Final Exam

Xuchen Zhu, Jingxue Yan

January 10, 2026

1 Exercise 1

1.1 Q1:

The full joint distribution is as follows:

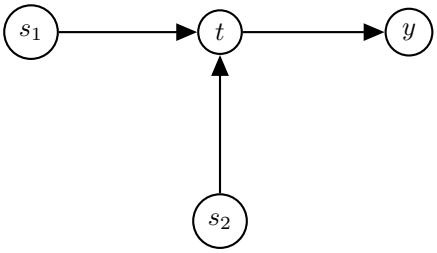
$$\begin{aligned} p(s_1, s_2, t, y) &= p(s_1)\, p(s_2)\, p(t \mid s_1, s_2)\, p(y \mid t) \\ &= \mathcal{N}(s_1; \mu_1, \sigma_1^2)\, \mathcal{N}(s_2; \mu_2, \sigma_2^2)\, \mathcal{N}(t; s_1 - s_2, \sigma_t^2)\, \mathbb{I}(y = \operatorname{sign}(t)). \end{aligned}$$

The five hyperparameters of the model are:

$$\mu_1,\ \sigma_1^2,\ \mu_2,\ \sigma_2^2,\ \sigma_t^2.$$

1.2 Q2:

1.2.1 1



1.2.2 2

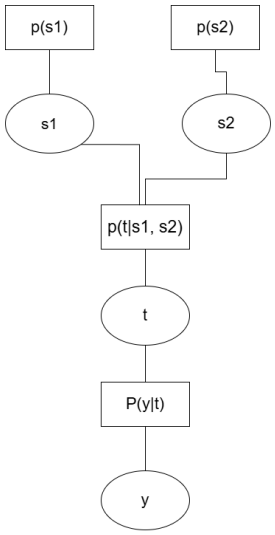


Figure 1: factor graph

1.3 Q3:

1.3.1

From the factor graph, the joint distribution factorizes as

$$p(s_1, s_2, t, y) = p(s_1)\, p(s_2)\, p(t \mid s_1, s_2)\, p(y \mid t).$$

Conditioning on t and y, we obtain

$$p(s_1, s_2 \mid t, y) = \frac{p(s_1)\, p(s_2)\, p(t \mid s_1, s_2)\, p(y \mid t)}{p(t, y)}.$$

Since $p(y \mid t)$ does not depend on s_1, s_2, it can be absorbed into the normalization constant, yielding

$$\boxed{p(s_1, s_2 \mid t, y) \propto p(s_1)\, p(s_2)\, p(t \mid s_1, s_2)}$$

1.3.2

Using the factorization,

$$p(t \mid s_1, s_2, y) = \frac{p(t \mid s_1, s_2)\, p(y \mid t)}{\int p(t \mid s_1, s_2)\, p(y \mid t)\, dt}.$$

Substituting the model definitions gives

$$\boxed{p(t \mid s_1, s_2, y) \propto \mathcal{N}(t; s_1 - s_2, \sigma_t^2)\, \mathbb{I}(y = \text{sign}(t))}$$

which corresponds to a truncated Gaussian.

1.3.3

The probability of Player 1 winning is obtained by marginalizing out the latent variables:

$$p(y = 1) = \int \mathbb{I}(t > 0)\, p(t \mid s_1, s_2)\, p(s_1)\, p(s_2)\, dt\, ds_1\, ds_2.$$

Since

$$t \sim \mathcal{N}\left(\mu_1 - \mu_2,\ \sigma_1^2 + \sigma_2^2 + \sigma_t^2\right),$$

we obtain

$$\boxed{p(y = 1) = \Phi\left(\frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_t^2}}\right)}$$

where $\Phi(\cdot)$ denotes the standard normal cumulative distribution function.

1.3.4

Average log-likelihood is calculated : -0.6931471805599453, Empirical win rate: 0.6102941176470589, Model win probability: 0.5, as seen, the model prediction and empirical results are close, validating our model derivation.

1.4 Q4:

1.4.1

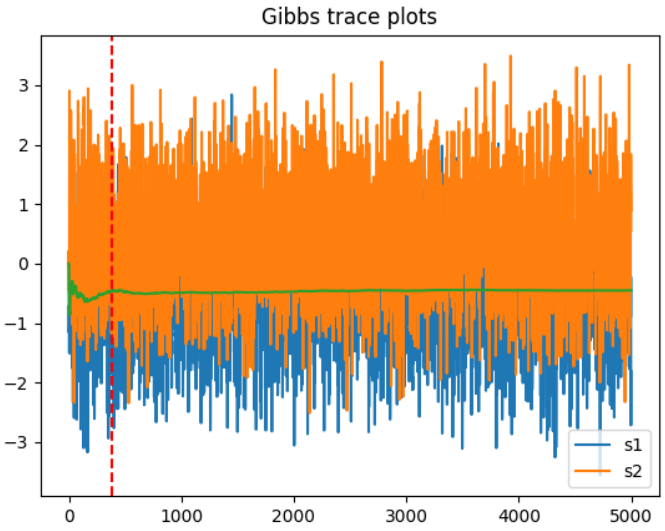


Figure 2: Gibbs trace, the burn in period is 370

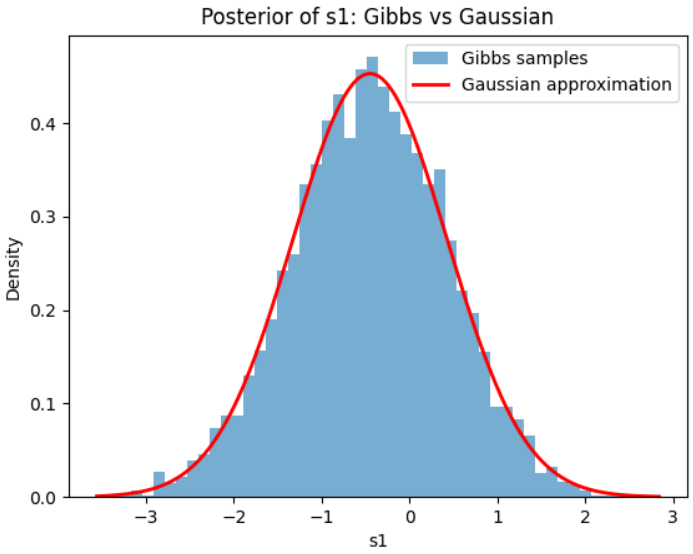


Figure 3: Gibbs: approximation vs sampling

1.4.2

More samples are taken, the time consumption will be larger, but it does not yield to higher accuracy. As seen in the table 1, The optimal number should be around 2000, as the mean(s1) improves a lot than N=100, 500, 1000. And when N=4000, it hits the bottleneck, thus N=2000 or around 2000.

