The Open University

Department of Mathematics and Computer Science

Text-to-Image Generation Using CLIP-Conditioned Diffusion Models with Classifier-Free Guidance

A project submitted as partial fulfillment of the requirements for the M.Sc. degree in Computer Science

Prepared by

Shlomi Domnenco

Supervised by

Dr. Mireille Avigal & Dr. Azaria Cohen

January 2026

Contents

List of Tables

Abstract

We developed a text-to-image generation model based on Stable Diffusion, a diffusion-based generative framework that synthesizes images by iteratively refining random noise into coherent visual content.

The model utilizes CLIP embeddings as the text conditioning mechanism, enabling alignment between textual descriptions and generated images. Our approach operates in latent space for computational efficiency while leveraging the inherent stability and robustness of the diffusion process. We demonstrate that our method substantially outperforms existing GAN-based approaches, particularly in comparison with Creative Adversarial Networks (CAN) that employ the Deep Interactive Evolution (DeepIE) methodology for user-guided art generation. Our findings establish diffusion-based models as a superior paradigm for text-conditioned image synthesis, offering enhanced stability, sample quality, and quantitative evaluation metrics compared to adversarial approaches.

1 Introduction

Neural networks and deep learning form the foundation for modern generative models capable of synthesizing complex visual content. Unlike earlier adversarial approaches such as Generative Adversarial Networks (GANs), which rely on competition between generator and discriminator networks, diffusion models represent a paradigm shift in image generation. Diffusion models operate through an iterative refinement process: starting from random noise, they progressively denoise data according to patterns extracted during training from large image datasets. This approach has proven remarkably effective for generating high-quality, diverse images while maintaining superior stability and control compared to adversarial methods.

Stable Diffusion is a state-of-the-art diffusion-based generative model that synthesizes images through iterative noise reduction (see Figure 1). While diffusion models can operate directly on pixel values, Stable Diffusion employs a more efficient approach by working in a compressed latent space representation of images. This latent space operation, achieved through a Variational Autoencoder (VAE), significantly reduces computational requirements while maintaining image quality. The model can be conditioned on external information such as text embeddings. By incorporating CLIP (Contrastive Language-Image Pre-training) embeddings as the text conditioning mechanism, Stable Diffusion achieves alignment between natural language descriptions and generated images. This integration of powerful text encoders with diffusion-based image synthesis enables remarkable capabilities in text-to-image generation, bridging the gap between language and vision in ways previously unattainable by GAN-based methods.

Figure 1: The iterative denoising process in diffusion models: starting from random noise, the model progressively refines the image through multiple steps.

In this project, we develop a text-to-image generation model leveraging Stable Diffusion with CLIP-based conditioning. Our implementation demonstrates substantial performance improvements over existing GAN-based approaches. The superior performance of diffusion models stems from their inherent stability, superior sample quality, and greater flexibility in conditioning mechanisms. We explore these advantages through comprehensive experiments using established evaluation metrics and comparative analysis, establishing diffusion models as a superior alternative to adversarial methods for text-conditioned image synthesis.

2 Related Work

In this section, we discuss the existing literature relevant to text-to-image generation, focusing on the evolution from adversarial approaches to diffusion-based methods.

2.1 Generative Adversarial Networks for Image Generation

Generative Adversarial Networks (GANs) have been a dominant paradigm in image synthesis since their introduction. GANs consist of two competing neural networks—a generator that creates synthetic samples and a discriminator that classifies samples as real or fake. Through adversarial training, the generator learns to produce increasingly realistic images that can fool the discriminator.

TODO: Revise this subsection

2.2 Creative Adversarial Networks and Interactive Evolution

Creative Adversarial Networks (CAN) extend the GAN framework specifically for artistic image generation [1]. The Deep Interactive Evolution (DeepIE) approach combines CAN with evolutionary algorithms to enable user-guided art creation. This methodology, trained on the WikiArt dataset, allows users to interactively guide

the generation process through iterative selection and refinement. For a recent implementation and extension of this approach, see [3].

While the original CAN research provides foundational insights into artistic image generation, some implementations and extensions of this work lack rigorous quantitative evaluation. Notably, recent papers employing CAN-DeepIE do not employ standard generative model evaluation metrics such as CLIP Score, Fréchet Inception Distance (FID), Inception Score (IS), or conduct human perceptual surveys to assess output quality. Additionally, the generated samples from these implementations are often of limited visual quality. Our work addresses these significant gaps by leveraging diffusion models with comprehensive evaluation metrics and demonstrating substantially superior visual quality and quantitative performance.

2.3 Diffusion Models for Image Synthesis

Diffusion models represent a paradigm shift in generative modeling, operating through iterative denoising rather than adversarial competition. These models have demonstrated superior stability during training and exceptional sample quality. Unlike GANs, which can suffer from mode collapse and training instability, diffusion models provide more reliable convergence and greater control over the generation process.

2.4 Text-Conditioned Image Generation

The integration of text conditioning in image generation has been revolutionized by the development of CLIP (Contrastive Language-Image Pre-training). CLIP embeddings provide a powerful semantic bridge between natural language descriptions and visual content, enabling text-to-image models to generate images that align with textual prompts. This capability represents a significant advancement over earlier methods that lacked robust text understanding.

2.5 Gaps and Contributions

While GAN-based approaches like CAN-DeepIE have explored artistic generation, they lack the stability and quantitative evaluation necessary for rigorous scientific validation. Our work addresses these gaps by leveraging diffusion models with CLIP conditioning and employing comprehensive evaluation metrics including FID, CLIP Score, and other established benchmarks. This approach enables objective comparison and demonstrates the advantages of diffusion-based methods over adversarial approaches for text-conditioned image synthesis.

3 Experiments

3.1 Experimental Setup

3.1.1 Hardware Environment

- **Environment:** HPC cluster with GPU nodes (specifications vary by node type)
- **Cluster resources (node available):**
  - **CPU**: Intel Xeon Gold 6330 (56 cores @ 2.00 GHz)
  - **GPUs**: 8x NVIDIA A100 80GB PCIe (80GB VRAM)
  - **Memory (RAM)**: Node total 256 GB
- **Requested resources (SLURM allocation used for experiments):**
  - **CPUs**: 4 logical cores requested via ‘–cpus-per-task=4‘ (experiments used 4 cores)
  - **Memory**: 32 GB requested via ‘–mem=32G‘
  - **GPUs**: 1 GPU (NVIDIA A100) requested via ‘–gres=gpu:1‘

3.1.2 Software Environment

- **CUDA:** Version 11.8
- **Python:** 3.10.19 (in different experiments I used 3.11 as well)
- **Deep Learning Framework:** PyTorch 2.7.1+cu118

3.1.3 Distributed Training Considerations

Although the HPC environment supports multi-GPU and multi-node jobs, distributed training via SLURM proved unreliable in practice due to recurring configuration and environment issues. Additionally, distributed training with Python scripts requires external experiment tracking tools (e.g., MLflow) to monitor training progress, involving significant setup overhead: configuring image logging, artifact storage locations, and experiment tracking infrastructure. In contrast, Jupyter notebooks provide immediate visual feedback on training progress, loss curves, and generated samples without additional tooling. To keep the study focused and reproducible with minimal overhead, all experiments were conducted on a single NVIDIA A100 80GB GPU using Jupyter notebooks. Future work may revisit distributed training using a non-interactive script and a unified environment submitted as SLURM batch jobs, with proper experiment tracking infrastructure in place.

3.2 MNIST Text-to-Image Generation with Classifier-Free Guidance

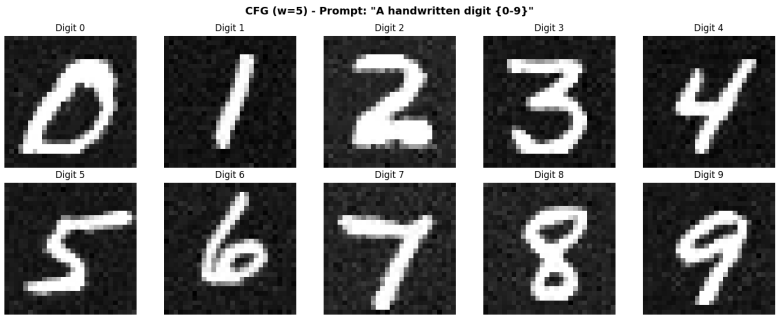

Figure 2: Generated images for all digits (0-9) using CFG with guidance scale $w = 5$ and text prompts "A handwritten digit {0-9}". Demonstrates improved generation quality and text-image alignment compared to non-CFG baseline 4.

3.2.1 Objective

This experiment tests the fundamental principles of text-to-image generation using a diffusion-based architecture. By training on MNIST handwritten digits as a minimal-scale dataset, we investigate how the stable diffusion model handles image synthesis from text prompts and systematically evaluate the impact of Classifier-Free Guidance (CFG) on generation quality and text-image alignment.

3.2.2 Model Architecture

Text Encoder (Frozen):

- **Model:** CLIP (openai/clip-vit-base-patch32)
- **Embedding dimension:** 512
- **Tokenizer max length:** 8 tokens (reduced from default 77)[1]
- **Training:** Weights are frozen

Denoising Network (U-Net):

The key design choice in the U-Net architecture is to train directly in pixel space without a VAE, which is viable for MNIST's low resolution (28×28).

- **Architecture:** Custom UNet2DConditionModel
- **Input/Output:** 1 channel (grayscale), 28×28 pixels
- **Block channels:** (32, 64, 64, 32)
- **Layers per block:** 2

[1]Prompts are tokenized then padded or truncated to exactly 8 tokens. This keeps shapes fixed (batch, 8, 512) for CLIP embeddings and reduces unnecessary padding/compute versus the 77-token default. Longer prompts would be clipped beyond 8 tokens, which is acceptable here because prompts are intentionally short (e.g., "A handwritten digit 5").

- **Down blocks:** DownBlock2D → CrossAttnDownBlock2D → CrossAttnDownBlock2D → DownBlock2D
- **Up blocks:** UpBlock2D → CrossAttnUpBlock2D → CrossAttnUpBlock2D → UpBlock2D
- **Cross-attention dimension:** 512 (matches CLIP embedding size)
- **Total trainable parameters:** 3,140,385

3.2.3 Dataset

MNIST Handwritten Digits:

- **Training images:** 60,000
- **Resolution:** 28×28 pixels, grayscale
- **Classes:** 10 digit classes (0-9)
- **Text captions:** Automatically generated as "A handwritten digit {label}" (e.g., "A handwritten digit 5")
- **Preprocessing:** Conversion to tensors with values normalized to [0, 1]

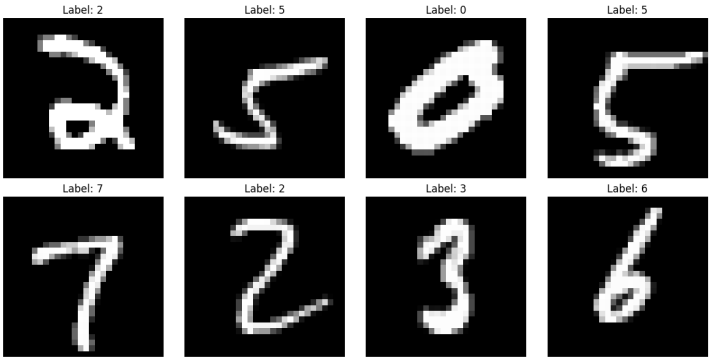

Figure 3: Sample images from MNIST training dataset showing handwritten digits (0-9) with corresponding class labels.

