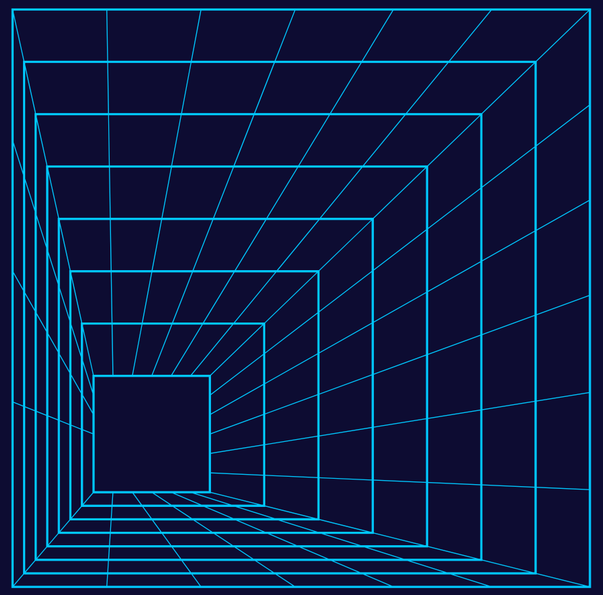


4th Week
The Bridge

Manifold distances
01/26 - 02/02

Josefa Fernández

ToDo Meeting

- ✓ Save Data in Google Drive
- ✓ FRGC, filter identities >=100 imgs
- ✓ Manifold distances
  - calculate the distances between manifolds
  - search from math papers to integrate in CS
- Distribution between manifold
  - filter number of img K, whose intrinsic dimension it is K-1
  - a single value for intrincic dimension
  - between same intrinsic dimension, compare angle the distances
    - orthogonal?
    - distribution angles close to 1 and if they are really small WHY?
    - angle between subpaces
  - Dimensionality estimation
    - Search for a well regarded dimensionality estimation - better if it is for face recognition
    - Compare to this found method to the PCA method

ToDo Meeting

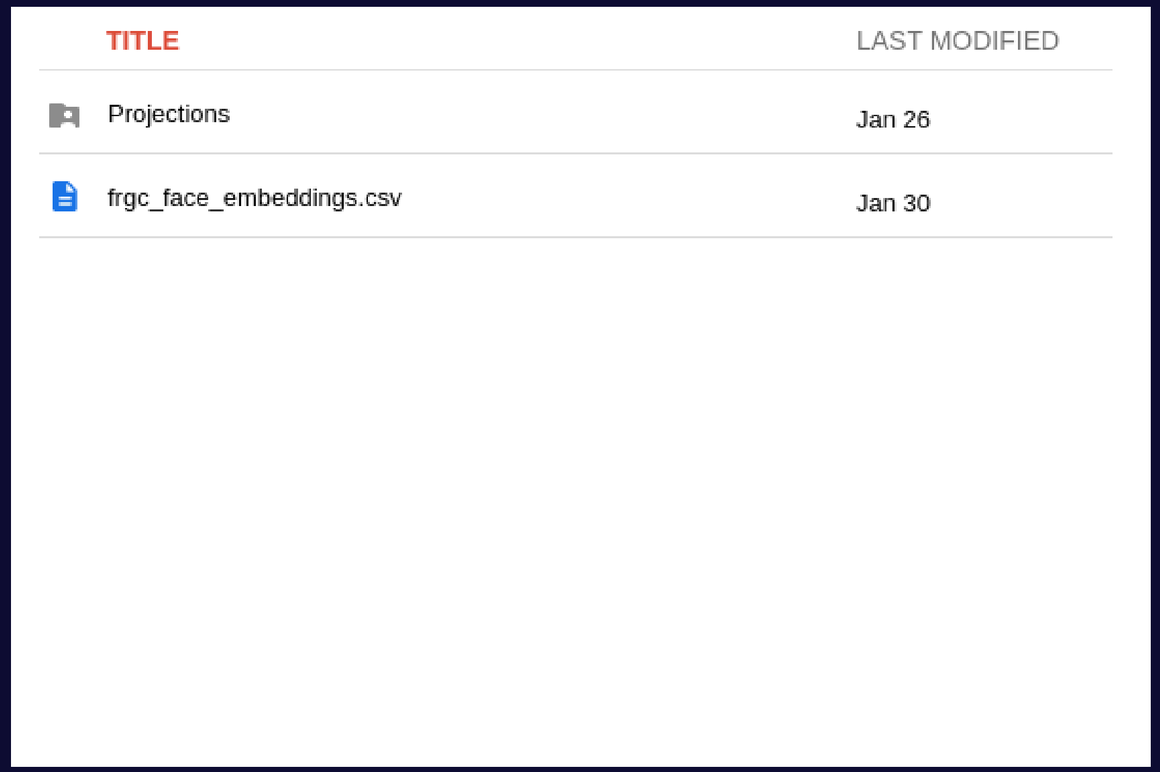
- Distribution of embeddings
  - Geometry structure
- Representation proximity
- Compare to face networks
  - ArcFace
  - MagFace
  - AdaFace
  - Facenet
  - Dlib

Main Info

- **ENTER TERMINAL:** ssh jferna27@cvrl-flynn-ws1.cse.nd.edu
- **GITHUB:** https://github.com/J0SEF4/The-Bridge/tree/main

Save Data in Google Drive

- Save data in Google Drive
  - fgrc embeddings data
  - projections data
  - send it to Prof. Flynn
  - write it on ReadME



https://drive.google.com/drive/folders/1FflEJ7RteaaUuj1Mp5bczcTsXdSb86dy?usp=sharing

Compare FRGC

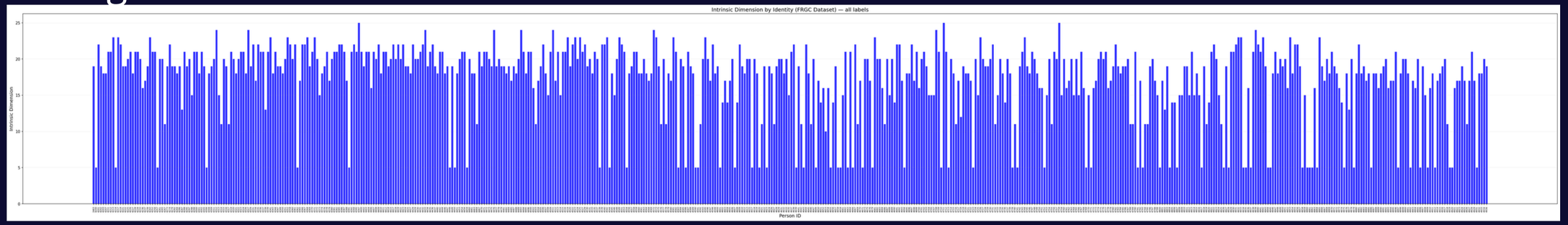
all imgs **VS** filter>= 100 imgs

```
Found frgc_face_embeddings.csv
Embeddings: (39327, 512)
Valid images: 39327
Unique person_ids: 568
```

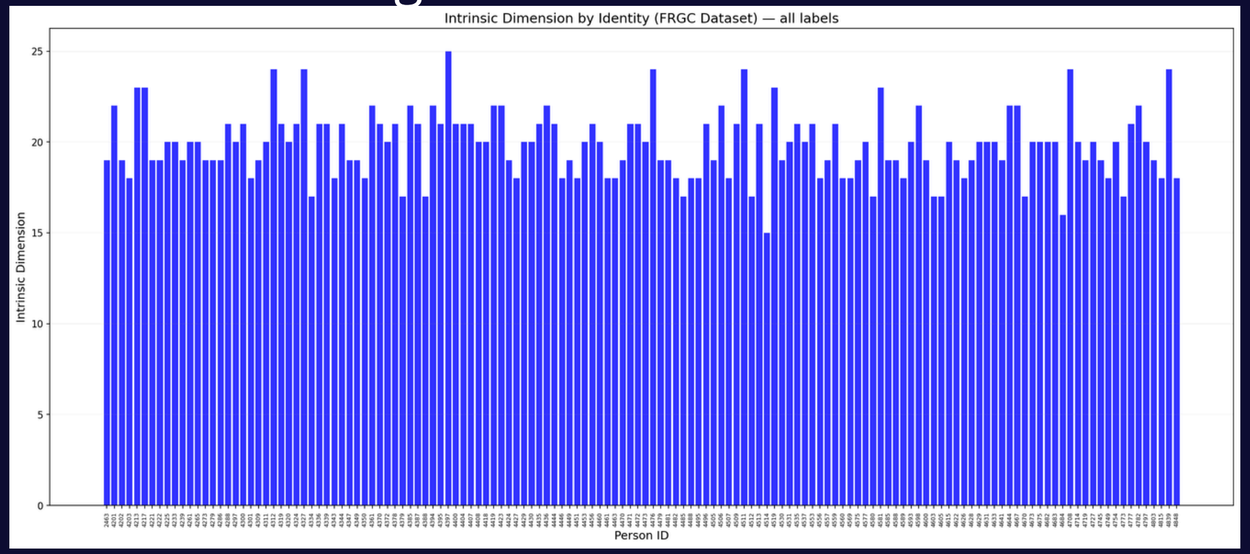
```
Identities with >= 100 images: 142/568
```