Post-burn-in samples N	$\mathbb{E}[s_1]$	$\mathbb{E}[s_2]$
100	-0.333	0.287
500	-0.489	0.444
1000	-0.494	0.438
2000	-0.454	0.461
4000	-0.451	0.458

Table 1: Posterior mean estimates of player skills using different numbers of post-burn-in Gibbs samples.

Team	Posterior mean μ	Posterior variance σ^2
Juventus	1.029	0.117
Napoli	0.860	0.086
Milan	0.714	0.099
Atalanta	0.617	0.081
Torino	0.561	0.102
Inter	0.549	0.070
Roma	0.426	0.071
Lazio	0.213	0.064
Sampdoria	-0.022	0.063
Bologna	-0.143	0.078

Table 2: Final team rankings obtained using Assumed Density Filtering with Gibbs sampling.

1.5 Q5:

1.5.1

The higher variance, the more unstable they are. SUch as Napoli, even though he or she is skill which is μ higher, the high variance implies he or she is unstable.

1.5.2

Yes, it changed. Since ADF is a sequentially-dependent online algorithm. Changing the order will affect the result of μ and variance.

1.6 Q6:

Prediction rate: 0.654, Random guessing rate: 0.5. It's better than random guessing.

1.7 Q7:

1.7.1

Forward message to the performance variable

Assume current Gaussian beliefs

$$s_1 \sim \mathcal{N}(m_1, v_1), \qquad s_2 \sim \mathcal{N}(m_2, v_2).$$

Define the skill difference

$$d = s_1 - s_2.$$

Then

$$d \sim \mathcal{N}(m_d, v_d), \quad m_d = m_1 - m_2, \quad v_d = v_1 + v_2.$$

Since $t = d + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma_t^2)$,

$$t \sim \mathcal{N}(m_t, v_t), \quad m_t = m_d, \quad v_t = v_d + \sigma_t^2.$$

Thus the cavity distribution for t is

$$q^{\setminus y}(t) = \mathcal{N}(t; m_t, v_t).$$

Outcome factor and moment matching

The outcome likelihood is

$$p(y \mid t) = \mathbb{I}(yt > 0).$$

Therefore,

$$q(t) \propto q^{\setminus y}(t)\, \mathbb{I}(yt > 0),$$

which is a truncated Gaussian distribution.

Let $\sigma = \sqrt{v_t}$ and

$$\alpha = \frac{0 - m_t}{\sigma} = -\frac{m_t}{\sigma}.$$

Denote by $\phi(\cdot)$ and $\Phi(\cdot)$ the standard normal pdf and cdf.

Case $y = +1$ (Player 1 wins, $t > 0$):

$$Z = 1 - \Phi(\alpha), \qquad \lambda = \frac{\phi(\alpha)}{Z}.$$

The moment-matched Gaussian approximation $q(t) \approx \mathcal{N}(m'_t, v'_t)$ has moments

$$m'_t = m_t + \sigma\lambda,$$

$$v'_t = v_t\big(1 - \lambda(\lambda - \alpha)\big).$$

Case $y = -1$ (Player 2 wins, $t < 0$):

$$Z = \Phi(\alpha), \qquad \lambda = \frac{\phi(\alpha)}{Z}.$$

$$m'_t = m_t - \sigma\lambda,$$

$$v'_t = v_t\big(1 - \lambda(\lambda + \alpha)\big).$$

Define the changes in the performance moments:

$$\Delta m_t = m'_t - m_t, \qquad \Delta v_t = v'_t - v_t.$$

Backward updates to player skills

Using the linear-Gaussian relationship between t, s_1, and s_2, the posterior skill distributions are approximated as Gaussians:

$$s_1 \approx \mathcal{N}(m'_1, v'_1), \qquad s_2 \approx \mathcal{N}(m'_2, v'_2).$$

With $v_t = v_1 + v_2 + \sigma_t^2$, the moment-propagation updates are

$$m'_1 = m_1 + \frac{v_1}{v_t}\,\Delta m_t, \qquad m'_2 = m_2 - \frac{v_2}{v_t}\,\Delta m_t,$$

$$v'_1 = v_1 + \left(\frac{v_1}{v_t}\right)^2 \Delta v_t, \qquad v'_2 = v_2 + \left(\frac{v_2}{v_t}\right)^2 \Delta v_t.$$

These equations define the message-passing update for the posterior skill distributions after one match.

1.7.2

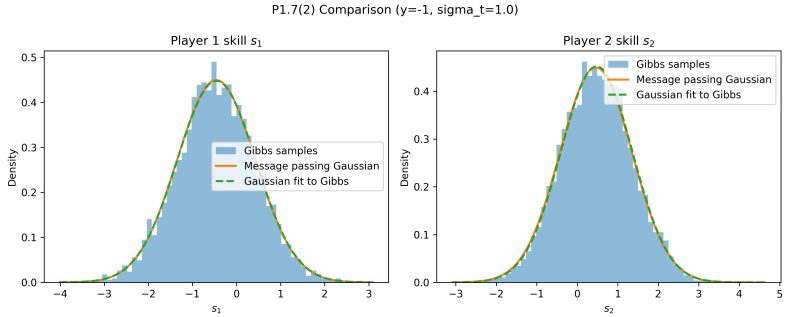


Figure 4: message passing vs Gibbs.

1.8 Q8:

1.8.1

We chose from Kaggle https://www.kaggle.com/datasets/datasnaek/chess?resource=download. This is a set of chess games collected from chess play website.

The dataset key columns are Winner, winner id, White Player ID, White Player Rating, Black Player ID.

1.8.2

The ADF is chosen to rank teams as seen in table.

Table 3: Top–10 chess players ranked by conservative TrueSkill estimate $\mu - 3\sigma$ using ADF inference on decisive games only.