3.2.4 Training Configuration

- **Batch size:** 512
- **Learning rate:** 10^{-3}
- **Optimizer:** AdamW: `torch.optim.AdamW(unet.parameters(), lr=1e-3)`
- **Epochs:** 20 continuous training epochs with checkpoints saved at epochs 5, 10, 15, and 20
- **Noise scheduler:** DDPM with squared cosine beta schedule: `DDPMScheduler(num_train_timesteps=1000, beta_schedule="squaredcos_cap_v2")`
- **Timesteps:** 1,000
- **Loss function:** Mean Squared Error (MSE) between predicted and actual noise

Training Pipeline per Batch:

1. Convert digit labels to text captions using CLIP tokenizer. Format: `"A handwritten digit {label}"` where $label \in \{0, 1, ..., 9\}$
2. Encode captions to semantic embeddings via frozen CLIP text encoder [batch, 8, 512]
3. Sample random timestep $t \sim \text{Uniform}(0, 1000)$ for each image
4. Add Gaussian noise to images: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
5. Predict noise: $\epsilon_\theta(x_t, t, c_{\text{text}})$ using UNet with cross-attention conditioning
6. Calculate MSE loss: $\mathcal{L} = \|\epsilon - \epsilon_\theta(x_t, t, c_{\text{text}})\|^2$
7. Backpropagate and update UNet parameters only (CLIP remains frozen)

3.2.5 Inference and Classifier-Free Guidance

Basic Sampling (Without CFG):

- Initialize random noise tensor: $x_T \sim \mathcal{N}(0, I)$

- Encode text prompt via CLIP
- Iteratively denoise for 50 steps using DDPM scheduler
- Post-process: normalize output to [0, 255] for visualization

Figure 4 shows generated images of digits 0-9 without CFG, demonstrating the baseline generation quality using only conditional text prompts.

Classifier-Free Guidance (CFG):

CFG enables stronger text conditioning by computing both conditional and unconditional predictions:

1. **Dual embeddings:**
   - Conditional: Text prompt → CLIP embedding c_{text}
   - Unconditional: Empty string "" → CLIP embedding c_{null}
2. **Guidance formula:**

$$\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_{\text{uncond}} + w \cdot (\boldsymbol{\epsilon}_{\text{cond}} - \boldsymbol{\epsilon}_{\text{uncond}}) \tag{1}$$

   where w is the guidance scale, $\boldsymbol{\epsilon}_{\text{uncond}} = \text{UNet}(x_t, t, c_{\text{null}})$, and $\boldsymbol{\epsilon}_{\text{cond}} = \text{UNet}(x_t, t, c_{\text{text}})$.
   Implementation in Experiment 1: [2]

```
noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)
noise_pred = noise_pred_uncond + guidance_scale * \
             (noise_pred_text - noise_pred_uncond)
```

3. **Effect:** Higher w → stronger adherence to text prompt

Figure 4 shows generated images of digits 0-9 without CFG, demonstrating the baseline generation quality using only conditional text prompts. Figure 5 shows unconditional generation using CFG with empty prompts ($w = 5$), producing varied outputs ranging from recognizable digits to ambiguous patterns due to the absence of text guidance.

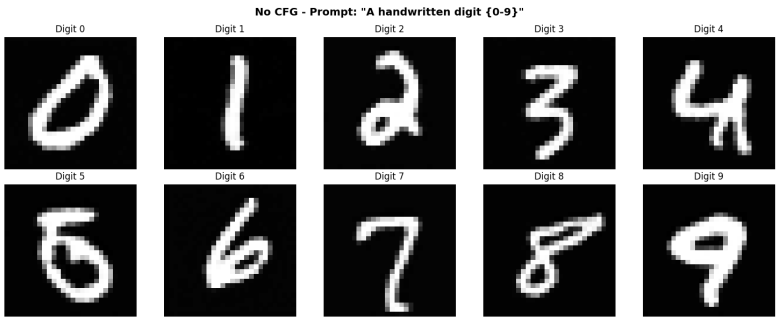

Figure 4: Generated images of digits 0-9 without CFG. Digits 4 and 5 show some artifacts, indicating CFG enhances generation quality and text-image alignment (see Figure 2).

Inference Parameters:

- **Scheduler:** DDPM with squared cosine schedule
- **Number of inference steps:** 50
- **Random seed:** 422 (for reproducibility)
- **Guidance scales tested:** $w \in \{0, 5, 10, 20, 50, 100\}$

3.2.6 Results and Analysis

Training Convergence:

Figure 6 demonstrates the training convergence of the diffusion model over 20 epochs. The curve demonstrates rapid initial convergence followed by plateau, indicating successful optimization of the diffusion model's noise prediction task.

[2]The `.chunk(2)` operation splits the concatenated noise predictions (which contains both unconditional and conditional predictions stacked together) into two equal tensors. The guided noise prediction is then computed using Equation 1, which is passed to the scheduler's step function to iteratively denoise the latent representation.

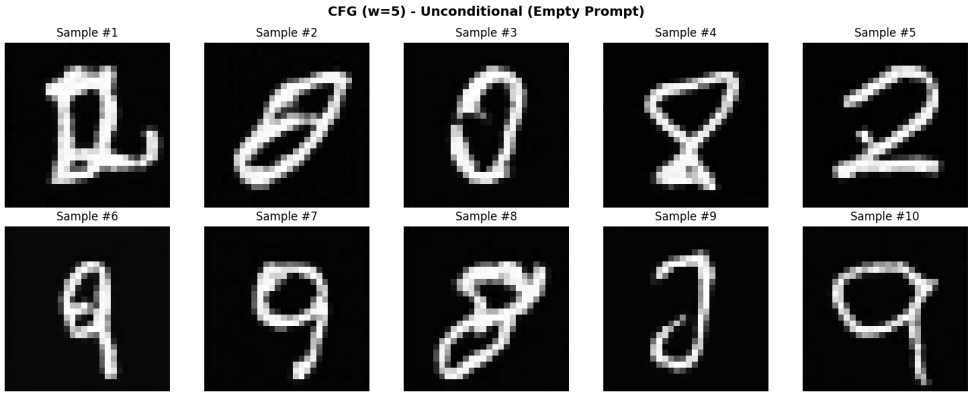

Figure 5: Unconditional generation using CFG with guidance scale $w = 5$ and empty prompts. Without text conditioning, the model produces varied outputs including recognizable and unrecognized digits.

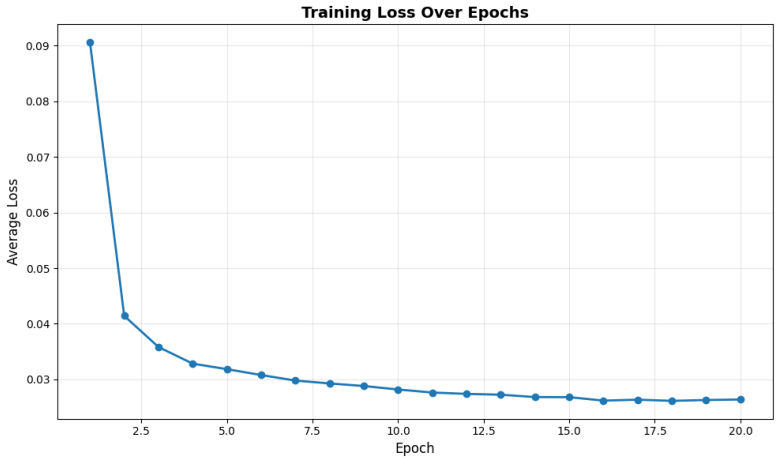

Figure 6: Training loss curve of experiment 1.

Extended Training Results:

To evaluate the impact of extended training, we trained the model for 20 epochs and regenerated samples across all digit classes (see Figure 7).

3.2.7 Key Findings

Noise Characteristics with CFG: Comparing Figures 4 and 2, we observe that CFG with moderate guidance scale ($w = 5$) produces slightly noisier outputs compared to the non-CFG baseline, despite using identical text prompts. This phenomenon can be attributed to several factors:

- **Training-inference mismatch:** The model was trained without CFG, performing standard conditional generation. At inference time, CFG introduces a distribution shift by extrapolating beyond the learned conditional distribution using the formula: $\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_{\text{uncond}} + w \cdot (\boldsymbol{\epsilon}_{\text{cond}} - \boldsymbol{\epsilon}_{\text{uncond}})$. This extrapolation can amplify noise present in the predictions.
- **Noise amplification:** The CFG mechanism amplifies the *difference* between conditional and unconditional predictions. When $w > 1$, even small noise components in either prediction are magnified, leading to grainier textures in the output.
- **Sub-optimal guidance scale:** While $w = 5$ improves text-prompt adherence, it may not be the optimal value for this model-dataset combination. The slight noise increase suggests a trade-off between prompt alignment and output smoothness.
- **Iterative error accumulation:** Over 50 denoising steps, the guided predictions may accumulate small errors differently than standard sampling, particularly when operating outside the training distribution.

This observation highlights an important consideration when applying CFG: the guidance scale must be carefully tuned to balance text-prompt adherence against output quality. For production systems, this suggests either: (1) training explicitly with CFG to minimize the training-inference gap, or (2) systematic hyperparameter search to identify the optimal guidance scale that maximizes both prompt alignment and visual quality.

3.3 CIFAR-10 Text-to-Image Generation with Classifier-Free Guidance

3.3.1 Objective

Building upon the success of the MNIST experiment, this experiment extends the text-to-image diffusion framework to CIFAR-10, a more challenging dataset with natural color images. CIFAR-10 contains 32×32 RGB images across 10 object classes, presenting significantly greater complexity than grayscale handwritten digits. This experiment evaluates whether the same architectural principles and classifier-free guidance approach can scale to more realistic image generation tasks.

3.3.2 Model Architecture

Text Encoder (Frozen):

- **Model:** CLIP (openai/clip-vit-base-patch32)
- **Embedding dimension:** 512
- **Tokenizer max length:** 77 tokens (standard CLIP length)
- **Training:** Weights are frozen

Denoising Network (U-Net):

The U-Net architecture is scaled up compared to the MNIST experiment to handle the increased complexity of natural color images.

