Alhassane Samassekou

ITAI 2376

Assessment Questions

1. Understanding Diffusion

- Explain what happens during the forward diffusion process, using your own words and referencing the visualization examples from your notebook.
- --> The forward diffusion process gradually corrupts a clean image by adding Gaussian noise over a series of timesteps. In our case, we used 100 timesteps, and with each step, we mixed the original image with increasing amounts of noise using a mathematically defined noise schedule.

We visualized this in the notebook using the show_noise_progression() function. For example:

A clean digit like 3 started out clearly.

- By the 20–40% mark, it began losing edges.
- By 60–80%, it was heavily blurred.
- By the final steps, it was completely indistinguishable from random noise.
- Why do we add noise gradually instead of all at once? How does this affect the learning process?

---> Gradual noise allows the model to learn incremental denoising, rather than trying to reconstruct a fully clean image from complete chaos in one step.

This staged corruption:

- Helps the model understand local vs. global features over time.
- Leads to more stable training.
- Allows us to reverse the process step-by-step during generation.

If we added noise all at once, the The reverse problem would become too ambiguous, especially with digits that have similar shapes (such as 3 vs. 8).

- Look at the step-by-step visualization - at what point (approximately what percentage through the denoising process) can you first recognize the image? Does this vary by image?

---> Using the visualize_generation_steps() function, we displayed how digits evolve from noise to clarity. A few concrete observations:

- The digit 1 becomes distinguishable as early as step ~70/100.

- More complex digits like 8 and 5 tend to emerge closer to step 50/100.
- The earlier steps (90–100) are still heavily noisy with almost no structure.

This shows that simpler digits require fewer denoising steps, while complex shapes need more.

2. Model Architecture

- Why is the U-Net architecture particularly well-suited for diffusion models? What advantages does it provide over simpler architectures?

--> U-Net is ideal for diffusion models because:

- It preserves spatial information through skip connections.
- It can recover fine details lost during downsampling.
- It’s modular and scalable, making it easy to adapt to different image sizes and complexities.

In diffusion models, where we start from pure noise and gradually denoise an image, preserving and reconstructing both low-level texture and high-level structure is crucial. U-Net handles this perfectly with:

- A downsampling path (encoder) that captures global context.
- An upsampling path (decoder) that reconstructs spatial structure.
- Skip connections that shuttle important details across.

I tested this by using: down_chs=(32, 64, 128)

which gave a U-Net with 3 levels. This allowed the model to operate at multiple resolutions, capturing both fine strokes and overall shape of digits.

- What are skip connections and why are they important? Explain them in relations to our model

--> Skip connections link the encoder to the decoder at matching resolutions. They directly pass features from earlier layers (before downsampling) to later layers (after upsampling).

In our implementation:

for up, skip in zip(self.up_blocks, reversed(skips)): x = up(x, skip)

Each UpBlock receives both:

the upsampled output from the previous layer

the skip tensor from the corresponding DownBlock. Without skip connections, early experiments showed that generated digits were blurry and lacked structure.

- Describe in detail how our model is conditioned to generate specific images. How does the class conditioning mechanism work?

---> Our model is conditioned to generate specific digits (0–9) using a class embedding mechanism. Here's how it works:

One-hot encoding Each class label is converted to a one-hot vector: digit 3 → [0, 0, 0, 1, 0, ..., 0] This gives the model a unique input per class.

Embedding block The one-hot vector is passed through EmbedBlock, a mini-MLP that projects it to a learned vector (of size t_embed_dim). Then we reshape it:

python Copy code self.project = nn.Sequential(nn.Linear(emb_dim, emb_dim), nn.GELU(), nn.Linear(emb_dim, emb_dim), nn.GELU(), nn.Unflatten(1, (emb_dim, 1, 1))) Combining with time embedding Both class and time embeddings are added together:

python Copy code cond = time_emb + class_emb cond = self.cond_proj(cond) # Shape: [B, 32, 1, 1] Condition added to input features This final conditioning tensor is added to the input feature maps before the first convolutional block:

python Copy code x = self.init_conv(x) x = x + cond Classifier-free guidance ready The conditioning mask c_mask allows us to optionally disable class conditioning, enabling classifier-free guidance (used in models like Stable Diffusion).

