)$, the forward process can be reformulated as:

$$\begin{aligned} q(\mathbf{x}_t|\mathbf{x}_{t-1}) &= \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \\ q(\mathbf{x}_{1:T}|\mathbf{x}_0) &= \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \end{aligned} \tag{1.2}$$

Here, $\sqrt{1-\beta_t}$ gradually decreases the contribution of $\mathbf{x}_t$, while β_t increases the noise component. Typically, $\beta_1 < \beta_2 < \cdots < \beta_T$. As $T \to \infty$, $\mathbf{x}_T$ approaches a pure Gaussian distribution: $q(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$ [**?**].



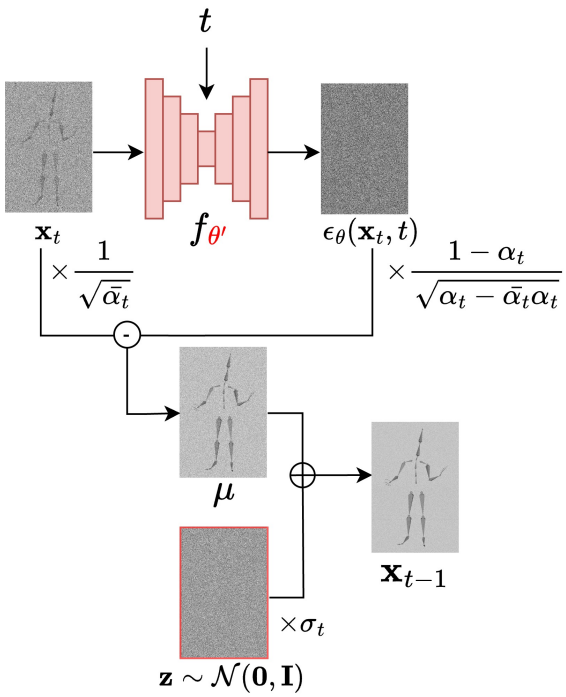
Hình 1.2: Noise addition during training

Since $\boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and fixed for each t, $\mathbf{x}_t$ can be computed directly from $\mathbf{x}_0$. Define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, then Equation 1.1 becomes:

$$\begin{aligned} \boldsymbol{x}_t &= \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_0 \\ &\sim \mathcal{N}\left(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\,\mathbf{I}\right) \end{aligned} \tag{1.3}$$

The evolution of $\sqrt{\alpha}$ and $\sqrt{1-\alpha}$ over diffusion steps is shown in Appendix ??.

1.1.2 Denoising Process



Hình 1.3: Denoising during inference

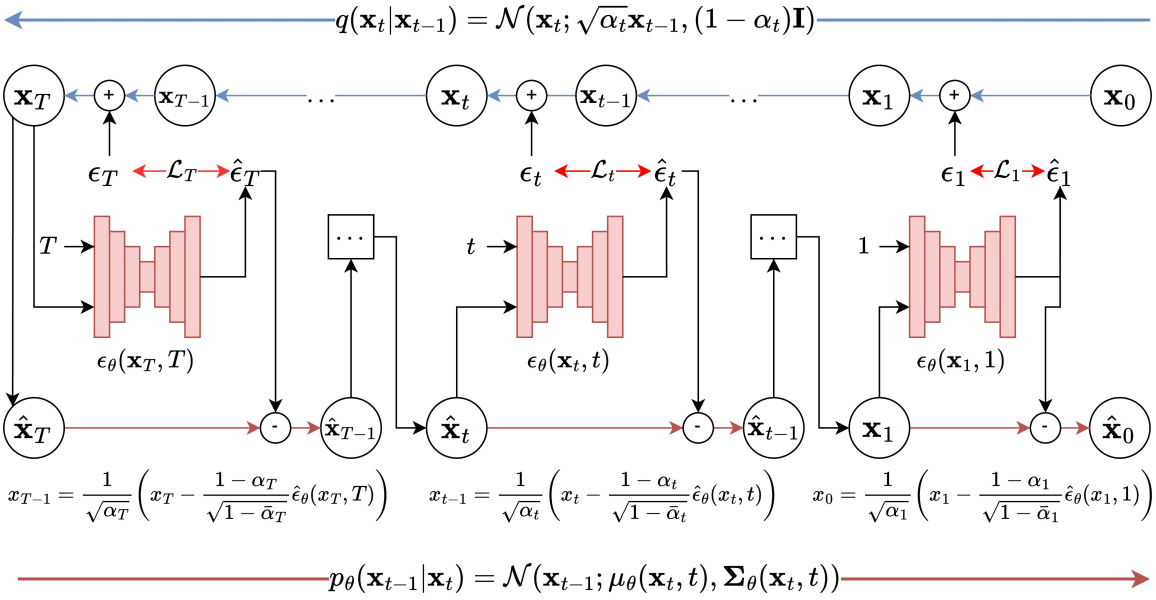
The denoising process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ starts from $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. A neural network $f_\theta(\mathbf{x}_t, t)$ is used to predict the noise $\hat{\epsilon}_t = f_\theta(\mathbf{x}_t, t)$.

The denoising distribution has mean and variance:

$$\begin{aligned} p_\theta(\mathbf{x}_{0:T}) &= p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \end{aligned} \tag{1.4}$$

where $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\mathbf{x}_t, t)\right)$.