FRGC Filter identities >= 100 imgs

All imgs

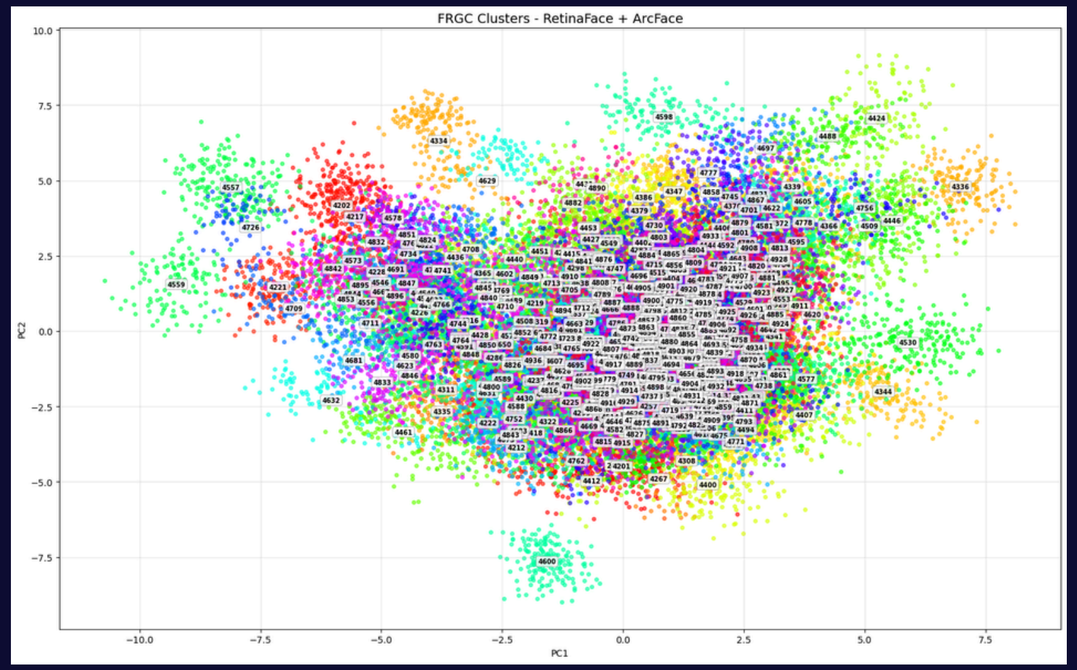


Filter >= 100 imgs

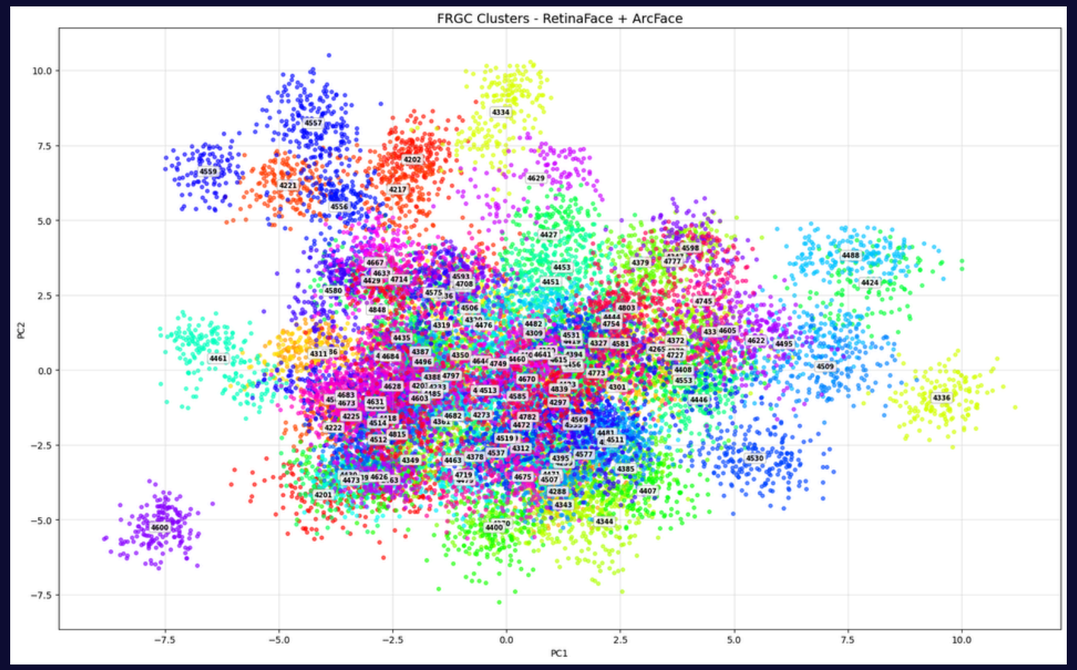


FRGC Filter identities >= 100 imgs

All imgs

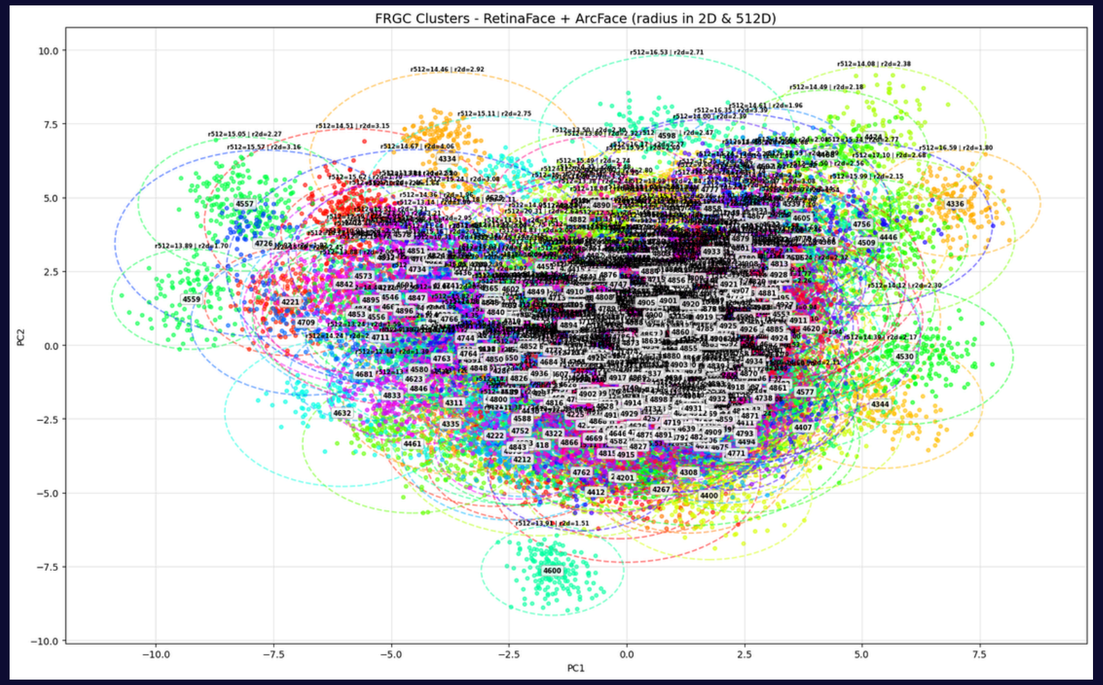


Filter >= 100 imgs

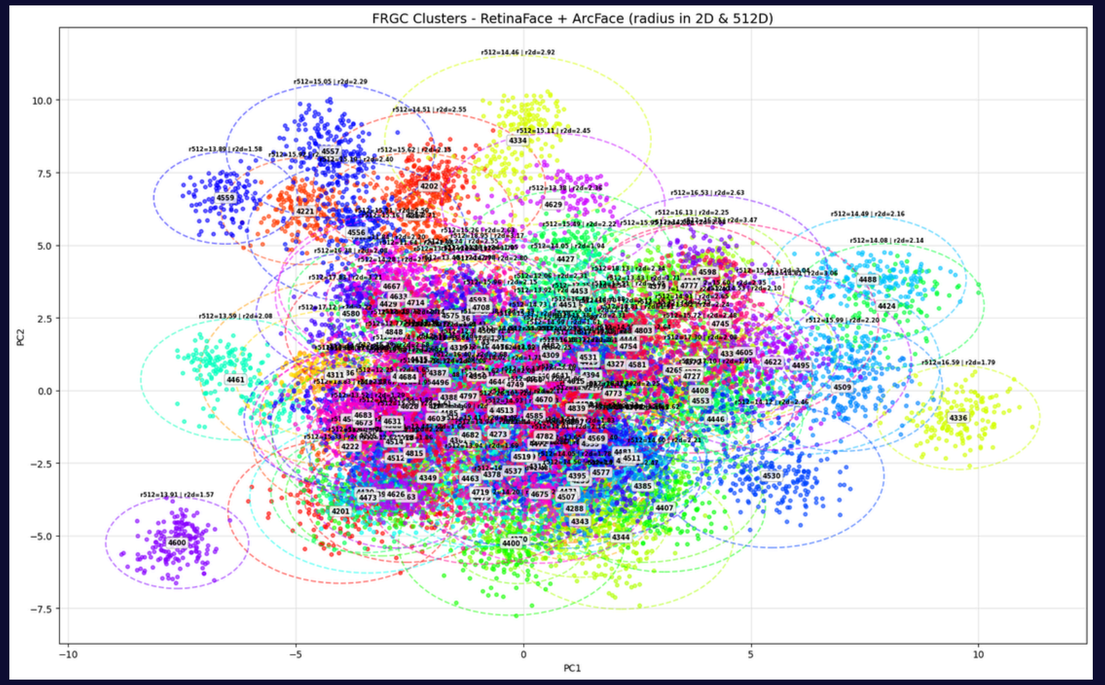


FRGC Filter identities >= 100 imgs

All imgs



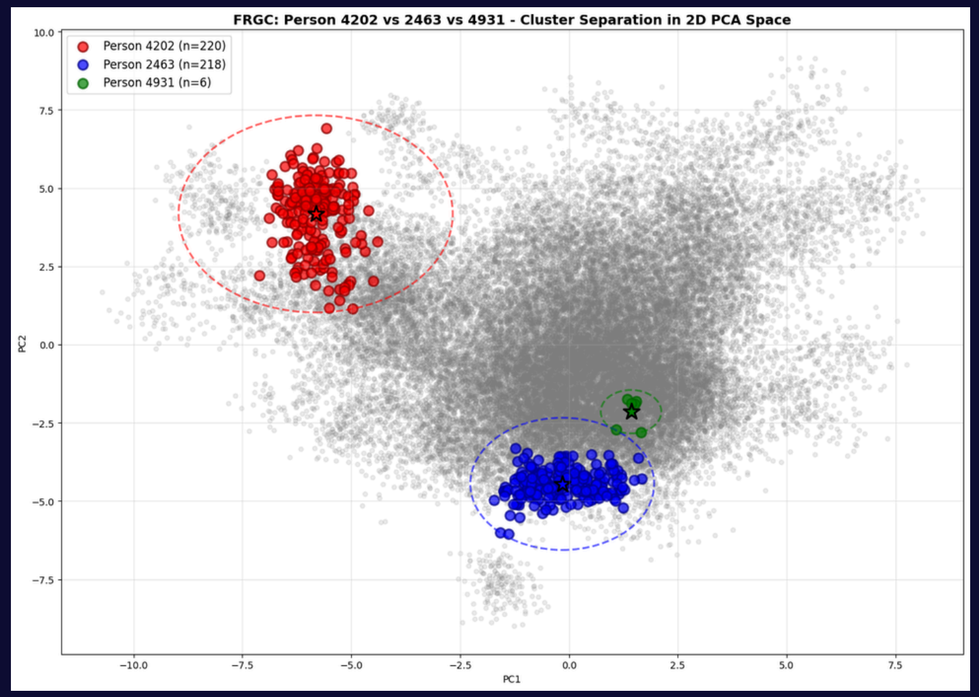
Filter >= 100 imgs



FRGC Filter identities >= 100 imgs

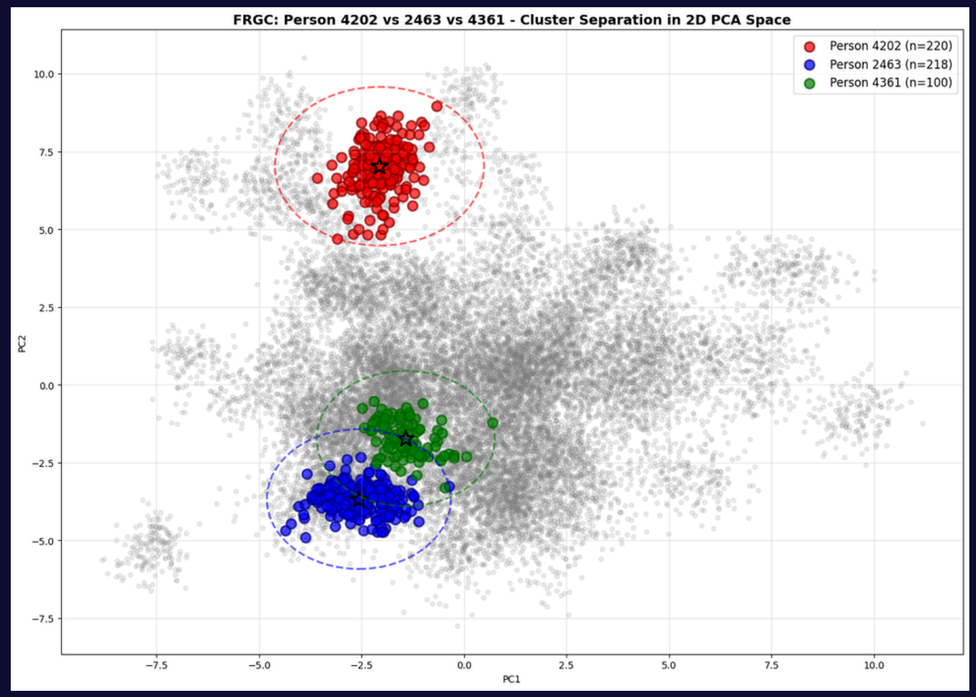
All imgs

```
Person 4202: 220 images in dataset
Person 2463: 218 images in dataset
Person 4931: 6 images in dataset
```



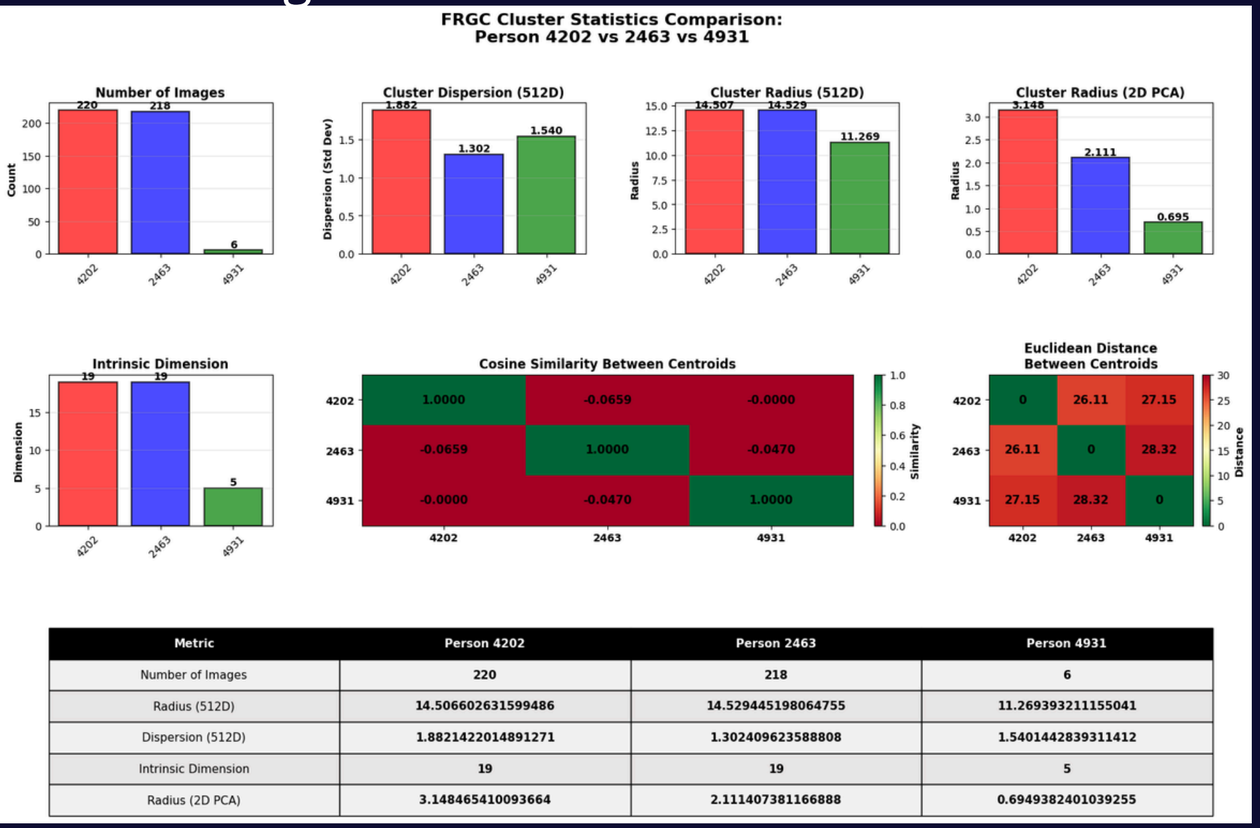
Filter >= 100 imgs

```
Person 4202: 220 images in dataset
Person 2463: 218 images in dataset
Person 4361: 100 images in dataset
```