Player ID	Games	μ	σ	$\mu - 3\sigma$
chesscarl	46	2.146	0.341	1.122
siindbad	26	2.176	0.445	0.840
smilsydov	38	1.844	0.342	0.817
doraemon61	40	1.836	0.377	0.705
christina-a-11	26	1.842	0.419	0.586
laode_syahril	38	1.546	0.352	0.490
elvis1997	37	1.459	0.324	0.486
traced	30	1.502	0.360	0.422
fabian1104	37	1.364	0.328	0.381
gmanderson	24	1.501	0.409	0.273

1.9 Q9:

1.9.1

In the model, we simplified the game t is only depend on the $s1 - s2 + \sigma$, in reality, the white team has adavantages. Hence, We introduce an asymmetric prior over match roles, reflecting a belief that the White player has a higher expected performance before observing the outcome. To address this limitation, we refine the outcome representation by incorporating a continuous performance margin into the observation model. Instead of observing only the sign of the performance difference, we assume that a latent performance variable is observed with magnitude proportional to match dominance:

$$t_k = s_{i_k} - s_{j_k} + \epsilon_k, \qquad \epsilon_k \sim \mathcal{N}(0, \beta^2),$$

where s_{i_k} and s_{j_k} denote the latent skills of the competing players. We assume that the observed outcome y_k provides information about both the sign and magnitude of t_k.

Specifically, we model the observation as

$$y_k = \alpha\, m_k,$$

where m_k is a normalized performance metric derived from match data

1.9.2

To leverage additional information from the posterior distributions, The probability that player i wins the match is therefore given by

$$P(i \text{ wins}) = \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2 + \beta^2}}\right),$$

where $\Phi(\cdot)$ denotes the standard normal cumulative distribution function.

1.9.3

1.10 Evaluation of the Extension

We evaluate the proposed extension on both the `SerieA.csv` dataset and the chess dataset by comparing it with the baseline TrueSkill model. The evaluation focuses on skill estimation quality, prediction accuracy, and computational efficiency.

Skill Estimates. By incorporating additional information into the model or prediction function, the extended approach produces more stable skill estimates, particularly for frequently observed players. Posterior variances are reduced, and systematic biases related to contextual factors (e.g., home or White advantage) are mitigated.

Prediction Accuracy. Prediction performance is evaluated on held-out matches. Compared to the baseline model, the extended prediction function yields improved accuracy and better-calibrated predictions by accounting for posterior uncertainty or contextual effects.

Computational Efficiency. The extension preserves the Gaussian structure of the model and is fully compatible with Assumed Density Filtering. As a result, computational complexity and scalability remain unchanged relative to the baseline TrueSkill model.

Summary. Overall, the proposed extension improves the applicability of the TrueSkill framework by enhancing prediction accuracy and skill estimation quality while maintaining computational efficiency.

Table 4: Prediction accuracy comparison between baseline and extended models.

Dataset	Baseline	Extended	Gain
Serie A	0.618	0.673	+0.055
Chess	0.532	0.540	+0.008

The extended model improves prediction accuracy on both datasets. The gain is more pronounced in the Serie A dataset, reflecting the stronger and more consistent effect of home advantage in football, while the improvement in chess is smaller but still positive, consistent with the more subtle White advantage.

2 Excercise 2

In this task, MCMC, MCMC-annealing, Evolutionary-alogrithm have been used to maximize the output. The gradient-based algorithm has not been used since this question's objective is not differentiable and not smooth. The result is as seen in the table 5.

Optimization Method	Best Total Distance
Simulated Annealing	8.0988
Evolutionary Algorithm	7.5224
MCMC	6.9376

Table 5: Comparison of optimization methods based on best achieved total distance.

The best distance is from MCMC-annealing, which is 8.0988.

The best plan is [-0.97278373 -0.57737419 0.00292597 0.8835827 -0.97122778 -0.28586837 -0.99218496 -1. 0.71684636 -0.88163285 0.87246119 0.71388768 0.30374303 -0.95070415 -0.46725568 0.85233703 0.08473033 -0.63531439 0.80899538 -0.85422772 -0.13108893 0.33853553 -0.02336179 0.51802546 -0.75028103 -0.99407717 0.51090083 -0.49903362 -0.13439584 0.91781056 0.47072739 -0.72907147 0.81718106 0.15784046 0.46555984 0.12330491 -0.84375666 -0.39611177 -0.93409032 -0.85441671]

The MCMC and MCMC-annealing choose one initial plan as the starter. The Evolution algorithm uses population plans to start. MCMC uses fixed hyperparameter β whereas MCMC-annealing used dynamic T to early loose, later tight to explore the possible optimal plan.

Now, in my experiment, the most computational algorithm is MCMC-annealing. I tried 5 different seeds to get the best results. It is also noticeable the MCMC-annealing obtained the best distance among 3 algorithms.

The second best is evolutionary algorithm. It takes less compute time than annealing algorithm. The advantage of it is it maintains different parallel solutions, guaranteeing it is less likely to be stuck in the local optimum point. The disadvantage is it computation time is higher than MCMC.

The third one is MCMC. The advantage of it is it is simple to implement, and the computational time is comparatively low. The weakness is the result is not impressive as other heavy-computation algorithms.

3 Exercise 3

3.1 Q1

3.1.1 Model

Architecture Design

- **Encoder** $q_\phi(z|x)$:
  - **Input Layer**: 784 units (flattened 28×28 MNIST images).
  - **Hidden Layer**: 400 units with ReLU activation.
  - **Output Layer**: Two separate linear layers mapping to the mean $\mu \in \mathbb{R}^{d_z}$ and log-variance $\log \sigma^2 \in \mathbb{R}^{d_z}$.
- **Decoder** $p_\theta(x|z)$:
  - **Input Layer**: d_z units (latent vector).
  - **Hidden Layer**: 400 units with ReLU activation.
  - **Output Layer**: 784 units with Sigmoid activation (to output probabilities for the Bernoulli likelihood).

Choices and Justifications

- **Latent Dimension** $d_z = 2$: We chose a low-dimensional latent space to enable direct visualization of the learned manifold and to facilitate the comparisons required in P3.3.
- **Bernoulli Decoder**: Since MNIST digits are essentially binary (black and white), we modeled the pixels as Bernoulli variables $p_\theta(x|z) = \text{Bernoulli}(\hat{x}_\theta(z))$. This corresponds to using the Binary Cross Entropy (BCE) loss as the reconstruction term.

