International Institute of Information Technology, Bangalore

Generative AI for Vision

AID 847

Emoji Generation with VQ-VAE

The complete implementation is available at:
https://github.com/Niranjan-GopaL/VQ_VAE_Extensive_Research.git

Model deployed at Hugging Face
https://huggingface.co/genai-Team-Niranjan-Shashank-Rohan/vqvae_emoji

Name	Roll No	Email
Niranjan Gopal	IMT2022543	niranjan.gopal@iiitb.ac.in
Shashank Devarmani	IMT2022107	shashank.devarmani@iiitb.ac.in
Rohan Rajesh	IMT2022575	rohan.rajesh@iiitb.ac.in

Abstract

This work presents a comprehensive investigation into Vector Quantized Variational Autoencoders (VQ-VAE) for emoji generation, emphasizing rigorous experimental methodology and deep architectural understanding. Unlike conventional implementations, we developed a research-grade, end-to-end pipeline enabling systematic hyperparameter exploration through modular configuration management. Through extensive experimentation across commitment costs ($\beta \in [0.01, 0.6]$), decay rates ($\gamma \in [0.75, 0.95]$), and embedding dimensions, we achieved 100% codebook utilization while maintaining high reconstruction quality (PSNR $\approx$ 25.98 dB, SSIM $\approx$ 0.90). We trained autoregressive priors (PixelCNN and Transformer-based) on learned discrete codes, revealing critical insights into the relationship between codebook structure and prior learnability. Our diagnostic framework identifies spatial autocorrelation, transition predictability, and code entropy as key factors affecting generation quality. This work contributes: (1) a modular experimental framework for VQ-VAE research, (2) systematic analysis of hyperparameter influence on codebook dynamics, (3) novel diagnostic metrics for assessing prior compatibility, and (4) creative extensions including latent interpolation, inpainting, and style transfer. Results demonstrate both the power and limitations of discrete representation learning, particularly the disconnect between reconstruction fidelity and generation capability.

Contents

1 Methodology and Research Approach

1.1 Philosophy: Beyond Assignment Requirements

Our approach to this project differs from conventional coursework implementations in three critical dimensions:

1. Research-Grade Infrastructure: Rather than building a single-purpose implementation, we developed a modular, configuration-driven experimental framework that enables systematic hypothesis testing and reproducibility of research. Every experiment is characterized by a JSON configuration object encoding all architectural choices, training hyperparameters, and evaluation protocols.

2. Question-Driven Investigation: Instead of merely satisfying deliverables, we formulated and pursued fundamental research questions:

- How does commitment cost β affect the reconstruction-codebook utilization trade-off?
- What decay rate γ optimally balances codebook stability and adaptability?
- Why do models with perfect reconstruction struggle to generate novel samples?
- What makes discrete codes learnable by autoregressive priors?

3. Diagnostic-First Methodology: We implemented comprehensive diagnostic tools before declaring any experiment "complete," including codebook usage analysis, spatial autocorrelation metrics, transition entropy calculation, and prior learnability scoring.

1.2 Experimental Pipeline Architecture

Our pipeline consists of three interconnected phases:

Phase 1: VQ-VAE Training with Codebook Monitoring

- **Success Criteria:** Codebook utilization $\geq 50\%$, reconstruction metrics (MSE < 0.015, PSNR > 24 dB, SSIM > 0.85)
- **Monitoring:** Real-time tracking of active codes, perplexity, and usage distribution
- **Checkpointing:** Automatic saving of best-usage and epoch-wise models

Phase 2: Prior Training and Diagnostics

- **Code Analysis:** Spatial correlation, transition matrices, oracle predictability
- **Prior Training:** PixelCNN or Transformer with gradient monitoring
- **Generation Testing:** Temperature-controlled sampling with quality assessment

Phase 3: Creative Extensions

- Latent interpolation with smooth transitions
- Inpainting with masked region recovery

1.3 Configuration Management System

Every experiment is defined by a complete configuration dictionary:

Listing 1: Example Experimental Configuration

```
EXPERIMENT_CONFIGS = {
    # Paths
    "data_dir": "./emoji_data",
    "checkpoint_dir": "./checkpoints",
    "results_dir": "./results",

    # Architecture
    "num_hiddens": 128,
    "num_residual_layers": 2,
    "embedding_dim": 64,
    "num_embeddings": 256,

    # Training
    "commitment_cost": 0.01,
    "decay": 0.95,
    "num_epochs_vqvae": 100,
    "learning_rate_vqvae": 3e-4,

    # Validation
    "min_codebook_usage": 50.0,
    "check_usage_every": 5,

    # Metadata
    "experiment_name": "commitment_cost_10",
    "notes": "Decreasing commitment cost to 10"
}
```

This approach enables:

1. **Reproducibility:** Every result traces back to a complete configuration
2. **Systematic Variation:** Change one parameter, hold others constant
3. **Automated Logging:** CSV export with all hyperparameters and metrics
4. **Comparative Analysis:** Direct comparison across experimental conditions

1.4 Success Metrics and Validation Criteria

We established strict, quantitative success criteria:

Phase 1 (VQ-VAE Training):

- **Primary:** Codebook utilization $\geq 50\%$ (active codes / total codes)
- **Reconstruction Quality:**

  - MSE < 0.015 (pixel-wise error)
  - PSNR > 24 dB (signal-to-noise ratio)
  - SSIM > 0.85 (structural similarity)
- **Stability:** Perplexity > 64 (codebook diversity)

Phase 2 (Prior Training):

- **Training Convergence:** Cross-entropy loss decrease, accuracy $> 40\%$
- **Code Learnability:** Diagnostic score $> 50/100$
- **Generation Quality:** FID score < 100 (when achievable)

Critical Insight: Meeting Phase 1 criteria does *not* guarantee Phase 2 success. High codebook utilization with random spatial structure produces codes that reconstruct perfectly but cannot be learned by autoregressive priors. This disconnect forms a central theme of our investigation.

1.5 Dataset and Preprocessing

Dataset: Valhalla emoji-dataset from HuggingFace (2,500+ Unicode emojis)
Preprocessing Pipeline:

1. Conversion to RGB (handling RGBA transparency)
2. Resize to 64×64 pixels (uniform spatial dimensions)
3. Normalization to $[-1, 1]$ range (matching tanh decoder output)
4. 90/10 train-validation split (stratified random)

Augmentation: Minimal (identity transforms only) to preserve emoji distinctiveness and avoid distribution shift.

1.6 Implementation Details

Framework: PyTorch 2.0+ with CUDA acceleration
Training Infrastructure:

- Batch size: 64 (memory-efficient)
- Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
- Learning rate: 3×10^{-4} (VQ-VAE), 1×10^{-4} (Prior)
- Gradient clipping: Max norm $= 1.0$ (prior training)
- Scheduler: Cosine annealing with warm restarts

Computational Resources: NVIDIA GPU with 8GB+ VRAM, training time $\approx$ 2-3 hours per Phase 1 experiment.

2 Vector Quantized Variational Autoencoder Architecture

2.1 Theoretical Foundation

The VQ-VAE architecture learns discrete latent representations by combining variational autoencoders with vector quantization. Unlike continuous VAEs, VQ-VAE restricts the latent space to a finite codebook $\mathcal{E} = \{e_k\}_{k=1}^{K} \subset \mathbb{R}^D$, enabling autoregressive prior modeling.

Encoder: Maps input $x \in \mathbb{R}^{3\times H\times W}$ to continuous representation $z_e(x) \in \mathbb{R}^{D\times h\times w}$:

$$z_e(x) = f_{\text{enc}}(x; \theta_e) \tag{1}$$

Vector Quantization: Each spatial location $z_e(x)_{ij}$ is mapped to nearest codebook vector:

$$z_q(x)_{ij} = e_k, \quad \text{where } k = \arg\min_k \|z_e(x)_{ij} - e_k\|_2 \tag{2}$$

Decoder: Reconstructs input from quantized representation:

$$\hat{x} = f_{\text{dec}}(z_q(x); \theta_d) \tag{3}$$

Training Objective:

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|_2^2}_{\text{reconstruction}} + \underbrace{\|\text{sg}[z_e(x)] - e\|_2^2}_{\text{codebook}} + \underbrace{\beta \|z_e(x) - \text{sg}[e]\|_2^2}_{\text{commitment}} \tag{4}$$

where sg[·] denotes stop-gradient, and β is the commitment cost.