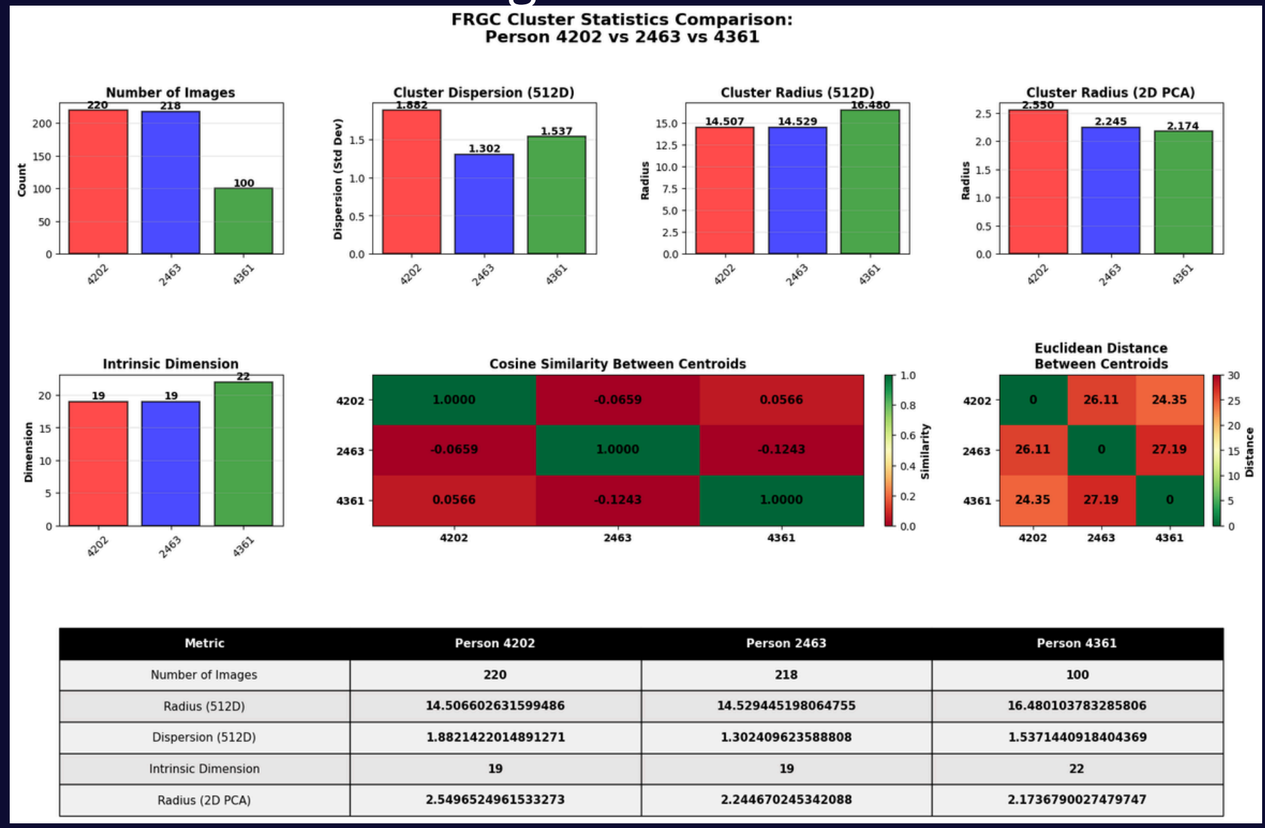
FRGC Filter identities >= 100 imgs

All imgs



Metric	Person 4202	Person 2463	Person 4931
Number of Images	220	218	6
Radius (512D)	14.506602631599486	14.529445198064755	11.269393211155041
Dispersion (512D)	1.8821422014891271	1.302409623588808	1.5401442839311412
Intrinsic Dimension	19	19	5
Radius (2D PCA)	3.148465410093664	2.111407381166888	0.6949382401039255

Filter >= 100 imgs



Metric	Person 4202	Person 2463	Person 4361
Number of Images	220	218	100
Radius (512D)	14.506602631599486	14.529445198064755	16.480103783285806
Dispersion (512D)	1.8821422014891271	1.302409623588808	1.5371440918404369
Intrinsic Dimension	19	19	22
Radius (2D PCA)	2.5496524961533273	2.244670245342088	2.1736790027479747

FRGC Filter identities >= 100 imgs

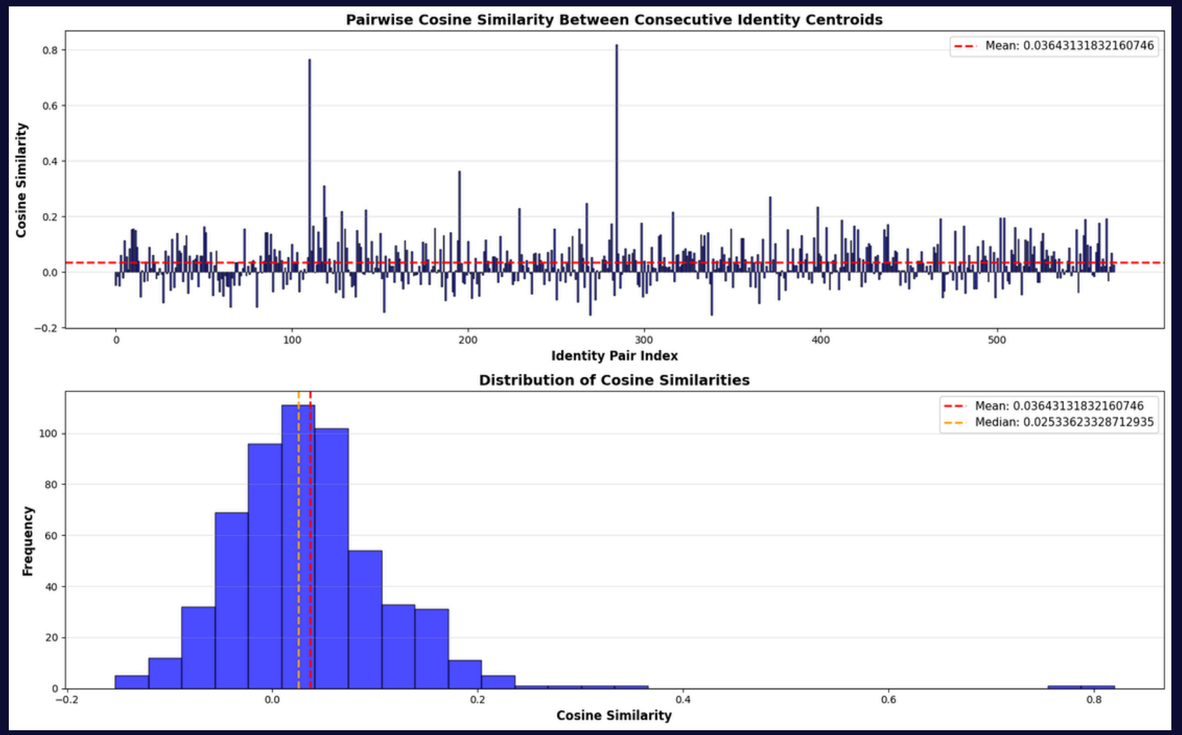
7. Cosine similarity between clusters

```
labels_arr = np.array(valid_person_ids)
unique_ids = np.unique(labels_arr)
similarities = []
identity_pairs = []

for i in range(len(unique_ids) - 1):
    lbl1 = unique_ids[i]
    lbl2 = unique_ids[i+1]
    c1 = cluster_stats[lbl1]['centroid']
    c2 = cluster_stats[lbl2]['centroid']
    sim = cosine_similarity(c1.reshape(1,-1), c2.reshape(1,-1))[0,0]
    similarities.append(sim)
    identity_pairs.append(f"{str(lbl1)[:3]} vs {str(lbl2)[:3]}")
    print(f"Cosine similarity {lbl1} vs {lbl2}: {sim}")
```

