Unsupervised Learning Algorithm Notes

Core Requirements Analysis

Before diving into algorithms, remember what your project needs:

1. **Clustering** to find natural groups in data (for visual differentiation)
2. **Latent space representation** for smooth traversal/generation
3. **Unsupervised** (no labels)
4. **Interpretable** (so you can map clusters to visual features)

---

1. K-Means (Mentioned in Spec)

How It Works:

Partitions data into K clusters where each point belongs to the cluster with the nearest mean (centroid).

```
# Pseudocode intuition
1. Randomly place K centroids
2. Assign each point to nearest centroid
3. Move centroids to mean of their assigned points
4. Repeat 2-3 until convergence
```

Pros:

- **Simple & Fast:** Easy to implement, scales well to large datasets
- **Deterministic Results:** With same initialization, gives same clusters
- **Interpretable:** Each cluster = group around a center; easy to explain
- **Works Well with Elbow Method:** Perfect for autonomous `k` determination as required
- **Low Memory:** Only stores centroids, not pairwise distances

Cons:

- **Assumes Spherical Clusters:** Struggles with non-convex shapes (like concentric circles)
- **Requires Specifying K:** Though elbow method helps
- **Sensitive to Outliers:** Centroids get pulled by extreme points

- **Poor with Varying Densities:** Assumes clusters of similar size/density
- **Initialization Sensitivity:** K-means++ helps but doesn't eliminate

Best For Project When:

- Latent space (from VAE/PCA) has roughly spherical clusters
- Speed for real-time adaptation
- Clear, interpretable cluster boundaries

---

2. Self-Organizing Maps (SOMs) (Mentioned in Spec)

How It Works:

A neural network that creates a low-dimensional (usually 2D) discretized representation of input space while preserving topological properties.

```
# Pseudocode intuition
1. Initialize a 2D grid of neurons with random weights
2. For each data point:
   a. Find the "winning" neuron (closest weight vector)
   b. Update winning neuron AND its neighbors to be more like input
   c. Decrease neighbor radius and learning rate over time
3. Result: A 2D map where similar data points activate nearby neurons
```

Pros:

- **Natural 2D Output:** Perfect for visualization - the SOM grid IS your visualization canvas
- **Preserves Topology:** Similar inputs map to nearby neurons
- **Great for Exploration:** You can "walk" the grid and see smooth transitions
- **Handles Non-linearities:** Can capture complex manifolds
- **Intuitive Interpretation:** Each grid cell represents a "prototype" of data

Cons:

- **Computationally Heavy:** Slower than K-Means for large datasets
- **Many Hyperparameters:** Grid size, topology (hex vs rectangular), learning rates, neighborhood functions
- **Less Standardized for Autonomous K:** Determining optimal grid size is less straightforward than elbow method
- **Fixed Grid Structure:** Artificially imposes grid topology on data

Best For Project When:

- Want the clustering to directly drive a 2D visualization grid
- Topological preservation is crucial (similar data = nearby in visualization)
- Willing to trade some speed for richer structure

---

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

My first recommended addition

How It Works:

Finds core samples of high density and expands clusters from them.

```
# Pseudocode intuition
1. For each point:
   a. Count neighbors within radius ε
   b. If ≥ minPts neighbors → mark as "core point"
2. Connect core points that are within ε of each other
3. All connected core points + their neighbors form a cluster
4. Points not in any cluster = noise
```

Pros:

- **No Need for K:** Automatically finds number of clusters (satisfies "autonomous clustering"!)
- **Handles Arbitrary Shapes:** Can find clusters of any shape
- **Robust to Outliers:** Explicitly labels noise points
- **Handles Varying Densities:** With parameter tuning
- **Works with Distance Metric of Choice:** Can use cosine, Euclidean, etc.