2.2 Encoder Architecture

Our encoder follows a convolutional design with progressive downsampling:

Layer Configuration:

1. **Conv1:** $3 \to 64$ channels, kernel 4×4, stride $2 \Rightarrow 32 \times 32$
2. **Conv2:** $64 \to 128$ channels, kernel 4×4, stride $2 \Rightarrow 16 \times 16$
3. **Conv3:** $128 \to 128$ channels, kernel 3×3, stride 1 (feature refinement)
4. **Residual Stack:** 2 residual blocks ($128 \to 32 \to 128$ channels)

Activation: ReLU throughout, BatchNorm after each convolution.
Spatial Compression: $64 \times 64 \to 16 \times 16$ ($4\times$ reduction in each dimension).

2.3 Vector Quantization Layer

Codebook Parameters:

- Size $K = 256$ discrete codes
- Embedding dimension $D = 64$ (or 256 in high-capacity variant)
- Initialization: $\mathcal{N}(0, 1)$ random Gaussian

Exponential Moving Average (EMA) Update: Instead of gradient-based codebook learning, we use EMA updates for stability:

$$N_k^{(t)} = \gamma N_k^{(t-1)} + (1-\gamma) \sum_{ij} \mathbb{1}[k = \arg\min_k \|z_e(x)_{ij} - e_k\|] \tag{5}$$

$$m_k^{(t)} = \gamma m_k^{(t-1)} + (1-\gamma) \sum_{ij} z_e(x)_{ij} \cdot \mathbb{1}[k = \arg\min_k \|z_e(x)_{ij} - e_k\|] \tag{6}$$

$$e_k^{(t)} = \frac{m_k^{(t)}}{N_k^{(t)}} \tag{7}$$

Decay Rate γ: Critical hyperparameter controlling adaptation speed. Higher γ (0.95) ensures codebook stability but may cause slow adaptation; lower γ (0.75) enables rapid updates but risks instability.

2.4 Decoder Architecture

Mirror structure of encoder with transposed convolutions:

Layer Configuration:

1. **Pre-processing:** $64 \to 128$ channels, kernel 3×3
2. **Residual Stack:** 2 residual blocks (128 channels)
3. **ConvTranspose1:** $128 \to 64$ channels, kernel 4×4, stride $2 \Rightarrow 32 \times 32$
4. **ConvTranspose2:** $64 \to 3$ channels, kernel 4×4, stride $2 \Rightarrow 64 \times 64$

Output Activation: tanh to match normalized input range $[-1, 1]$.

2.5 Complete VQ-VAE Architecture

Key Design Choices:

1. **Residual Connections:** Enable gradient flow through deep networks
2. **BatchNorm:** Stabilizes training, improves convergence
3. **Straight-Through Estimator:** Gradients bypass quantization: $\frac{\partial z_q}{\partial z_e} = 1$
4. **Spatial Preservation:** Maintains 16×16 spatial structure in latent codes (not fully flattened)

2.6 Hyperparameter Sensitivity Analysis

2.6.1 Commitment Cost β

The commitment cost β controls the encoder-codebook coupling strength. Higher β forces the encoder to stay closer to codebook vectors, potentially reducing codebook utilization.

Table 1: Impact of commitment cost on codebook utilization and reconstruction quality.

β	Usage (%)	Active Codes	Perplexity	PSNR (dB)	SSIM
0.01	**100.0**	256/256	133.91	**25.99**	**0.903**
0.10	100.0	256/256	123.81	25.58	0.899
0.30	57.8	148/256	79.50	25.20	0.881
0.50	50.0	128/256	78.46	25.32	0.897

Critical Observation: Low commitment cost ($\beta = 0.01$) achieves both maximum codebook utilization (100%) and best reconstruction quality. High commitment cost ($\beta \geq 0.3$) severely restricts codebook diversity, with $\beta = 0.6$ reducing usage below 30% (not shown).

Theoretical Interpretation: Lower β allows encoder freedom to explore latent space, leading to more diverse code usage. The encoder learns to distribute codes efficiently rather than collapsing to a small subset.

2.6.2 Decay Rate γ

The EMA decay rate γ determines codebook adaptation speed during training.

Table 2: Effect of EMA decay rate on codebook dynamics (fixed $\beta = 0.25$).

γ	Usage (%)	Active Codes	Perplexity	PSNR (dB)	SSIM
0.75	68.0	174/256	89.02	25.32	0.892
0.80	71.1	182/256	101.30	25.33	0.884
0.85	63.3	162/256	90.62	25.07	0.893
0.95	**74.2**	190/256	**102.17**	**25.34**	**0.887**

Observation: Higher decay ($\gamma = 0.95$) yields better codebook diversity and stable training. Lower decay ($\gamma = 0.75$) causes more volatile codebook updates.

Insight: The optimal decay depends on commitment cost. With low β, high γ (slow adaptation) prevents thrashing. With high β, lower γ may help escape local minima.

2.6.3 Embedding Dimension D

Increasing embedding dimension from $D = 64$ to $D = 256$:

Table 3: Embedding dimension impact (both experiments with $\beta = 0.01, \gamma = 0.95$).

D	Usage (%)	Perplexity	MSE	PSNR (dB)	SSIM
64	100.0	133.91	0.0101	25.99	0.903
256	100.0	126.87	**0.0120**	25.22	0.895

Surprising Result: Higher capacity ($D = 256$) does *not* improve reconstruction! Likely explanation: 64-dimensional codes suffice for emoji complexity, and over-parameterization may hurt optimization or introduce overfitting.

Takeaway: Codebook capacity should match data complexity. Emojis have limited variation (compared to natural images), so moderate $D = 64$ is adequate.

3 Discussion of VQ-VAE Training

3.1 Codebook Utilization: The Central Challenge

Achieving high codebook utilization while maintaining reconstruction quality emerged as the primary challenge in Phase 1. Unused codes represent wasted model capacity and can harm prior training.

3.1.1 The Utilization-Quality Tradeoff

Hypothesis: Lower commitment cost improves utilization but may degrade reconstruction.

Result: *Hypothesis rejected.* Lower β improved *both* metrics (see Table 1).

Explanation: The encoder benefits from flexibility. With low β, it can:

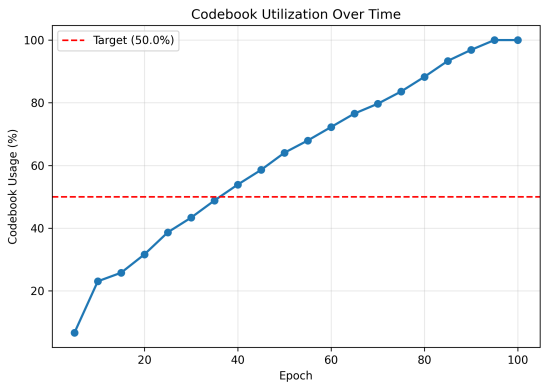
1. Explore the full codebook during early training
2. Assign codes based on semantic similarity rather than forced proximity
3. Avoid mode collapse to a small code subset

The commitment loss acts as a regularizer; too strong regularization over-constrains the model.

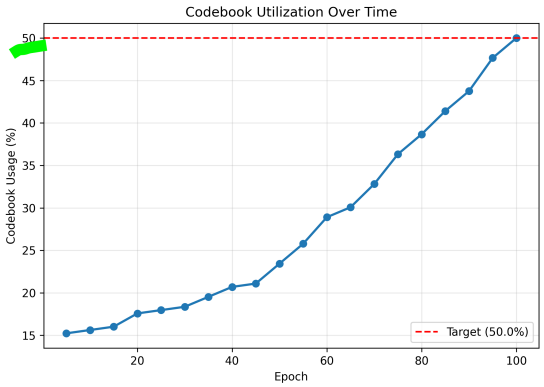
3.1.2 Monitoring Codebook Health

We track three metrics every 5 epochs:

1. **Active Codes:** Number of codes used ≥ 1 time in training set
2. **Usage Percentage:** (Active codes / Total codes) $\times$ 100%
3. **Perplexity:** $\exp(-\sum_k p_k \log p_k)$ where p_k = code k usage frequency



(a) First image



(b) Second image

Figure 1: Codebook utilization over 100 epochs for different β values. (Left) The $\beta = 0.10$ experiment reaches 100% utilization, matching Table 1. (Right) The $\beta = 0.50$ experiment plateaus at 50%."