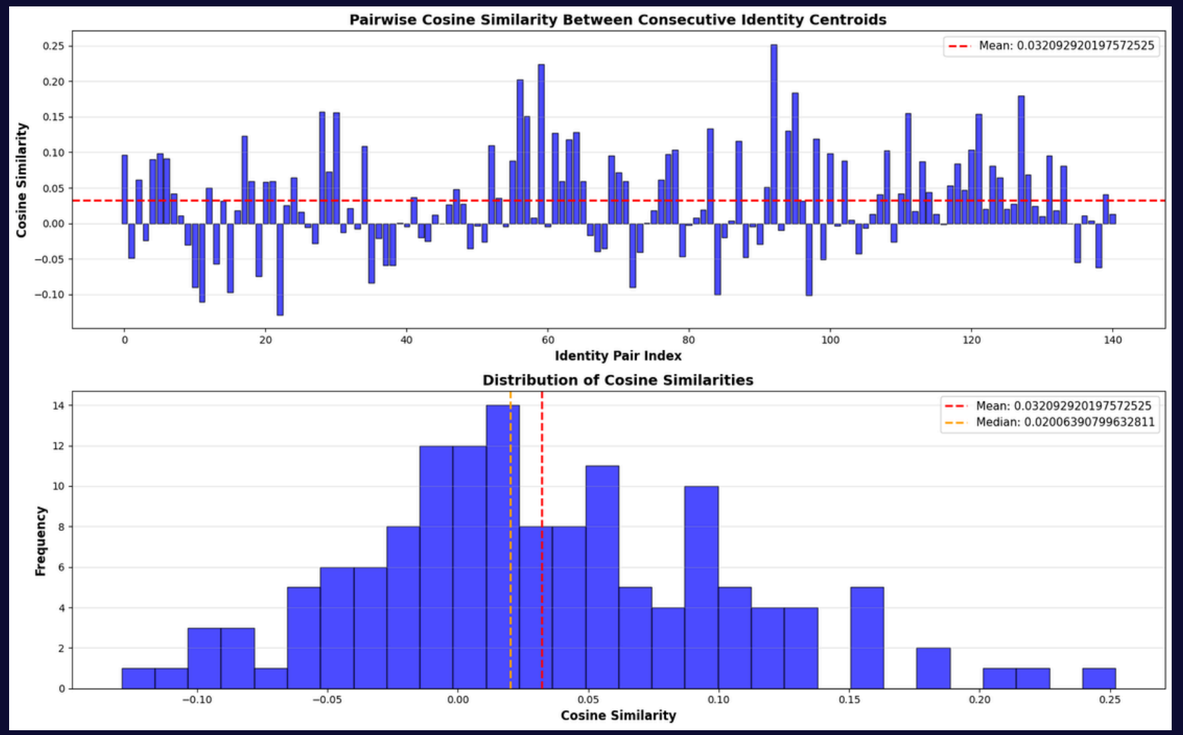
✓ 0.0s

FRGC Filter identities >= 100 imgs

All imgs

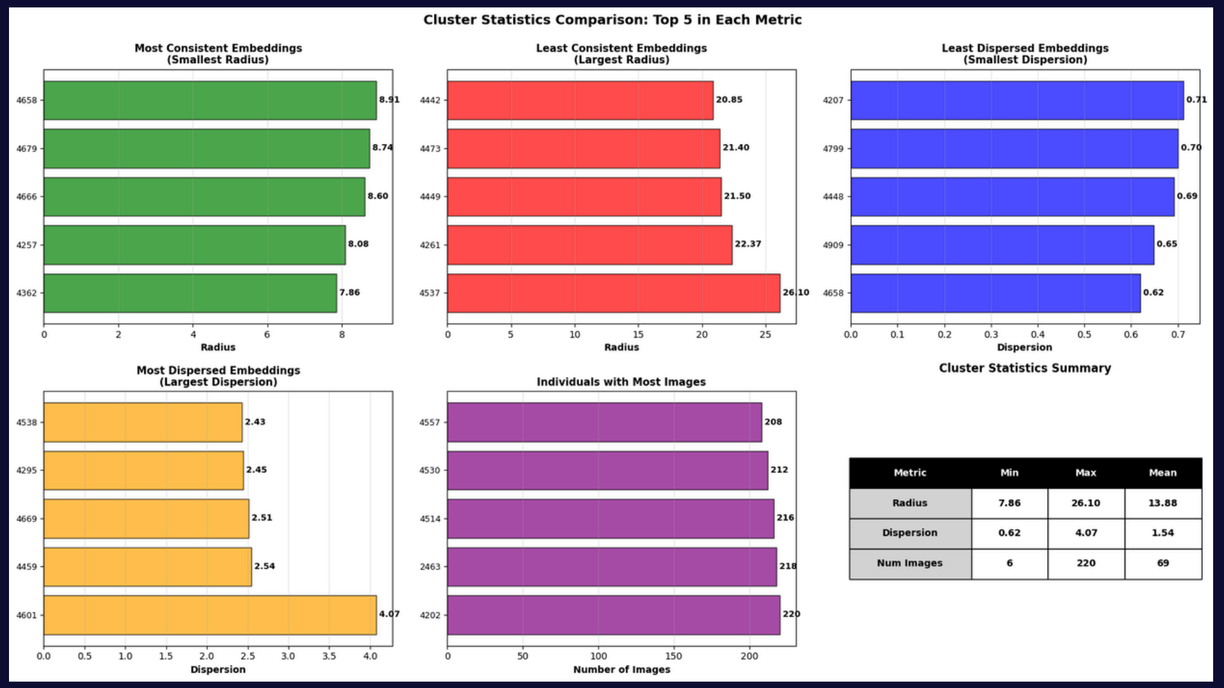


Filter >= 100 imgs

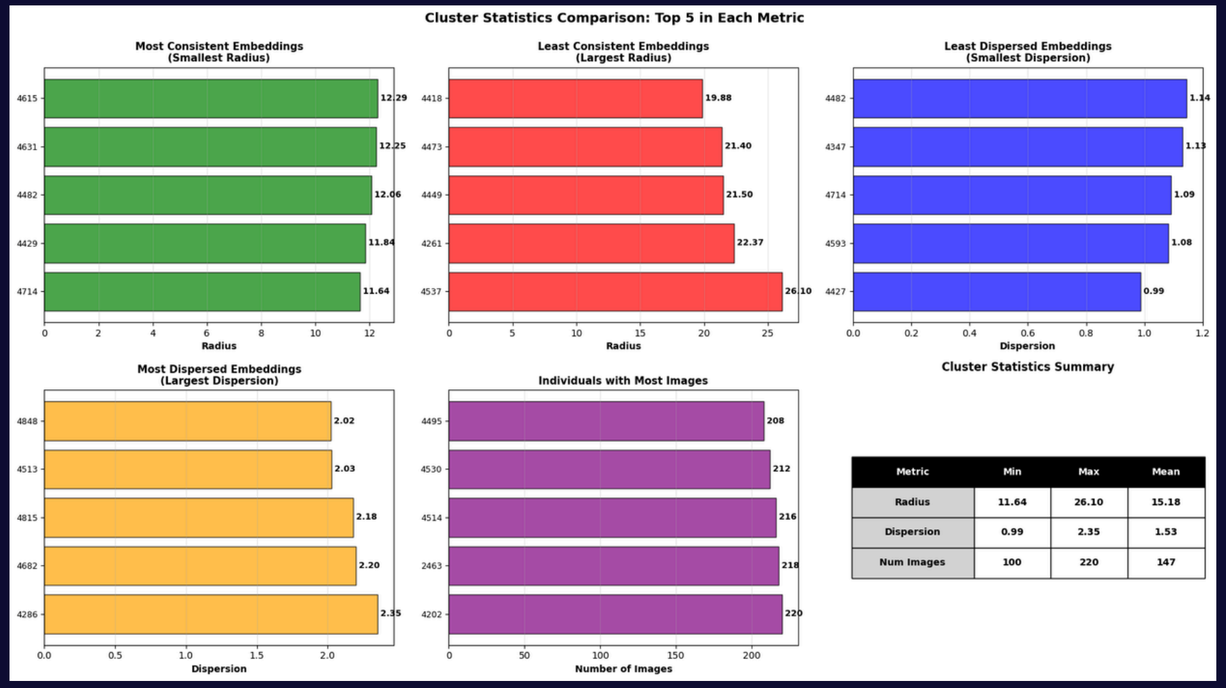


FRGC Filter identities >= 100 imgs

All imgs

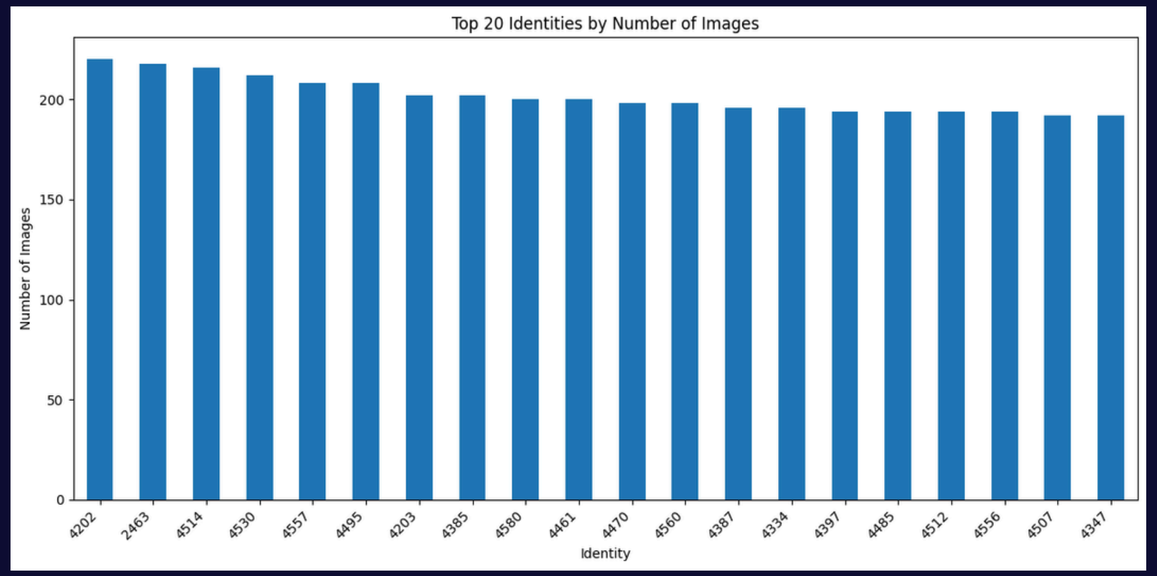


Filter >= 100 imgs

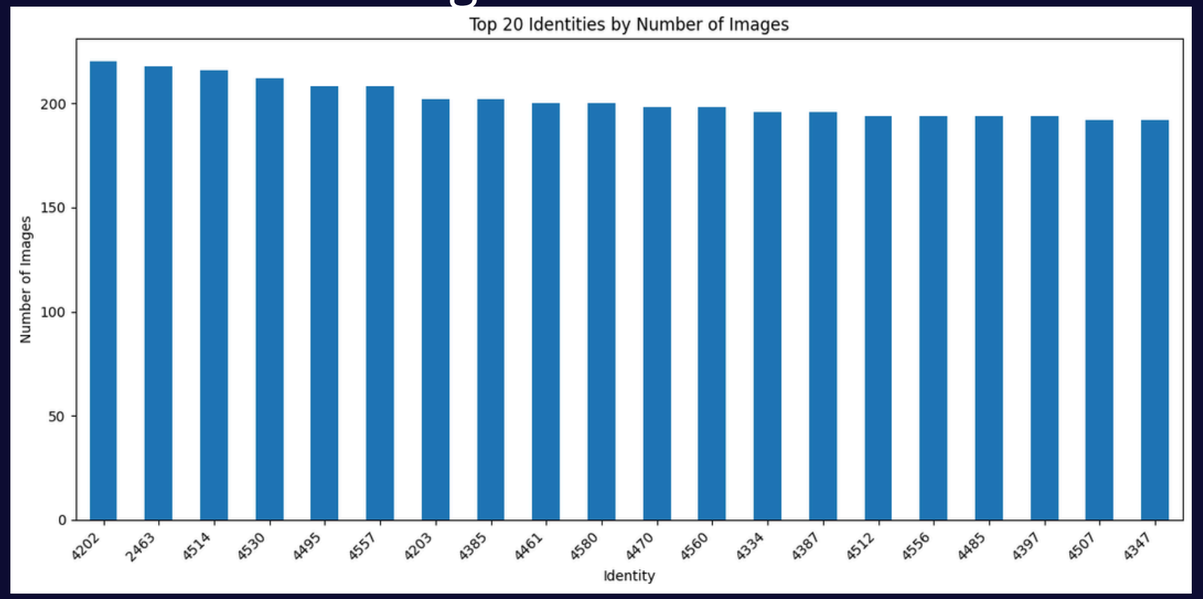


FRGC Filter identities >= 100 imgs

All imgs

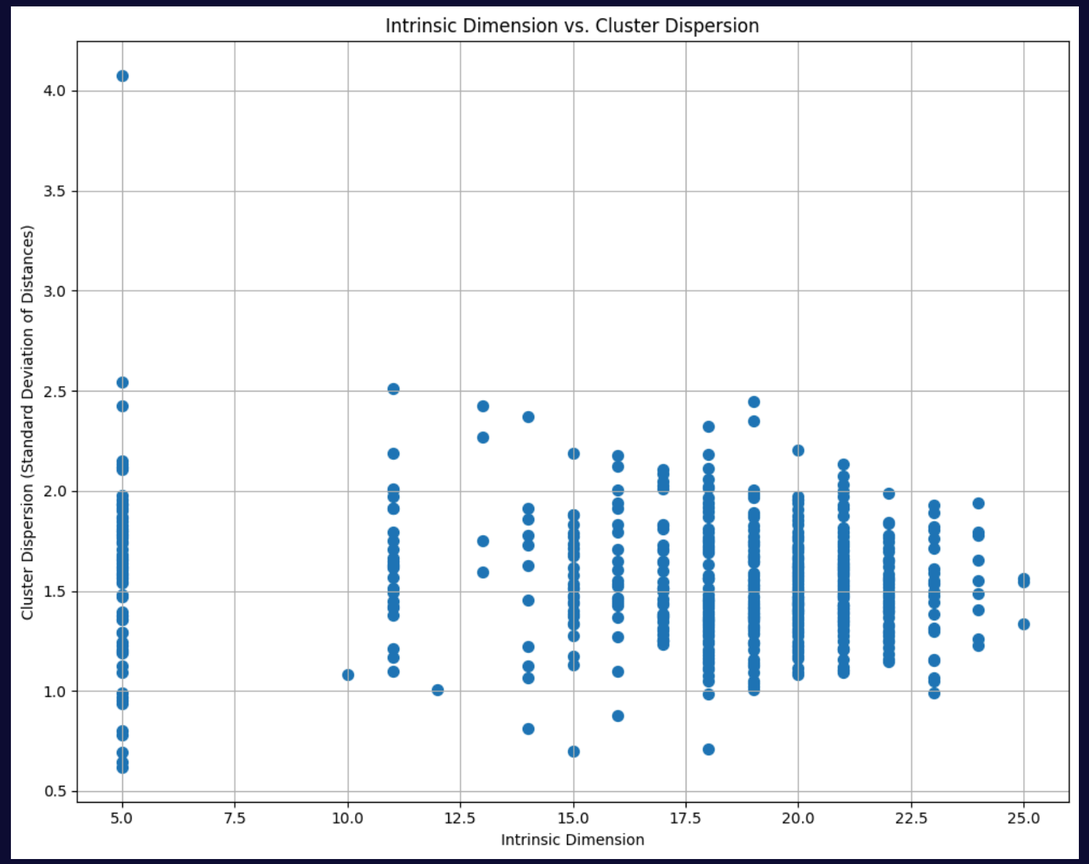


Filter >= 100 imgs

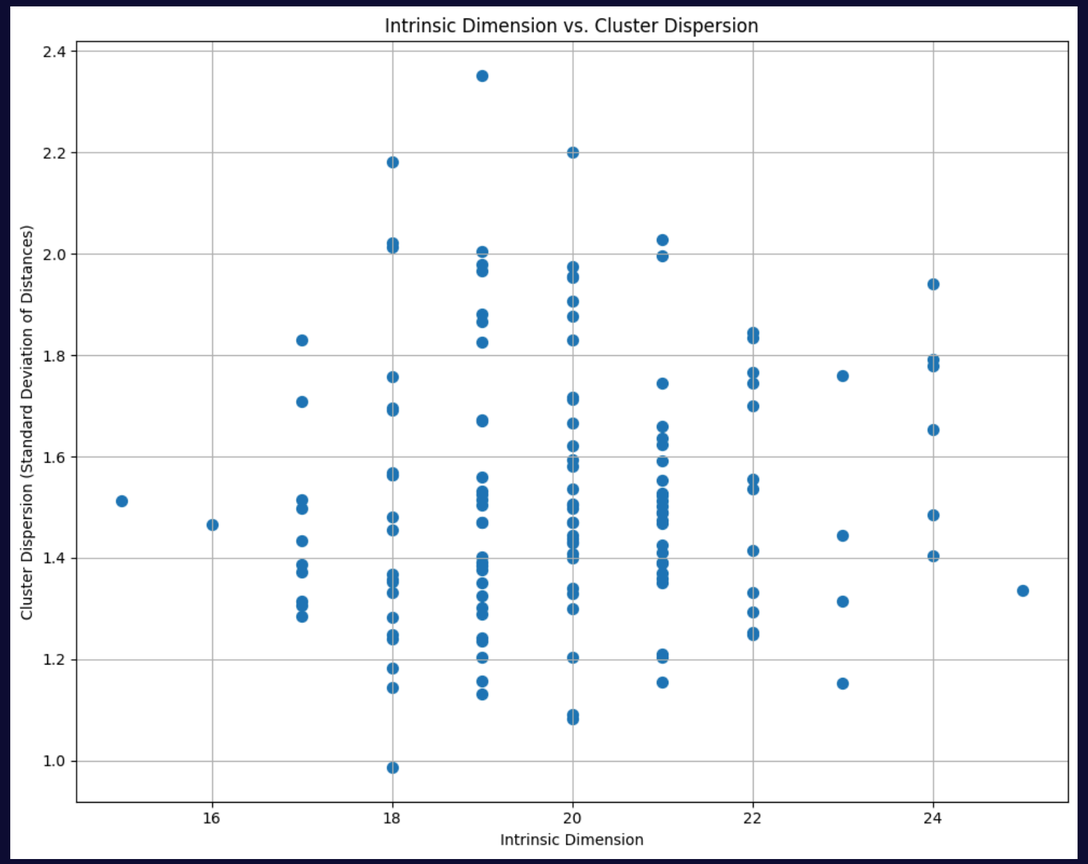


FRGC Filter identities >= 100 imgs

All imgs

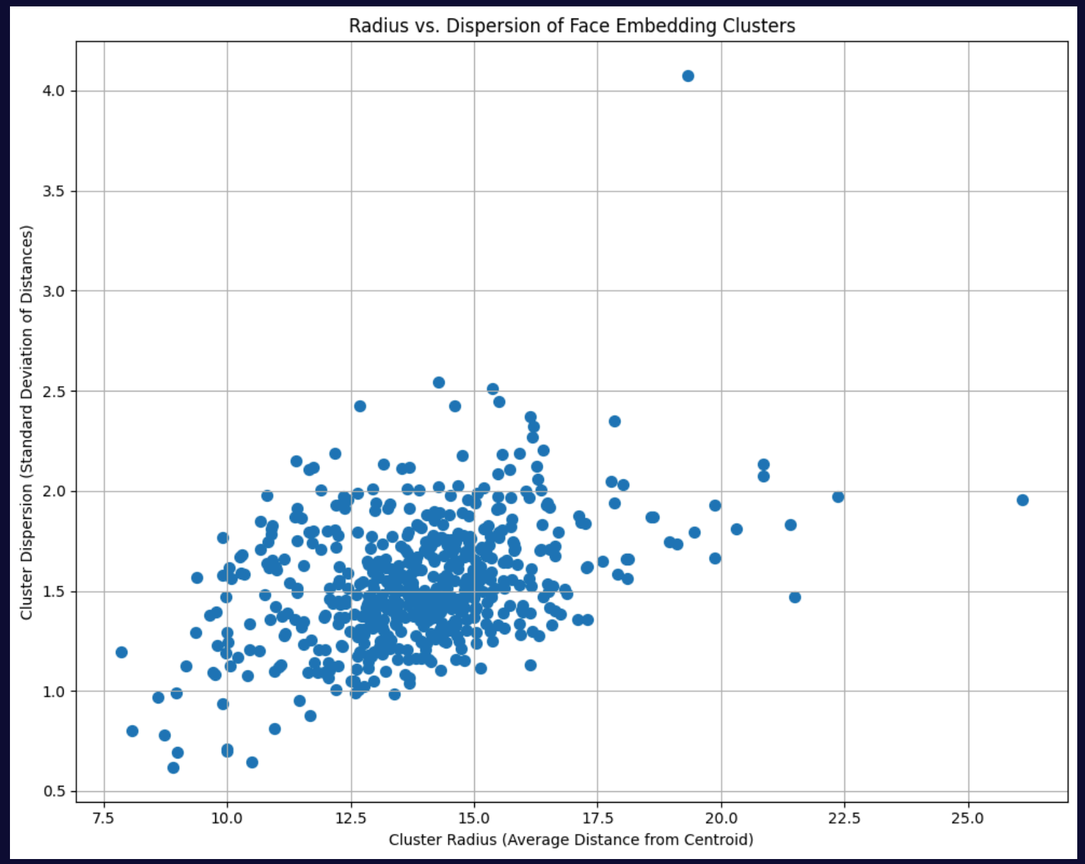


Filter >= 100 imgs

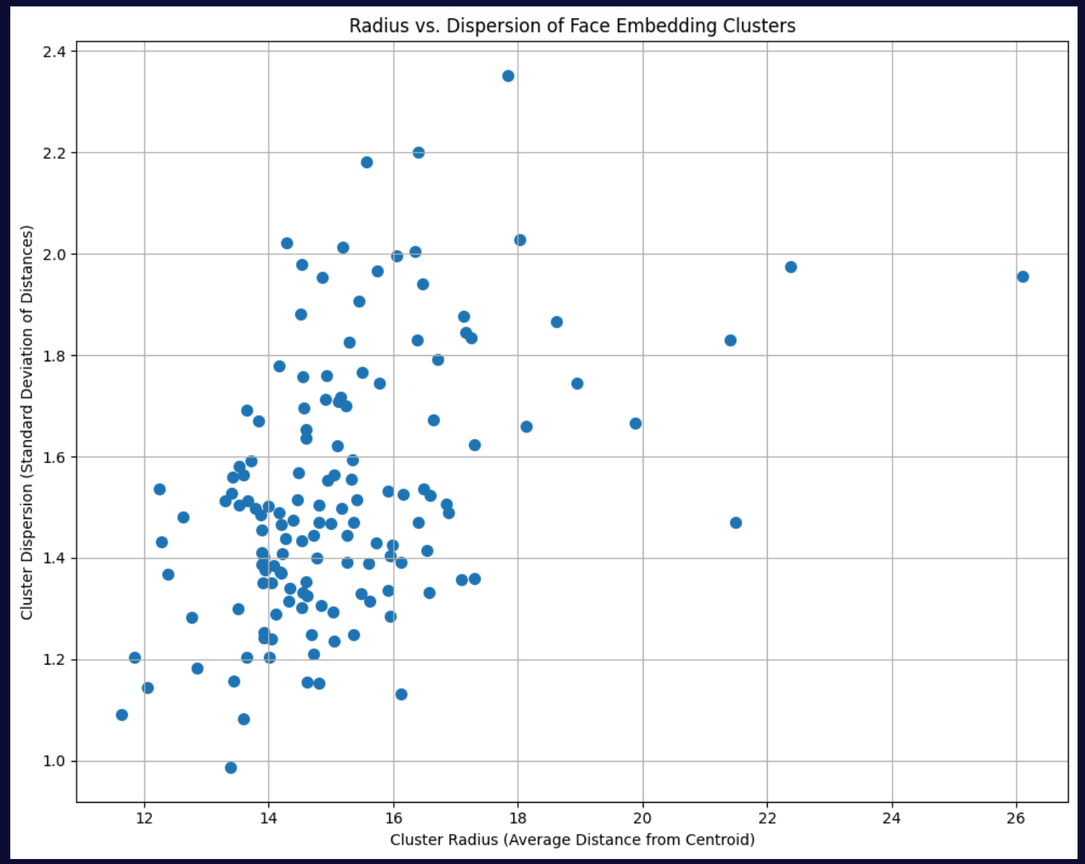


FRGC Filter identities >= 100 imgs

All imgs

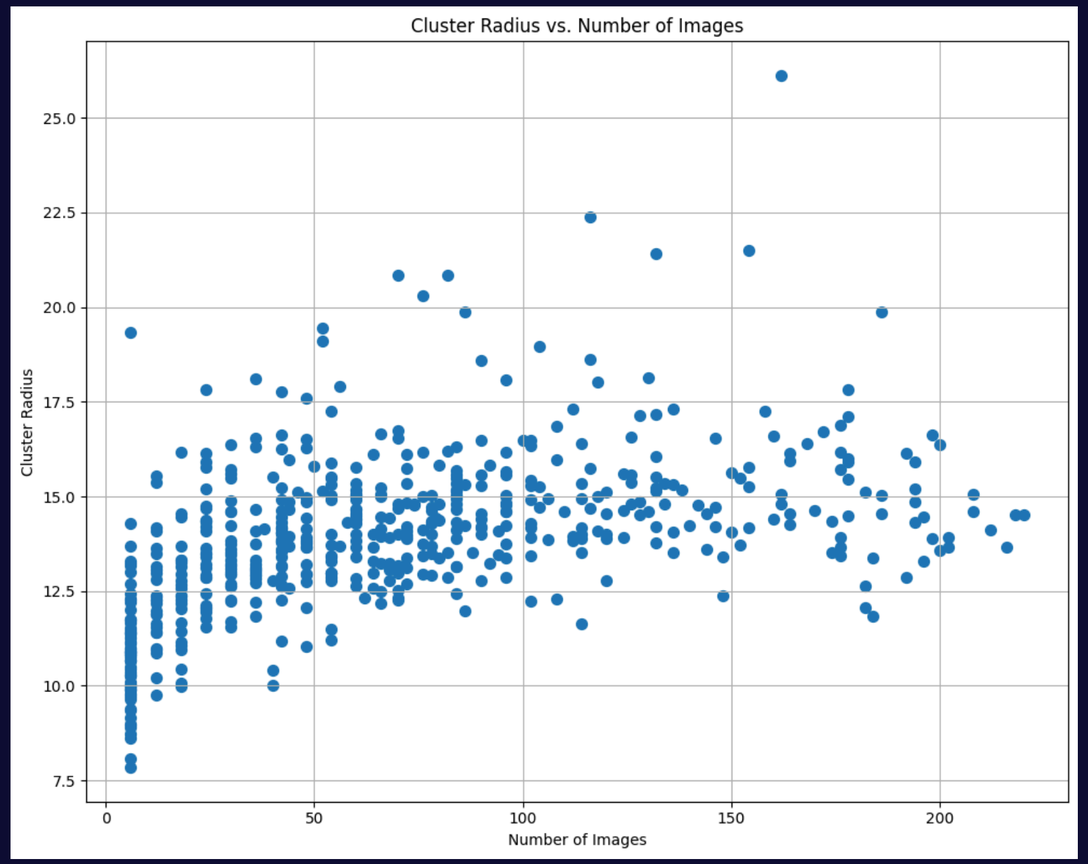


Filter >= 100 imgs

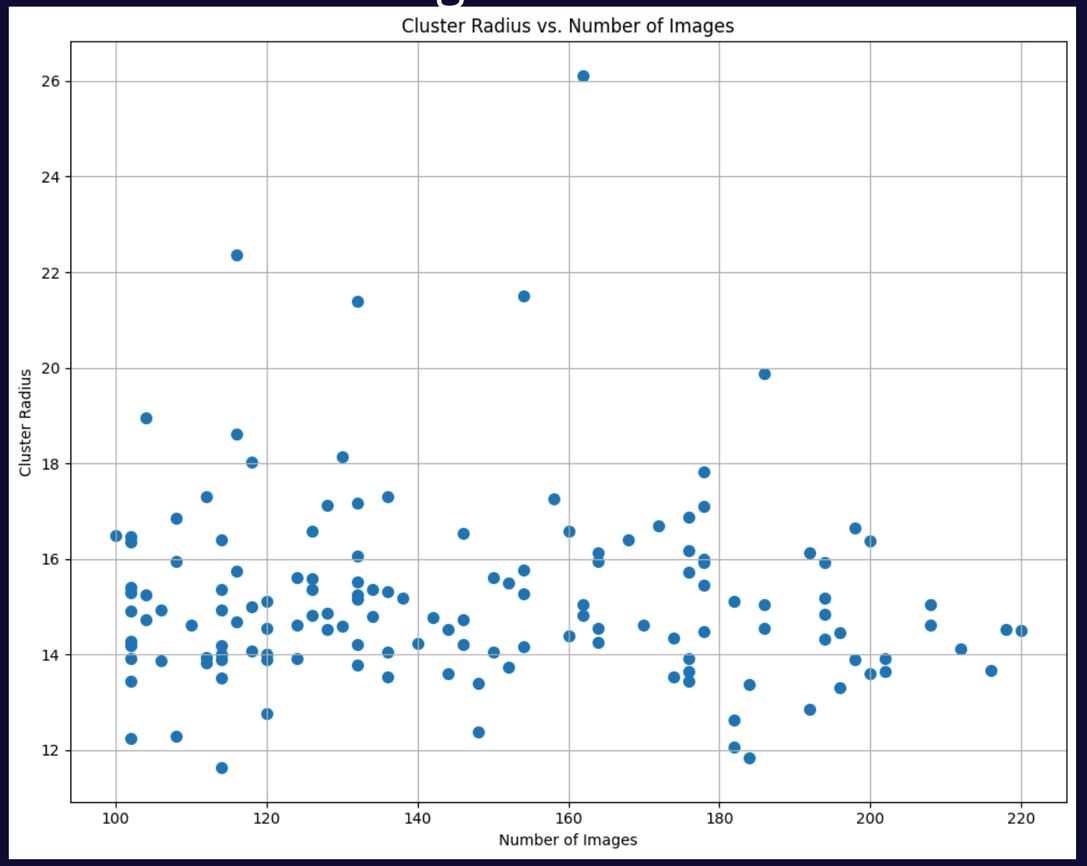


FRGC Filter identities >= 100 imgs

All imgs



Filter >= 100 imgs



FRGC Filter identities >= 100 imgs

All imgs

```
Identities with the 10 highest intrinsic dimensions:
      intrinsic_dimension
4397                   25
4758                   25
4711                   25
4433                   24
4511                   24
4327                   24
4312                   24
4708                   24
4839                   24
