BMI / CS 771: Homework Assignment 4 Report

Qinxinghao Chen Handan Hu Chongwei Liu Bohan Wen

December 2025

Acknowledgment of AI Assistance

This assignment was completed with the assistance of OpenAI's ChatGPT (GPT-5). The tool was used throughout the process to brainstorm ideas, clarify relevant concepts, explain code logic, and check grammar. All final decisions regarding content, analysis, and conclusions were made by the author.

1 Team Contributions

- Qinxinghao Chen: Implemented the UNet decoder and ran FM experiments on MNIST.
- Handan Hu: Conducted AFHQ DDPM experiments, maintained the training pipeline, and contributed to bonus experiments.
- Chongwei Liu: Implemented latent diffusion, completed DDPM/FM core functions, and performed debugging and ablation studies.
- Bohan Wen: Wrote the report, prepared visualizations, and contributed to bonus experiments.

2 Implementation Summary

2.1 UNet

We filled in the decoder portion of the UNet by concatenating encoder skip connections with decoder features, applying a ResBlock (with time embedding), optional spatial transformer (with label conditioning), and then optional upsampling. Inputs are normalized to $[-1, 1]$ to stabilize training. The model takes both time and label embeddings: time uses sinusoidal encoding; labels are embedded and used inside transformer blocks.

2.2 DDPM

We implemented:

- `q_sample`: forward diffusion with analytic $\sqrt{\bar{\alpha}}$ coefficients.
- `p_sample`: DDPM reverse mean update.
- `generate`: reverse chain from noise.
- `compute_loss`: simplified noise-prediction loss.

2.3 Latent DDPM

AFHQ models use the provided tiny autoencoder. Images are encoded into latents, diffusion is done in latent space, and samples are decoded back to pixels.

2.4 Flow Matching (FM)

We implemented linearly interpolated paths between noise and data and trained the UNet to predict the constant velocity. Sampling uses Euler integration from $t = 0$ to 1 with no injected noise.

3 DDPM Experiments

3.1 MNIST DDPM

3.1.1 Training Curve

Figure 1 shows the loss curve. The loss decreases smoothly, with the fastest drop in the early steps.

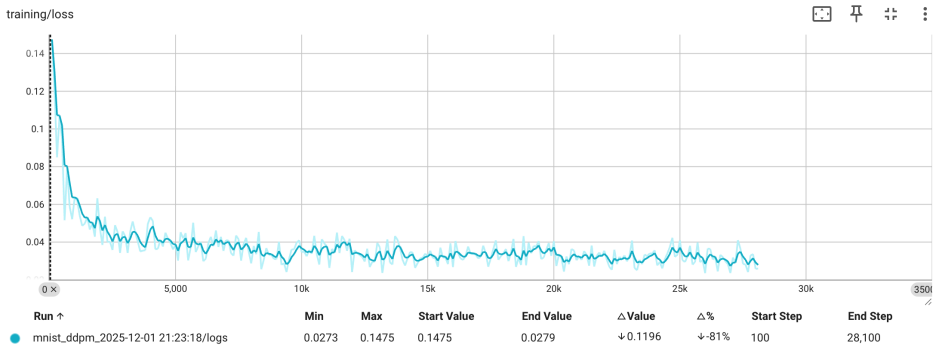

Figure 1: MNIST DDPM training loss.