1.1.3 Training Process in the Basic Diffusion Model



Hình 1.4: Basic diffusion model

The diffusion model learns the parameters θ of the noise prediction function $f_\theta(\mathbf{x}_t, t)$ (or ϵ_θ). During denoising, we minimize the loss between the predicted noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ and actual noise $\boldsymbol{\epsilon}_t$ at each step t:

$$\begin{aligned}\mathcal{L}^t &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right] \\ &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\right]\end{aligned} \tag{1.5}$$

The total loss is $\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}^t$.

Here, $f_\theta(x_t, t)$ or ϵ_θ is a U-Net model used to encode and decode the data for noise prediction. The computation process is illustrated in Figure 1.4.

Algorithm 1 DDPM Training Algorithm

1. Precompute $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, and $\sqrt{\bar{\alpha}_t}$ for $t = 1 \to T$. Define noise schedule $\{\alpha_t \in (0,1)\}_{t=1}^{T}$ with $\alpha_1 < \alpha_2 < \cdots < \alpha_T$.
2. Sample label $\mathbf{x}_0$ from the normalized data distribution.
3. Generate random noise $\boldsymbol{\epsilon}_t$ for each step $t = 1 \to T$, where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
4. Apply forward diffusion to get $\mathbf{x}_t$:
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t$$
5. Randomly select $t \in [1, T]$.
6. Feed $\mathbf{x}_t$ and t into the model to predict noise: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$.
7. Compute the gradient:
$$\nabla_{\theta_t} \left\| \boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2$$
And loss:
$$\mathcal{L}^t = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\| \boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t) \|^2 \right]$$
8. Repeat step 6 until convergence to obtain optimal weights θ'.

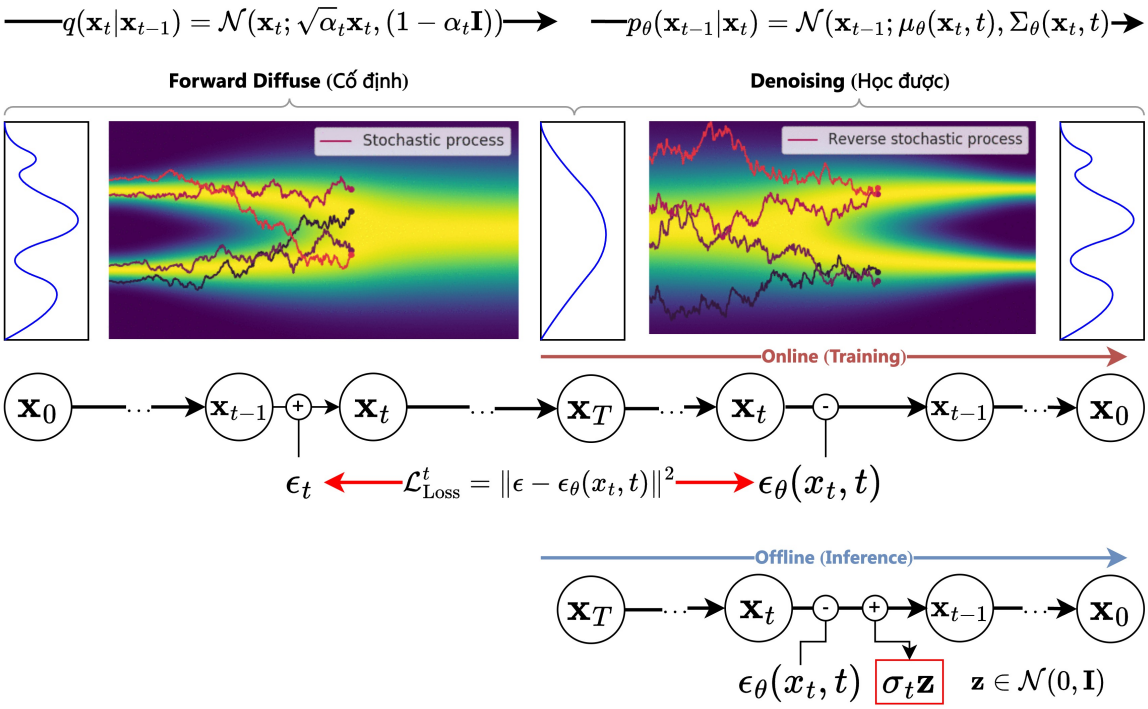
1.1.4 Basic Sampling Process in Diffusion Models

After obtaining the weights θ', the denoising function is used to denoise from noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The transformation from pure noise $\mathbf{x}_T$ to the prediction $\hat{\mathbf{x}_0}$ is as follows:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_{\theta'}(\mathbf{x}_t, t)\right) + \sqrt{1-\alpha_t}\tilde{\epsilon}_t \tag{1.6}$$

Note that ϵ_t is fixed and randomly generated noise created before the training process, and reused during the forward diffusion process Subsection 1.1.2 in formula Equation 1.1. As shown in Figure 1.5, the error ϵ_t corresponds to the noise at each step t, and the loss function $\mathcal{L}^t$ is computed per time step t.