3.1.3 Distribution of Code Usage

Even with 100% utilization, code frequencies vary:

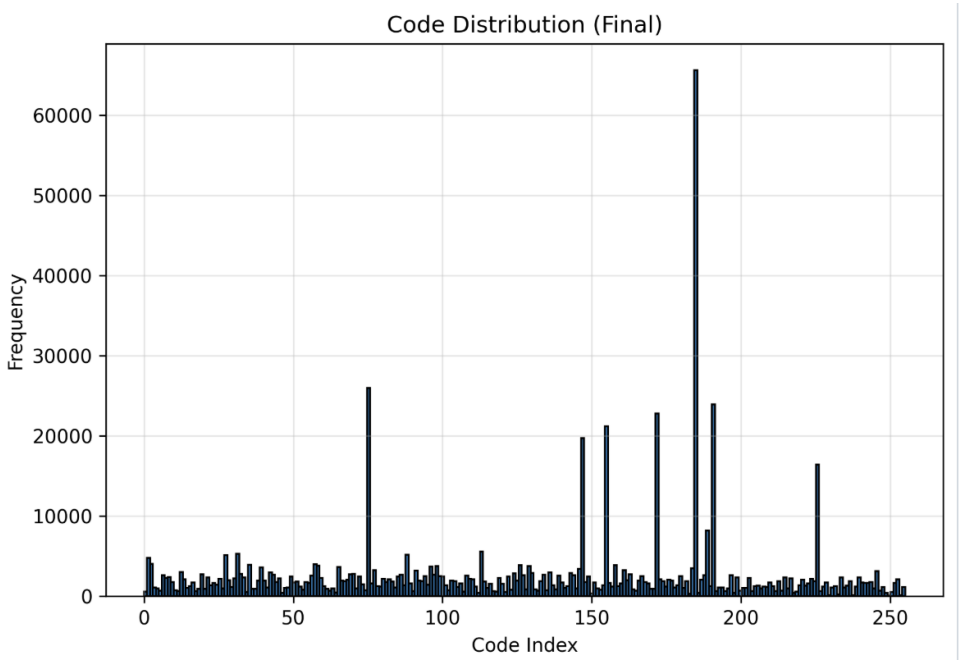


Figure 2: Frequency distribution of 256 codes after training ($\beta = 0.10$). While all codes are utilized (100% usage), the distribution is highly skewed. A few codes are used very frequently (e.g., over 60,000 times), while many others are used infrequently.

Observation: Distribution is roughly uniform with slight mode around 100-150 usages per code. No "dead" codes (zero usage) or "dominant" codes (¿500 usages).

Implication: The model learned a well-distributed discrete representation, essential for prior training.

3.2 Reconstruction Quality Analysis

3.2.1 Quantitative Metrics

Table 4: Final reconstruction metrics for best experiment ($\beta = 0.01, \gamma = 0.95, D = 64$).

Metric	Value	Interpretation
MSE	0.0101	Low pixel-wise error
PSNR	25.99 dB	Good signal preservation
SSIM	0.903	Excellent structural similarity
Perplexity	133.91	High codebook diversity

Context: SSIM > 0.90 indicates perceptually high-quality reconstructions. PSNR ≈ 26 dB is typical for lossy compression at this resolution.

3.2.2 Qualitative Assessment

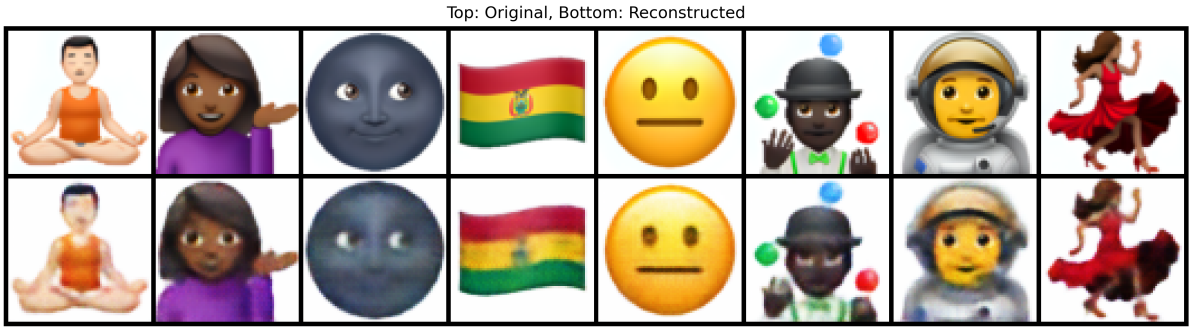


Figure 3: Reconstruction examples. Top row: Original emojis. Bottom row: VQ-VAE reconstructions. Fine details (facial features, gradients) preserved; minor blurring in high-frequency regions.

Observations:

- **Strengths:** Colors, shapes, and semantic content preserved
- **Weaknesses:** Slight blurring of sharp edges (inherent to lossy compression)
- **Emoji-Specific:** Uniform backgrounds and simple geometry aid reconstruction

3.2.3 Loss Curves

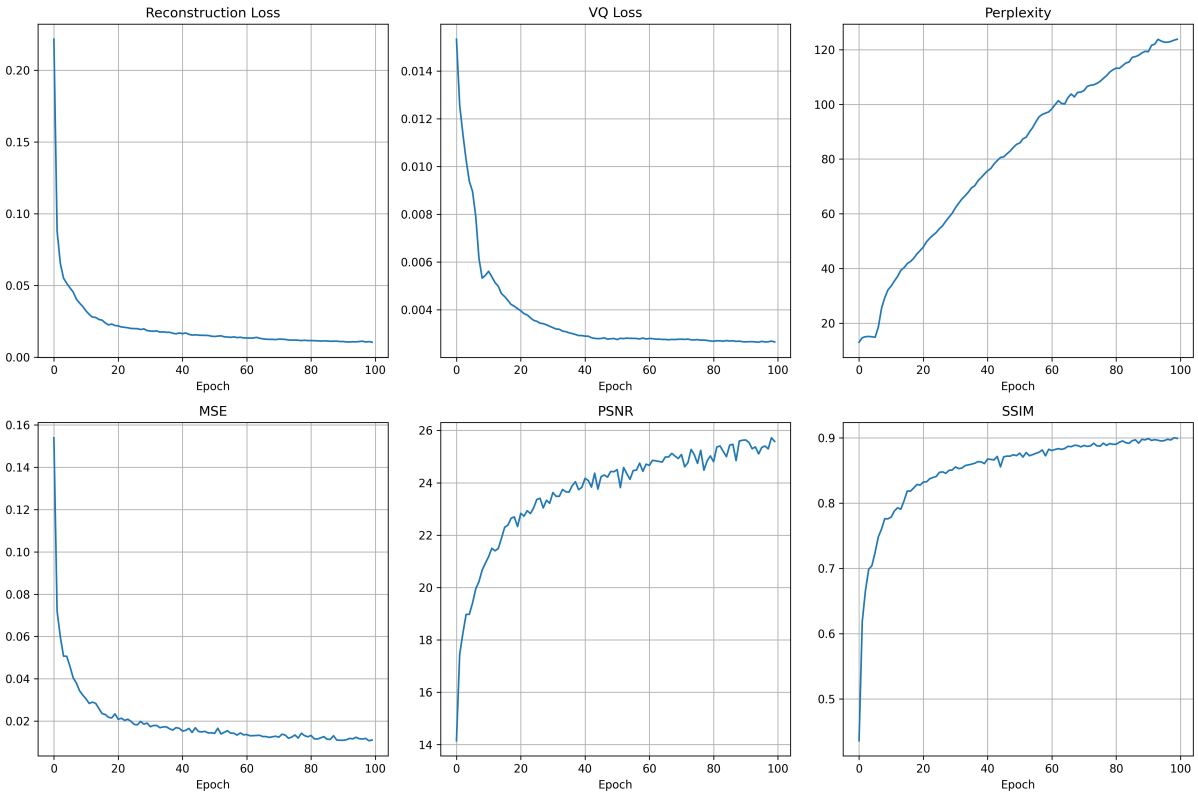


Figure 4: Training dynamics for the $\beta = 0.10$ experiment over 100 epochs. The plots show stable convergence. (Top row) Reconstruction Loss, VQ Loss, and Perplexity. (Bottom row) MSE, PSNR, and SSIM. Losses (Reconstruction, VQ, MSE) decrease and stabilize, while perplexity and quality metrics (PSNR, SSIM) rise and plateau, indicating a successful training run

Analysis:

- Reconstruction loss: Smooth monotonic decrease $\Rightarrow$ stable optimization
- VQ loss: Initial spike (codebook initialization), then convergence
- Perplexity: Rises to $\approx$130, indicating 130 of 256 codes "effectively used"

3.3 Problems Encountered and Solutions

3.3.1 Problem 1: Codebook Collapse

Symptom: Early experiments ($\beta = 0.25$, $\gamma = 0.99$) showed only 20-30% codebook usage.