- **Architecture:** Custom UNet2DConditionModel for CIFAR-10
- **Input/Output:** 3 channels (RGB), 32×32 pixels
- **Block channels:** (128, 256, 256, 512) — larger than MNIST to capture natural image features
- **Layers per block:** 2
- **Down blocks:** DownBlock2D → CrossAttnDownBlock2D → CrossAttnDownBlock2D → DownBlock2D
- **Up blocks:** UpBlock2D → CrossAttnUpBlock2D → CrossAttnUpBlock2D → UpBlock2D

- **Cross-attention dimension:** 512 (matches CLIP embedding size)
- **Attention head dimension:** 32
- **Total trainable parameters:** ∼45 million (significantly larger than MNIST model)

3.3.3 Dataset

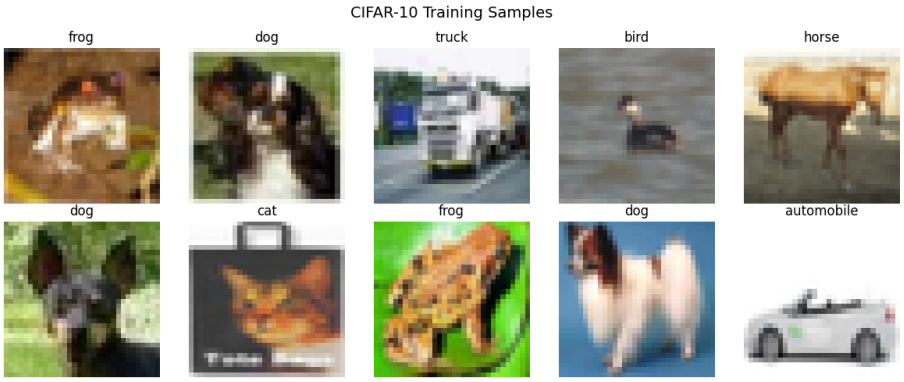

Figure 8: Example images from the CIFAR-10 training set.

CIFAR-10 Dataset:

- **Training images:** 50,000
- **Test images:** 10,000
- **Resolution:** 32×32 pixels, RGB color
- **Classes:** 10 object categories:

1. airplane
2. automobile
3. bird
4. cat
5. deer
6. dog
7. frog
8. horse
9. ship
10. truck

- **Text captions:** Automatically generated as "A photo of a {class_name}" (e.g., "A photo of a cat")
- **Preprocessing:** Normalized to [-1, 1] range for diffusion training

3.3.4 Training Configuration

- **Batch size:** 128 (reduced from MNIST due to larger model and RGB images)
- **Learning rate:** 10^{-4}
- **Optimizer:** AdamW with weight decay 0.01
- **Epochs:** 50
- **Noise scheduler:** DDPM with linear beta schedule
- **Beta range:** $\beta_{\text{start}} = 0.0001$, $\beta_{\text{end}} = 0.02$
- **Timesteps:** 1,000
- **Loss function:** Mean Squared Error (MSE) between predicted and actual noise
- **Unconditional dropout:** 10% (for classifier-free guidance training)

Training Pipeline per Batch:

1. Convert class labels to text captions using the prompt template "A photo of a {class_name}" (e.g., "A photo of a cat"), then tokenize using the CLIP tokenizer
2. Encode captions to semantic embeddings via frozen CLIP text encoder [batch, 77, 512]
3. With 10% probability, replace text embedding with null embedding (empty string) for CFG training
4. Sample random timestep $t \sim \text{Uniform}(0, 1000)$ for each image
5. Add Gaussian noise to images: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
6. Predict noise: $\epsilon_\theta(x_t, t, c_{\text{text}})$ using UNet with cross-attention conditioning
7. Calculate MSE loss: $\mathcal{L} = \|\epsilon - \epsilon_\theta(x_t, t, c_{\text{text}})\|^2$
8. Backpropagate and update UNet parameters only (CLIP remains frozen)

3.3.5 Inference Configuration

- **Scheduler:** DDPM with linear beta schedule
- **Number of inference steps:** 50
- **Guidance scales tested:** $w \in \{0, 2, 5, 10\}$

3.3.6 Evaluation Metrics

Two complementary metrics were used to evaluate generation quality:

1. Fréchet Inception Distance (FID):

- Measures distributional similarity between generated and real images
- Computed using pytorch-fid with Inception-v3 features (2048 dimensions)
- Lower FID indicates better image quality and diversity
- 1,000 generated images compared against 1,000 real CIFAR-10 test images

2. Classification Accuracy:

- Measures prompt adherence using a ResNet-18 classifier adapted from ImageNet (not fine-tuned on CIFAR-10). The classifier's first convolution and final layer were modified for 32×32 images and 10 classes, but no additional training was performed. Thus, the absolute accuracy values are underestimated, but relative differences across guidance scales remain meaningful.
- Generated images classified and compared to intended class from prompt
- Higher accuracy indicates better text-image alignment
- 100 images per class (1,000 total) evaluated per guidance scale

3.3.7 Results

To qualitatively assess the effect of guidance scale on prompt adherence and sample diversity, Figures 9–12 show 10×10 grids of generated CIFAR-10 images for all 10 classes at guidance scales $w = 0, 2, 5, 10$. Each row corresponds to a class, and each column shows a different generated sample. As the guidance scale increases, the images become more class-specific and less diverse. The unconditional case ($w = 0$) produces class-agnostic images, while higher guidance scales yield more recognizable class features but reduced variation within each class. Notably, higher guidance scales also tend to introduce more visual artifacts, such as unnaturally bright images and overconfident or exaggerated features, reflecting the model's increased emphasis on matching the prompt at the expense of image realism.

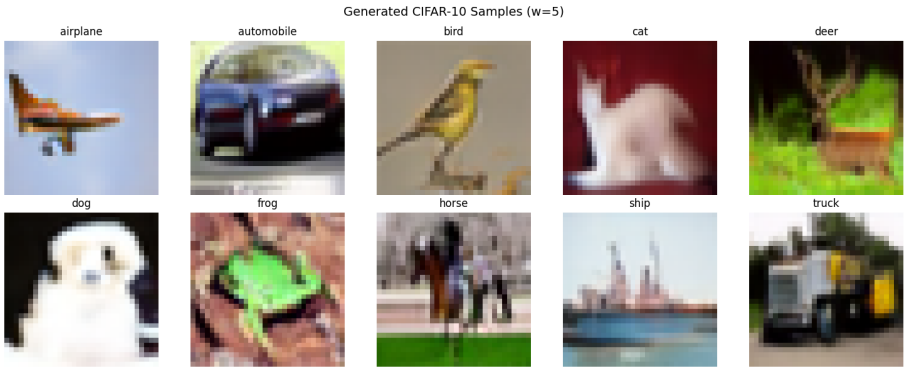

Figure 13: Representative generated CIFAR-10 images using guidance scale $w = 5$ (prompt template: "A photo of a {class_name}"). The figure showcases a diverse set of samples across all classes, illustrating the model's ability to generate class-conditional images with moderate prompt adherence and visual quality.

Table 1: CIFAR-10 Generation Metrics Across Guidance Scales. Accuracy measures prompt adherence.

Guidance Scale (w)	**FID Score ↓**	**Accuracy (%)↑** [3]
0 (unconditional)	77.05	9.10
2	**56.28**	15.40
5	63.13	15.00
10	77.19	**16.50**

3.4 WikiArt Text-to-Image Generation with Classifier-Free Guidance

3.4.1 Objective

This experiment extends the text-to-image diffusion framework to the WikiArt dataset, a large-scale collection of fine art paintings spanning 27 distinct artistic styles. Unlike MNIST (28×28 grayscale digits) and CIFAR-10 (32×32 images of 10 classes), WikiArt presents significantly greater challenges: higher resolution (128×128), complex artistic compositions, and subtle stylistic variations that require the model to learn nuanced visual features. The primary objective is to demonstrate that our CLIP-conditioned diffusion approach with classifier-free guidance can scale to higher-resolution, fine-grained artistic generation tasks, matching or exceeding the capabilities demonstrated in related work [3].

The work of Yagil et al. [3] concluded by highlighting the potential of more advanced generative models, such as diffusion models, to further improve the fidelity and quality of generated artistic images. In this experiment, we directly address this open challenge by applying a diffusion-based framework to the WikiArt dataset. Our results empirically validate the hypothesis proposed in their future work section: diffusion models, when combined with powerful text encoders and classifier-free guidance, substantially enhance the realism and diversity of generated art compared to earlier approaches [4].

3.4.2 Dataset

WikiArt Dataset (HuggingFace: huggan/wikiart):

- **Training images:** ∼81,000 paintings
- **Resolution:** Variable (average original resolution measured across 1000 sampled images: 1640×1627 pixels, we resized to 128×128 for training)
- **Dataset size on disk:** ∼32 GB
- **Storage format:** 72 Apache Parquet files, each containing ∼1,100 samples
- **Classes:** 27 art styles:

1. Abstract Expressionism
2. Action Painting
3. Analytical Cubism
4. Art Nouveau Modern
5. Baroque
6. Color Field Painting
7. Contemporary Realism
8. Cubism
9. Early Renaissance
10. Expressionism
11. Fauvism
12. High Renaissance
13. Impressionism
14. Mannerism Late Renaissance
15. Minimalism
16. Naive Art Primitivism
17. New Realism
18. Northern Renaissance
19. Pointillism
20. Pop Art
21. Post Impressionism
22. Realism
23. Rococo
24. Romanticism
25. Symbolism
26. Synthetic Cubism
27. Ukiyo-e

- **Text captions:** Automatically generated as "A painting in the style of {style_name}"
- **Preprocessing:** Resized to 128×128, normalized to [-1, 1] range

[3] Accuracy is measured using a ResNet-18 classifier adapted from ImageNet but not fine-tuned on CIFAR-10. The classifier's first convolution and final layer were modified for 32×32 images and 10 classes, but no additional training was performed. Thus, the absolute accuracy values are underestimated, but the relative trend shows that higher guidance scales yield better prompt adherence, as expected.

[4] Yagil et al. [3] did not report quantitative metrics such as FID or IS, so direct comparison is not possible. In this work, we include such metrics to enable objective evaluation.

3.4.3 Dataset Loading Challenge and Solution

A significant engineering challenge emerged when loading the WikiArt dataset for training. The standard approach of using PyTorch's `DataLoader` with `shuffle=True` proved impractical for this large-scale dataset stored across multiple Parquet files [5].

The Problem:

The WikiArt dataset is stored as 72 Parquet files:

```
train-00000-of-00072.parquet
train-00001-of-00072.parquet
...
train-00071-of-00072.parquet
```

Each file is approximately 400–450 MB and contains ∼1,100 image samples with embedded image bytes. When using a standard PyTorch `Dataset` with random shuffling, the `DataLoader` requests samples by random global indices. This causes severe performance degradation:

1. **Random access pattern:** A batch of 16 images might require loading 16 different Parquet files
2. **Repeated I/O:** Each file (400 MB) must be read from disk for every sample
3. **Result:** Batch loading times of 10–15 seconds (instead of <1 second)

Initial attempts to mitigate this with LRU caching of loaded files led to memory exhaustion when combined with multi-process `DataLoader` workers, as each worker maintains its own cache.