4493                   24

Identities with the 10 lowest intrinsic dimensions:
      intrinsic_dimension
4448                    5
4771                    5
4837                    5
4459                    5
4583                    5
4694                    5
4586                    5
4391                    5
4835                    5
4306                    5
```

Filter >= 100 imgs

```
Identities with the 10 highest intrinsic dimensions:
      intrinsic_dimension
4397                   25
4511                   24
4839                   24
4312                   24
4476                   24
4327                   24
4708                   24
4581                   23
4519                   23
4217                   23

Identities with the 10 lowest intrinsic dimensions:
      intrinsic_dimension
4514                   15
4684                   16
4580                   17
4485                   17
4773                   17
4512                   17
4388                   17
4670                   17
4379                   17
4334                   17
```

FRGC Filter identities >= 100 imgs

All imgs

Top 10 Identities with Highest Intrinsic Dimensions

Identity	Intrinsic Dim	Radius	Dispersion	Num Images
4397	25	15.92	1.34	194
4711	25	14.40	1.55	90
4758	25	14.35	1.56	78
4312	24	13.87	1.48	106
4327	24	16.70	1.79	172
4433	24	14.21	1.26	90
4476	24	15.96	1.40	108
4493	24	15.58	1.55	96
4511	24	14.61	1.65	110
4572	24	12.32	1.23	62

Filter >= 100 imgs

Top 10 Identities with Highest Intrinsic Dimensions

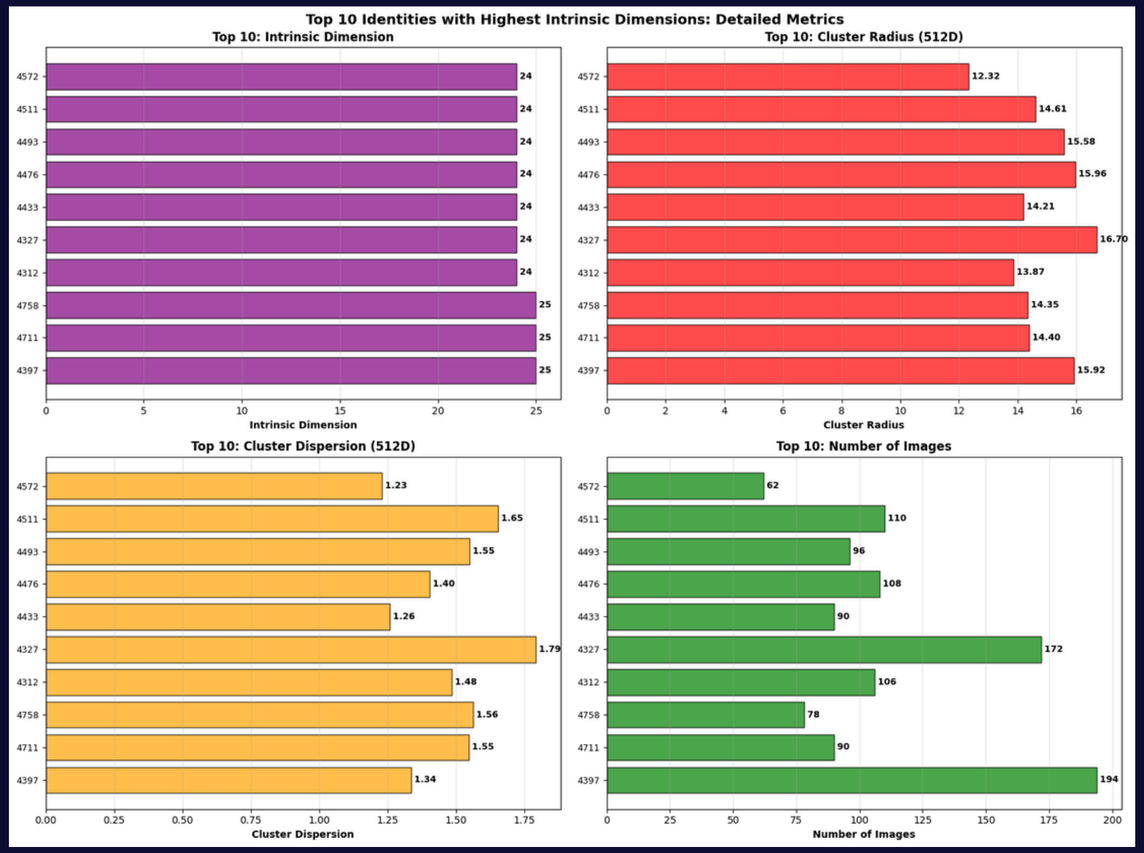
Identity	Intrinsic Dim	Radius	Dispersion	Num Images
4397	25	15.92	1.34	194
4312	24	13.87	1.48	106
4327	24	16.70	1.79	172
4476	24	15.96	1.40	108
4511	24	14.61	1.65	110
4708	24	14.18	1.78	102
4839	24	16.48	1.94	102
4213	23	15.26	1.44	104
4217	23	15.62	1.31	150
4519	23	14.93	1.76	114

Cluster Statistics Summary

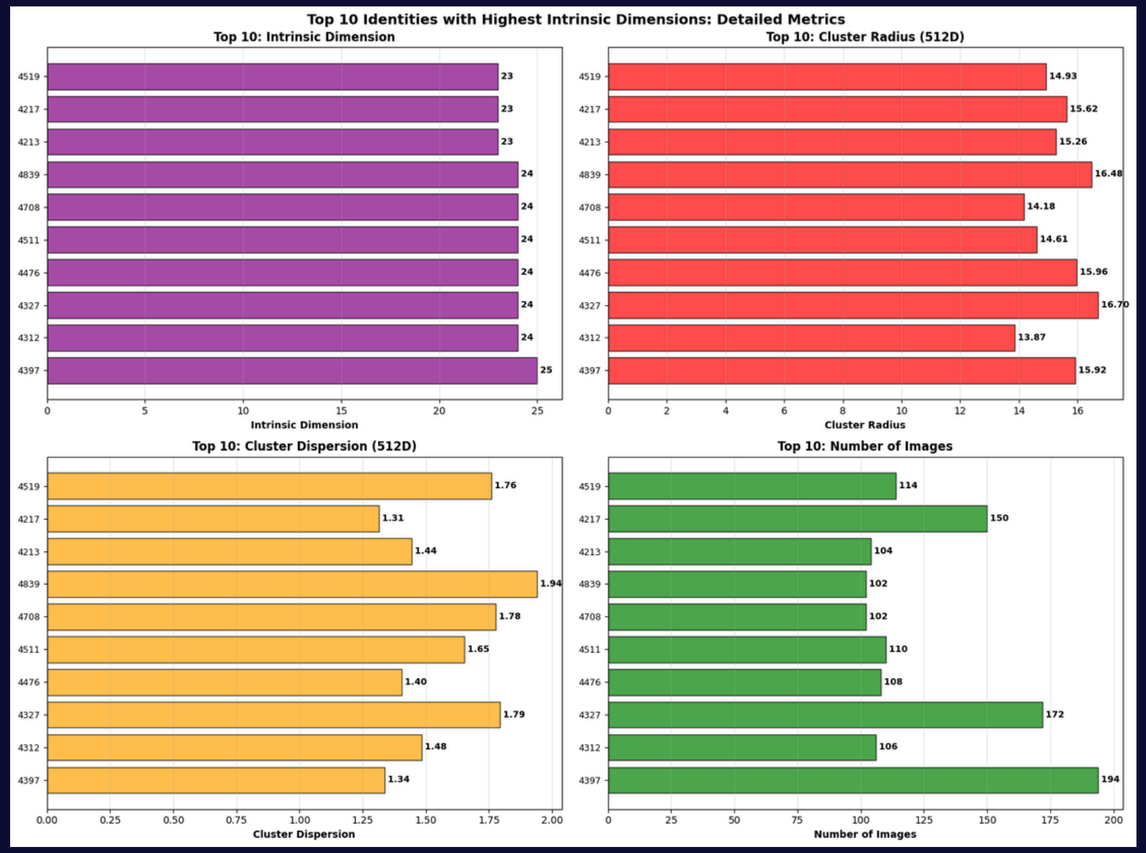
Metric	Max	Min	Mean
Radius (512D)	26.10	7.86	13.88
Dispersion	4.07	0.62	1.54
Num Images	220	6	69

FRGC Filter identities >= 100 imgs

All imgs



Filter >= 100 imgs



Cluster Statistics Summary

Metric	Max	Min	Mean
Radius (512D)	26.10	7.86	13.88
Dispersion	4.07	0.62	1.54
Num Images	220	6	69

Compare all imgs

FRGC vs LFW

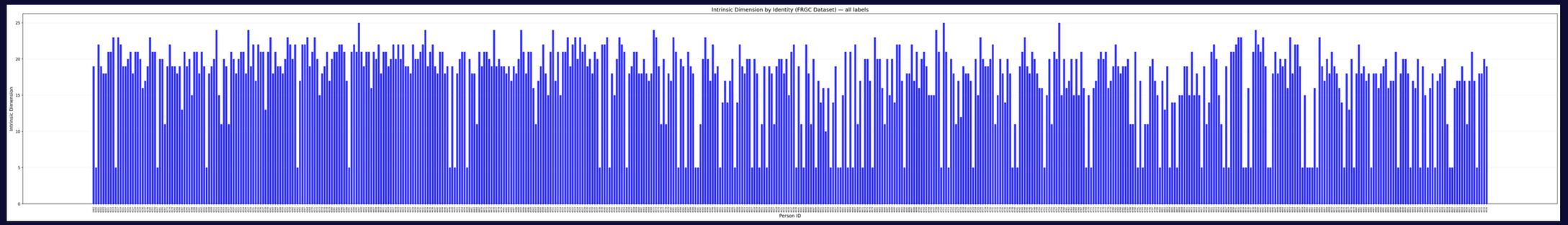
```
Found frgc_face_embeddings.csv
Embeddings: (39327, 512)
Valid images: 39327
Unique person_ids: 568
```

```
Processing images: 100%|██████████| 4324/4324
Extracted embeddings shape: (4324, 512)
```

