# **Diffusion Model Implementation and Troubleshooting Report**

## **Introduction**

This project began with ambitious intentions: training a conditional diffusion model on the CelebA dataset, aiming to generate high-resolution, attribute-conditioned facial images. CelebA, with its complex feature space and attribute vectors, seemed like the perfect playground to stretch the limits of diffusion models.

However, after burning through nearly 190 compute units, suffering multiple hallucinated outputs (later identified as numerical instability), training collapses, and a progressive erosion of sanity, the project scope was adapted — first to CIFAR-10, and ultimately, to MNIST.

In a way, this descent mirrors the trajectory of the project itself: big dreams slowly constrained by the realities of limited compute power, unstable training pipelines, and the relentless clock of assignment deadlines. Despite these setbacks, the core mission remained unchanged: to build, train, and analyze a class-conditional diffusion model using a UNet-based architecture, and to observe its capacity for reconstructing structured image data under a diffusion framework.

What follows is not merely a technical methodology, but a documentation of the compromises, adaptations, and lessons learned in the process of dragging a working model into existence under less-than-ideal conditions. The Methodology section below outlines the specific training pipeline, dataset handling, architectural choices, and sampling strategies employed for the various datasets prior to choosing MNIST dataset for the purpose of documentation and learning value.

## **Methodology**

The core objective was the implementation and training of a class-conditional Denoising Diffusion Probabilistic Model (DDPM) based on the provided notebook template structure. The methodology underwent significant evolution across attempts with CelebA, CIFAR-10, and Fashion-MNIST datasets, incorporating progressively more complex techniques to address observed shortcomings.

**1. Core Diffusion Process (DDPM):**

* Forward Process (Noising): Implemented as described in Ho et al. (2020). Gaussian noise was incrementally added to clean images x0​ over T discrete timesteps according to a predefined variance schedule βt​. The noisy image xt​ at any step t can be sampled directly using the closed-form solution:  
  q(xt​∣x0​)=N(xt​;αˉt​​x0​,(1−αˉt​)I), where αt​=1−βt​ and αˉt=∏i=1tαi​. The add\_noise(x\_0, t) function realized this, taking a clean image batch x0​ and a batch of timesteps t to produce the corresponding noisy versions xt​ and the added noise ϵ. Visualizations using show\_noise\_progression confirmed the correct implementation of this forward process across different schedules.
* Reverse Process (Denoising/Sampling): The objective is to learn the reverse conditional distribution pθ​(xt−1​∣xt​,c) to iteratively denoise a sample starting from pure noise xT​∼N(0,I). This was approximated by training a neural network ϵθ​(xt​,t,c) (the U-Net) to predict the noise ϵ that was added to obtain xt​. The mean μθ​(xt​,t) of the reverse step distribution pθ​(xt−1​∣xt​,c) was then calculated using the predicted noise:  
  μθ​(xt​,t)=αt​​1​(xt​−1−αˉt​βt​​ϵθ(xt​,t,c)).  
  Sampling xt−1​ involves adding Gaussian noise with variance σt2​=βt​ (for DDPM):  
  xt−1​=μθ​(xt​,t)+βt​​z, where z∼N(0,I) for t>0, and x0​=μθ​(x1​,1) for t=1. A small epsilon (1e−9) was added to the denominator term 1−αˉt​​ for numerical stability. This iterative process was implemented within the various generation functions (remove\_noise, generate\_samples\_cfg, generate\_number, visualize\_generation\_steps).

**2. Noise Schedules:**

* **Linear Schedule (Template Default):** The initial configuration, primarily intended for MNIST, used a linear schedule where βt​ ramps from βstart​=0.0001 to βend​=0.02 over T=100 steps. This was used in the final fallback attempts.
* **Cosine Schedule (Optimization Attempt):** To potentially improve performance on more complex datasets (CIFAR-10, Fashion-MNIST) and align with common practices suggesting smoother sample quality, a cosine schedule (Nichol & Dhariwal, 2021) was implemented. This schedule defines αˉt​ based on a cosine function: $ f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2 $, $\bar{\alpha}\_t = \frac{f(t)}{f(0)} $, and βt​=1−αˉt−1αˉt​. It typically adds noise more slowly initially. This was tested with T=1000 (CIFAR-10) and T=200 (Fashion-MNIST), using the standard s=0.008. The cosine\_beta\_schedule function was implemented to calculate the corresponding βt​ values, ensuring βt​ was clipped between 0 and 0.999 for stability.