Cons:

- **Difficulty with Varying Densities:** If clusters have different densities, hard to choose single ε
- **Border Points Ambiguity:** Points on cluster edges might be assigned arbitrarily
- **Parameter Sensitivity:** ε and minPts choice is critical
- **Not Deterministic:** Border points can flip between clusters
- **Scales $O(n^2)$ with naive implementation** (though optimized versions exist)

Best For Project When:

- Expecting clusters of irregular shapes in your latent space
- Outliers (noise) should be visually distinct
- Can't assume similar cluster densities
- Data has clear density variations

---

4. Gaussian Mixture Models (GMM) / Expectation-Maximization

My second recommended addition

How It Works:

Assumes data comes from a mixture of several Gaussian distributions with unknown parameters.

```
# Pseudocode intuition (EM algorithm):
1. Initialize K Gaussian distributions randomly
2. E-step: For each point, calculate probability it came from each Gaussi
an
3. M-step: Update Gaussian parameters (mean, covariance, weight) based on
these probabilities
4. Repeat 2-3 until convergence
```

Pros:

- **Soft Clustering:** Points have probabilities of belonging to each cluster (rich for visualization!)
- **Handles Overlapping Clusters:** Natural for ambiguous data points
- **Flexible Cluster Shapes:** Covariance matrix captures elliptical shapes (not just spherical)
- **Probabilistic Foundation:** Can use Bayesian Information Criterion (BIC) for autonomous K selection
- **Generative Model:** Can sample new points from learned distributions (perfect for generative art!)

Cons:

- **Assumes Gaussian Distribution:** May fail for non-Gaussian clusters
- **Slower than K-Means:** EM algorithm has more computation
- **Sensitive to Initialization:** Can get stuck in local optima
- **Singularities Possible:** If a cluster has only one point, covariance becomes singular

Best For Project When:

- Want smooth, probabilistic transitions between clusters (great for latent traversal!)

- Overlap between clusters is expected
- Want to generate new "synthetic" data points from clusters
- Elliptical (not just circular) clusters are expected

Comparative Analysis Table

Algorithm	Autonomous K?	Cluster Shapes	Speed	Output for Visualization	Best For Generative Art Because...
K-Means	With elbow method	Spherical only	⚡⚡⚡ Fast	Hard cluster assignments	Simple mapping: cluster ID → visual style
SOMs	Grid size fixed	Preserves topology	⚡⚡ Medium	2D activation grid	Direct spatial mapping to visual canvas
DBSCAN	✅ Automatic	Arbitrary shapes	⚡⚡ Medium	Core/border/noise labels	Can show outliers as "special" visual elements
GMM	With BIC/AIC	Elliptical	⚡⚡ Medium	Soft probabilities	Smooth transitions; can sample from distributions

---

Primary Choice: Gaussian Mixture Models (GMM) paired with VAE

Why GMM:

1. **Natural for Latent Traversal:** The soft assignments mean as you move through latent space, you get smooth blends of cluster memberships → smooth visual transitions
2. **Generative Capability:** You can sample new points from learned Gaussians to create "synthetic but plausible" data for visualization
3. **Rich Visual Mapping:** Instead of just "cluster 1 = blue, cluster 2 = red", you can map:
    - `probability(cluster A)` → opacity of visual element A
    - `probability(cluster B)` → opacity of visual element B
    - Creates beautiful overlays and blends

4. **Autonomous K with BIC:** Bayesian Information Criterion gives principled way to choose number of components
5. **Fits VAE Philosophy:** Both are probabilistic models that assume latent space has Gaussian structure

Secondary Choice: DBSCAN

If data exploration shows clusters have weird shapes and densities vary a lot.

Implementation Strategy:

```
# Suggested pipeline for your project
1. VAE → creates smooth, Gaussian-ish latent space
2. GMM on latent space → soft clusters with probabilities
3. For visualization:
   - Position in latent space → x,y coordinates on screen
   - GMM probabilities → blend of visual styles/colors
   - Cluster means → "attractor points" for visual motifs
```

What About SOMs?

SOMs are tempting because they give you a 2D grid naturally. However:

- They're more complex to implement correctly
- The grid structure is rigid
- Less standard methods for autonomous sizing

Consider this hybrid approach: Use VAE for dimensionality reduction, then use **GMM for clustering**, but visualize on a **hexagonal grid inspired by SOMs** for aesthetic appeal.