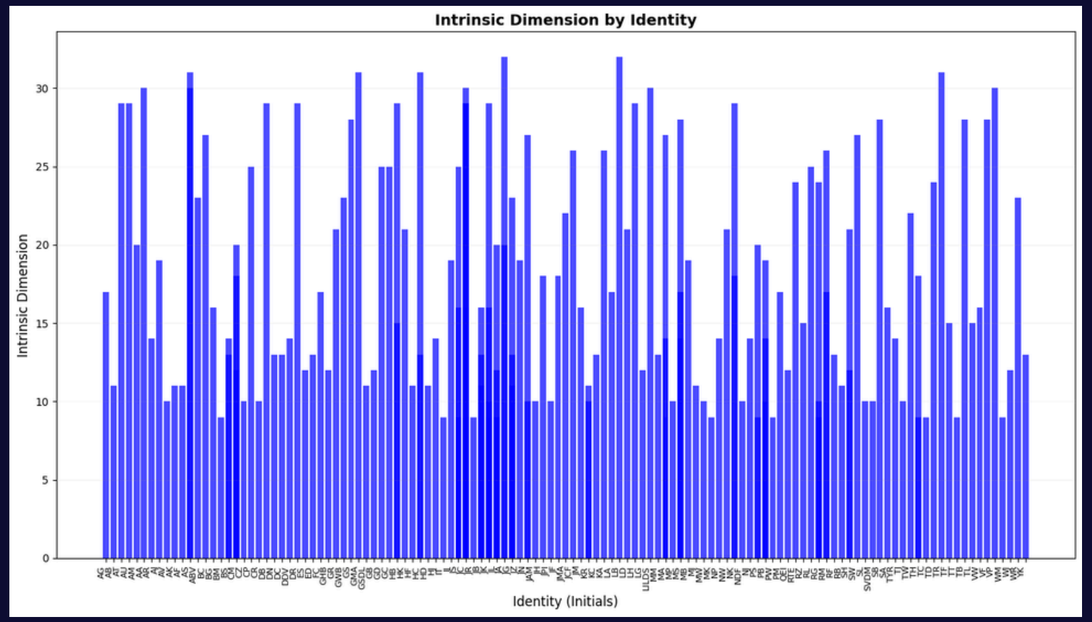
```
Images shape: (4324, 62, 47), Identities: 158
```