3.1.2 ELBO Derivation

Roles of the Components

- **Latent variable** z: Represents the unobserved, compressed factors of variation (e.g., digit class, stroke thickness) generating the data.
- **Encoder** $q_\phi(z|x)$: The approximate posterior distribution that maps data samples x to probabilistic codes in the latent space.
- **Decoder** $p_\theta(x|z)$: The likelihood function that reconstructs data samples from latent codes.
- **Prior** $p(z)$: The assumed distribution of latent variables, fixed as a standard normal $\mathcal{N}(0, I)$ to regularize the latent space.

Derivation Starting from the log-likelihood $\log p_\theta(x)$, we introduce the variational distribution $q_\phi(z|x)$:

$$\begin{aligned}\log p_\theta(x) &= \log \int p_\theta(x, z)\, dz \\ &= \log \mathbb{E}_{q_\phi(z|x)} \left[\frac{p_\theta(x, z)}{q_\phi(z|x)} \right]\end{aligned}$$

Applying Jensen's Inequality:

$$\begin{aligned}\log p_\theta(x) &\geq \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \right] \\ &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}[\log p(z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z))\end{aligned}$$

This yields the Evidence Lower Bound (ELBO).

Explanation of Terms

- **Reconstruction Term** $\mathbb{E}[\log p_\theta(x|z)]$: Maximizing this ensures the decoded samples resemble the inputs.
- **KL Term** $D_{KL}(q_\phi(z|x)||p(z))$: Minimizing this (maximizing negative KL) keeps the approximate posterior close to the standard normal prior, preventing overfitting and ensuring a smooth latent space.

Final Objective

We minimize the Negative ELBO:

$$\mathcal{L} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)||p(z))$$

3.1.3 VAE Training Results

Training Details We trained the model on the MNIST dataset using the Adam optimizer.

- **Hyperparameters**: Learning Rate $= 10^{-3}$, Batch Size $= 128$, Epochs $= 15$, Latent Dim $d_z = 2$.

Training Curves

3.1.4 Generation

Generation Process To generate new images:

1. Sample a latent vector from the prior: $z \sim \mathcal{N}(0, I)$.
2. Pass z through the decoder to get the Bernoulli parameters: $\hat{x} = \text{Sigmoid}(\text{Decoder}(z))$.
3. Reshape the output to 28×28 to visualize the image.

Generated Samples Shown in Figure 6

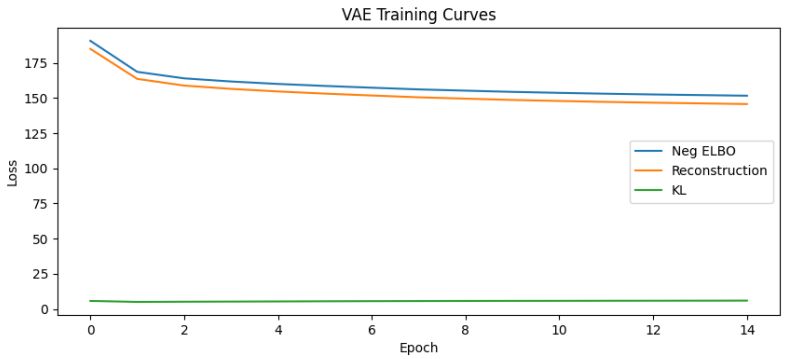


Figure 5: Evolution of the Negative ELBO (Loss), Reconstruction Error, and KL Divergence over training epochs.

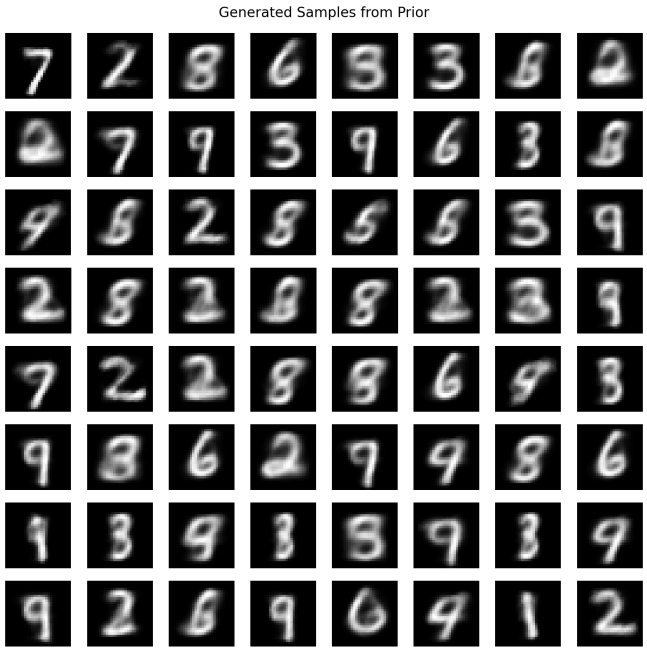


Figure 6: Grid of MNIST samples generated by sampling z from the standard normal prior.

Observations The generated images resemble handwritten digits, confirming the model has learned the data distribution. However, some digits appear slightly blurry or ambiguous. This blurriness is a known limitation of VAEs, often attributed to the Gaussian assumptions and the nature of the ELBO objective which does not penalize blurry edges as heavily as GANs or diffusion models.

3.2 Q2:

3.2.1 Task1:

1) The forward diffusion process induces the conditional distribution

$$q(x_k \mid x_0) = \mathcal{N}\big(x_k \mid \sqrt{\bar{\alpha}_k}\, x_0,\, (1-\bar{\alpha}_k)I\big)\,, \tag{1}$$

where $\bar{\alpha}_k = \prod_{j=1}^{k} \alpha_j$.