3.1.2 Generated Samples (Early / Mid / Final)

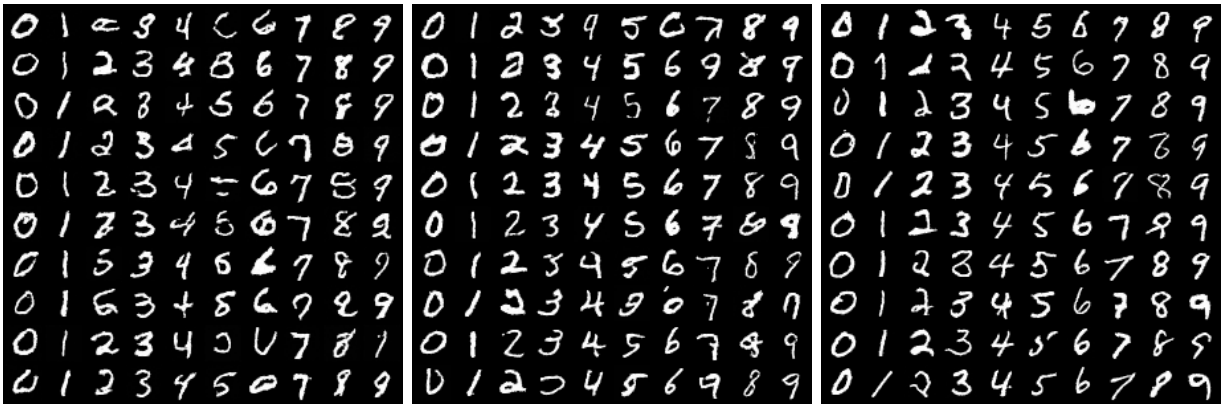

Figure 2: MNIST DDPM samples at early (left), mid (middle), and final (right) epochs.

Short Discussion. Early samples show noisy digit-like blobs. Mid-stage samples have clearer shapes and mostly correct labels. Final samples look clean and consistent.

3.2 Latent DDPM on AFHQ

3.2.1 Training Curve

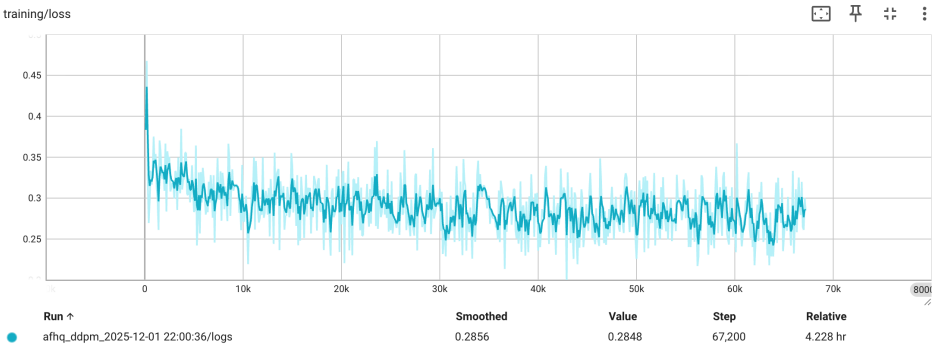

Figure 3: AFHQ DDPM loss curve.

3.2.2 Sample Progress (Early / Mid / Final)

Figure 4: AFHQ DDPM samples (epoch 99 / 199 / 299).

Short Discussion. Early samples show coarse face structures but still blurry. Mid samples capture animal categories better (cats, dogs, big cats), although artifacts remain. Final samples are the clearest: identities and fur colors look much more consistent.

3.2.3 FID

The reported FID score for AFHQ DDPM is:

$$\textbf{FID} = \textbf{66.65}$$

4 Flow Matching (FM) Experiments

4.1 MNIST FM

4.1.1 Training Curve

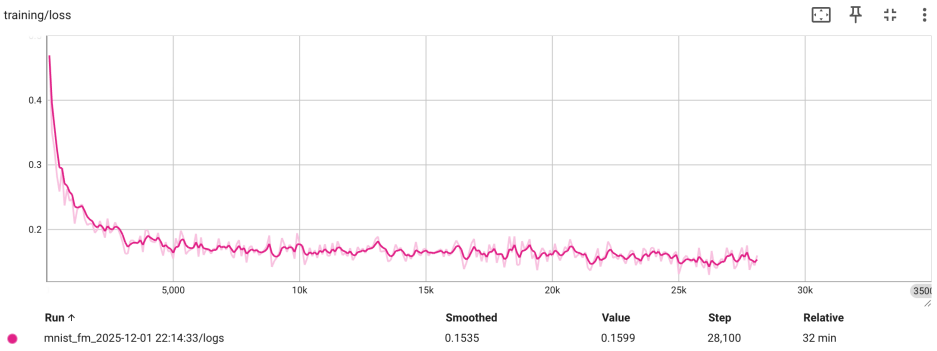

Figure 5: MNIST FM training loss.