**3. Model Architecture (U-Net):**

* **Base Architecture:** A U-Net architecture, standard for diffusion models due to its multi-scale processing and skip connections, was employed based on the template. Key blocks included:
  + GELUConvBlock: Conv2d, GroupNorm, GELU activation. Group size adjustment logic was included for compatibility.
  + RearrangePoolBlock: einops.Rearrange for 2x pixel shuffling downsampling, followed by a GELUConvBlock.
  + DownBlock (Template): Combined two GELUConvBlocks and a RearrangePoolBlock.
  + UpBlock (Template): ConvTranspose2d, skip connection concatenation, two GELUConvBlocks.
* **Parameterization:**
  + *Template MNIST:* Shallow settings: down\_chs=(32, 64, 128), t\_embed\_dim=8.
  + *Advanced (CIFAR/Fashion):* Deeper configurations: down\_chs=(64, 128, 256) or (64, 128, 256, 512), larger embedding dimensions (t\_embed\_dim=256 or 512). The target embedding dimension for time/class was set to match the bottleneck channel count (down\_chs[-1]).
* **Skip Connection Correction:** A critical modification was implemented to address a structural issue in the template's DownBlock which prevented easy access to features *before* pooling. The U-Net's \_\_init\_\_ was restructured to separate the feature extraction layers (down\_blocks\_features containing nn.Sequential of GELUConvBlocks) from the pooling layers (down\_blocks\_pool containing RearrangePoolBlock). The forward pass was then modified to iterate through these separated lists, appending the output of down\_blocks\_features to a skips list *before* applying down\_blocks\_pool, ensuring the correct feature maps were passed to the corresponding UpBlock layers via skips.pop().

**4. Conditioning Mechanisms:**

* **Time Embedding:** Timesteps t were encoded using SinusoidalPositionEmbedBlock, followed by an MLP (Linear -> GELU -> Linear) projecting to the target\_embed\_dim. An nn.Unflatten layer reshaped the output to [B, target\_embed\_dim, 1, 1] for addition to feature maps.
* **Class Embedding:** Class labels c (integers) were first converted to one-hot vectors within the EmbedBlock's forward pass using F.one\_hot(c.long(), N\_CLASSES).float(). These were then passed through an MLP (similar to time embedding) and also unflattened to [B, target\_embed\_dim, 1, 1].
* **Embedding Injection:** Both t\_emb and class\_emb were added element-wise to the feature map at the U-Net's bottleneck (after the middle\_blocks).
* **Conditioning Mask (c\_mask):** A boolean tensor of shape [B] controlled the application of embeddings for CFG. It was reshaped to [B, 1, 1, 1] and converted to float within the forward pass. Embeddings were applied only if the mask value was True (1.0): x = x + t\_emb \* mask + class\_emb \* mask.

**5. Training Procedure:**

* **Objective:** Train the U-Net ϵθ​ to predict the noise ϵ added in the forward process, minimizing the MSE loss: L=Et,x0​,ϵ∣∣ϵ−ϵθ(xt​,t,c)∣∣2.
* **Optimization:** Adam optimizer (e.g., lr=5e-4 or 1e-3, weight\_decay=1e-5) with ReduceLROnPlateau LR scheduling (monitoring validation loss, patience=3 or 5, factor=0.5). Gradient clipping (max\_norm=1.0) was used.
* **Classifier-Free Guidance (CFG) Training:** Implemented in the train\_step for advanced setups. A probability p\_uncond (e.g., 0.1) determined the chance of masking the condition for each sample in a batch. A c\_mask tensor (boolean [B]) was generated (torch.rand(...) > p\_uncond). This mask was passed to the U-Net forward method to control embedding addition. The U-Net thus learned both conditional and unconditional predictions.
* **Validation & Checkpointing:** Validation loss (MSE) was computed on a separate validation set after each epoch without CFG (c\_mask all True). Model checkpoints (best\_diffusion\_model\_...pth) were saved using safe\_save\_model whenever validation loss improved, incorporating basic backup logic. Early stopping (patience=10) was included.

**6. Sampling/Generation Strategies:**

* **Basic (Template):** The remove\_noise function implemented the standard DDPM reverse step using only the conditional prediction ϵθ​(xt​,t,c) (passed with c\_mask=True).
* **CFG Sampling (Advanced):** Generation functions (generate\_samples\_cfg, etc.) implemented CFG sampling. At each step t:
  1. Compute unconditional prediction ϵuncond​=ϵθ​(xt​,t,cnull​) (using c\_mask=False).
  2. Compute conditional prediction ϵcond​=ϵθ​(xt​,t,c) (using c\_mask=True).
  3. Combine using extrapolation: ϵ^=ϵuncond​+s⋅(ϵcond​−ϵuncond​), with guidance scale s (cfg\_scale).
  4. Use ϵ^ in the DDPM reverse step calculation for μθ​(xt​,t).
  5. Optionally clamp ϵ^ before calculating the mean (tested during troubleshooting).