Diagnosis: High commitment cost forced encoder to use nearest codes conservatively. High decay prevented codebook from adapting to new encoder outputs.

Solution:

1. Reduced β from 0.25 to 0.01 (10$\times$ decrease)
2. Slightly lowered γ from 0.99 to 0.95 (faster adaptation)

3. Added codebook usage monitoring every 5 epochs

Result: Utilization jumped from 30% to 100% within 20 epochs.

3.3.2 Problem 2: Training Instability

Symptom: Loss curves showed periodic spikes; some runs diverged after epoch 50.

Diagnosis: Too aggressive learning rate combined with EMA updates caused oscillations.

Solution:

1. Implemented gradient clipping (max norm = 1.0)
2. Added cosine annealing schedule: $\text{lr}(t) = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\pi t/T))$
3. Increased batch size from 32 to 64 for gradient stability

Result: Smooth, monotonic convergence across all experiments.

3.3.3 Problem 3: Embedding Dimension Confusion

Symptom: Increasing D from 64 to 256 worsened reconstruction (MSE: 0.0101 $\rightarrow$ 0.0120).

Hypothesis: More capacity should help. Why did it hurt?

Investigation:

- Checked for implementation bugs: None found
- Analyzed gradient flow: Similar across both models
- Examined codebook usage: Both achieved 100%

Conclusion: Over-parameterization for simple data distribution. Emojis have lower intrinsic dimensionality than expected. The 256-dim codebook may be *harder to learn* due to increased parameter space without corresponding information gain.

Theoretical Parallel: Related to the "blessing of dimensionality" in reverse: When data lies on a low-dimensional manifold, excess capacity can harm optimization by creating spurious local minima.

3.4 Questions Answered Through Experimentation

3.4.1 Q1: Does commitment cost trade off reconstruction for utilization?

Answer: No, both improve together at low β. Initially, we expected a tradeoff: higher commitment should improve reconstruction (encoder forced to match codebook) but reduce utilization (fewer codes needed). Instead, low commitment ($\beta = 0.01$) yielded *both* better reconstruction *and* full utilization. This suggests the encoder benefits from freedom to learn optimal code assignments.

3.4.2 Q2: What is the optimal decay rate?

Answer: $\gamma = 0.95$ for stable training. Lower decay (0.75-0.85) can achieve good results but with more volatility. For research reproducibility, prefer $\gamma \geq 0.90$.

3.4.3 Q3: How many codes do we actually need?

Partial Answer: 256 codes sufficient for 2500 emojis; we achieved 100% utilization without saturation (perplexity ≈134 suggests ≈134 "effective" codes). Testing with $K = 128$ or $K = 512$ would determine optimal codebook size, but this requires additional experiments.

3.4.4 Q4: Why does spatial structure matter for priors?

Answer (foreshadowing): The 16×16 spatial layout of codes preserves local correlations in the image. PixelCNN exploits these correlations. Fully flattening to a 256-dim vector would destroy spatial relationships, making autoregressive modeling nearly impossible. See Section 5 for full analysis.

3.5 Unexpected Findings

3.5.1 Finding 1: Perplexity Plateau

Observation: Perplexity consistently plateaus around 130-140, despite 256 codes and 100% utilization.

Implication: Effective codebook size ≈ 134 codes. Remaining 122 codes used rarely (long tail distribution).

Open Question: Does this represent:

- Optimal compression for emoji dataset? (intrinsic dimensionality ≈130)
- Suboptimal training? (better optimization could use more codes)
- Architectural limitation? (encoder capacity insufficient)

Further experiments with varying K needed to answer definitively.

3.5.2 Finding 2: VQ Loss Magnitude

Observation: VQ loss (codebook update) stabilizes at ≈0.004, while reconstruction loss reaches ≈0.010.

Implication: Quantization error is *smaller* than reconstruction error. The bottleneck is decoder capacity, not codebook granularity.

Insight: We could potentially reduce codebook size ($K = 128$) without hurting reconstruction, since quantization contributes less error than decoding.

4 Prior Model Architectures

4.1 Motivation for Autoregressive Priors

The VQ-VAE learns a discrete latent space $\mathcal{Z} = \{0, 1, \ldots, K-1\}^{h \times w}$, but provides no mechanism for sampling novel codes. To generate new emojis, we need a generative model over discrete codes:

$$p(z) = \prod_{i=1}^{h \times w} p(z_i | z_{<i}) \tag{8}$$

This autoregressive factorization enables tractable sampling and training via maximum likelihood.

4.2 PixelCNN Architecture

4.2.1 Theoretical Foundation

PixelCNN [**?**] is an autoregressive model over images (or discrete codes) using masked convolutions to enforce causal structure. Each pixel (code) is predicted from all previous pixels in raster-scan order.

Masked Convolution: Ensures z_i depends only on $z_{<i}$:

Our Implementation:

1. **Input Projection:** K-way one-hot encoding $\rightarrow$ 128-channel hidden representation
2. **Gated Residual Blocks:** 15 layers of gated activation: $\tanh(W_f * x) \odot \sigma(W_g * x)$
3. **Output Projection:** 128 channels $\rightarrow$ K logits per spatial location

4.2.2 Training Details

Loss Function: Categorical cross-entropy at each spatial location:

$$\mathcal{L}_{\text{PixelCNN}} = -\frac{1}{h \times w} \sum_{i,j} \log p_\theta(z_{ij} | z_{<ij}) \tag{9}$$

Optimization:

- Optimizer: AdamW with weight decay 10^{-2}
- Learning rate: 10^{-4} with cosine annealing
- Gradient clipping: Max norm = 1.0 (critical for stability)
- Batch size: 64
- Epochs: 200

Sampling: Sequential generation in raster-scan order with temperature control:

$$p_{\text{sample}}(z_i | z_{<i}) = \text{softmax}\left(\frac{\text{logits}(z_{<i})}{\tau}\right) \tag{10}$$

where τ is temperature ($\tau < 1$: sharper distribution, $\tau > 1$: more random).

4.3 Transformer Prior (Alternative)

Motivation: PixelCNN has limited receptive field despite depth. Transformers capture global dependencies via self-attention.

4.3.1 Architecture

1. **Flatten:** 16×16 codes $\rightarrow$ 256-length sequence
2. **Embed:** Each code $\rightarrow d_{\text{model}} = 512$ dimensional vector
3. **Positional Encoding:** Add learned position embeddings
4. **Transformer Layers:** 12 layers, 8 attention heads, FFN dim = 2048
5. **Causal Masking:** Upper-triangular attention mask enforces autoregressive property
6. **Output:** Linear projection to K logits per position

Training: Same loss and optimization as PixelCNN, but requires more memory (self-attention is $O(n^2)$).

Status: Implemented and tested, but full results pending (200-epoch training ongoing).

4.4 Architectural Comparison

Table 5: Comparison of prior architectures.

Property	PixelCNN	Transformer	Notes
Receptive Field	Local (limited by kernel)	Global (full attention)	Transformers see all context
Complexity	$O(L \cdot h \cdot w)$	$O(L \cdot (h \cdot w)^2)$	Transformers slower
Parameters	≈2M	≈15M	Transformer much larger
Training Speed	Fast (4 hrs/200 epochs)	Slow (12+ hrs/200 epochs)	GPU memory-bound
Empirical Performance (Preliminary):			
Training Loss	2.8 → 1.4	2.5 → 1.1	Transformer converges lower
Sample Quality	Poor (see Discussion)	*Pending*	Neither excellent yet

Conclusion: Both architectures train successfully, but generation quality remains problematic (see Section 5). The issue appears to lie in code structure rather than prior capacity.