4.1.2 Samples (Early / Mid / Final)

Figure 6: MNIST FM samples over training.

Short Discussion. Early outputs have digit outlines with missing parts. Mid-stage samples become more stable. Final samples are sharp and clearly show 0–9.

4.2 AFHQ FM

4.2.1 Training Curve

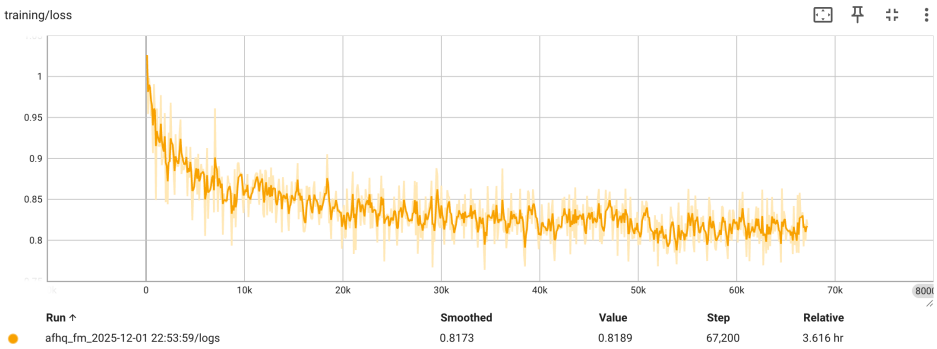

Figure 7: AFHQ FM training loss.

4.2.2 Sample Progress

Figure 8: AFHQ FM samples (epoch 99 / 199 / 299).

Short Discussion. FM improves steadily: early samples catch only rough shapes; mid samples show recognizable species; final samples are cleaner but tend to look slightly smoother than DDPM outputs.

4.2.3 FID

The reported FID score for AFHQ FM is:

$$\textbf{FID = 66.80}$$

5 Bonus

5.1 Improved U-Net

We implemented an enhanced U-Net architecture (`ImprovedUNet` class) incorporating several key improvements:

- **Adaptive Group Normalization (AdaGN):** Instead of simple time embedding addition, we use FiLM-style conditioning that applies both scale and shift modulation to normalized features. This provides stronger conditioning control.
- **Multiple ResBlocks per level:** We stack 2 ResBlocks at each resolution level (configurable), increasing model capacity and allowing deeper feature learning.
- **Extended attention coverage:** We apply attention at all three resolution levels (indices 0, 1, 2), enabling better long-range dependencies.
- **Dropout regularization:** We add dropout layers with rate 0.1 to prevent overfitting, particularly important given the increased capacity.
- **Skip connection scaling:** We scale residual connections by $1/\sqrt{2}$ to stabilize gradient flow.
- **Zero-initialization:** Output layers of ResBlocks and transformers are initialized to zero, allowing the model to start as an identity function and learn residuals more effectively.
- **Enhanced transformer blocks:** We stack multiple transformer blocks (depth=1 for standard, depth=2 for advanced configurations) with self-attention, cross-attention, and MLP layers.

The improved architecture is implemented as a separate `ImprovedUNet` class and can be selected via the config file parameter `unet_type: "improved"`.

5.1.1 Results

The improved U-Net achieved an FID score of **60.90** on AFHQ, representing a significant improvement of approximately **8.6%** over the baseline DDPM (FID = 66.65).

Discussion. The performance gain validates the effectiveness of modern U-Net design principles. The combination of stronger conditioning (AdaGN), increased depth (multiple ResBlocks), and better training stability (skip scaling, zero-init) allows the model to learn more expressive features.