Following the training process, the DDPM sampling algorithm begins from pure noise, i.e., $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, where the initial data is entirely noise. The values $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, and $\sqrt{\bar{\alpha}_t}$, obtained from the training phase, are used during sampling to reconstruct the original data $\mathbf{x}_0$. The next step is to compute the noise adjustment coefficient σ_t, based on the noise schedule α_t defined during train-



Hình 1.5: Training and Sampling process in a standard Diffusion model

ing. These values influence the amount of noise added in the reverse sampling process.

The sampling process (Algorithm 2) proceeds from step T down to 1, and at each step, a random noise $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ is generated and added to the predicted result. At each step t, the model predicts the noise $\boldsymbol{\epsilon}_{\theta'}$ based on the noisy data $\mathbf{x}_t$ and time step t, then uses this prediction to compute the value μ, an estimate of $\mathbf{x}_0$. Finally, a noise term $\sigma_t\mathbf{z}$ is added to μ to obtain $\hat{\mathbf{x}}_{t-1}$, the noisy data at step $t-1$. This process continues until $t = 1$, at which point $\hat{\mathbf{x}}_0$ — the final prediction of the original data — is obtained through denoising.

Algorithm 2 Sampling algorithm in DDPM

1. Start with noise: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
2. Retrieve values $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, and $\sqrt{\bar{\alpha}_t}$ from the training process.
3. Compute the noise adjustment coefficient σ_t from α_t at each step $t : 1 \to T$:

$$\sigma_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(1-\alpha_t)}$$

4. For each t, iterate **sequentially** from $[T, \ldots, 1]$.
5. Generate random noise $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.
6. Feed $\mathbf{x}_t$ into the model to infer noise: $\boldsymbol{\epsilon}_{\theta'} = \boldsymbol{\epsilon}_{\theta'}(\mathbf{x}_t, t)$.
7. Use the predicted noise to subtract from $\mathbf{x}_t$ at step t:

$$\mu = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\theta'}(\mathbf{x}_t, t)\right)$$

8. Add noise: $\hat{\mathbf{x}}_{t-1} = \mu + \sigma_t \mathbf{z}$.
9. When $t = 1$, obtain $\hat{\mathbf{x}}_0$ from the denoising process.

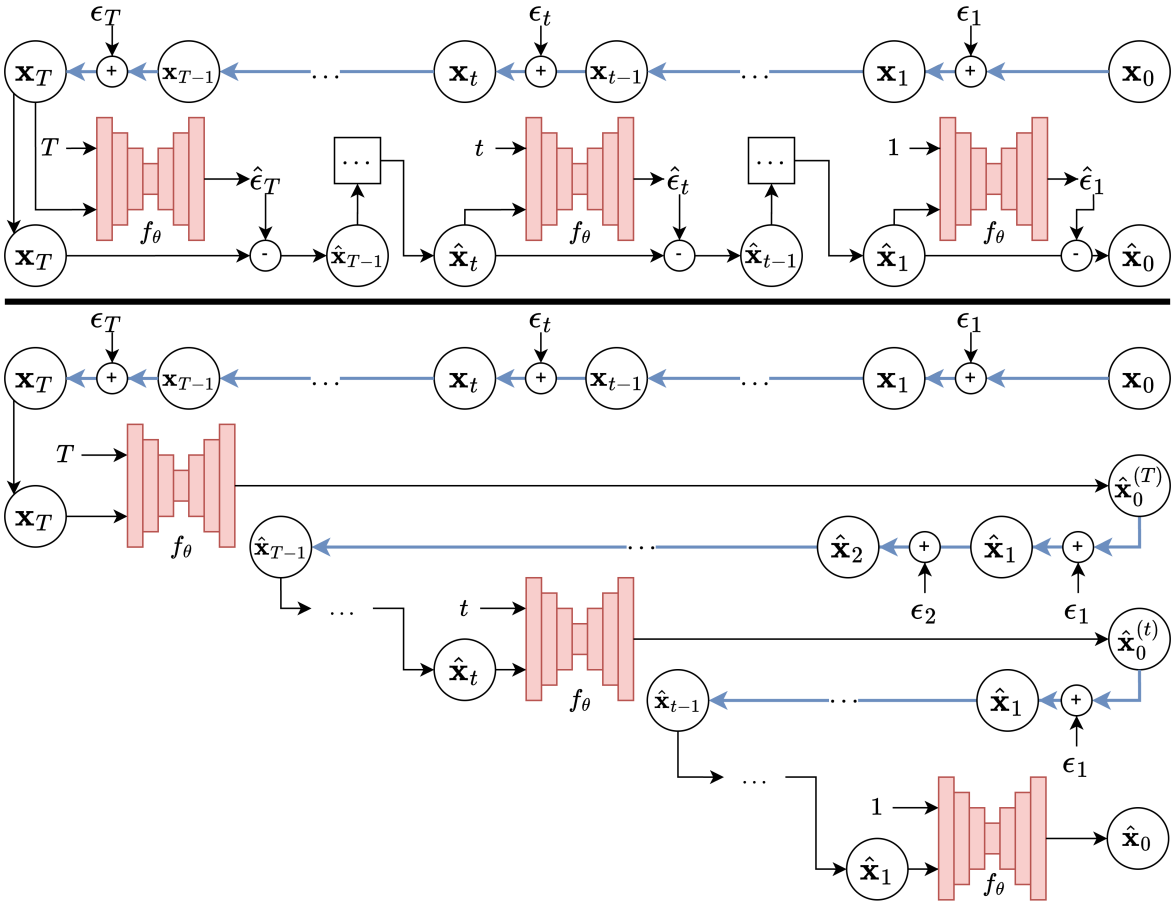
The most important aspect of the denoising sampling process is to add a noise term $\mathbf{z} \in \mathcal{N}(0, \mathbf{I})$, with $\mathbf{z}$ sequentially added at each step t using the scaling factor σ_t. The goal of noise ϵ is to shape the marginal noise distribution so that model f_θ (or ϵ_θ) can learn to denoise, whereas $\mathbf{z}$ is used to increase diversity in generation and improve stability during sampling. The decay of σ_t is detailed in Appendix **??**.