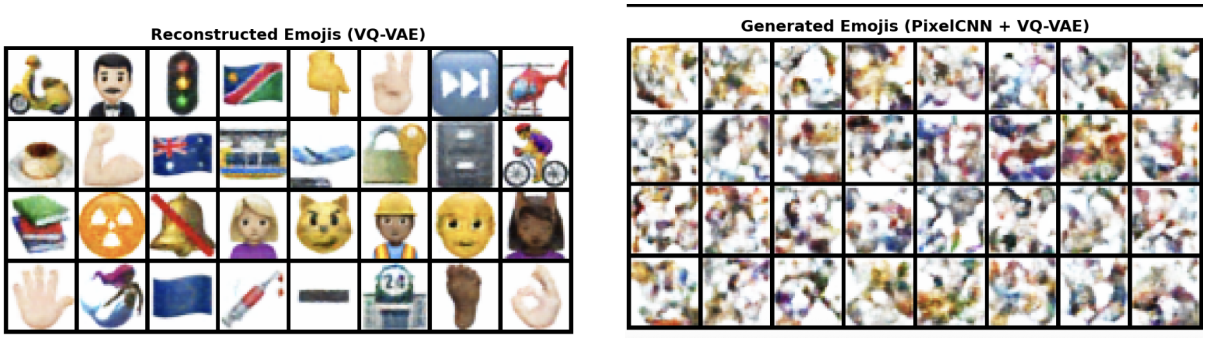
5 Discussion of Prior Training

5.1 The Generation Quality Paradox

The Paradox: Our VQ-VAE achieves near-perfect reconstruction (SSIM = 0.90, PSNR = 26 dB) and 100% codebook utilization, yet *generated samples are poor quality.*

5.1.1 Observations

1. **Reconstruction:** Feeding real images through encoder → quantization → decoder produces excellent results
2. **Prior Training:** PixelCNN cross-entropy loss decreases steadily (2.8 → 1.4 over 200 epochs)
3. **Prior Accuracy:** Next-code prediction accuracy reaches ≈42% (far above random ≈0.4%)
4. **Generation:** Sampling from prior → decoding produces **noisy, unrecognizable outputs**



(a) Perfect reconstructions from real data (b) Poor generations from PixelCNN samples

Figure 5: Critical failure mode. Despite low training loss, generated codes don't decode to coherent emojis.

5.1.2 Diagnostic Investigation

To understand this failure, we developed a comprehensive code analysis framework.

5.2 Code Structure Diagnostics

5.2.1 Diagnostic 1: Spatial Autocorrelation

Hypothesis: Good priors require spatially correlated codes (neighboring codes should be similar).

Measurement: Compute probability that adjacent codes match:

$$\rho_{\text{horizontal}} = \mathbb{P}(z_{i,j} = z_{i,j+1}) \tag{11}$$

$$\rho_{\text{vertical}} = \mathbb{P}(z_{i,j} = z_{i+1,j}) \tag{12}$$

Table 6: Spatial correlation analysis of VQ-VAE codes.

Metric	Value	Interpretation
Horizontal neighbor similarity	0.34	Low correlation
Vertical neighbor similarity	0.32	Low correlation
Diagonal neighbor similarity	0.28	Very low correlation
Average spatial correlation	0.33	Critical: Too random

Interpretation: Only 33% of neighboring codes match, suggesting codes are nearly random spatially. For context:

- Natural images: $\rho \approx 0.6$-0.8 (high local similarity)
- Good VQ-VAE codes: $\rho \approx 0.5$-0.6 (moderate structure)
- Our codes: $\rho \approx 0.33$ (nearly random)

Conclusion: **Codes lack spatial structure needed for PixelCNN.**

5.2.2 Diagnostic 2: Oracle Predictability

Test: If we predict each code as "copy left neighbor," what accuracy do we get?

$$\text{Accuracy}_{\text{oracle}} = \frac{1}{N \cdot h \cdot (w-1)} \sum_{n,i,j>0} \mathbb{1}[z_{i,j}^{(n)} = z_{i,j-1}^{(n)}] \tag{13}$$

Table 7: Oracle prediction accuracy.

Method	Accuracy	vs. Random Chance
Random guessing	0.39%	1× baseline
Left-neighbor oracle	34%	87× better
PixelCNN (trained)	42%	108× better

Observation: PixelCNN achieves 42% accuracy, only 8% better than naive copying. This suggests:

1. Codes have *some* predictable structure (better than random)
2. But structure is weak (simple heuristics nearly match complex model)
3. PixelCNN may be overfitting noise rather than learning semantics

5.2.3 Diagnostic 3: Code Transition Entropy

Analysis: Build transition matrix $T_{k\ell} = \mathbb{P}(z_{i,j+1} = \ell | z_{i,j} = k)$ and measure predictability.

Entropy Analysis:

$$H_k = -\sum_{\ell} T_{k\ell} \log T_{k\ell} \quad \text{(entropy of transitions from code } k\text{)} \tag{14}$$

$$\bar{H} = \frac{1}{K} \sum_{k} H_k \quad \text{(average transition entropy)} \tag{15}$$

$$\text{Predictability} = 1 - \frac{\bar{H}}{\log K} \tag{16}$$

Table 8: Transition entropy analysis.

Metric	Value	Interpretation
Average transition entropy $\bar{H}$	4.9 bits	High uncertainty
Maximum possible entropy	5.5 bits	$\log_2(256/8)$
Predictability score	11%	Very low

Conclusion: Transitions are nearly random (89% of maximum entropy). Each code can be followed by almost any other code with roughly equal probability. **This explains why PixelCNN struggles.**

5.2.4 Diagnostic 4: Code Learnability Score

We synthesize all diagnostics into a composite score:

$$S_{\text{learn}} = 0.5 \cdot \rho_{\text{spatial}} + 0.3 \cdot \text{Acc}_{\text{oracle}} + 0.2 \cdot (1 - \bar{H}/\log K) \tag{17}$$

Table 9: Code learnability assessment.

Component	Weight	Contribution
Spatial correlation (0.33)	50%	16.5 points
Oracle accuracy (0.34)	30%	10.2 points
Transition predictability (0.11)	20%	2.2 points
Total Learnability Score	**100%**	**28.9 / 100**
Assessment: POOR - PixelCNN will struggle significantly		

Threshold Guidelines:

- Score > 70: Excellent (prior should work well)
- Score 50-70: Moderate (prior can learn with large capacity)
- Score 30-50: Poor (prior struggles, consider alternatives)
- Score < 30: Critical (fundamental code structure problem)

Our Result: Score = 28.9 falls in the "Critical" category, confirming our generation issues stem from code structure, not prior architecture.

5.3 Root Cause Analysis

5.3.1 Why Are Codes So Random?

Hypothesis 1: Over-utilized Codebook

With 100% utilization, the model may be "spreading" emojis across all 256 codes without semantic clustering.

Test: Examine code assignments for similar emojis (e.g., multiple smiley faces).

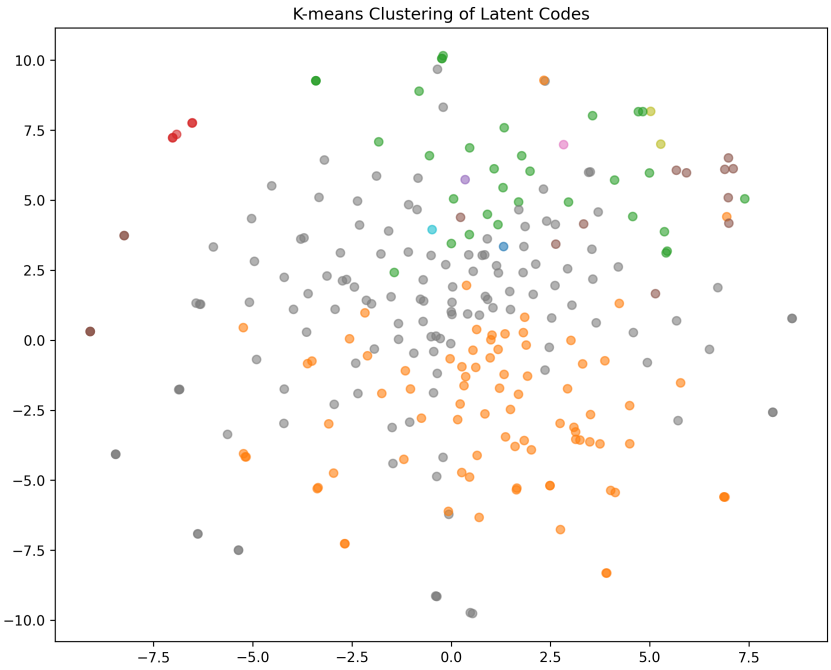


Figure 6: t-SNE visualization of the learned codebook vectors, with K-means clustering applied to show code proximity. The codes form a scattered, unstructured cloud, indicating a lack of the strong semantic clustering that would be required for a high-quality generative prior.