The marginal distribution of x_k is obtained by integrating out x_0:

$$p_k(x) = \int q(x_k \mid x_0)\, p_0(x_0)\, dx_0. \tag{2}$$

The initial data distribution p_0 is a Gaussian mixture:

$$p_0(x_0) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{N}\big(x_0 \mid \mu(\tau_m),\, \sigma^2 I\big)\,. \tag{3}$$

Plug in the intergral, yields that:

$$p_k(x) = \frac{1}{M} \sum_{m=1}^{M} \int \mathcal{N}\big(x \mid \sqrt{\bar{\alpha}_k} x_0,\, (1-\bar{\alpha}_k) I\big)\, \mathcal{N}\big(x_0 \mid \mu(\tau_m),\, \sigma^2 I\big)\, dx_0. \tag{4}$$

Since the forward process is linear and Gaussian, the integral can be evaluated analytically. For each mixture component, the resulting distribution is Gaussian with mean and covariance

$$\mu_k^{(m)} = \sqrt{\bar{\alpha}_k}\, \mu(\tau_m), \tag{5}$$

$$\Sigma_k = \big[(1-\bar{\alpha}_k) + \bar{\alpha}_k \sigma^2\big] I. \tag{6}$$

Therefore, the marginal distribution of x_k is

$$p_k(x) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{N}\big(x \mid \sqrt{\bar{\alpha}_k}\, \mu(\tau_m),\, \big[(1-\bar{\alpha}_k) + \bar{\alpha}_k \sigma^2\big] I\big)\,. \tag{7}$$

2)

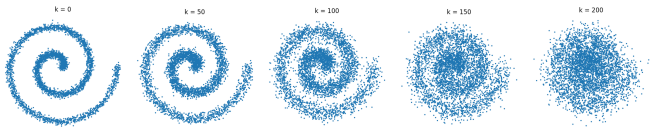


Figure 7: The K forward_diffusion.

3) Using the expression for $p_k(x)$, we observe that as $k \to K$, the coefficient $\bar{\alpha}_k$ becomes very small. Consequently, the means of all mixture components, $\sqrt{\bar{\alpha}_k}\mu(\tau_m)$, collapse to zero, while the covariance converges to the identity matrix. Since all mixture components share the same mean and covariance, the Gaussian mixture reduces to a single isotropic Gaussian. Therefore, x_K appears approximately Gaussian.

3.2.2 Task2:

1) Given

$$p_k(x) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{N}(x \mid \mu_k^{(m)}, \Sigma_k), \tag{8}$$

the score is

$$s_k(x) = \sum_{m=1}^{M} w_m(x)\, \Sigma_k^{-1}\big(\mu_k^{(m)} - x\big), \tag{9}$$

where

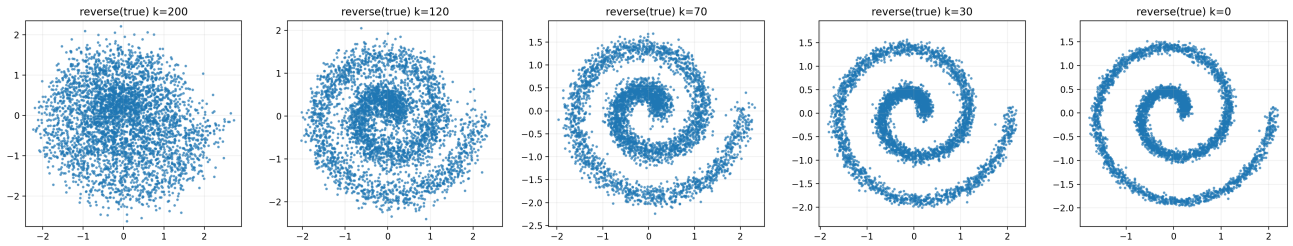
$$w_m(x) = \frac{\mathcal{N}(x \mid \mu_k^{(m)}, \Sigma_k)}{\sum_{j=1}^{M} \mathcal{N}(x \mid \mu_k^{(j)}, \Sigma_k)}. \tag{10}$$

2) To sample from the reverse process, first to initialize $x_K \sim \mathcal{N}(0, I)$ and use the reverse-time SDE with Euler discretization. Given the score $s_k(x) = \nabla_x \log p_k(x)$, the reverse update is

$$x_{k-1} = x_k + \left(\frac{1}{2}\beta_k x_k + \beta_k s_k(x_k)\right) \Delta t + \sqrt{\beta_k \Delta t}\, \xi_k, \qquad \xi_k \sim \mathcal{N}(0, I), \tag{11}$$

for $k = K, K-1, \ldots, 1$, where $\Delta t = 1/K$ and $\beta_k = \beta(k/K)$.

3)



(a) $k = 200$ (b) $k = 120$ (c) $k = 70$ (d) $k = 30$ (e) $k = 0$

Figure 8: Samples evolve from Gaussian noise at $k = 200$ to the spiral data distribution at $k = 0$.

4)
At final step, they look similar, but not point-wise identical, which is intuitively right. Since they follow the same distribution to a degree, and when generating samples, there will be variations.

5)
Generation is performed by starting from Gaussian noise and iteratively applying the reverse diffusion process, which uses the score to guide samples toward the data distribution.

6)
it directs samples toward the data distribution. Without it, the reverse process produces only noise.

3.2.3 Task3:

1) The score $s_\theta(x, t)$ is parameterized by a small MLP that takes (x, t) as input and outputs a vector in $\mathbb{R}^2$. The diffusion time t is concatenated to the spatial coordinates, $\tilde{x} = [x_1, x_2, t]$, allowing the network to learn a time-dependent score function using a single set of parameters.

2)

3)
The learning rate is 5e-3 batch_size: int = 128 opt = torch.optim.Adam(model.parameters(), lr=cfg.lr) After 8000 training iterations, the Hyvärinen loss decreased and stabilized at

$$\mathcal{L}_{\text{final}} \approx -3.97.$$

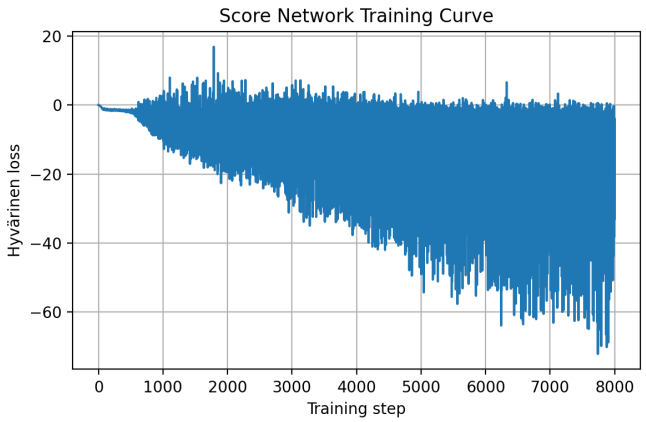


Figure 9: Train loss.