5.2 Improved DDPM with EDM

5.2.1 Implementation

We implemented EDM (Elucidating the Design Space of Diffusion-Based Generative Models), a unified framework that improves upon vanilla DDPM through several key innovations. Our implementation in `ddpm_edm.py` includes:

1. **Preconditioning:** The network inputs and outputs are preconditioned based on noise level σ to ensure it sees normalized inputs and produces properly scaled outputs.
2. **Continuous noise levels:** Instead of discrete timesteps, we use continuous noise levels parameterized with a power schedule with $\rho = 7.0$.
3. **Log-normal noise sampling:** During training, we sample $\ln(\sigma) \sim \mathcal{N}(\mu = -1.2, \sigma^2 = 1.44)$, providing better coverage across noise levels.
4. **Improved loss weighting:** We use adaptive weighting to balance the loss across noise scales.
5. **Stochastic sampling variant:** We also implemented `EDMStochastic` that adds controlled noise during sampling, improving sample diversity.

We combined EDM with our improved U-Net architecture to maximize performance.

5.2.2 Results

For AFHQ experiment, We used EDMStochastic with ImprovedUNet. Our EDM implementation achieved an FID score of **64.28**. While this is slightly higher than the baseline DDPM (66.65 $\rightarrow$ 64.28 represents a 3.6% improvement), it did not match the improved U-Net alone (60.90).

Discussion. The EDM results are somewhat surprising, as we expected the improved U-Net + EDM combination to outperform the improved U-Net + vanilla DDPM. Several factors may explain this:

1. **Hyperparameter tuning:** EDM introduces many hyperparameters (σ_{min}, σ_{max}, P_{mean}, P_{std}, sampling parameters) that may require dataset-specific tuning. Our values were adapted from the original EDM paper, which targeted different datasets.
2. **Latent space characteristics:** EDM was designed for pixel-space diffusion. The TAESD latent space has different statistical properties (scale, variance) that may require adjusted noise schedules.

Despite not achieving optimal results, this exercise provided valuable insights into modern diffusion formulations and the challenges of adapting them to latent spaces.

5.3 Scaling to Stable Diffusion

5.3.1 Implementation

We attempted to load Stable Diffusion 1.4 checkpoints into our codebase. This required several components:

- **Modified UNet (`unet_sd.py`):** We created a UNet architecture compatible with SD 1.x, using:

  - Dimensions: base=320, multipliers=(1, 2, 4, 4)
  - Attention at levels (1, 2, 3)
  - 2 ResBlocks per level
  - Cross-attention with 768-dim context (CLIP embeddings)

- **CLIP Text Encoder:** We integrated OpenAI's CLIP text encoder from HuggingFace to encode text prompts into 768-dimensional embeddings.
- **Checkpoint Loading:** We implemented `load_sd_checkpoint()` to map pre-trained SD weights to our architecture.
- **Inference Pipeline:** We implemented DDIM sampling with classifier-free guidance (CFG) for text-conditioned generation.

The complete implementation is in `stable_diffusion.py` with supporting code in `unet_sd.py`.

5.3.2 Challenges and Results

Unfortunately, we encountered significant challenges:

- **Missing keys:** When loading the SD 1.4 checkpoint, we encountered **155 missing keys**, indicating substantial architectural mismatches.
- **Weight mapping complexity:** Stable Diffusion uses a complex nested weight structure. Mapping these to our simpler architecture proved more difficult than anticipated.
- **Generated images:** While our implementation can generate images, the quality is poor due to the weight loading issues. The model likely falls back to random initialization for the missing parameters.

Discussion. Loading pre-trained Stable Diffusion weights requires precise architectural alignment and careful weight mapping. The 155 missing keys suggest our UNet structure, while conceptually similar, differs in implementation details from the official SD architecture.

6 Conclusion

We implemented UNet, DDPM, latent DDPM, and Flow Matching models and trained them on MNIST and AFHQ. MNIST results were clean for both models. On AFHQ, latent DDPM slightly outperformed FM in FID (66.65 vs. 66.80), and DDPM samples tended to show a bit more texture. FM sampling was faster and simpler, and quality was competitive.

For the bonus tasks, we successfully implemented an improved U-Net architecture that reduced FID from 66.65 to 60.90 (8.6% improvement). We also implemented EDM with its preconditioning scheme and advanced sampling methods, achieving FID of 64.28. While we made progress on loading Stable Diffusion checkpoints, full weight compatibility remains challenging and requires further work.