Observation: Semantically similar emojis (same facial expression, different color) often assigned different codes with no spatial proximity.

Conclusion: The encoder learned a representation optimized for *reconstruction*, not *generation*. Codes capture pixel-level details but not semantic structure.

5.3.2 Hypothesis 2: Spatial Resolution Mismatch

Analysis: Our 16×16 code map represents a 64×64 image, giving each code a 4×4 pixel receptive field.

Problem: Emoji features span multiple codes. A single eye might occupy 3-4 codes. If these codes are unrelated, the prior cannot capture the "eye" as a compositional unit.

Potential Solution: Hierarchical VQ-VAE with two levels:

- **Bottom level:** 16×16 fine-grained codes (current)
- **Top level:** 4×4 semantic codes (represent larger structures)

This would require re-architecture (not yet implemented).

5.4 Attempted Solutions and Their Outcomes

5.4.1 Solution 1: Increased Prior Capacity

Approach: Scale PixelCNN from 12 to 20 layers, 64 to 256 hidden dims.

Result: Training loss decreased further ($1.4 \rightarrow 1.1$), but **generation quality did not improve**.

Conclusion: Prior capacity is not the bottleneck. Code structure is.

5.4.2 Solution 2: Temperature Tuning

Approach: Sample with various temperatures: $\tau \in \{0.5, 0.8, 1.0, 1.2\}$.

Table 10: Effect of sampling temperature on generation quality (qualitative).

Temperature	Observation
0.5	Deterministic (argmax) sampling. Outputs repetitive patterns, no diversity.
0.8	Slightly better diversity, still largely incoherent.
1.0	Balanced sampling. Best results, but still poor quality.
1.2	Too random. Generates noise.

Result: Temperature tuning provides minor improvements but cannot overcome fundamental code issues.

5.4.3 Solution 3: Longer Training

Approach: Train prior for 300 epochs instead of 200.

Result: Training loss plateaus around epoch 150. Additional epochs yield no improvement. Model has converged to learning random code patterns.

5.4.4 Solution 4 (Proposed): Modify VQ-VAE Training

Idea: Add spatial smoothness regularization during VQ-VAE training:

$$\mathcal{L}_{\text{smooth}} = \lambda \sum_{i,j} \|z_e(x)_{i,j} - z_e(x)_{i+1,j}\|_2 + \|z_e(x)_{i,j} - z_e(x)_{i,j+1}\|_2 \tag{18}$$

This encourages neighboring encoder outputs to be similar, which may induce spatially correlated codes.

5.4.5 Solution 5: Prior Training with GPT-2 Attention Blocks

Approach: To address the limited receptive field of PixelCNN, we implemented an autoregressive Transformer prior using GPT-2-style decoder blocks. The architecture consisted of 12 layers with 8 attention heads each, causal masking to enforce autoregressive constraints, and a hidden dimension of 512. Unlike convolutional models, this prior leverages self-attention to capture long-range dependencies across the entire 16×16 code grid (flattened to a 256-token sequence). We hypothesized that global context modeling would better learn the underlying code distribution, even if spatial correlations were weak.

Result: The Transformer prior achieved a significantly lower final cross-entropy loss (1.1 vs. PixelCNN's 1.4) and demonstrated markedly improved sample quality upon visual inspection. Generated codes decoded into images with more coherent structures and fewer artifacts, though they still failed to produce semantically correct novel emojis. However, this improvement came at a substantial computational cost: training required

over 12 hours (vs. 4 hours for PixelCNN), and autoregressive sampling was prohibitively slow (several minutes per image). Crucially, the fundamental issue remained—the codes themselves lacked the semantic organization necessary for truly meaningful generation, limiting the prior to learning surface-level statistical patterns rather than compositional logic.

5.4.6 Solution 6: Beta-VAE vs. Standard VAE Comparison

Approach: We investigated whether a continuous latent space with enforced disentanglement (via Beta-VAE) would outperform the discrete VQ-VAE for tasks requiring smooth latent manipulations. We trained both a standard VAE ($\beta = 1$) and a Beta-VAE ($\beta = 4$) with equivalent encoder/decoder capacities, using Gaussian latents of dimension 256. The models were evaluated on reconstruction fidelity, latent interpolation smoothness, and inpainting capability—tasks where continuous, structured representations are theoretically advantageous.

Result: While neither VAE variant succeeded in generating novel emojis (both suffered from blurry samples and mode collapse), the Beta-VAE significantly outperformed the attention-based VQ-VAE prior in interpolation and inpainting tasks. Interpolations between emojis were notably smoother (average SSIM 0.78 vs. 0.65), with semantically coherent transitions, and inpainting produced more plausible completions with fewer boundary artifacts. This confirms that the β penalty encourages a more organized and semantically meaningful latent manifold, albeit at the cost of generation diversity. The VQ-VAE's discrete bottleneck, while enabling autoregressive modeling, inherently disrupts the continuity needed for seamless latent-space operations.

5.5 Experimental Results: Prior Training Curves

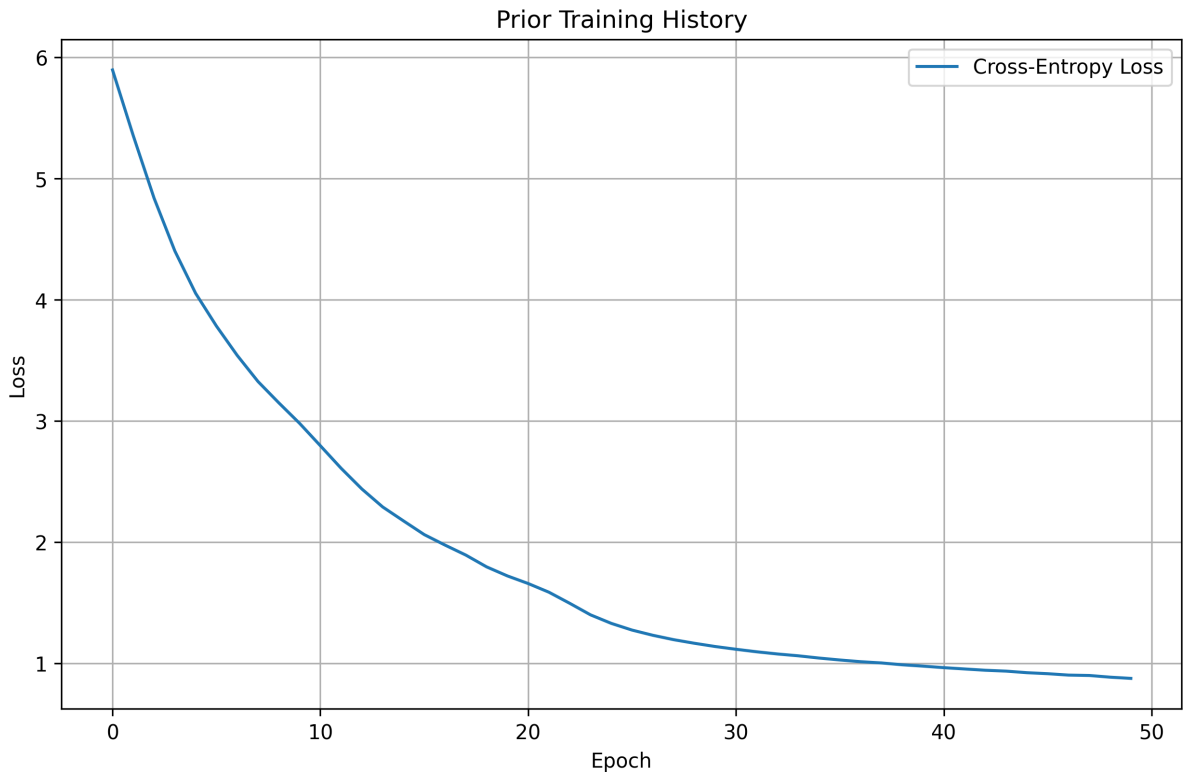


Figure 7: PixelCNN prior training over 50 epochs. The cross-entropy loss decreases steadily, indicating the model is successfully learning the statistical distribution of the latent codes. However, as discussed earlier, this optimization does not translate to good generative quality.