1.1.5 Improved Diffusion Model with $\mathbf{x}_0$ Prediction Instead of ϵ_t

Based on Equation 1.3, it is clear that given $\mathbf{x}_t$, we can infer $\mathbf{x}_0$. Conversely, given $\mathbf{x}_0$, we can explicitly infer $\mathbf{x}_t$ for any t by adding the noise ϵ_t from the forward diffusion process.

From this observation, the authors in [**?**] proposed an improvement to DDPM: instead of using the neural network $f_\theta(\mathbf{x}_t, t)$ to predict ϵ_t as in Figure 1.4, the function $f_\theta(\mathbf{x}_t, t)$ is repurposed to directly predict $\mathbf{x}_0$, after which

noise is added using Equation 1.3.



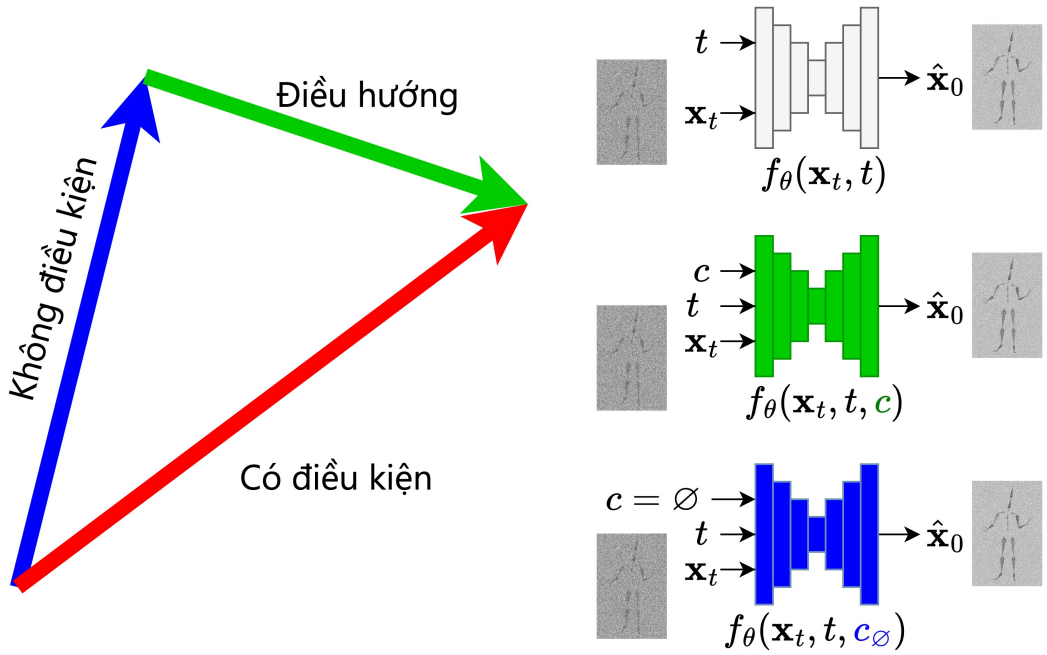
Hình 1.6: Comparison between ϵ objective (top) and $\mathbf{x}_0$ objective (bottom)

As shown in Figure 1.6, we start with forward diffusion from $t : 1 \to T$ to obtain $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. After acquiring $\mathbf{x}_T$, we feed it into f_θ to predict $\hat{\mathbf{x}}_0$, and from $\hat{\mathbf{x}}_0$ we add noise to obtain $\hat{\mathbf{x}}_{t-1}$. This continues until $t = 1$, when $\hat{\mathbf{x}}_0$ is ultimately obtained. We can compare the two methods as follows:

- ϵ **objective**: the model predicts the noise. Begin with forward noise injection to get $\mathbf{x}_T$, then use $\mathbf{x}_T \in \mathcal{N}(0, \mathbf{I})$ as input for denoising. During denoising, the model predicts the noise $\hat{\epsilon}_t$ added in the forward step, i.e., ϵ_t, and minimizes the error between the predicted and actual forward noise.
- $\mathbf{x}_0$ **objective**: similarly, the model applies forward diffusion to get $\mathbf{x}_T$, then uses $\mathbf{x}_T \in \mathcal{N}(0, \mathbf{I})$ for denoising. The model directly predicts $\mathbf{x}_0$, then reintroduces noise to compute $\mathbf{x}_{t-1}$, and continues using $\mathbf{x}_{t-1}$ as input to predict $\mathbf{x}_0$ again.