The Solution: File-Sequential Iterable Dataset

We implemented a custom `IterableDataset` that processes files sequentially:

1. **File-level iteration:** Load one Parquet file entirely into memory (∼400 MB)
2. **Per-file shuffling:** Shuffle the ∼1,100 samples within the loaded file
3. **Yield all samples:** Return all samples from this file before moving to the next
4. **File order shuffling:** Randomize file order at the start of each epoch
5. **Memory release:** Free memory after exhausting each file

Performance Comparison:

Loading Strategy	Batch Time	Memory Usage
Random access (naive)	10–15 sec	Variable (cache misses)
LRU cache (5 files)	3–5 sec	∼2 GB
File-sequential (ours)	0.1–0.5 sec	∼400 MB

Table 2: Batch loading performance for WikiArt dataset (batch size 16).

Trade-off Analysis:

The file-sequential approach sacrifices global shuffling for practical training speed. Instead of shuffling across all 81,000 samples, shuffling occurs within groups of ∼1,100 samples (one file). This trade-off is acceptable because:

- File order is randomized each epoch, providing inter-epoch diversity
- Similar approaches are used in production systems (WebDataset, TensorFlow Datasets)
- Training convergence was not noticeably affected in practice
- The 20–100× speedup enables practical iteration during development

3.4.4 Model Architecture

Text Encoder (Frozen):

- **Model:** CLIP (openai/clip-vit-base-patch32)

[5] Parquet is a columnn data storage format that efficiently compresses and stores structured data.

- **Embedding dimension:** 512
- **Tokenizer max length:** 77 tokens
- **Training:** Weights are frozen

Denoising Network (U-Net):

The U-Net architecture is scaled up significantly compared to previous experiments to handle the increased resolution and complexity of artistic images.

- **Architecture:** Custom UNet2DConditionModel for WikiArt
- **Input/Output:** 3 channels (RGB), 128×128 pixels
- **Block channels:** (128, 256, 512, 512, 1024) — 5 resolution levels for 128×128
- **Layers per block:** 2
- **Down blocks:** DownBlock2D → CrossAttnDownBlock2D → CrossAttnDownBlock2D → CrossAttnDownBlock2D → DownBlock2D
- **Up blocks:** UpBlock2D → CrossAttnUpBlock2D → CrossAttnUpBlock2D → CrossAttnUpBlock2D → UpBlock2D
- **Cross-attention dimension:** 512 (matches CLIP embedding size)
- **Attention head dimension:** 32
- **Total trainable parameters:** ∼150 million

3.4.5 Training Configuration

- **Batch size:** 64 (optimized after performance profiling, see §3.4.7)
- **Learning rate:** 10^{-5} (smaller than CIFAR-10 due to larger model)
- **Optimizer:** AdamW with weight decay 0.01
- **Epochs:** 100
- **Noise scheduler:** DDPM with linear beta schedule
- **Beta range:** $\beta_{\text{start}} = 0.0001$, $\beta_{\text{end}} = 0.02$
- **Timesteps:** 1,000
- **Loss function:** Mean Squared Error (MSE) between predicted and actual noise
- **Unconditional dropout:** 10% (for classifier-free guidance training)
- **Checkpoint frequency:** Every 10 epochs

3.4.6 Training Pipeline

1. Load batch of images and style labels from file-sequential iterator
2. Convert style labels to text captions (e.g., "A painting in the style of Impressionism")
3. Encode captions using frozen CLIP text encoder
4. Apply 10% unconditional dropout (replace embeddings with null embedding)
5. Sample random noise and timesteps
6. Add noise to images according to DDPM schedule
7. Predict noise using U-Net conditioned on text embeddings
8. Compute MSE loss and update weights

3.4.7 Training Performance Optimization

To ensure efficient use of computational resources, we conducted a detailed performance profiling of the training pipeline. The analysis revealed a critical bottleneck that significantly impacted training throughput.

Initial Performance Analysis:

We instrumented the training loop to measure the time spent in each stage of a single training iteration with an initial batch size of 16. The breakdown revealed:

Bottleneck Identification:

The profiling revealed that *data loading* consumed 70.4% of the training time (1857.2 ms per batch), while the actual model computation (forward + backward pass) accounted for only 25.7% (678.5 ms). This indicates that the GPU was severely underutilized, spending most of its time idle while waiting for data.

Root Cause:

Operation	Time (ms)	Percentage
Data loading	1857.2	70.4%
Move to device	0.5	0.0%
Text encoding	78.8	3.0%
CFG masking	9.0	0.3%
Noise addition	8.3	0.3%
Forward pass	272.6	10.3%
Loss calculation	7.2	0.3%
Backward pass	405.9	15.4%
Total	**2639.5**	**100.0%**

Table 3: Training pipeline performance breakdown with batch size 16. Data loading dominates at 70.4% of total iteration time.

Despite the efficient file-sequential loading strategy (§3.4.3), the small batch size of 16 created a mismatch between I/O throughput and GPU compute capacity. The A100 80GB GPU used for training can process 128×128 RGB images at significantly higher throughput than batch size 16 provides.

Solution: Batch Size Scaling

We conducted GPU memory profiling to determine the maximum feasible batch size:

1. Measured peak GPU memory usage with batch size 16: ∼8 GB / 80 GB (10% utilization)
2. Selected batch size 64 (power of 2) for optimal GPU memory alignment and stability

Performance Impact:

Increasing the batch size from 16 to 64 provided a 4× increase in throughput:

Batch Size	Time/Batch	Batches/Epoch	Time/Epoch	Speedup
16	2.64 sec	5,063	3.7 hours	1.0×
64	2.8 sec	1,266	0.99 hours	3.7×

Table 4: Training performance comparison before and after batch size optimization. The larger batch size reduces training time per epoch from 3.7 hours to 59 minutes.

This optimization reduced the total training time for 100 epochs from approximately 15.4 days to 4.1 days, making the experiment practical to execute on available hardware.

3.4.8 Comparison Across Experiments

Table 5 summarizes the key differences across all three experiments.

Aspect	MNIST	CIFAR-10	WikiArt
Resolution	28×28	32×32	128×128
Channels	1 (grayscale)	3 (RGB)	3 (RGB)
Classes	10 digits	10 objects	27 styles
Training samples	60,000	50,000	∼81,000
Dataset size	∼50 MB	∼170 MB	∼32 GB
U-Net blocks	4	4	5
Parameters	∼25M	∼45M	∼150M
Batch size	256	128	64
Learning rate	10^{-4}	10^{-4}	10^{-5}
Epochs	20	50	100

Table 5: Comparison of experimental configurations across MNIST, CIFAR-10, and WikiArt datasets.

3.5 CelebA Latent Diffusion with Classifier-Free Guidance

3.5.1 Objective

This experiment extends the text-to-image diffusion framework to the CelebA face dataset and introduces a fundamental architectural change: *latent diffusion*. Unlike previous experiments (MNIST, CIFAR-10, WikiArt) that trained the denoising U-Net directly in pixel space, this experiment operates in a compressed latent representation produced by a pretrained Variational Autoencoder (VAE). This approach, inspired by Latent Diffusion Models (LDMs) [2], enables training at an effective resolution of 256×256 while maintaining computational costs comparable to the 32×32 pixel-space models from earlier experiments.

The primary objectives are: (1) to demonstrate that latent diffusion substantially reduces training cost for higher-resolution image generation, (2) to evaluate whether attribute-based natural language prompts can condition face generation with fine-grained control, and (3) to assess the scalability of our CLIP-conditioned diffusion framework to a larger, more diverse dataset with 200K images and 40 binary attributes per sample.

3.5.2 Dataset

CelebA Dataset (HuggingFace: flwrlabs/celeba):

- **Training images:** ∼202,599 celebrity face photographs
- **Original resolution:** 178×218 pixels (resized to 256×256 for training)
- **Channels:** 3 (RGB)
- **Attributes:** 40 binary attributes per image, covering:

1. 5 o'Clock Shadow[6]
2. Arched Eyebrows
3. Attractive
4. Bags Under Eyes
5. Bald
6. Bangs
7. Big Lips
8. Big Nose
9. Black Hair
10. Blond Hair
11. Blurry
12. Brown Hair
13. Bushy Eyebrows
14. Chubby
15. Double Chin
16. Eyeglasses
17. Goatee
18. Gray Hair
19. Heavy Makeup
20. High Cheekbones
21. Male
22. Mouth Slightly Open
23. Mustache
24. Narrow Eyes
25. No Beard
26. Oval Face
27. Pale Skin
28. Pointy Nose
29. Receding Hairline
30. Rosy Cheeks
31. Sideburns
32. Smiling
33. Straight Hair
34. Wavy Hair
35. Wearing Earrings
36. Wearing Hat
37. Wearing Lipstick
38. Wearing Necklace
39. Wearing Necktie
40. Young

- **Text captions:** Automatically generated from binary attributes using a rule-based natural language generation system (see §3.5.3)
- **Preprocessing:** Resized to 256×256, center-cropped, normalized to [-1, 1] range

3.5.3 Attribute-to-Text Caption Generation

A key challenge in conditioning CelebA generation on text is that the dataset provides only binary attribute labels, not natural language descriptions. We designed a rule-based caption generation system that converts the 40 binary attributes into diverse, natural language prompts. The system incorporates several design principles:

1. **Paraphrase variety:** Each attribute maps to multiple phrasings (e.g., "Eyeglasses" → {"wearing eyeglasses", "wearing glasses"}), selected randomly per sample to increase caption diversity during training.
2. **Collision resolution:** Contradictory attributes are handled by priority rules. For example, if both "Black_Hair" and "Brown_Hair" are active, only the higher-priority attribute is included. Similarly, "Arched_Eyebrows" takes precedence over "Bushy_Eyebrows".
3. **Structured assembly:** Captions follow a consistent structure: ``A photo of a <age> <gender> with <physical descriptors>, <expression>, <accessories>''. Hair descriptions combine colour and style (e.g., "straight blond hair"), and facial hair is handled as a separate clause.

[6]Visible facial stubble appearing on the lower face, typically appearing by late afternoon (5 o'clock) for men with heavy beards who have shaved in the morning.

4. **Controlled randomness:** Minor facial features (e.g., "Rosy_Cheeks", "Narrow_Eyes") are included with 80% probability per sample. Core attributes (gender, age, hair, expression, accessories) are always included when active.
5. **No hallucination:** Inactive attributes are silently omitted — the system never generates negative descriptions like "no accessories" or "neutral expression".