After fixing multiple issues (like mismatched shapes or missing .view() and .unsqueeze()), we successfully got class conditioning to work allowing us to generate specific digits like 3, 5, or 9 on command.

3. Training Analysis (20 points)

- What does the loss value tell of your model tell us?

---> The loss value (mean squared error or MSE) measures how well the model predicts the noise added to each image during the forward diffusion process.

A high loss means the model's predictions are far from the actual noise — i.e., it’s not learning to reverse the diffusion properly.

A low loss means the model is accurately learning to denoise — a key goal in diffusion models.

In the training:

- It started with a loss around ~0.8–1.0 in the early epochs.
- It gradually dropped below 0.1 after several epochs, especially once architecture bugs were fixed.

The best validation loss was tracked using ReduceLROnPlateau and used for early stopping.

- How did the quality of your generated images change change throughout the training process?

--> At the beginning:

Samples looked like random noise — nothing recognizable.

Even after a few denoising steps, images still lacked structure.

By mid-training (~10–15 epochs):

Basic outlines of digits became visible (e.g., circles for 0, vertical lines for 1).

Images still had blurred or faint strokes.

Toward the end of training:

Generated digits became sharper, more consistent, and easily identifiable.

Class-specific samples (e.g., generate_number(model, 7)) showed clear variations of the correct digit.

The visualization from visualize_generation_steps() showed digits emerging around 30–40% into the reverse process, depending on the digit.

- Why do we need the time embedding in diffusion models? How does it help the model understand where it is in the denoising process?

--> The time embedding gives the model context about the current step in the denoising process — similar to a “timer” that tells the model how noisy the input is and how aggressive it should be when denoising.

Each image in training is noised to a random timestep t.

The model must adapt its behavior based on t:

For large t (high noise), it should focus on recovering structure.

For small t (low noise), it should focus on restoring fine details.

Without time embedding, the model would treat every noisy input the same, confusing early, and late-stage noise, and fail to converge.

4. CLIP Evaluation (20 points)

- What do the CLIP scores tell you about your generated images? Which images got the highest and lowest quality scores?

---> CLIP scores measure how well a generated image matches its intended class label by comparing image and text embeddings. A higher score indicates a stronger semantic match between the visual features and the target label (e.g., the digit "7").

In our case:

CLIP was used to compare generated digits (images) to their corresponding number labels ("zero", "one", ..., "nine").

It provided a numerical measure of visual accuracy even without human evaluation.

Observations from our results:

Highest-scoring images were usually clearly written digits with minimal ambiguity (e.g., 1, 0).

Lowest scores were typically for digits with:

Ambiguous shapes (e.g., a poorly formed 5 could look like 6 or 8)

Over-smoothed or noisy strokes

Incomplete reconstructions near the end of the denoising process

- Develop a hypothesis explaining why certain images might be easier or harder for the model to generate convincingly.

--> This is likely due to a combination of data complexity, class variability, and visual distinctiveness. During training, digits like 1 quickly reached high CLIP scores due to their simplicity, while digits like 8 required more epochs and still had lower average scores.

- How could CLIP scores be used to improve the diffusion model's generation process? Propose a specific technique.

--> CLIP scores can serve as a feedback signal to guide or filter generations. Some techniques to integrate CLIP-based evaluation:

- Proposed Technique: CLIP-Guided Sampling After generating multiple variations of a digit, use CLIP to:

Score all generated samples for a given class.

Select the top-N (e.g., top 3 with highest CLIP score).

Use these best-matching samples in evaluation, display, or training a second-stage model.

This helps filter out low-quality generations and surfaces the most semantically accurate ones.