1.1.6 Conditional Diffusion Models

To control the generation process under different conditions, generation must be conditioned on c—that is, we aim to model $p(\mathbf{x}|c)$ given condition c. The authors in [**?**] proposed using a separate function f_ϕ to train the conditional component. However, this approach presents difficulties when the conditions change, and combining or updating weights in a separate model is challenging for scalability.



Hình 1.7: Conditional Diffusion via Guidance Vector

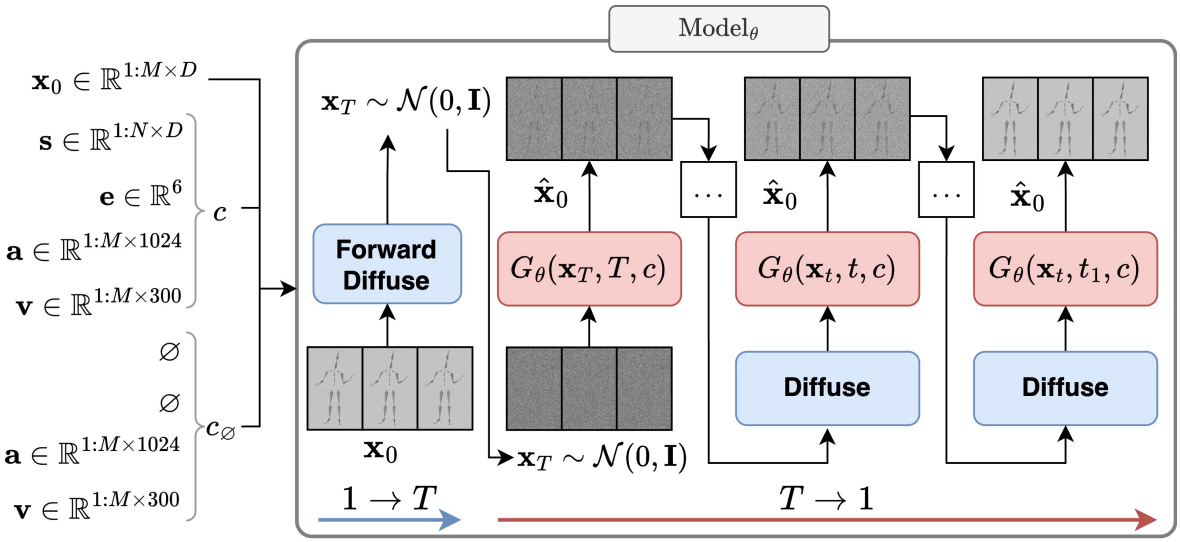
As shown in Figure 1.5, to allow inference to generate different $\mathbf{x}_0$ samples controllably based on condition c, a guidance term must be added at each step t. The authors proposed Classifier-Free Diffusion Guidance [**?**], in which the result $\mathbf{x}_0$ is updated by combining the conditional and unconditional outputs:

$$\hat{\mathbf{x}}_{0\gamma,c,c\varnothing} = \gamma \cdot f_\theta\left(\mathbf{x}_t, t, c\right) + (1-\gamma) \cdot f_\theta\left(\mathbf{x}_t, t, c_\varnothing\right) \tag{1.7}$$

Here, c is the condition, $c_\varnothing$ is the null (empty) condition, and the conditional generation is controlled by the parameter γ: the larger γ is, the closer the result aligns with condition c; conversely, a smaller γ leads to a result closer to the unconditional output.

As illustrated in Figure 1.7, the top function f_θ is unconditional, the middle one is conditional diffusion, and the bottom one is diffusion under an empty condition.

1.2 Proposed OHGesture Model



Hình 1.8: Overview of the OHGesture model

The proposed model **OHGesture** (**O**pen **H**uman **Gesture** Generation) in this thesis is based on the **DiffuseStyleGesture** model [**?**], which applies the Diffusion model [**?**] with conditional guidance [**?**] (Classifier-Free Diffusion Guidance) to control features during the denoising process.

The similarities and differences of applying the diffusion model to the gesture generation task compared to image generation are as follows:

Similarities

- Uses the Diffusion model (Section 1.1) on gesture data $\mathbf{x}^{1:M\times D}$, with M temporal frames and $D = 1141$ representing motion coordinates per frame (analogous to image width and height).
- Uses conditional Diffusion (Subsection 1.1.6) with $\mathbf{x}_0$ objective (Subsection 1.1.5).
- In stages *4. Feature Encoding* and *6. Feature Decoding* in **??**, the model uses a latent vector of dimension 256.

Differences

- Conditional gesture generation:

  - Emotional condition: $c = \left[\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}\right]$ and $c_{\varnothing} = \left[\varnothing, \varnothing, \mathbf{a}, \mathbf{v}\right]$.
  - Emotion state interpolation between $\mathbf{e}_1, \mathbf{e}_2$ using: $c = \left[\mathbf{s}, \mathbf{e}_1, \mathbf{a}, \mathbf{v}\right]$ and $c_{\varnothing} = \left[\mathbf{s}, \mathbf{e}_2, \mathbf{a}, \mathbf{v}\right]$.

- In stage *5. Feature Fusion* **??**, the model uses Self-Attention: learning the relationship between emotions, seed gestures, and each frame (similar to DALL-E 2's text-image alignment).

- In stage *5. Feature Fusion* **??**, the model concatenates speech and text (analogous to ControlNet's pixel-wise condition).