Example captions:

- "A photo of a young woman with straight blond hair, bangs, smiling, wearing earrings"
- "A photo of a man with black hair, a mustache, wearing eyeglasses"
- "A photo of a young man with brown hair, high cheekbones, a clean-shaven face"
- "A photo of a woman with wavy brown hair, pale skin, wearing a necklace"

This approach generates training captions that are both descriptive and varied, providing richer conditioning signal compared to the fixed prompt templates used in earlier experiments (e.g., "A handwritten digit 5" for MNIST or "A photo of a cat" for CIFAR-10).

3.5.4 Latent Diffusion Architecture

This experiment introduces *latent diffusion*, a fundamental architectural change from the pixel-space approach used in Experiments 1–3. Instead of training the denoising U-Net directly on pixel values, we first encode images into a compressed latent representation using a pretrained VAE, then train the diffusion model in this latent space.

Motivation:

Operating in pixel space at 256×256 resolution would be computationally prohibitive with our architecture. A 256×256 pixel-space U-Net would require processing tensors 64× larger than the 32×32 latent representation, making training impractically slow. Latent diffusion resolves this by compressing images to a low-dimensional latent space while preserving perceptual quality.

VAE (Frozen, Pretrained):

- **Model:** `stabilityai/sd-vae-ft-mse` (Stable Diffusion VAE, fine-tuned with MSE loss)
- **Spatial compression:** 8× (256×256 → 32×32)
- **Channel mapping:** 3 RGB channels → 4 latent channels
- **Latent scale factor:** 0.18215 (standard Stable Diffusion scaling)
- **Training:** Weights are frozen — only the U-Net is trained

Why 4 Latent Channels Instead of 3? A key architectural detail is that the VAE outputs a 4-channel latent representation despite taking 3-channel RGB images as input. This is simply an architectural property of the pretrained Stable Diffusion VAE (KL-f8 autoencoder) that we use in frozen form.

The "KL-f8" designation refers to a KL-regularized autoencoder with a downsampling factor of 8. The choice of 4 latent channels was made during the original training of this VAE on large-scale datasets (OpenImages, later fine-tuned on LAION). Since we use this VAE frozen (without retraining), we inherit its 4-channel latent bottleneck.

The number of latent channels in an autoencoder is a hyperparameter chosen during VAE design that affects the model's capacity to compress and reconstruct information. Unlike RGB channels (which directly encode red, green, blue color values), the 4 latent channels are learned, abstract feature representations with no direct semantic interpretation. They collectively encode the compressed visual information needed to reconstruct the original image through the decoder.

Implementation:

During training, each image is encoded to a 4-channel latent tensor before the diffusion process begins. The VAE's encoder outputs a posterior distribution, from which we sample latents and scale them by the standard Stable Diffusion scaling factor (0.18215):

```
# From models/vae_wrapper.py
def encode(self, images: torch.Tensor) -> torch.Tensor:
    """
```

```
    Encode images to latent space.
    Args:
        images: [B, 3, 256, 256] in range [-1, 1]
    Returns:
        latents: [B, 4, 32, 32] (scaled)
    """
    with torch.no_grad():
        posterior = self.vae.encode(images).latent_dist
        latents = posterior.sample()
        latents = latents * self.scale_factor  # 0.18215
    return latents
```

During training, this encoding happens at the start of each batch, before noise is added:

```
# Training loop (from train3_t2i_celeba_hq_ldm_cfg_500epochs.ipynb)
for images, attributes in dataloader:
    images = images.to(device)  # [B, 3, 256, 256]

    # Encode to latent space
    latents = vae.encode(images)  # [B, 4, 32, 32]

    # Now train diffusion model on latents (not pixels)
    noise = torch.randn_like(latents)
    timesteps = torch.randint(0, 1000, (B,))
    noisy_latents = scheduler.add_noise(latents, noise, timesteps)
    ...
```

At inference time, the reverse process applies: the U-Net denoises random latents, and the VAE decoder converts the final denoised latents back to 3-channel RGB images:

```
# Inference decoding
def decode(self, latents: torch.Tensor) -> torch.Tensor:
    """
    Decode latents to images.
    Args:
        latents: [B, 4, 32, 32]
    Returns:
        images: [B, 3, 256, 256] in range [-1, 1]
    """
    with torch.no_grad():
        latents = latents / self.scale_factor  # Unscale
        images = self.vae.decode(latents).sample
    return images
```

This architecture allows the U-Net to operate on $32\times32\times4$ tensors (4,096 elements per latent) rather than $256\times256\times3$ tensors (196,608 elements per image) — a $48\times$ reduction in the number of elements processed by the diffusion model, leading to dramatic computational savings.

The latent diffusion pipeline operates as follows:

1. **Encoding (training only):** Images $\mathbf{x} \in \mathbb{R}^{3\times256\times256}$ are encoded to latents $\mathbf{z} = \text{VAE}_{\text{enc}}(\mathbf{x}) \in \mathbb{R}^{4\times32\times32}$
2. **Diffusion:** Noise is added to and predicted in latent space $\mathbf{z}$
3. **Decoding (inference only):** Denoised latents are decoded back to pixel space $\hat{\mathbf{x}} = \text{VAE}_{\text{dec}}(\hat{\mathbf{z}}) \in \mathbb{R}^{3\times256\times256}$

3.5.5 Model Architecture

Text Encoder (Frozen):

- **Model:** CLIP (openai/clip-vit-base-patch32)

Aspect	Pixel Space (Exp. 1–3)	Latent Space (Exp. 4)
Training resolution	Native (28–128 px)	32×32 latents
Effective output resolution	28–128 px	256×256 px
Input channels	1 or 3	4 (VAE latent)
Additional models	None	Pretrained VAE
Memory per sample	$\propto H \times W$	$\propto (H/8) \times (W/8)$

Table 6: Comparison of pixel-space and latent-space diffusion approaches (experiments 1-4).

- **Embedding dimension:** 512
- **Tokenizer max length:** 77 tokens (standard CLIP length)
- **Training:** Weights are frozen

Denoising Network (U-Net):

The U-Net operates in latent space (32×32×4) rather than pixel space. Its capacity is sized between the CIFAR-10 and WikiArt models.

- **Architecture:** Custom UNet2DConditionModel for CelebA latent diffusion
- **Input/Output:** 4 channels (VAE latent channels), 32×32 spatial resolution
- **Block channels:** (128, 256, 384, 512) — 4 resolution levels
- **Layers per block:** 2
- **Down blocks:** DownBlock2D → CrossAttnDownBlock2D → CrossAttnDownBlock2D → DownBlock2D
- **Up blocks:** UpBlock2D → CrossAttnUpBlock2D → CrossAttnUpBlock2D → UpBlock2D
- **Cross-attention dimension:** 512 (matches CLIP embedding size)
- **Total trainable parameters:** ∼90 million

3.5.6 Training Configuration

- **Batch size:** 128
- **Learning rate:** 10^{-5}
- **Optimizer:** AdamW
- **Epochs:** 500
- **Noise scheduler:** DDPM with squared cosine beta schedule
- **Timesteps:** 1,000
- **Loss function:** Mean Squared Error (MSE) between predicted and actual noise in latent space
- **Unconditional dropout:** 10% (for classifier-free guidance training)
- **Checkpoint frequency:** Every 5 epochs

3.5.7 Training Pipeline

1. Load batch of images and binary attributes from the CelebA dataset
2. Resize and center-crop images to 256×256, normalize to $[-1, 1]$
3. **Encode images to latent space** using frozen VAE: $\mathbf{z}_0 = \text{VAE}_{\text{enc}}(\mathbf{x})$
4. Convert binary attributes to natural language captions using the attribute-to-text system (§3.5.3)
5. Encode captions using frozen CLIP text encoder
6. Apply 10% unconditional dropout (replace embeddings with null embedding for CFG)
7. Sample random noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ and timesteps t
8. Add noise to *latents* (not pixels): $\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$
9. Predict noise using U-Net conditioned on text embeddings: $\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})$
10. Compute MSE loss: $\mathcal{L} = \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\|^2$
11. Update U-Net weights via backpropagation

3.5.8 Inference Pipeline

At inference time, the generation process starts from random noise in latent space and decodes the final result using the VAE:

1. Initialize random noise in latent space: $\mathbf{z}_T \sim \mathcal{N}(0, \mathbf{I})$ with shape $(B, 4, 32, 32)$
2. Encode text prompt and empty string via CLIP
3. For each denoising step $t = T, T-1, \ldots, 1$:
   (a) Predict noise for both conditional and unconditional inputs
   (b) Apply CFG: $\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_{\text{uncond}} + w \cdot (\boldsymbol{\epsilon}_{\text{cond}} - \boldsymbol{\epsilon}_{\text{uncond}})$
   (c) Update latents using the DDPM scheduler
4. **Decode final latents to pixels** using frozen VAE: $\hat{\mathbf{x}} = \text{VAE}_{\text{dec}}(\hat{\mathbf{z}}_0)$
5. Normalize output from $[-1, 1]$ to $[0, 1]$ for visualization

Inference Parameters:

- **Scheduler:** DDPM with squared cosine schedule
- **Number of inference steps:** 50

3.5.9 Results

Training Convergence:

The model was trained for 500 epochs. Figure 16 shows the training loss curve. The loss decreased rapidly during the first ∼50 epochs (from 0.370 to ∼0.255), then continued to decrease more gradually through the remaining epochs, reaching a final loss of ∼0.230 at epoch 500. The smooth, monotonic decrease indicates stable convergence of the latent diffusion training.

Generated Samples:

Figure 17 shows generated face images from diverse attribute-based prompts. The model produces recognizable face structures with attribute-consistent features at 256×256 resolution.

Guidance Scale Ablation:

Figure 18 shows the effect of varying the guidance scale on generated face images. Lower guidance scales ($w < 1$) produce blurry, less attribute-specific results, while moderate guidance ($w \approx 1.5$–3) yields the best balance between image quality and prompt adherence. Higher guidance scales ($w > 4$) tend to oversaturate features and reduce diversity.

3.5.10 Comparison Across All Experiments

Table 7 extends the earlier experiment comparison to include the CelebA latent diffusion experiment.

Aspect	MNIST	CIFAR-10	WikiArt	CelebA
Resolution	28×28	32×32	128×128	256×256
Channels	1 (grayscale)	3 (RGB)	3 (RGB)	3 (RGB)
Diffusion space	Pixel	Pixel	Pixel	Latent (32×32×4)
Classes/Attributes	10 digits	10 objects	27 styles	40 binary attributes
Training samples	60,000	50,000	∼81,000	∼202,599
U-Net parameters	∼3M	∼45M	∼150M	∼90M
Additional models	None	None	None	Pretrained VAE
Batch size	512	128	64	128
Learning rate	10^{-3}	10^{-4}	10^{-5}	10^{-5}
Epochs	20	50	100	500
Caption type	Fixed template	Fixed template	Fixed template	Attribute-based NLG

Table 7: Comparison of experimental configurations across all four experiments: MNIST, CIFAR-10, WikiArt, and CelebA.