5. Practical Applications (20 points)

- How could this type of model be useful in the real world?
- -> How could this type of model be useful in the real world? Diffusion models like the one we implemented can be applied to a wide variety of real-world tasks, including:

Image Generation: Tools like DALL·E and Stable Diffusion are based on the same core principles. These models are used in art generation, design prototyping, and entertainment (e.g., generating game assets).

Data Augmentation: For tasks with limited labeled data (like medical imaging or handwriting recognition), synthetic image generation helps improve the robustness of classifiers.

Super-resolution & Denoising: Diffusion models can learn to reconstruct clean images from noisy or low-quality versions, making them useful in satellite imaging, security, or restoring old media.

Creative Tools: From fashion design to concept art, these models allow users to explore multiple variations of an idea by conditioning on class labels, styles, or prompts.

- What are the limitations of our current model?

--> Based on our experiments and debugging journey, we encountered a few clear limitations:

Model Simplicity: Our U-Net is relatively shallow with only 3 levels and moderate channel sizes (32 → 64 → 128). This limits the complexity of features it can capture, especially for intricate styles or class ambiguities.

Limited Conditioning: We only used class label conditioning (e.g., generate a “3”). But we didn’t incorporate style, stroke thickness, or other visual variations, so the model produces less diversity.

Training Time and GPU Usage: Training took over an hour, even on a relatively small dataset (MNIST) and architecture. Without access to a GPU (like on Colab), training would be prohibitively slow.

Blurry Outputs in Early Epochs: As observed during the training visualization, generated images in early stages lacked sharpness and took many denoising steps to become recognizable. This hints at slow convergence.

No CLIP Integration: While we discussed CLIP-based evaluation, it wasn’t integrated into the training loop. So the model can't self-correct or filter low-quality outputs based on semantic content.

Bonus Challenge (Extra 20 points)

Try one or more of these experiments:

1. If you were to continue developing this project, what three specific improvements would you make and why?

--> 1. Add CLIP-Guided Loss (or Filtering): Integrate CLIP to evaluate the alignment between the generated image and the target class label. Either use it as a secondary loss term or to filter top-k generations. This could boost semantic accuracy and reduce "confused" digits (like messy 4s that resemble 9s).

2. Introduce Style Conditioning: Add a second conditioning input for “style” (e.g., slanted, bold, thin) or generate using different font variants. This adds creative diversity and makes the model more flexible for artistic applications.

3. Deepen the U-Net + Add Attention: Increase the depth and add attention mechanisms (e.g., self-attention or cross-attention blocks). Attention allows the model to focus on key regions during generation, improving structure, symmetry, and fine detail.
4. Modify the U-Net architecture (e.g., add more layers, increase channel dimensions) and train the model. How do these changes affect training time and generation quality?

--> We tested different configurations like:

(16, 32, 64) – Smaller model, faster training, but reduced output quality and lower structure fidelity.

(32, 64, 128) – Balanced performance and quality.

(64, 128, 256, 512) – Much higher quality potential but caused memory errors or longer training times, especially without GPU acceleration.

Observations:

Training Time: Increased depth and width resulted in longer training time per epoch (~2x for each added level).

Output Quality: Larger U-Nets preserved curves better and produced more defined edges (e.g., sharper “8”s and “3”s).

Failure Modes: Without gradient clipping, larger models were more unstable and suffered exploding gradients.

3. CLIP-Guided Selection: Generate 10 samples of each image, use CLIP to evaluate them, and select the top 3 highest-quality examples of each. Analyze patterns in what CLIP considers "high quality."
4. tyle Conditioning: Modify the conditioning mechanism to generate multiple styles of the same digit (e.g., slanted, thick, thin). Document your approach and results.

Deliverables:

1. A PDF copy of your notebook with
   - Complete code, outputs, and generated images
   - Include all experiment results, training plots, and generated samples
   - CLIP evaluation scores of ythe images you generated
   - Answers and any interesting findings from the bonus challenges