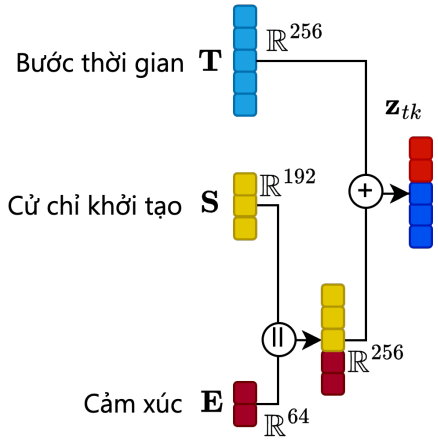
Here, $\mathbf{x}_0$ is a sequence of M gesture frames $\mathbf{x} \in \mathbb{R}^{1:M \times D}$ ($D = 1141$), with condition $c = [\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}]$ including seed gesture $\mathbf{s}$, emotion $\mathbf{e}$, speech $\mathbf{a}$ corresponding to the gesture, and text $\mathbf{v}$.

The model's objective is to learn parameters θ of the generative function G_θ with inputs being the noisy gesture matrix $\mathbf{x}_t \in \mathbb{R}^{1:M \times D}$, timestep t, and condition c. An overview of the proposed **OHGesture** model is illustrated in Figure 1.8. As with standard diffusion models, it includes two processes: the diffusion process q and the denoising process p_θ with weights θ. The *1. Preprocessing* stage will be presented in **??**.

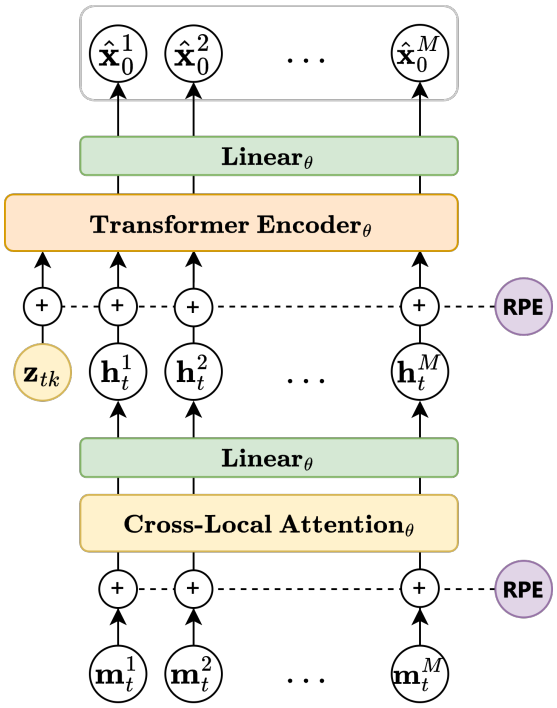
Cross-local Attention is performed with $\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{m}_t$. Following the idea of the Routing Transformer method [**?**], Cross-local Attention highlights the importance of constructing intermediate feature-vector representations before passing them through the Transformer Encoder layer, as shown in **??**. The feature vectors are augmented with a relative position-encoding vector **RPE** (**R**elative **P**osition **E**ncoding) to preserve temporal ordering before entering the Cross-Local Attention layer.

After Cross-Local Attention, the model is forwarded through a linear layer, as in **??**, to align with the M frames and obtain the matrix $\mathbf{h}_t \in \mathbb{R}^{1:M \times D}$.

1.2.0.1 Global Feature Fusion with the Transformer Encoder



(a) How to concatenate features to obtain $\mathbf{z}_{tk}$



(b) Frame-wise feature-fusion process

Following MDM [**?**], the vector $\mathbf{z}_{tk}$ is the first token that encodes information for the entire frame sequence, analogous to the `CLS` token in BERT [**?**], which summarizes an entire text segment. Here, the thesis uses $\mathbf{z}_{tk}$, with $\mathbf{z}_{tk} \in \mathbb{R}^{256}$ (**??**), as the first token representing global features for the whole sequence of M frames.

$$\mathbf{X}_0 = \text{Transformer Encoder}\big(\text{concat}(\mathbf{z}_{tk} \parallel \mathbf{h}_t^{1:M})\big) \tag{1.8}$$

The vectors $\mathbf{h}_t$ represent the sequence of M frames. Similar to Reformer [**?**], before entering the Self-Attention layer of the Transformer Encoder, the model employs Relative Position Encoding (RPE) instead of absolute position encoding, improving efficiency on long sequences. Within the Transformer Encoder layer [**?**], relationships among the data sequences are computed. The Transformer Encoder applies the same self-attention mechanism as in **??** but without the **Mask**, enabling correlations across the entire sequence to be captured.

1.2.1 Feature Decoding Stage

In stage *6. Feature Decoding* (**??**), once feature correlations are computed, the goal is to upsample the data back to its original dimensionality.

As illustrated in **??**, the latent matrix $\mathbf{X}_0$, after passing through the Transformer Encoder to capture correlations among heterogeneous data types, is fed into a linear projection layer $\hat{\mathbf{x}}_0 = \text{Linear}_\theta(\mathbf{X}_0)$ to restore the latent matrix to its original size, yielding $\hat{\mathbf{x}}_0 \in \mathbb{R}^{1:M \times D}$.

The final rendering step is presented in **??**.