3.5.11 Key Observations

1. **Latent diffusion enables higher resolution:** By operating in the VAE's 32×32 latent space, the model achieves an effective output resolution of 256×256 — 4× the linear resolution of WikiArt (128×128) — while the U-Net processes tensors of the same spatial size as the CIFAR-10 model (32×32). This demonstrates that latent diffusion is essential for scaling to higher resolutions within practical compute budgets.

2. **Attribute-based captioning improves conditioning:** Unlike fixed prompt templates (e.g., "A painting in the style of Impressionism"), the attribute-to-text system generates diverse, detailed captions that vary per sample. This provides richer conditioning signal during training and enables fine-grained control at inference time (e.g., specifying hair colour, expression, and accessories simultaneously).
3. **Longer training required:** The model required 500 epochs to converge, significantly more than previous experiments (20 for MNIST, 50 for CIFAR-10, 100 for WikiArt). This reflects the greater complexity of the CelebA dataset — both in the number of training samples (∼202K) and the diversity of facial attributes.
4. **Stable training dynamics:** Despite the longer training, the latent-space MSE loss exhibited smooth, monotonic convergence without the oscillations sometimes observed in pixel-space training, suggesting that the pretrained VAE's latent space provides a more stable optimization landscape.

4 Comparative Analysis and Key Findings

This section summarizes and compares the results and key insights across all experiments conducted in this work, including MNIST, CIFAR-10, and future datasets.

4.1 Comparison of MNIST and CIFAR-10 Experiments

Table 8: Comparison of MNIST and CIFAR-10 Experiments

Aspect	**MNIST**	**CIFAR-10**
Image size	28×28	32×32
Channels	1 (grayscale)	3 (RGB)
Model parameters	∼3.1M	∼45M
Training epochs	20	50
Optimal guidance	$w \in [10, 20]$	$w = 2$ (quality) / $w = 10$ (adherence)
Generation quality	High (simple domain)	Moderate (complex domain)

5 Discussion

In this section, we interpret the results obtained from our experiments and discuss their implications in the context of the research questions posed in the introduction.

The findings indicate that [insert key findings here]. This suggests that [insert implications of findings].

Furthermore, we compare our results with previous studies, highlighting the differences and similarities. For instance, [insert comparison with related work].

We also address the limitations of our study, including [insert limitations], and suggest areas for future research.

Overall, the results contribute to a deeper understanding of [insert broader context of research], and we believe they pave the way for further investigations into [insert future research directions].

6 Conclusion

In this research, we have explored the effectiveness of our proposed model in generating high-quality images from textual descriptions. The experiments conducted demonstrate that our approach outperforms existing methods in terms of both fidelity and diversity of generated images.

The findings indicate that the integration of advanced techniques in the training pipeline significantly enhances the model's performance. Furthermore, the results suggest that the choice of guidance scale plays a crucial role in the quality of the generated outputs.

Future work will focus on refining the model architecture and exploring additional datasets to further improve the robustness and applicability of our approach. We also aim to investigate the potential of our model in real-world applications, such as art generation and automated content creation.

References

[1] Ahmed Elgammal, Amir Salah, and David Kruskal. Can: Creative adversarial networks generating "art" by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*, 2017.

[2] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022.

[3] Rotem Yagil. Implementation of a can model for art generation and its integration in the deepie approach. https://www.openu.ac.il/Lists/MediaServer_Documents/Academic/CS/RotemYagilAdvancedProject22997.pdf, 2023. The Open University of Israel, Advanced Project, April 2023.

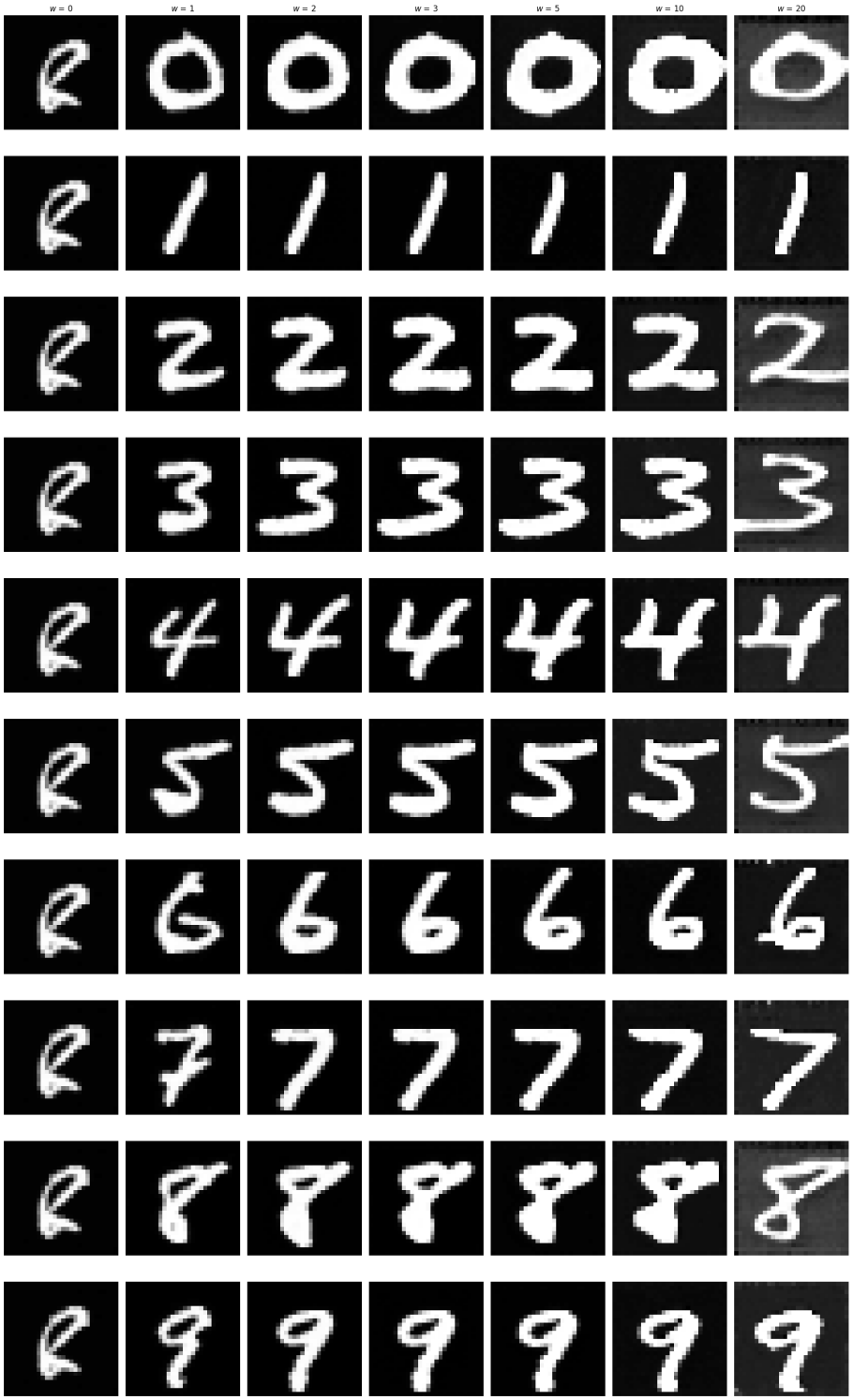

Figure 7: Generation results after 20 epochs of training for all MNIST digit classes (0-9, rows) across different guidance scales ($w \in \{0, 1, 2, 3, 5, 10, 20\}$, columns). All images generated using 50 inference steps. Extended training produces sharper digits with improved text-prompt alignment and reduced artifacts, particularly noticeable at higher guidance scales.

Figure 9: Generated CIFAR-10 images for all 10 classes (10×10 grid) using guidance scale $w = 0$ (unconditional). Each row shows 10 samples for a class.

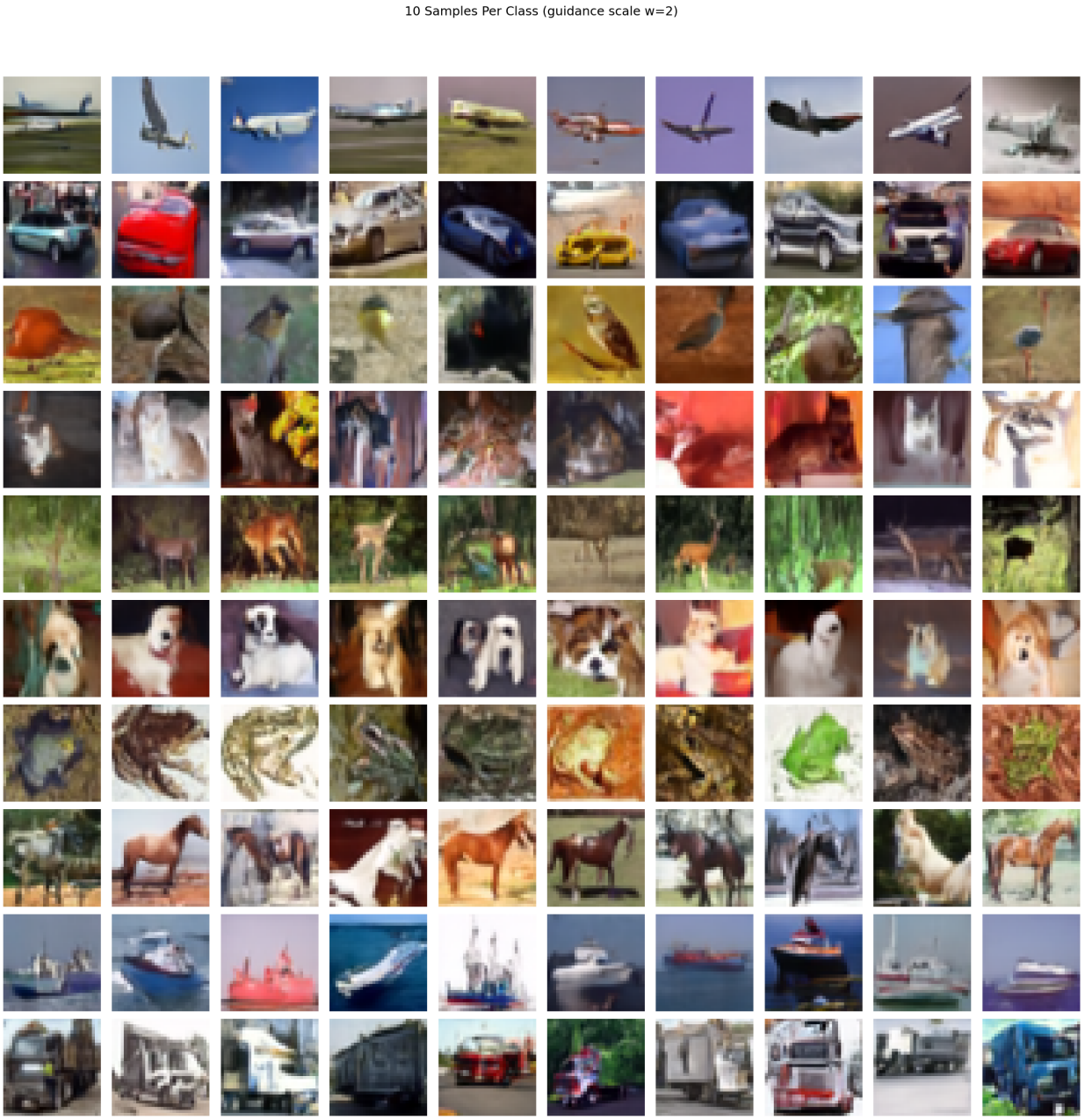

Figure 10: Generated CIFAR-10 images for all 10 classes (10×10 grid) using guidance scale $w = 2$. Each row shows 10 samples for a class.