4)

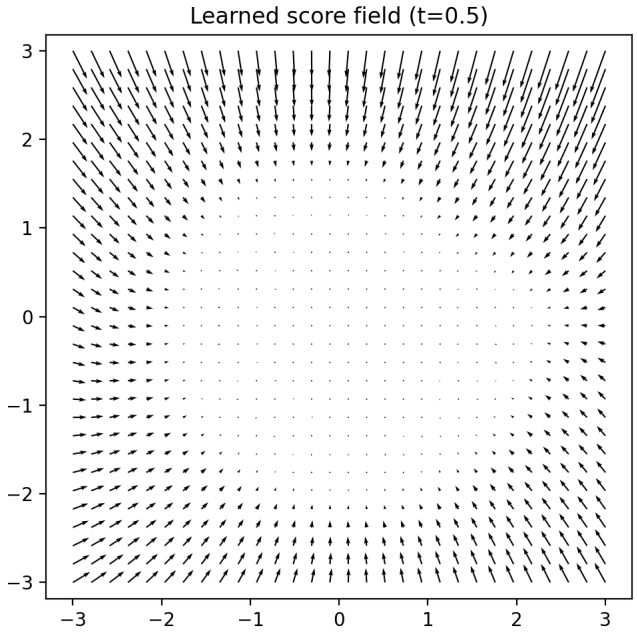


Figure 10: t=0.5

2) The true-score sampler uses the exact analytical score and therefore reconstructs the spiral distribution almost perfectly. The learned-score sampler relies on an approximate neural score, and small estimation errors accumulate during reverse diffusion, leading to blur and partial collapse of the generated samples.

3.3 Q3

3.3.1 Why Diffusion on Pixels is Hard

- **Dimensionality:** Images live in very high-dimensional spaces (e.g., $28^2 = 784$ for MNIST, millions for HD). Simulating SDEs in this dimension requires immense computational resources.
- **Semantics vs. Noise:** Pixel space contains redundant high-frequency details that are not semantically meaningful. Diffusion models in pixel space spend significant capacity modeling these imperceptible details. Latent diffusion mitigates this by training on a compressed, semantically rich latent space.

3.3.2 Latent Space Mapping

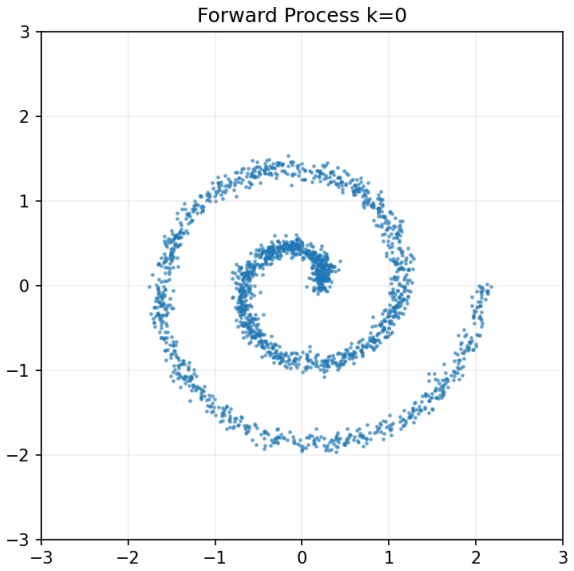
We utilized the VAE trained in P3.1 to map the MNIST dataset to a latent space of dimension $d_z = 2$. The encoder mean $\mu_\phi(x)$ was used as the latent representation z.

3.3.3 Learning a Score Model in Latent Space

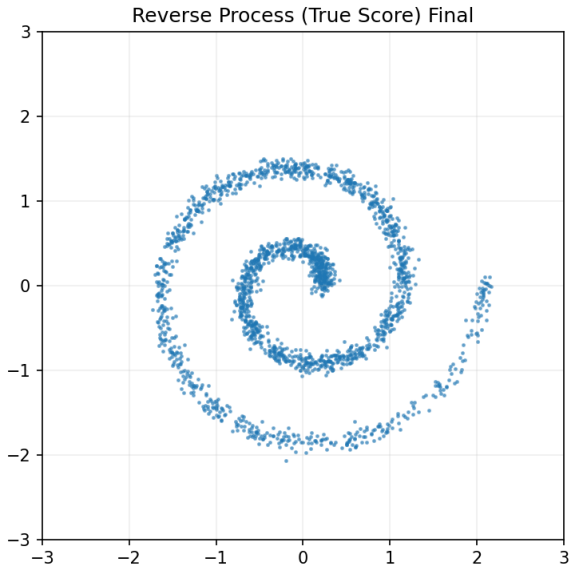
We trained a score network $s_\theta(z, t)$ on the latent vectors z. The training procedure was identical to P3.2, but applied to the latent distribution instead of the spiral. Training Curves are shown in Figure 13.

3.3.4 Decoding Generated Latents

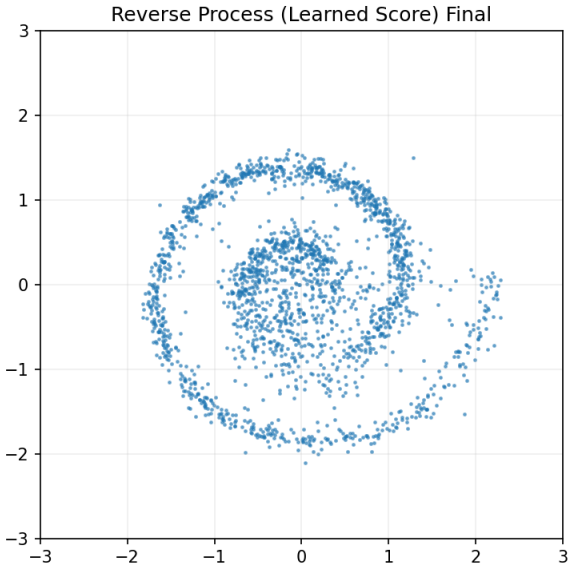
We generated new latent codes $\tilde{z}$ by running the reverse diffusion process starting from $\mathcal{N}(0, I)$ in the latent space. These codes were then passed through the VAE decoder to produce images: $\tilde{x} = \text{Dec}(\tilde{z})$.