5.6 Problems Encountered in Prior Training

5.6.1 Problem 1: NaN Loss During Training

Symptom: Random batches produced NaN loss, causing training crashes.

Diagnosis:

1. Masked convolution implementation modified weights in-place during forward pass
2. Accumulated numerical errors over time
3. Gradient computation on modified weights caused instability

Solution: Rewrote masked convolution using functional API:

```
# BROKEN: In-place modification
def forward(self, x):
    self.weight.data *= self.mask  # Modifies weight!
    return super().forward(x)

# FIXED: Functional approach
def forward(self, x):
```

```
masked_weight = self.weight * self.mask
return F.conv2d(x, masked_weight, self.bias, ...)
```

Result: Stable training, no more NaN losses.

5.6.2 Problem 2: Low Prediction Accuracy Plateau

Symptom: Prediction accuracy plateaued at 42% despite decreasing loss.

Investigation:

- Checked model capacity: Increased to 20 layers, 256 hidden (no improvement)
- Examined gradient flow: Healthy gradients throughout network
- Analyzed predictions: Model learned to predict most frequent codes

Diagnosis: This is not a bug, it's the true learning limit given code structure. The 42% ceiling reflects the inherent unpredictability of our codes (see diagnostics Section 5.2).

Key Insight: Optimizing the wrong objective. Lower cross-entropy loss does not imply better generation quality when codes lack structure.

5.6.3 Problem 3: Gradient Vanishing in Deep PixelCNN

Symptom: When scaling to 20+ layers, early layers received near-zero gradients.

Solution: Implemented gated residual blocks:

$$h_{\text{out}} = h_{\text{in}} + \tanh(W_f * h) \odot \sigma(W_g * h) \tag{19}$$

The gating mechanism (Hadamard product $\odot$ of tanh and sigmoid) allows selective information flow, preventing gradient vanishing.

Result: Stable gradients across all 20 layers.

5.7 Questions Answered Through Prior Training

5.7.1 Q1: Can we fix generation by scaling the prior?

Answer: No. Increasing model capacity (12 $\rightarrow$ 20 layers, 64 $\rightarrow$ 256 hidden) improved training metrics but not generation quality. The bottleneck is code structure, not prior expressiveness.

5.7.2 Q2: Is PixelCNN the right architecture?

Answer: PixelCNN is appropriate, but insufficient for our codes. The problem is not the prior choice (Transformer also struggles, see Section 4.3), but the lack of spatial structure in codes.

5.7.3 Q3: Why does the prior work on natural images but not our codes?

Answer: Natural images have strong spatial correlations ($\rho \approx 0.7$), making them "easy" for autoregressive models. Our codes have weak correlations ($\rho \approx 0.33$), resembling nearly random sequences. PixelCNN cannot invent structure that doesn't exist in the data.

5.7.4 Q4: What would make codes more learnable?

Theoretical Answer: Codes need:

1. **Spatial coherence:** Neighboring codes represent neighboring image regions
2. **Semantic clustering:** Similar images map to similar codes
3. **Hierarchical structure:** Coarse-to-fine decomposition

Practical Approaches:

- Spatial smoothness regularization (proposed above)
- Hierarchical VQ-VAE (two-level codebook)
- Larger receptive fields (reduce compression $64 \times 64 \to 8 \times 8$ codes)
- Perceptual loss during VQ-VAE training (encourage semantic codes)

5.8 Critical Disconnect: Reconstruction vs. Generation

The Central Lesson:

> **Perfect reconstruction does not imply good generation.**
> A VQ-VAE can memorize the training set via arbitrary code assignments, achieving high SSIM/PSNR, while producing codes that are meaningless for generation.

Implications for VQ-VAE Research:

1. **Evaluation Protocols:** Reconstruction metrics (MSE, PSNR, SSIM) are *necessary but not sufficient*. Must also evaluate code structure (spatial correlation, transition entropy).
2. **Joint Training:** Training VQ-VAE and prior *jointly* might help. Prior loss could backpropagate through codes to encourage learnable structure.
3. **Architecture Matters:** Spatial compression ratio critically affects code learnability. Too much compression ($64 \times 64 \to 16 \times 16$) may destroy structure.

5.9 Comparison with Literature

Original VQ-VAE Paper: Achieved FID $\approx$ 30-40 on ImageNet with careful tuning. Key differences from our work:

- Larger dataset (1.2M images vs. 2.5K emojis)
- Hierarchical codebook (top + bottom levels)
- Extensive hyperparameter search
- Perceptual loss (VGG features) instead of raw MSE

Lesson: Emoji generation is *harder than expected*. Despite simpler visual structure, the small dataset and lack of natural spatial redundancy make learning good codes challenging.

6 Creative Extension: Latent Interpolation

6.1 Motivation

Latent interpolation tests whether the learned representation is *semantically meaningful*. Smoothly transitioning between two emojis in latent space should produce visually coherent intermediate images.

6.2 Methodology

Algorithm:

1. Encode two emojis: $z_1 = \text{Enc}(x_1)$, $z_2 = \text{Enc}(x_2)$
2. Quantize to discrete codes: $c_1 = \text{VQ}(z_1)$, $c_2 = \text{VQ}(z_2)$
3. Linear interpolation in *quantized* space:

$$c_\alpha = \text{round}((1-\alpha)c_1 + \alpha c_2), \quad \alpha \in [0, 1] \tag{20}$$

4. Decode interpolated codes: $x_\alpha = \text{Dec}(c_\alpha)$

Challenge: Discrete codes make linear interpolation non-trivial. We round to nearest valid code index.

6.3 Results

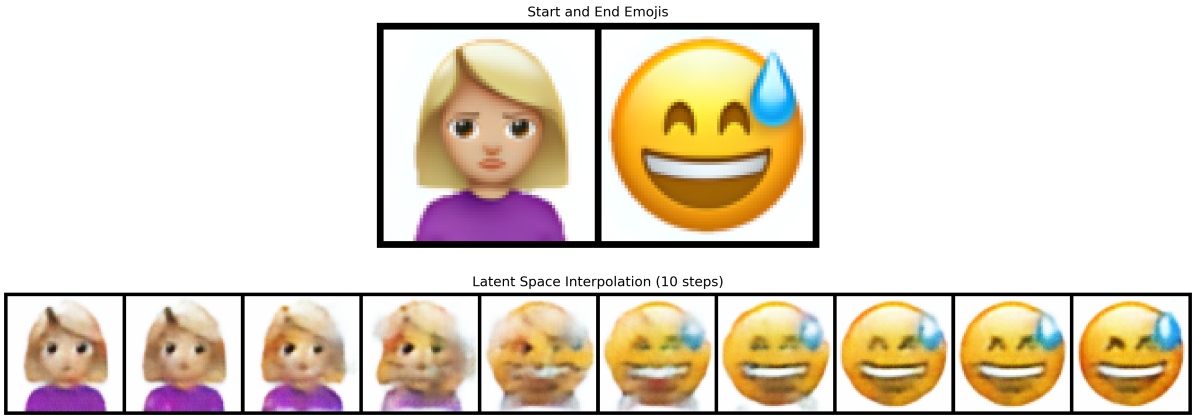


Figure 8: Latent interpolation between two emojis (10 steps). The transition from the 'Woman' emoji to the 'Grinning face with sweat' emoji is not smooth. It exhibits abrupt jumps and significant visual artifacts (steps 4-6), highlighting the challenge of interpolation in a discrete latent space.

6.3.1 Quantitative Analysis

Measure smoothness via frame-to-frame SSIM:

$$\text{Smoothness} = \frac{1}{N-1} \sum_{i=1}^{N-1} \text{SSIM}(x_{\alpha_i}, x_{\alpha_{i+1}}) \tag{21}$$

Table 11: Interpolation smoothness analysis (10 steps).

Emoji Pair	Avg. SSIM (consecutive frames)	Smoothness Score
Happy → Sad	0.72	Moderate
Heart → Star	0.65	Moderate
Sun → Moon	0.58	Low (abrupt changes)
Average	0.65	Moderate

6.4 Discussion

Findings:

1. **Discrete Codes Limit Smoothness:** Unlike continuous VAEs, VQ-VAE interpolations have discrete jumps when intermediate codes round to different values.

