Apendix of the project

Ana del Río Darío González

July 7, 2025

1 Introduction

In this appendix we gather supplementary material that complements the main report: first, a detailed overview of all hyperparameters used (their meaning, how they were tuned, and why we chose their final values); next, a description of initial ideas that proved unworkable and how we adapted our approach; and finally, a log of the technical issues encountered during implementation and the solutions applied to ensure a robust, reproducible code base.

2 Hyperparameters

2.1 Hyperparameter descriptions

We tuned the following key hyperparameters in our pipeline:

VAE latent_dim (D) Dimensionality of the continuous latent code.

Learning rate (η) Step size for the Adam optimizer in both VAE and VQ-VAE.

Batch size (B) Number of samples per gradient update.

Commitment weight (β) Scales the code book commitment loss in VQ-VAE.

Code book size (K) Number of discrete embeddings in the VQ and Geo-VQ code book.

k-NN neighbors (k) Number of neighbors in the latent-space graph used to compute geodesic distances.

Number of clusters (K_{geo}) Number of clusters for Geodesic k-means.

AR seq_len (M) Sequence length (history) for the auto regressive RNN.

AR embed_dim & hidden_dim Embedding and hidden sizes in the GRU for code prediction.

2.2 Hyperparameter selection process

To pick sensible defaults, we performed:

- *Grid search* over $D \in \{10, 20, 30\}$ and observed ELBO on validation set.
- *Sweep* of $\beta \in \{0.1, 0.25, 0.5\}$ to balance reconstruction vs. commitment loss.
- *Graph sensitivity* tests for $k \in \{5, 10, 15\}$ and clusters $K_{\text{geo}} \in \{32, 64, 128\}$.
- *Optimizer tuning* by trying $\eta \in \{1e^{-3}, 5e^{-4}, 1e^{-4}\}$ for stable convergence.
- *AR configuration* comparing GRU vs. LSTM, and embed_dim$\in \{32, 64\}$, hidden_dim$\in \{128, 256\}$ based on NLL.

2.3 Chosen values and justification

After evaluating validation performance, we fixed:

Param	Value	Reason
D	20	Best ELBO vs. model complexity trade-off.
η	10^{-3}	Fast initial convergence without instability.
B	128	Balanced GPU-memory use and gradient estimate quality.
β	0.25	Reduced commitment loss without harming reconstruction.
K	64	Sufficient code diversity with manageable memory.
k	10	Captures local geometry accurately in the k-NN graph.
K_{geo}	64	Matched the VQ-VAE codebook size for fair comparison.
M	32	Provides enough context for AR sampling without overfitting.
embed_dim/hidden_dim	64/128	GRU with these sizes achieved lowest NLL (3.13 bits/token).

These settings offered the best balance of reconstruction quality, code book compactness, and generative performance.

3 Failed Ideas

Post-hoc Euclidean Quantization At the start, we tried applying standard k-means clustering directly on the continuous VAE latents using euclidean distance. However, this produced a test ELBO of roughly 1.40×10^6—significantly worse than the baseline VAE—and reconstructions were noticeably degraded. The flat euclidean metric failed to respect the curved structure of the latent manifold. We therefore abandoned euclidean quantization and adopted geodesic k-means on a k-NN graph, which yielded coherent clusters and recovered ELBO comparable to the VAE.

EMA-based Code book Updates We also implemented the Exponential Moving Average (EMA) variant of the VQ-VAE code book update (van den Oord et al., 2017). Although EMA can stabilize code book learning in theory, our version suffered from erratic embedding updates and occasional collapse, resulting in highly variable commitment losses. Consequently, we reverted to the simpler commitment-loss-only formulation (no EMA), which converged reliably and delivered consistent reconstruction performance.

4 Failures and issues

During the development of the project we faced and solved several problems:

- **Missing paths and modules.** Running `train_vae.py` (and other scripts) from the project root produced `ModuleNotFoundError: No module named 'models'` or `No module named 'utils'`. We fixed this by pre-prepending each script with:

```
sys.path.append(str(Path(__file__).resolve().parent.parent))
```

  and defining `ROOT`, `CHECKPOINT_DIR` and `LOG_DIR` via `pathlib.Path`.
- **Model and log saving issues.** Checkpoints and log directories (`experiments/checkpoints`, `experiments/logs`) were not created automatically, so nothing was saved. We added `Path.mkdir(parents=True, exist_ok=True)` and switched CSV writes from write mode (`w`) to append mode (`a`) to avoid overwriting and `PermissionError`s.
- **Shape mismatches in reconstruction evaluation.** In `evaluate_reconstructions.py` the GeoVQ decoder returned tensors of shape $(N, D, 784)$ while targets were $(N, 784)$, causing BCE errors. We corrected this by:
  - Flattening both predictions and targets to $(N, 28 \times 28)$.
  - Loading geodesic codes with `np.load(...)` instead of `torch.load(...)`.
  - Adding missing `import torch.nn.functional as F`.
- **Errors in `plot_comparison.py`.** We saw `IndexError` from mismatched titles vs. axes, and `TypeError` when plotting vectors of length 784 without reshaping. We fixed it by reshaping each image via `.view(28,28)` before plotting.
- **Auto regressive model dimension mismatches.** The generation script expected embedding dimension 32 but the checkpoint used 64. We aligned all scripts to `EMBED_DIM=64` and added a check to ensure `models/autoregressive_model.pt` exists, with a clear error message otherwise.
- **Dependency and reproducibility issues.** We had version conflicts between PyTorch releases and utility packages, so, we created a strict `requirements.txt` and tested installation in a clean virtual environment (`venv` or `conda`) to guarantee reproducibility.