**7. Datasets Attempted:**

* **CelebA:** Initial ambition, quickly abandoned due to compute/time constraints.
* **CIFAR-10:** 32x32 RGB. Advanced setup (Cosine T=1000, CFG, deeper U-Net). Stable training loss but unstable sampling.
* **Fashion-MNIST:** 28x28 Grayscale. Advanced setup (Cosine T=200, CFG, U-Net (64, 128, 256)). Stable training loss, promising early visuals, but unstable sampling persisted.
* **MNIST:** 28x28 Grayscale. Final fallback. Tested with advanced setup (unstable sampling) and basic setup (stable but poor visuals).

## **Troubleshooting and Attempted Optimizations**

The path towards a functional diffusion model was fraught with challenges, necessitating a cascade of scope reductions and targeted debugging efforts. The initial ambition of tackling CelebA was rapidly curtailed by the sheer computational demands, leading to a pivot towards the more manageable CIFAR-10 dataset.

1. Early Hurdles (CIFAR-10):

Training on CIFAR-10, even with a reduced scope, presented immediate obstacles. Using deeper U-Net architectures (e.g., down\_chs=(64, 128, 256, 512)) and standard batch sizes (e.g., 128) frequently triggered CUDA Out-of-Memory (OOM) errors on the available Colab GPU (even A100). This required reducing BATCH\_SIZE significantly (to 64 or even 32) and simplifying the architecture (down\_chs=(64, 128, 256)). Initial training runs, while showing decreasing loss, produced visually incoherent samples, prompting the implementation of established diffusion model enhancements.

2. Implementing Advanced Techniques:

To improve potential sample quality and training stability, several techniques were integrated:

* **Cosine Schedule:** Replaced the default linear schedule, hoping for better sample quality, particularly with the increased step count (T=1000).
* **Classifier-Free Guidance (CFG):** Both the training mechanism (10% unconditional training probability) and the sampling method (cfg\_scale typically tested around 3.0-7.0) were implemented, as CFG is crucial for high-fidelity conditional generation.
* **Architectural Corrections:** The U-Net was modified to ensure correct skip connection handling, a necessary fix from the template's initial structure.

3. The Persistent Problem: Sampling Instability:

Despite these enhancements, a consistent and critical issue emerged across both CIFAR-10 and subsequent Fashion-MNIST attempts (using T=200, Cosine, CFG): numerical instability during the reverse sampling phase.

* **Diagnosis via Logging:** Extensive debug prints were added to the generate\_samples\_cfg function to track tensor statistics (min, max, mean, std) at key points within the denoising loop (x\_t, eps\_cond, eps\_uncond, combined eps, mean, updated x\_{t-1}).
* **Key Finding:** The training logs consistently showed stable loss reduction (MSE often < 0.1) and stable tensor values within the train\_step. However, the generation logs revealed that while the model's *predicted noise* (ϵθ​) remained bounded, the intermediate *image representation* (xt​) and the calculated *mean* (μθ​) exploded to extremely large values (magnitudes of 102 to 103 or higher) as the sampling progressed from t=T−1 towards t=0.
* **Visual Consequence:** This numerical explosion inevitably led to corrupted outputs, appearing as noise or abstract artifacts, regardless of the low training loss. Promising structures sometimes visible in early/mid-sampling steps (like the Fashion-MNIST boot shape observed around Epoch 5) were overwhelmed by the instability by the end of the process.

4. Attempts to Mitigate Instability:

Several standard and specific strategies were employed to combat the sampling instability:

* **Noise Clamping:** torch.clamp(eps, -C, C) (with C=1.5 or 1.0) was applied to the CFG-combined noise estimate ϵ^ before its use in the mean calculation. This is a common heuristic to prevent outlier predictions from destabilizing the process. **Result:** Failed to prevent the explosion of xt​ values, suggesting the issue was more fundamental than just occasional large noise predictions.
* **CFG Scale Adjustment:** The guidance scale s (cfg\_scale) was varied during inference (tested 1.0, 3.0, 5.0, 7.0, 9.0). **Result:** Did not establish a stable regime; the explosion persisted across scales.
* **Reducing Diffusion Steps (T):** The number of steps for the Cosine/CFG setup was reduced from 200 back to 100 for Fashion-MNIST, hypothesizing that fewer iterations might limit error accumulation. **Result:** The numerical explosion during sampling remained evident even with 100 steps.
* **Dataset Simplification (Fashion-MNIST → MNIST):** The problem persisted even on the simpler grayscale Fashion-MNIST dataset, indicating the instability wasn't solely tied to the complexity of CIFAR-10. The final fallback to MNIST was made.
* **Basic Setup Trial:** Reverting to the template's original MNIST configuration (Linear schedule, T=100, simple U-Net, no CFG) yielded numerically stable generation but produced very poor, blurry visual results after comparable training epochs (e.g., 10-20).
* **CFG on Basic Model:** Applying CFG during generation *only* to the model trained with the basic setup offered slight visual improvement over no CFG but remained far from sharp, likely due to the limitations of the features learned by the basic model without CFG during training.
* **External Code Suggestions (Deepseek):** A fix suggested by another source, targeting potential shape mismatches in the U-Net's embedding application (EmbedBlock output shape, mask application in forward), was briefly considered and incorporated into one of the basic MNIST test runs. **Result:** As the diagnosed instability occurred during the sampling loop calculation (not the U-Net forward pass during training, which appeared stable), this fix did not address the core issue.

5. Resource and Time Cost:

The extensive debugging cycles, involving retraining models with different parameters and datasets, consumed a significant amount of computational resources (estimated ~150-190 Colab compute units) and time, highlighting the practical difficulties inherent in troubleshooting deep generative models.

## **Analysis & Examination**

The iterative process of implementation, training, and debugging across multiple datasets and configurations consistently pointed to a core challenge: **numerical instability arising specifically during the reverse diffusion (sampling) phase.** This instability manifested as divergent, exploding values in the intermediate image representations (xt​), rendering the final generated outputs visually incoherent despite achieving low numerical loss during training.

**Key Analytical Points:**

1. **Loss-Perception Discrepancy:** The project served as a stark illustration of the loss-perception gap. Low Mean Squared Error (MSE) loss, while indicating the model learned to predict the average noise component accurately pixel-wise, failed to guarantee the generation of perceptually coherent global structures. The model learned the *task* numerically but failed at the *synthesis* step due to instability.
2. **Sampling Loop as Failure Point:** Debug logs unequivocally isolated the numerical divergence to the iterative application of the DDPM reverse step formula. Stable single-step noise predictions (ϵθ​) contrasted sharply with the exploding intermediate states (xt​) and calculated means (μθ​). This strongly suggests an accumulation of errors, potentially amplified by the schedule coefficients (αt​,βt​,αˉt​) and CFG scaling, leading to divergence over the sampling trajectory (T=100 to T=1000).
3. **Impact of Design Choices:**
   * **CFG:** While crucial for sample quality, the extrapolation inherent in CFG (ϵuncond​+s⋅(ϵcond​−ϵuncond​)) likely increased the sensitivity to prediction errors and coefficient magnitudes, contributing to instability, especially at higher scales (s). The training phase, which only involved single predictions (either conditional or unconditional), remained stable.
   * **Cosine Schedule & High Step Count:** The combination of the Cosine schedule and a large number of steps (T=200 or 1000), though theoretically sound, provided ample opportunity for numerical errors to compound. The instability persisted even when T was reduced to 100 in the advanced setup, suggesting the schedule/CFG interaction was particularly problematic.
   * **U-Net Architecture:** While the U-Net was deepened for complex datasets and corrected for skip connections, its specific internal dynamics (e.g., normalization, activation choices) might have interacted unfavorably with the sampling process, contributing to the instability.
4. **Failure of Standard Fixes:** The inability of noise clamping (a common heuristic) to prevent the explosion suggests the problem wasn't just isolated large noise predictions but possibly related to the scaling factors (1/αt​​ and βt​/1−αˉt​​) within the DDPM update itself becoming problematic over the trajectory defined by the Cosine schedule.
5. **Basic Setup Trade-offs:** The basic MNIST setup demonstrated numerical stability during generation but yielded poor visual results without CFG, emphasizing the practical need for guidance mechanisms for efficient high-quality generation, even if they introduce stability challenges.

**Hypothesized Root Cause:**

The persistent numerical instability likely stems from an unfavorable interaction between the **DDPM sampling equation's coefficients**, the **Cosine noise schedule's specific variance progression over 100+ steps**, the **extrapolation effect of CFG**, and potentially the **learned dynamics of the specific U-Net implementation**. Small inaccuracies in the U-Net's noise prediction ϵθ​, magnified by the CFG scale s, are further scaled by schedule-dependent coefficients that might become very large or lead to subtractive cancellation issues within the iterative update for xt−1​. This error accumulation leads to divergence over the full sampling chain. The linear schedule over 100 steps appeared more robust, albeit yielding lower quality samples without CFG.