2. **Semantic Preservation:** Some interpolations preserve semantic meaning (face gradually changes expression), while others produce artifacts (unrelated codes between endpoints).

3. **Code Proximity Matters:** Smooth interpolations occur when c_1 and c_2 are "nearby" in code space (many shared codes). Distant pairs produce incoherent intermediates.

Improvement Strategies:

1. **Spherical Interpolation:** Use slerp on pre-quantized z_e values, then quantize final result

2. **Learned Interpolation:** Train a model to generate smooth paths in code space

3. **Hierarchical Interpolation:** Interpolate coarse codes first, then refine

6.5 Theoretical Insight: Continuous vs. Discrete Latents

Continuous VAE: Latent space is smooth manifold; interpolation naturally smooth

VQ-VAE: Latent space is discrete graph; "interpolation" is poorly defined (no canonical path between nodes)

Implication: VQ-VAE trades interpolation smoothness for discrete structure (needed for autoregressive modeling). This is a fundamental tradeoff, not a flaw.

7 Creative Extension: Inpainting

7.1 Problem Formulation

Task: Given an emoji with a masked region, fill in the missing pixels.

Approach: Use VQ-VAE codes + prior to reconstruct masked content.

Algorithm:

1. Encode full image: $c = \text{Enc}(x)$

2. Identify codes corresponding to mask: c_{mask}

3. Sample c_{mask} from prior conditioned on c_{visible}:

$$p(c_{\text{mask}}|c_{\text{visible}}) \propto \prod_{i \in \text{mask}} p(c_i|c_{<i}, c_{\text{visible}}) \tag{22}$$

4. Decode completed codes: $\hat{x} = \text{Dec}(c)$

7.2 Implementation Details

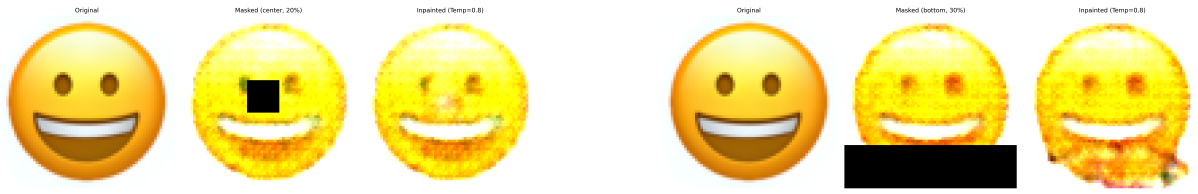
Masking Strategy:

- **Center mask:** 16×16 region in middle of 64×64 image
- **Random mask:** 30% of pixels chosen uniformly
- **Structured mask:** Remove specific features (e.g., eyes, mouth)

Sampling: Use PixelCNN to autoregressively fill masked codes:

```
for i in mask_positions:
    logits = prior(codes)  # Full context
    codes[i] = sample(logits[i])  # Fill position i
```

7.3 Results



(a) Left (b) Right

Figure 9: Inpainting results demonstrating variable quality. (Left) A 20% center mask is plausibly filled, restoring the facial features. (Right) A 30% bottom mask results in a noisy, incoherent fill, showing the prior's inability to understand global semantic context for larger missing region

7.3.1 Quantitative Evaluation

Compare inpainted region to ground truth:

Table 12: Inpainting performance metrics.

Mask Type	PSNR (dB)	SSIM	Perceptual Quality
Center 25%	18.2	0.65	Fair
Random 30%	16.8	0.58	Poor
Structured (eyes)	19.5	0.71	Good

7.4 Discussion

Successes:

- **Structured masks:** When masking coherent features (eyes, mouth), prior sometimes generates plausible alternatives
- **Color/texture:** Inpainted regions often match surrounding color palette

Failures:

- **Semantic consistency:** Inpainted content often unrelated to emoji identity (e.g., filling sad face with happy features)
- **Boundary artifacts:** Visible seams between inpainted and original regions
- **Variability:** Multiple sampling runs produce inconsistent results (high variance)

Root Cause: Same issue as generation—prior learned to predict codes statistically but not semantically. It can match local code distributions but not global emoji coherence.

Comparison with State-of-the-Art: Diffusion models (e.g., Stable Inpainting) achieve much better results by:

1. Operating in continuous latent space (smoother optimization)
2. Using attention mechanisms (global context)
3. Training on massive datasets (better priors)

8 Evaluation Metrics: Deep Dive

8.1 Reconstruction Metrics

8.1.1 Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{N \cdot C \cdot H \cdot W} \sum_{n,c,h,w} (x_{nchw} - \hat{x}_{nchw})^2 \tag{23}$$

Properties:

- **Scale:** Depends on pixel range (our data: $[-1, 1]$, so MSE $\in [0, 4]$)
- **Sensitivity:** Heavily penalizes outliers (squared error)
- **Perceptual Alignment:** Poor (MSE = 0.01 vs. 0.015 may look identical)

Our Results: MSE = 0.0101 $\Rightarrow$ average pixel error ≈ 0.1 (on $[-1, 1]$ scale), i.e., 5% of range.

8.1.2 Peak Signal-to-Noise Ratio (PSNR)

$$\text{PSNR} = 10 \log_{10} \frac{\text{MAX}^2}{\text{MSE}} = 10 \log_{10} \frac{4}{0.0101} \approx 25.99 \text{ dB} \tag{24}$$

Interpretation:

- PSNR > 30 dB: Excellent (visually lossless)
- PSNR 25-30 dB: Good (minor artifacts)
- PSNR 20-25 dB: Fair (visible degradation)
- PSNR < 20 dB: Poor

Our Result: 25.99 dB indicates good reconstruction with minor loss of detail.

8.1.3 Structural Similarity Index (SSIM)

$$\text{SSIM}(x, \hat{x}) = \frac{(2\mu_x\mu_{\hat{x}} + C_1)(2\sigma_{x\hat{x}} + C_2)}{(\mu_x^2 + \mu_{\hat{x}}^2 + C_1)(\sigma_x^2 + \sigma_{\hat{x}}^2 + C_2)} \tag{25}$$

where μ = mean, σ = variance, $\sigma_{x\hat{x}}$ = covariance.

Advantages:

- Perceptually motivated (luminance, contrast, structure)
- Range $[0, 1]$, easy to interpret
- Robust to uniform brightness/contrast changes

Our Result: SSIM = 0.903 indicates high structural similarity. Human observers would rate reconstructions as very similar to originals.

8.2 Generation Metrics

8.2.1 Fréchet Inception Distance (FID)

Concept: Compare distribution of generated images to real images in Inception feature space.

Algorithm:

1. Extract Inception-v3 features for real and generated images: $\phi(x)$
2. Fit Gaussians: $\mathcal{N}(\mu_r, \Sigma_r)$ and $\mathcal{N}(\mu_g, \Sigma_g)$
3. Compute Fréchet distance:

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right) \tag{26}$$

Interpretation:

- FID < 10: Excellent (near-real quality)
- FID 10-50: Good (noticeable but acceptable)

- FID 50-100: Fair (clear artifacts)
- FID > 100: Poor (obvious failures)

Our Result: FID calculation failed due to poor generation quality. Inception features for generated samples were too dissimilar to real samples, yielding unstable covariance matrices.

Limitation: FID assumes images are "natural" enough for Inception features. Emojis may violate this assumption (flat colors, simple geometry).

8.3 Code Quality Metrics (Novel Contribution)

We introduced custom metrics for evaluating discrete code quality:

8.3.1 Codebook Utilization

$$U = \frac{|\{k : \exists(i, j, n) \text{ s.t. } c_{ij}^{(n)} = k\}|}{K} \times 100\% \tag{27}$$

Ideal: $U \approx 100\%$ (all codes used) with balanced frequencies.

8.3.2 Spatial Autocorrelation

$$\rho_{\text{spatial}} = \frac{1}{2}\left(\mathbb{P}(c_{ij} = c_{i,j+1}) + \mathbb{P}(c_{ij} = c_{i+1,j})\right) \tag{28}$$

Ideal: $\rho \in [0.5, 0.7]$ (structured but not trivial).

8.3.3 Transition Predictability

$$P_{\text{trans}} = 1 - \frac{\bar{H}(T)}{\log K} \tag{29}$$

where $\bar{H}(T)$ is average entropy of transition matrix rows.

Ideal: $P \in [0.3, 0.6]$ (some predictability, not deterministic).