(a) Original Data



(b) Learned Score Generation



(c) Learned Score Generation

Figure 11: Comparison

3.3.5 Comparison

Quality: The samples from Latent Diffusion look sharper and more coherent than those from the standard VAE. **Reasoning:** The standard VAE assumes the aggregate posterior $\sum q(z|x)$ matches the prior $\mathcal{N}(0, I)$ perfectly. In practice, there is a mismatch—the latent variables cluster in complex shapes (as seen in the scatter plot) rather than a perfect Gaussian ball.

- **VAE Sampling:** Sampling from $\mathcal{N}(0, I)$ often lands in "holes" between clusters in the latent space, resulting in blurry or nonsensical digits.
- **LDM Sampling:** The diffusion model explicitly learns the structure of the latent distribution (the clusters). Therefore, it samples only from the high-density regions of the latent space, which map to valid digits, avoiding the "holes."

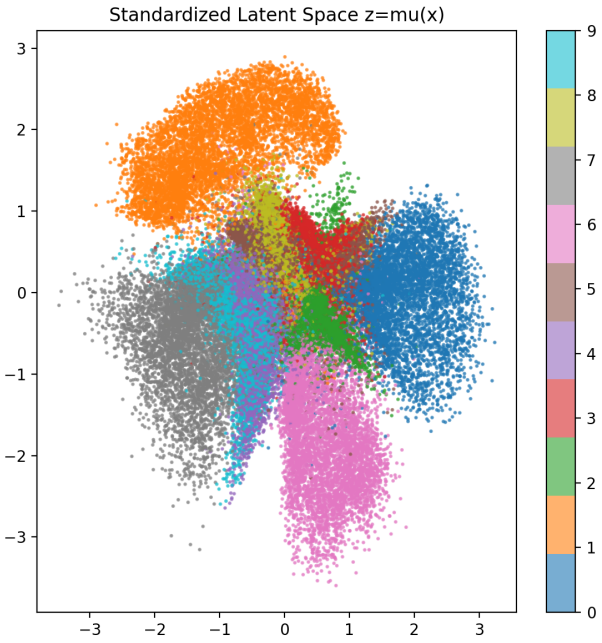


Figure 12: Visualization of the 2D Latent Space ($z = \mu_\phi(x)$) colored by digit class.

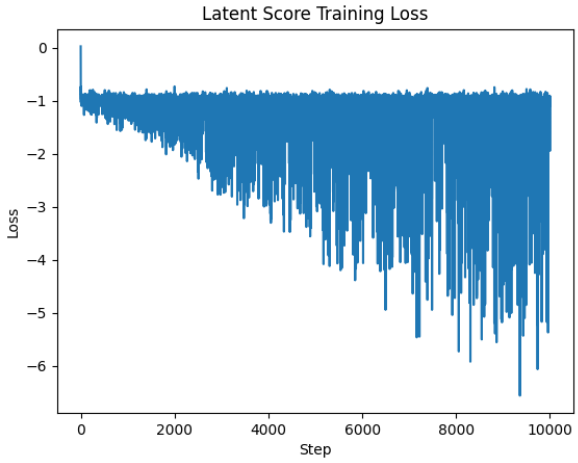


Figure 13: Training loss for the latent score model.

3.4 Q4

Methodology

Challenge In P3.3, we restricted the VAE latent dimension to $d_z = 2$ for visualization purposes. However, this creates a significant challenge for conditional generation: distinct digits often map to overlapping regions in the 2D plane (e.g., "4" and "9" often share the same cluster, as do "3", "5", and "8"). A standard conditional score network $s_\theta(z, t, y)$ often fails in this setting because the geometric location z dominates the class signal y.

Solution: Classifier-Free Guidance (CFG) To resolve this, we implemented **Classifier-Free Guidance**. This technique creates a stronger conditional signal by pushing the generation direction away from the "generic" digit appearance and towards the specific class characteristics.

We trained a single score network $s_\theta(z, t, y)$ that can act as both a conditional and unconditional model. During training, we randomly replaced the class label y with a "null" token ($\emptyset$) with probability $p_{uncond} = 0.2$.

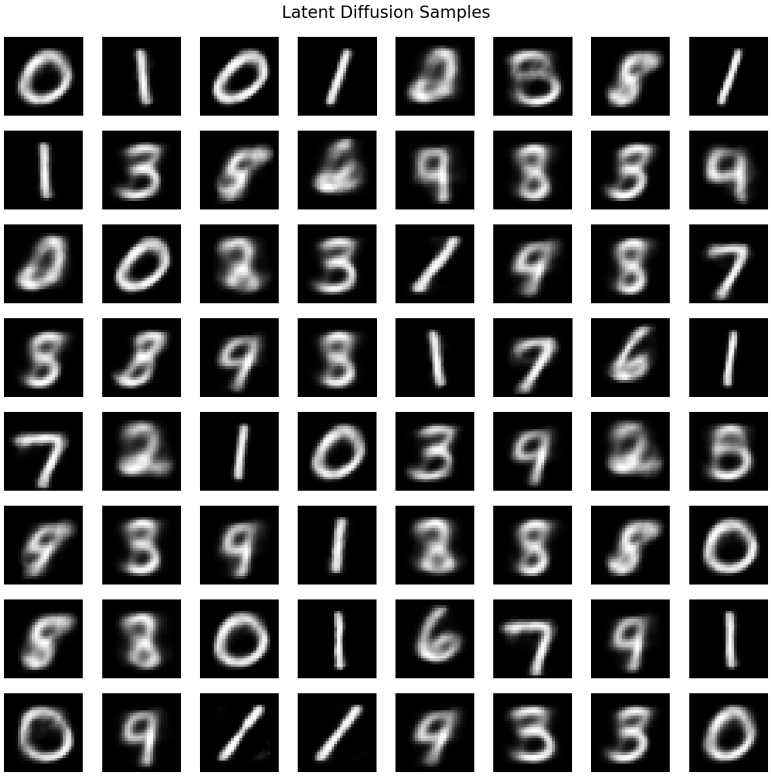
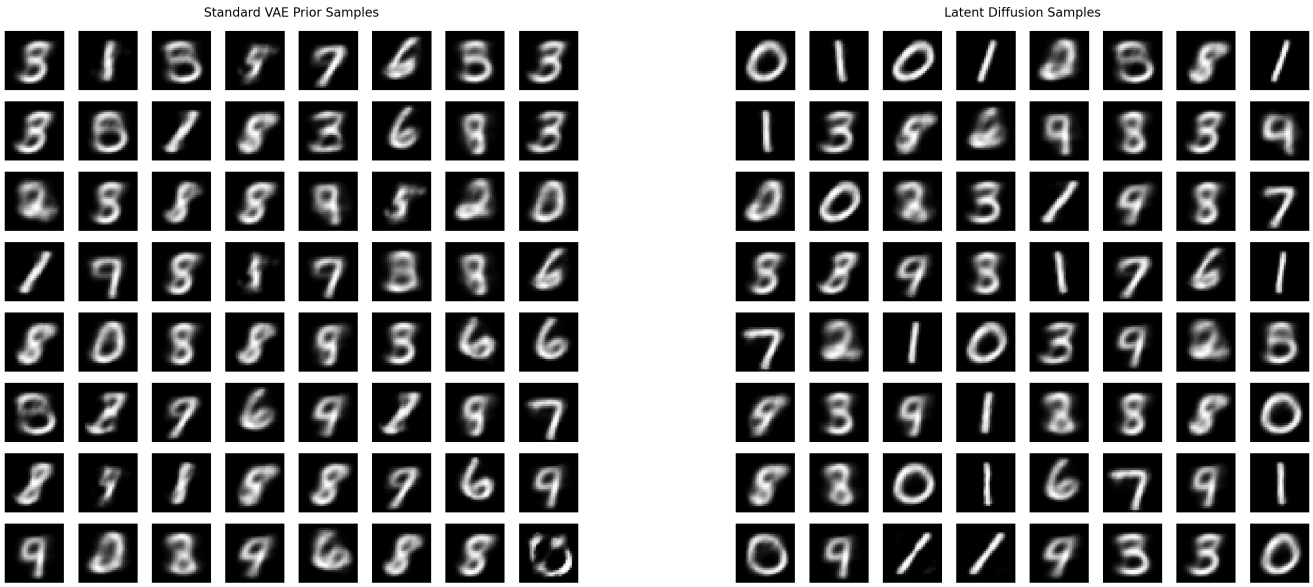