1.2.2 Emotion Control in Gesture Generation

The preceding steps enable the model to learn gesture generation. To incorporate emotions across different contexts, each emotion is parameterized and varied so that the predictions faithfully express the designated affect.

Analogous to conditional denoising models [**?** **?**], the thesis uses the condition vector $c = [\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}]$, where $\mathbf{s}$ is the seed gesture, $\mathbf{e}$ the emotion, $\mathbf{a}$ the associated speech, and $\mathbf{v}$ the text. The conditional diffusion model injects c at every timestep t in the denoising network $\text{G}_\theta(\mathbf{x}_t, t, c)$, with $c_\varnothing = [\varnothing, \varnothing, \mathbf{a}, \mathbf{v}]$ (unconditional) and $c = [\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}]$ (conditional). A random mask applied to the seed-gesture and emotion vectors conveniently switches labels, allowing optimization under diverse conditions.

$$\hat{\mathbf{x}}_{0\,c,c_\varnothing,\gamma} = \gamma\, G(\mathbf{x}_t, t, c) + (1-\gamma)\, G(\mathbf{x}_t, t, c_\varnothing) \tag{1.9}$$

Classifier-free guidance [**?**] further enables interpolation between two emotions $\mathbf{e}_1$ and $\mathbf{e}_2$ by setting $c = [\mathbf{s}, \mathbf{e}_1, \mathbf{a}, \mathbf{v}]$ and $c_\varnothing = [\mathbf{s}, \mathbf{e}_2, \mathbf{a}, \mathbf{v}]$:

$$\hat{x}_{0\,\gamma,c_1,c_2} = \gamma\, G(x_t, t, c_1) + (1-\gamma)\, G(x_t, t, c_2).$$