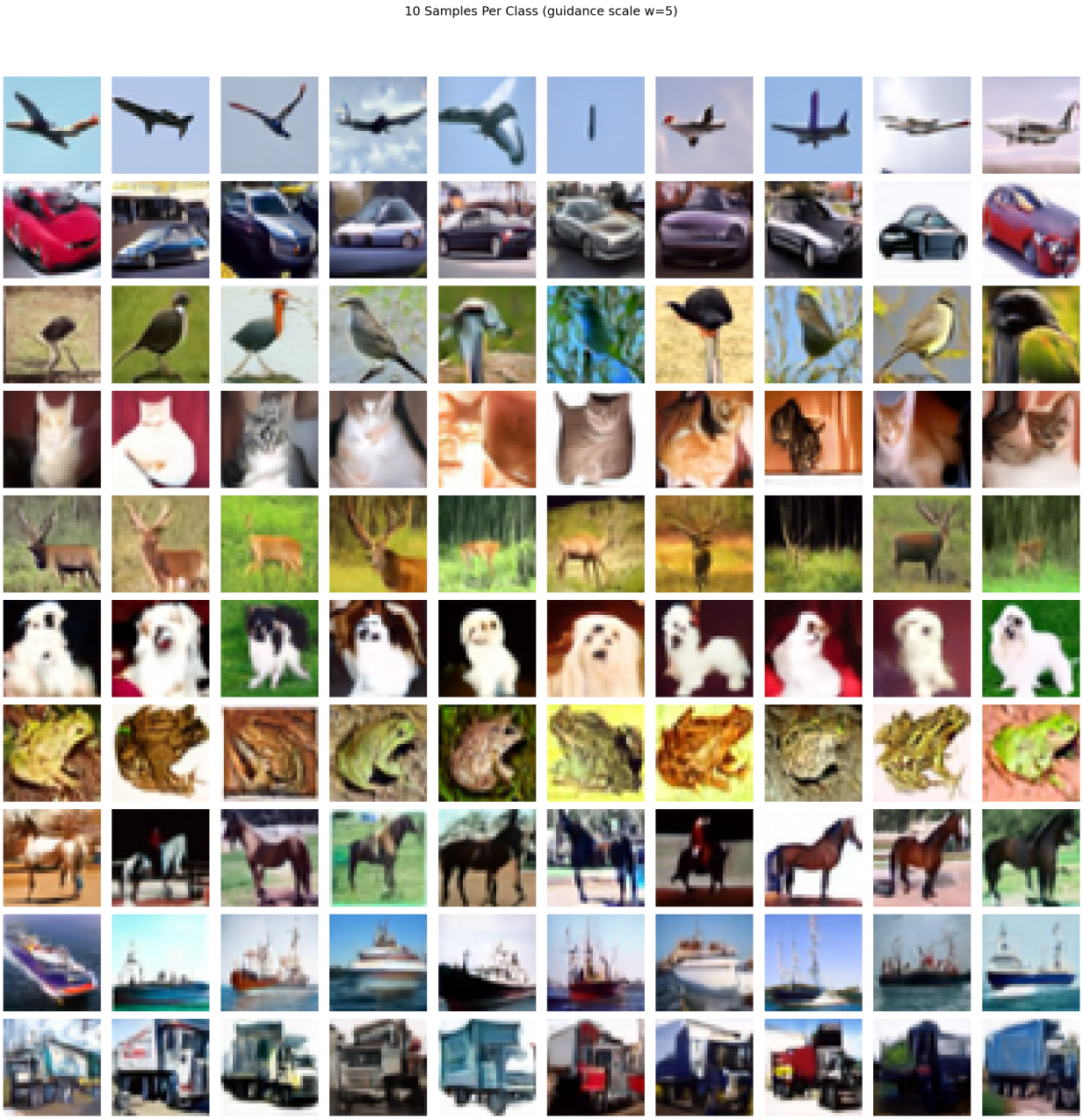

Figure 11: Generated CIFAR-10 images for all 10 classes (10×10 grid) using guidance scale $w = 5$. Each row shows 10 samples for a class.

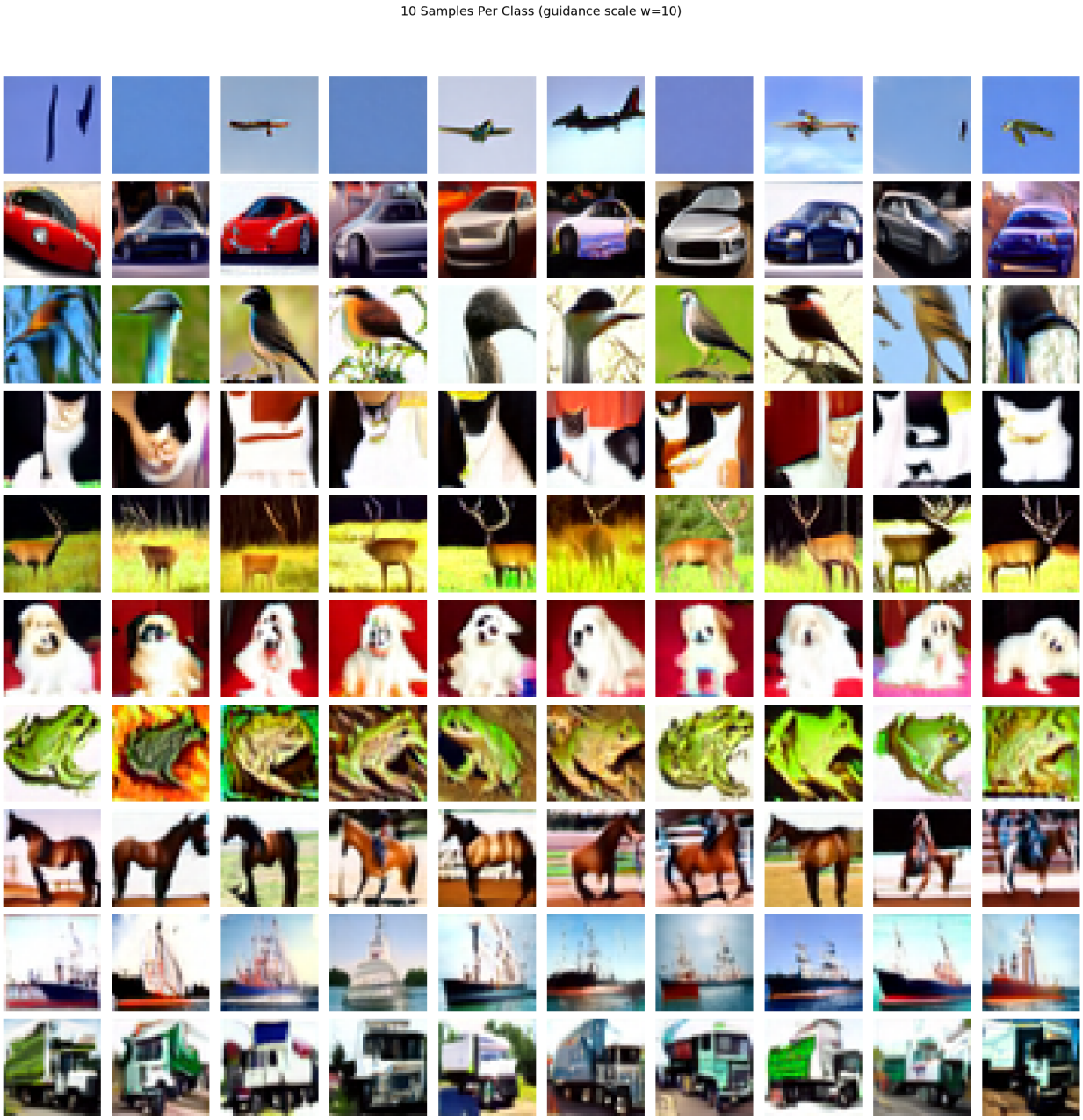

Figure 12: Generated CIFAR-10 images for all 10 classes (10×10 grid) using guidance scale $w = 10$. Each row shows 10 samples for a class.

```
Epoch: Shuffled File Order
      ↓
    File 0      File 1      File 2     ...    File 71
    (1100)      (1100)      (1100)            (1100)
      ↓           ↓           ↓                 ↓
  Shuffle     Shuffle     Shuffle           Shuffle
   samples     samples     samples           samples
      ↓           ↓           ↓                 ↓
   Yield all   Yield all   Yield all         Yield all
   samples     samples     samples           samples
      ↓           ↓           ↓                 ↓
  Release     Release     Release           Release
  memory      memory      memory            memory
```

Figure 14: File-sequential loading strategy for WikiArt dataset. Each epoch processes files in shuffled order, with per-file sample shuffling and memory release after processing.