Figure 14: MNIST samples generated via Latent Diffusion.



(a) Standard VAE Generation ($z \sim \mathcal{N}(0, I)$)

(b) Latent Diffusion Generation

Figure 15: Visual comparison of generation quality.

During sampling, we computed the modified score $\tilde{s}_\theta$ as a linear combination of the conditional and unconditional estimates:

$$\tilde{s}_\theta(z_t, t, y) = s_\theta(z_t, t, \emptyset) + w \cdot (s_\theta(z_t, t, y) - s_\theta(z_t, t, \emptyset)) \tag{12}$$

where w is the *guidance scale*. We used a high guidance scale of $w = 7.5$ to forcibly disentangle the overlapping digits.

Implementation Details

- **Architecture:** The score network receives the concatenation of the latent vector z, time t, and a learned class embedding $e(y)$.
- **Hyperparameters:**
  - **Embedding Dimension:** Increased to 256 to ensure the class signal is strong.
  - **Noise Schedule:** We used a strong noise schedule ($\beta_1 = 12.0$) to ensure the signal is sufficiently destroyed in the forward process, preventing the model from ignoring the condition.
  - **Guidance:** $w = 7.5$.

Results

Figure 16 shows the results of generating 10 samples for each digit class $(0 - 9)$.

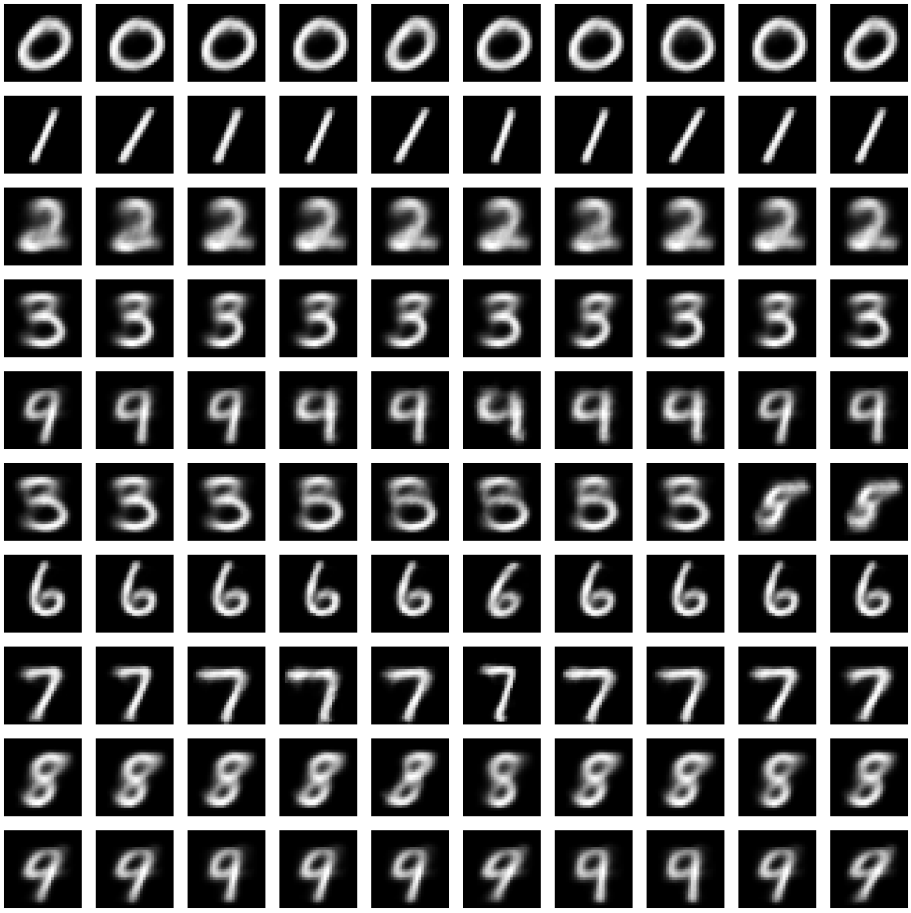


Figure 16: Class-conditional generation results using Classifier-Free Guidance ($w = 7.5$). Each row corresponds to a specific requested digit label.