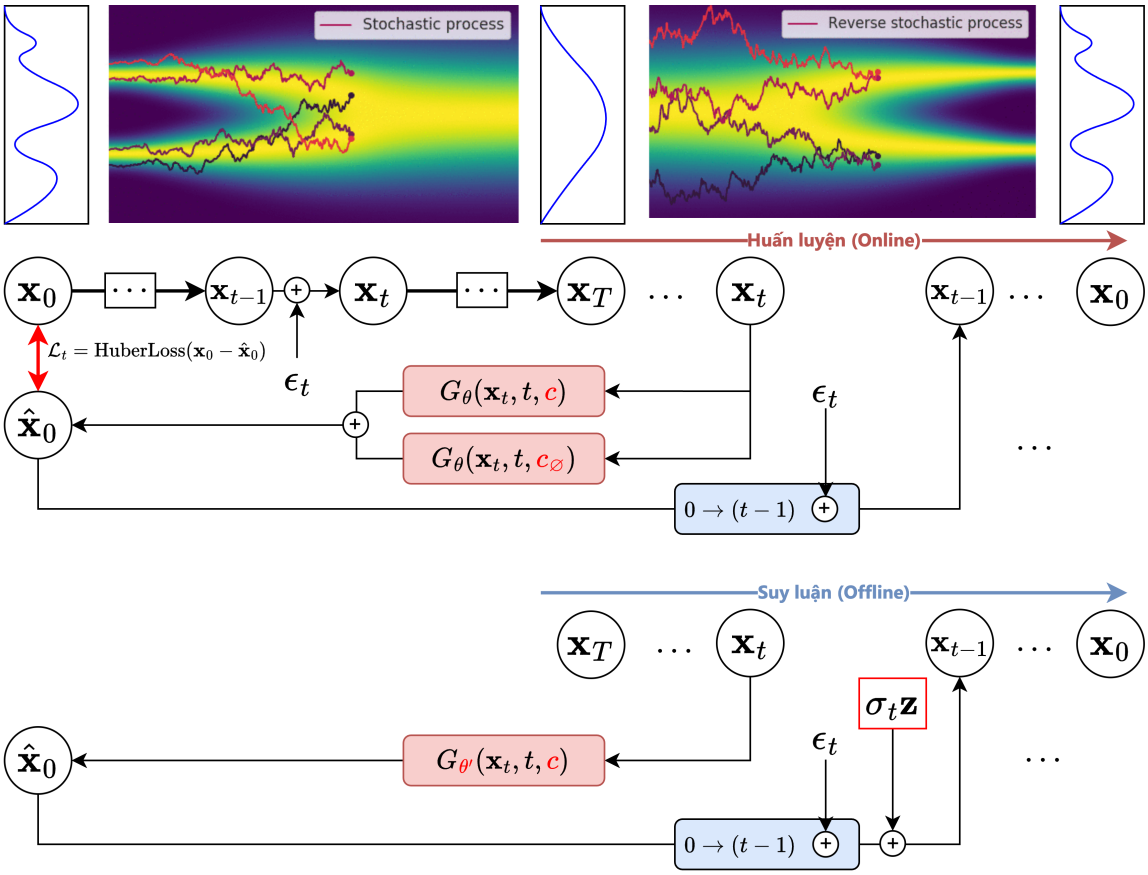
1.2.3 Training Procedure

Algorithm 3 Training in OHGesture

1. Pre-compute γ, $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, $\sqrt{\bar{\alpha}_t}$, and random noise $\boldsymbol{\epsilon}_t$ for each timestep $t : 1 \rightarrow T$. Define the noise schedule $\{\alpha_t \in (0,1)\}_{t=1}^{T}$.
2. Sample the initial label $\mathbf{x}_0$ from the normalized data distribution.
3. Randomly generate Bernoulli masks $c_1 = [\mathbf{s}, \mathbf{e_1}, \mathbf{a}, \mathbf{v}]$, $c_2 = [\mathbf{s}, \mathbf{e_2}, \mathbf{a}, \mathbf{v}]$, or $c_2 = [\varnothing, \varnothing, \mathbf{a}, \mathbf{v}]$.
4. Add noise to obtain the noisy gesture $\mathbf{x}_t$:
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}_t.$$
5. Sample t **uniformly** from $[1, T]$.
6. Given $\mathbf{x}_t$, t, and masks c_1, c_2, predict the gesture sequence:
$$\hat{\mathbf{x}}_{0\,\gamma,c_1,c_2} = \gamma\, G_\theta(\mathbf{x}_t, t, c_1) + (1-\gamma)\, G_\theta(\mathbf{x}_t, t, c_2).$$
7. Compute the loss and gradient to update θ:
$$\mathcal{L}^t = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}_t}\big[\text{HuberLoss}(\mathbf{x}_0, \hat{\mathbf{x}}_0)\big].$$
8. Repeat from step 6 until convergence, obtaining the optimal parameters θ'.

Algorithm 3 trains the OHGesture model by first computing the required values and hyper-parameters—γ, $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, $\sqrt{\bar{\alpha}_t}$, and $\boldsymbol{\epsilon}_t$—for every timestep t ($1 \ldots T$). The initial label $\mathbf{x}_0$, representing the ground-truth gesture, is drawn from the normalized data distribution. Random Bernoulli masks c_1 and c_2 emulate different conditions (gesture, emotion, speech, or text), with one mask possibly lacking emotion information. Noise is then added to create the noisy gesture $\mathbf{x}_t$. A timestep t is sampled uniformly, and $\mathbf{x}_t$ with the masks is fed into the model to predict the original gesture sequence as a weighted combination of conditional outputs. The Huber loss between ground-truth and prediction is used to update θ. This cycle repeats until the model converges, yielding the optimal parameters θ'.

1.2.4 Sampling Process



Hình 1.10: Offline (Training) and Online (Inference) Phases

To generate gestures of arbitrary length, the original sequence is segmented into clips of length M. During training, the seed gesture can be chosen by randomly selecting a gesture from the dataset or by averaging the clipped segments—here, the mean rotation angles are used. Generated frames are processed sequentially, with the last $N = 8$ frames taken as the seed for the next iteration. For each clip, the gesture $\mathbf{x}_t$ is denoised via $\hat{\mathbf{x}}_0 = G_{\theta'}(\mathbf{x}_t, t, c)$; noise is re-added to obtain $\mathbf{x}_{t-1}$, and the procedure repeats until $t = 1$, yielding $\mathbf{x}_0$.

Algorithm 4 Sampling in OHGesture

1. Initialize with noise: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
2. Retrieve $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, and $\sqrt{\bar{\alpha}_t}$ from training; precompute σ_t from α_t for each timestep $t : 1 \to T$.
3. Split each 4-second speech segment into $\mathbf{a} \in \mathbb{R}^{64000}$. The initial seed gesture $\mathbf{s}$ is the data mean and is later updated from the inferred gesture segment. Select the desired emotion, obtain the transcript $\mathbf{v}$ from speech $\mathbf{a}$, and form the condition $c = [\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}]$.
4. For each timestep, take t **sequentially** from $[T, \ldots, 1]$.
5. Sample random noise $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.
6. Infer $\hat{\mathbf{x}}_0^{(t)} = G_{\theta'}(\mathbf{x}_t, t, c)$.
7. Diffuse $\hat{\mathbf{x}}_0^{(t)}$ from step $0 \to t$ to obtain $\hat{\mathbf{x}}_{t-1}^{(t)}$.
8. Add noise: $\hat{\mathbf{x}}_{t-1} = \hat{\mathbf{x}}_{t-1}^{(t)} + \sigma_t \mathbf{z}$.
9. Return to step 4. When $t = 1$, output the denoised gesture $\hat{\mathbf{x}}_0$.

Algorithm 4 starts by initializing the noisy gesture $\mathbf{x}_T$ from $\mathcal{N}(0, \mathbf{I})$. The values $\sqrt{\alpha_t}$, $\sqrt{1-\alpha_t}$, and $\sqrt{\bar{\alpha}_t}$ obtained during training, together with σ_t, are employed at each timestep $(1 \ldots T)$. Each 4-second speech segment is represented by $\mathbf{a}$, and the seed gesture $\mathbf{s}$ is taken as the data mean or from the previously inferred segment. The desired emotion and the transcript form the condition $c = [\mathbf{s}, \mathbf{e}, \mathbf{a}, \mathbf{v}]$. The algorithm proceeds sequentially from T to 1: random noise $\mathbf{z}$ is generated, the model predicts $\hat{\mathbf{x}}_0^{(t)}$ from $\mathbf{x}_t$, t, and c, then $\hat{\mathbf{x}}_{t-1}^{(t)}$ is computed and updated with noise. This loop continues until $t = 1$, after which the algorithm outputs the final denoised gesture $\hat{\mathbf{x}}_0$.