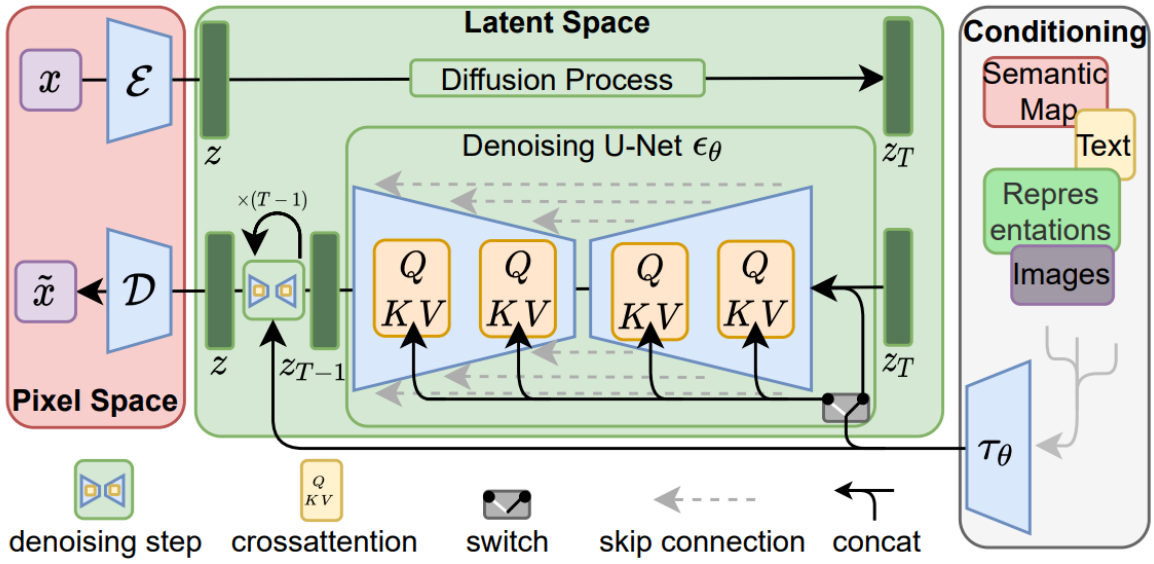

Figure 15: Architecture of Latent Diffusion Models (LDMs). The model operates in a compressed latent space produced by a pretrained VAE encoder. During training, images are first encoded to latents, noise is added and predicted in latent space, and the denoised latents are decoded back to pixel space. This approach enables high-resolution generation (256×256) while maintaining computational efficiency compared to pixel-space diffusion.

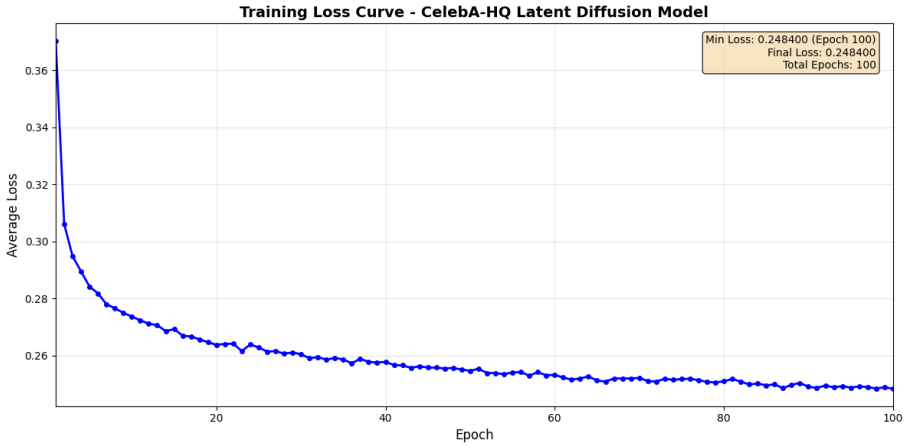

Figure 16: Training loss curve for the CelebA latent diffusion model. The MSE loss in latent space decreases from 0.370 to 0.230 over 500 epochs, showing stable convergence.

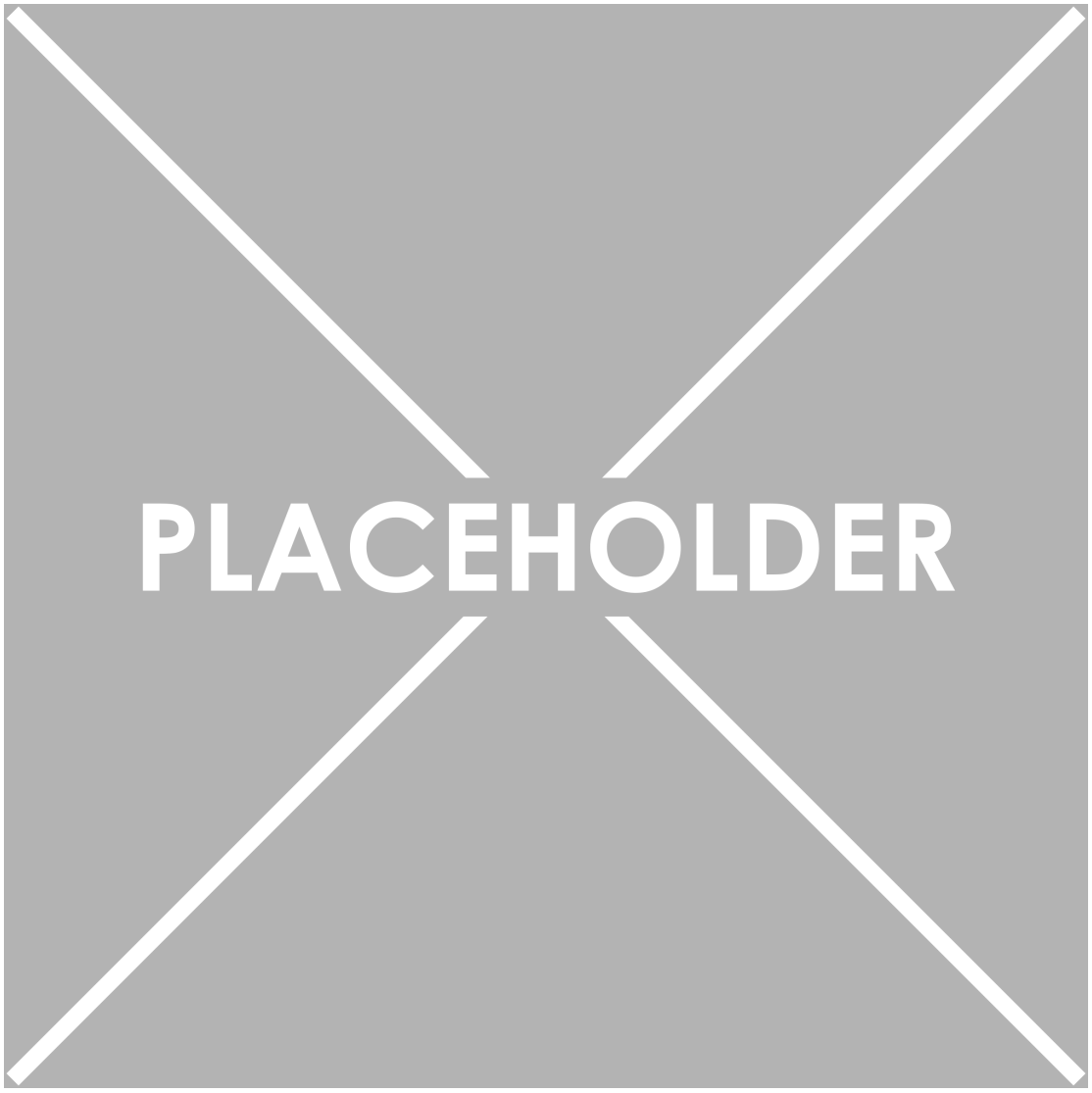

Figure 17: Generated 256×256 face images using diverse attribute-based text prompts with guidance scale $w = 2$. Each image is conditioned on a different combination of attributes.

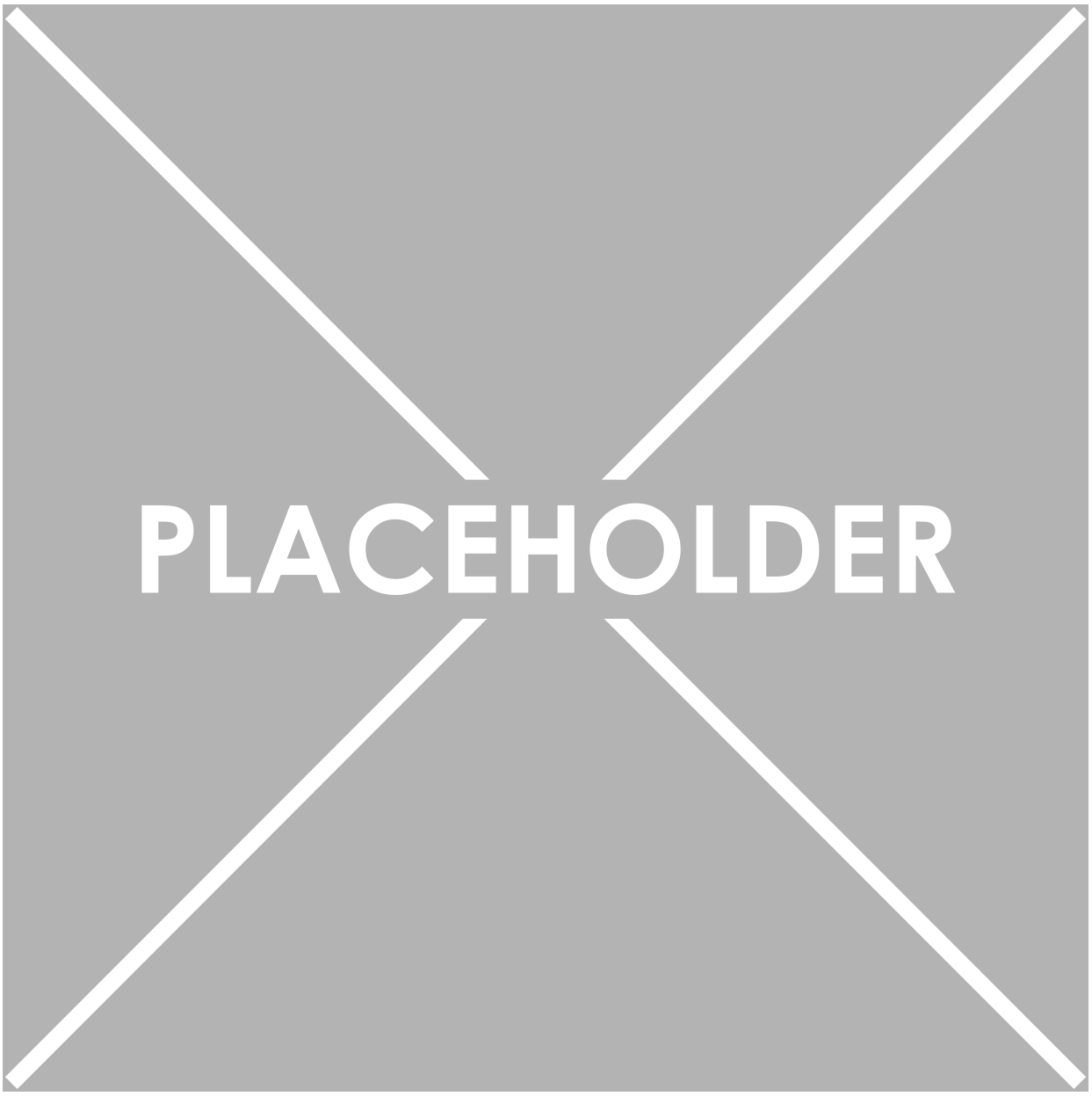

Figure 18: Effect of guidance scale w on CelebA face generation. Columns show $w \in \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$, rows show different prompts.