## **Future Direction**

The challenges encountered underscore the sensitivity of diffusion models and suggest several promising directions for future investigation to achieve stable, high-quality generation:

1. **Adopt DDIM Sampling:** Denoising Diffusion Implicit Models (DDIM) offer a deterministic sampling process derived from a non-Markovian perspective. The DDIM update step differs from DDPM and often exhibits greater numerical stability, especially when using accelerated sampling (fewer steps). Implementing the DDIM sampler and applying it to the trained U-Net checkpoints (both basic and advanced) would be the most immediate and potentially impactful next step to circumvent the DDPM sampling instability.
2. **Refine Hyperparameters:**
   * **Learning Rate Strategy:** Experiment with lower initial learning rates (e.g., 1e−4,2e−4), potentially combined with a cosine annealing schedule instead of ReduceLROnPlateau, to encourage smoother convergence to more robust model weights.
   * **CFG Scale Optimization:** Systematically evaluate the impact of cfg\_scale during inference across a wider range (e.g., 0.0 to 10.0) to identify if a stable and effective guidance level exists for the trained models. Explore adaptive CFG techniques.
   * **Noise Schedule Parameters:** For the Cosine schedule, tune the offset parameter s. For the Linear schedule, experiment with different βstart​ and βend​ values optimized specifically for MNIST/Fashion-MNIST stability over 100 steps. Consider exploring Variance Preserving (VP) or Variance Exploding (VE) SDE-based schedules.
3. **Architectural Modifications:**
   * **Normalization:** Replace GroupNorm with BatchNorm or LayerNorm, or experiment with adaptive normalization layers (AdaGN).
   * **Regularization:** Introduce dropout within the U-Net blocks, particularly for more complex datasets, as suggested in some literature. Adjust weight\_decay.
   * **Attention Mechanisms:** Ensure attention blocks (if used, though potentially omitted in the basic template) are correctly implemented and perhaps strategically placed only at lower resolutions.
4. **Alternative Loss Functions:** While MSE is standard, investigate the impact of using L1 loss (∣∣ϵ−ϵθ​∣∣1​) for training, which is sometimes reported to be more robust to outliers.
5. **Implementation Verification:** Conduct a meticulous comparison of the implemented DDPM sampling loop mathematics against canonical implementations (e.g., from original papers or established libraries like diffusers) to rule out subtle coding errors in coefficient calculation or the update step.
6. **Simplified Baseline:** Retrain the basic MNIST setup with the simplest possible U-Net (minimal channels/depth) and linear schedule to establish a definitively stable, albeit low-quality, baseline before re-introducing complexity.

## **Conclusion**

This project embarked on the implementation of conditional diffusion models, initially targeting complex datasets like CelebA and CIFAR-10, but ultimately requiring adaptation to simpler datasets like Fashion-MNIST and MNIST due to persistent challenges. While the training phase consistently demonstrated successful numerical convergence, indicated by decreasing MSE loss on both training and validation sets, a critical failure point repeatedly emerged during the image generation (reverse sampling) process.

Extensive debugging and analysis, aided by detailed logging of intermediate tensor values, identified catastrophic numerical instability within the iterative DDPM denoising loop as the primary obstacle. This instability, manifesting as exploding values in the image representation xt​, occurred particularly when using the theoretically advantageous Cosine noise schedule combined with Classifier-Free Guidance over extended step counts (T=100-1000), leading to visually corrupted outputs despite low training error. Standard stabilization techniques, such as clamping the predicted noise, proved insufficient. Reverting to the basic template setup (Linear schedule, T=100, no CFG) achieved sampling stability but at the cost of significantly degraded visual quality and slower convergence.

The core conclusion is that the interplay between the DDPM sampling formulation, the specific properties of the Cosine schedule across numerous steps, the amplification effects of CFG, and the dynamics of the implemented U-Net architecture created a numerically sensitive system prone to divergence during iterative synthesis. While the project did not culminate in high-fidelity generation for the initially targeted datasets, it provided invaluable hands-on experience with the diffusion model pipeline, the practical challenges of training deep generative models (OOM errors, hyperparameter tuning), and the critical importance of robust sampling algorithms. The documented troubleshooting journey underscores the potential fragility of the DDPM sampling process under certain configurations and highlights promising future directions, most notably the implementation of the more stable DDIM sampler, alongside careful hyperparameter and architectural refinement, to achieve the desired generative capabilities. The final successful MNIST run (detailed separately) ultimately validated the corrected core components when applied within a known stable configuration.