FRGC Filter identities >= 100 imgs

FRGC

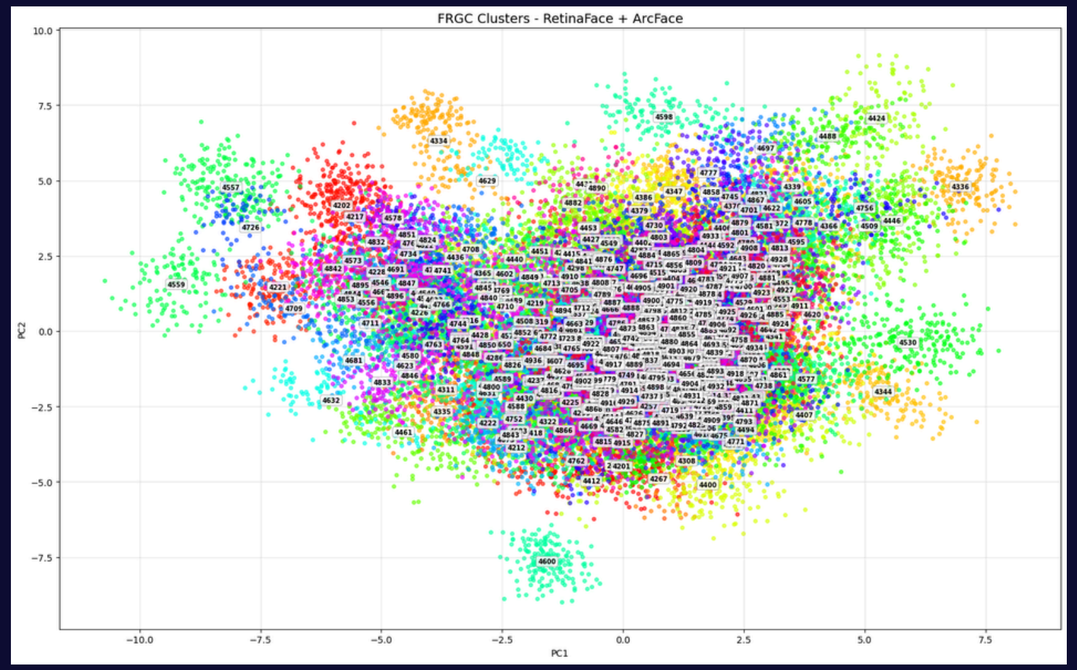


LFW

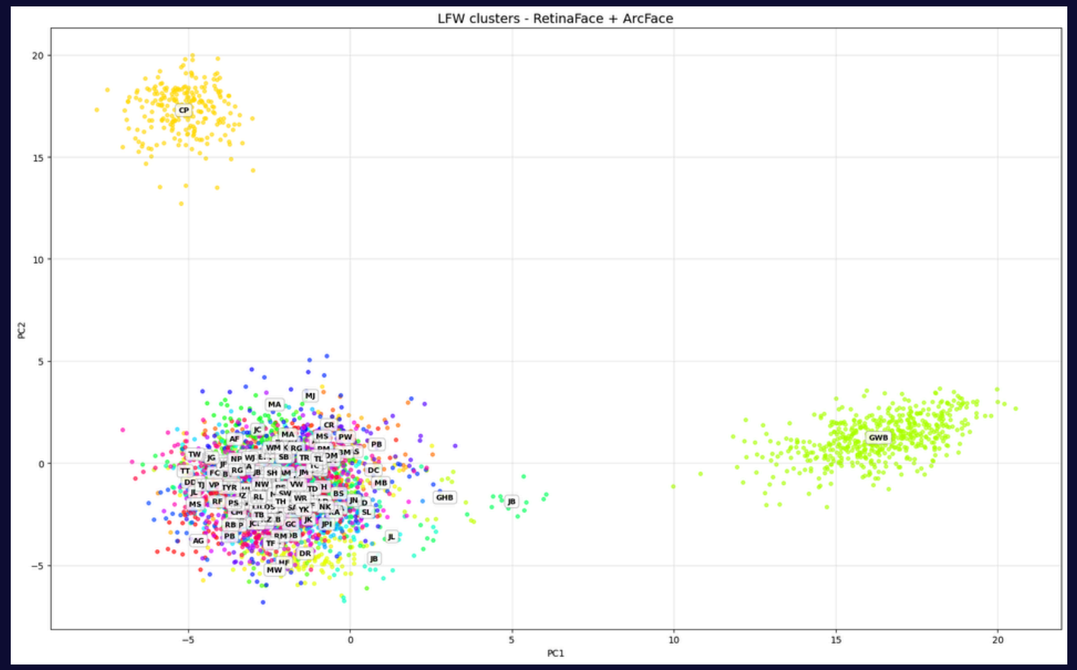


FRGC Filter identities >= 100 imgs

FRGC

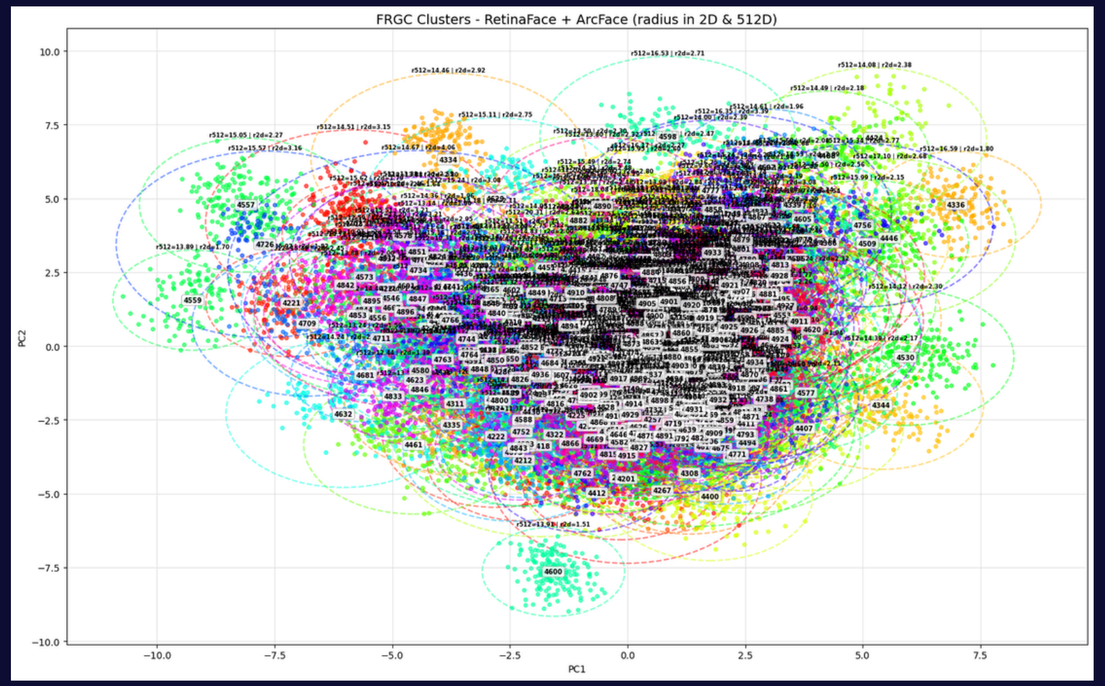


LFW

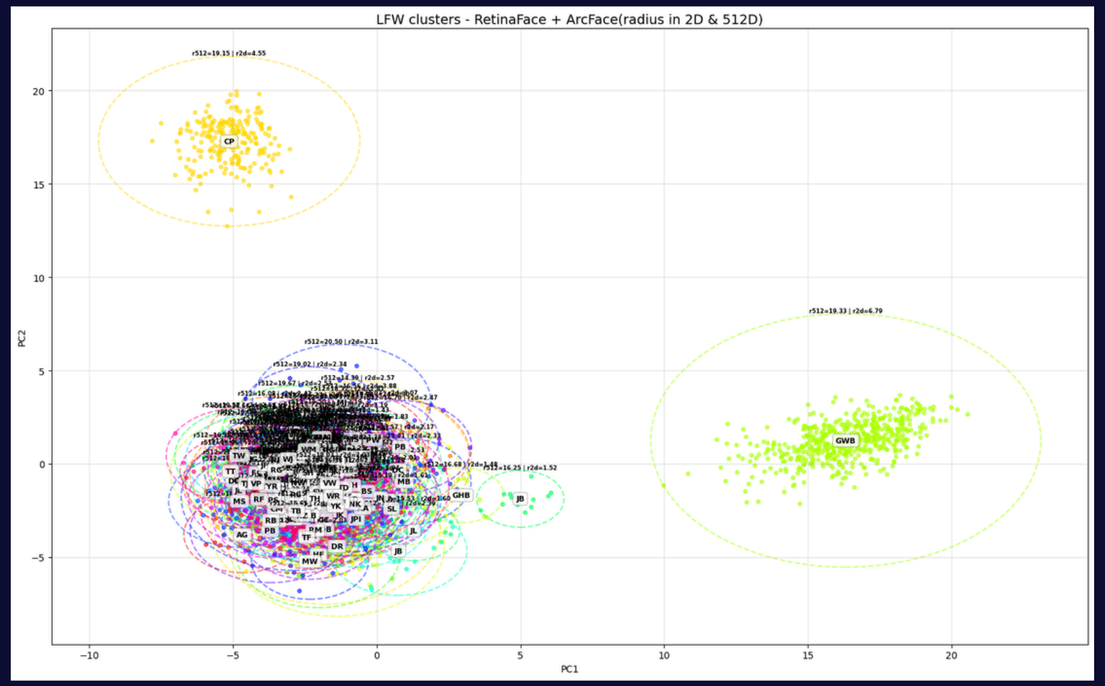


FRGC Filter identities >= 100 imgs

FRGC



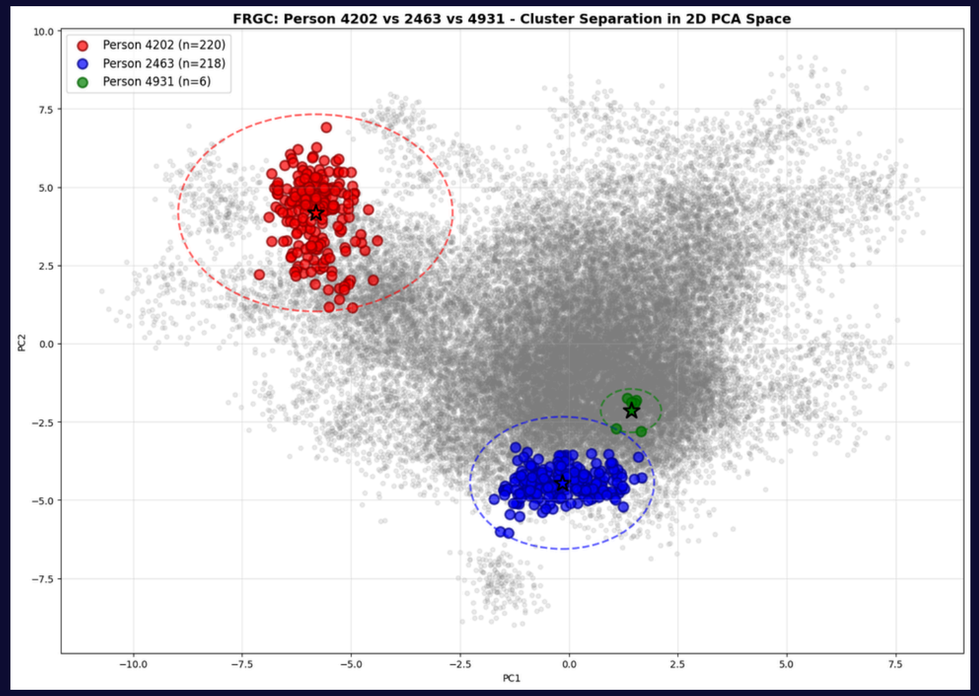
LFW



FRGC Filter identities >= 100 imgs

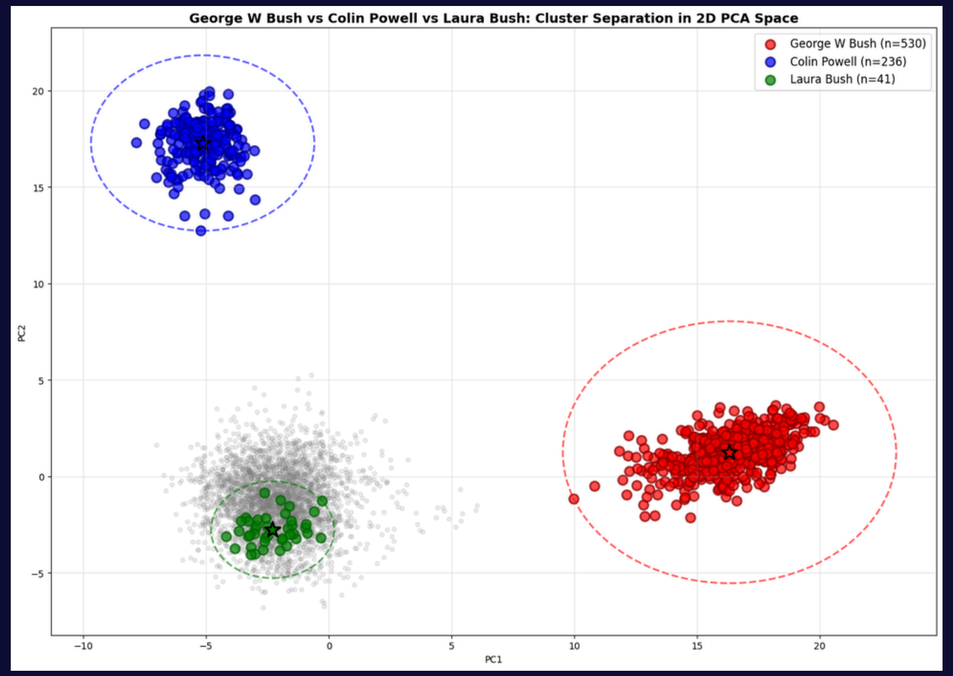
FRGC

```
Person 4202: 220 images in dataset
Person 2463: 218 images in dataset
Person 4931: 6 images in dataset
```



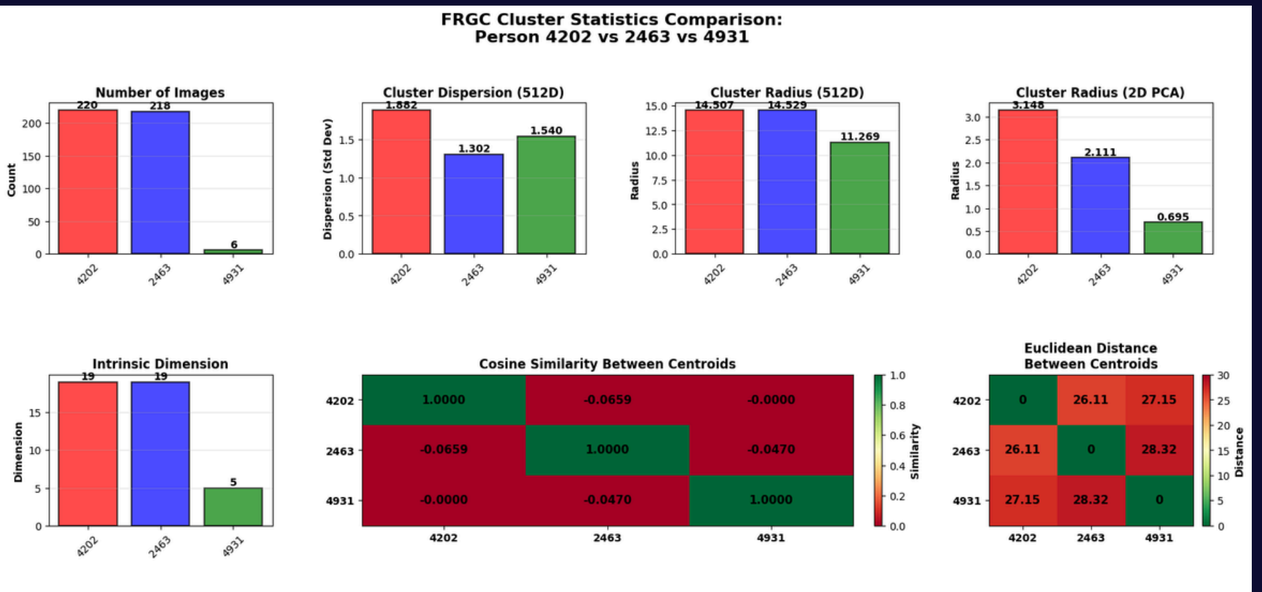
LFW (here is not the least just a random)

```
George W Bush (Label 35): 530 images in dataset
Colin Powell (Label 23): 236 images in dataset
Laura Bush (Label 91): 41 images in dataset
```



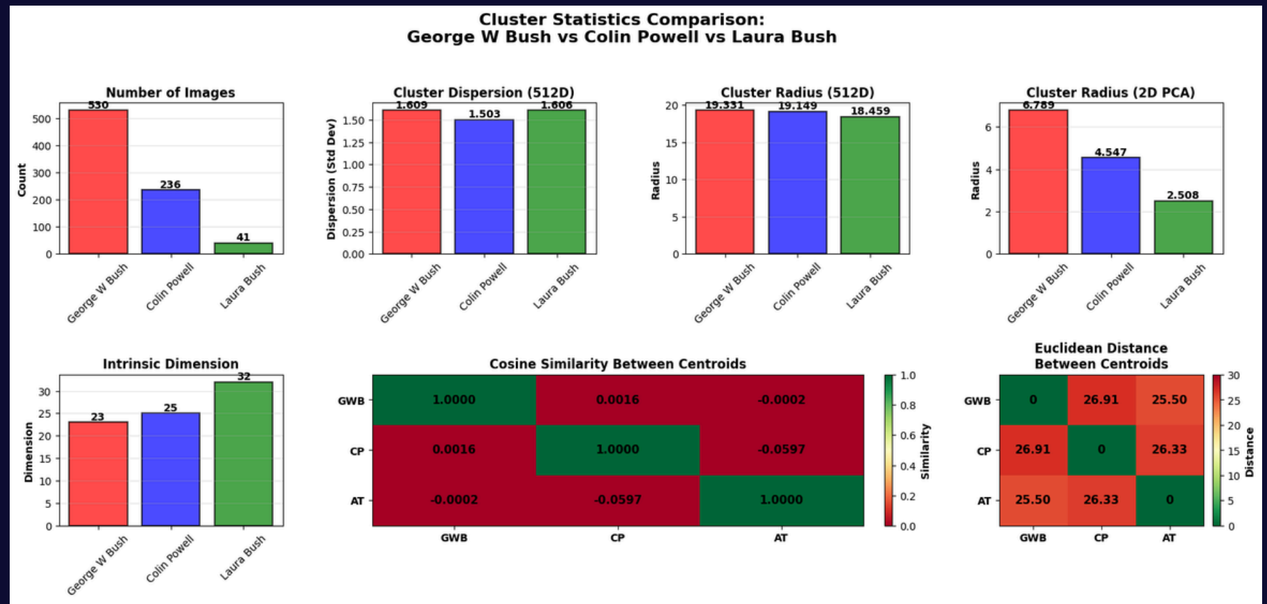
FRGC Filter identities >= 100 imgs

FRGC



Metric	Person 4202	Person 2463	Person 4931
Number of Images	220	218	6
Radius (512D)	14.506602631599486	14.529445198064755	11.269393211155041
Dispersion (512D)	1.8821422014891271	1.302409623588808	1.5401442839311412
Intrinsic Dimension	19	19	5
Radius (2D PCA)	3.148465410093664	2.111407381166888	0.6949382401039255

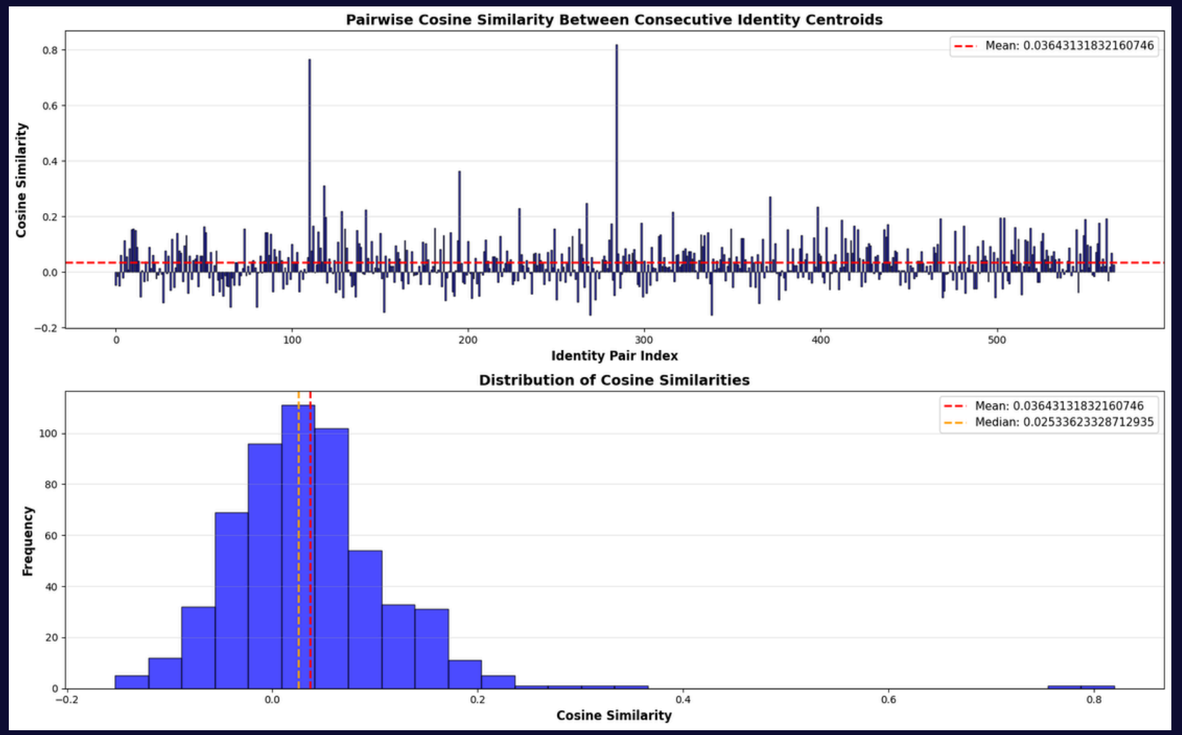
LFW



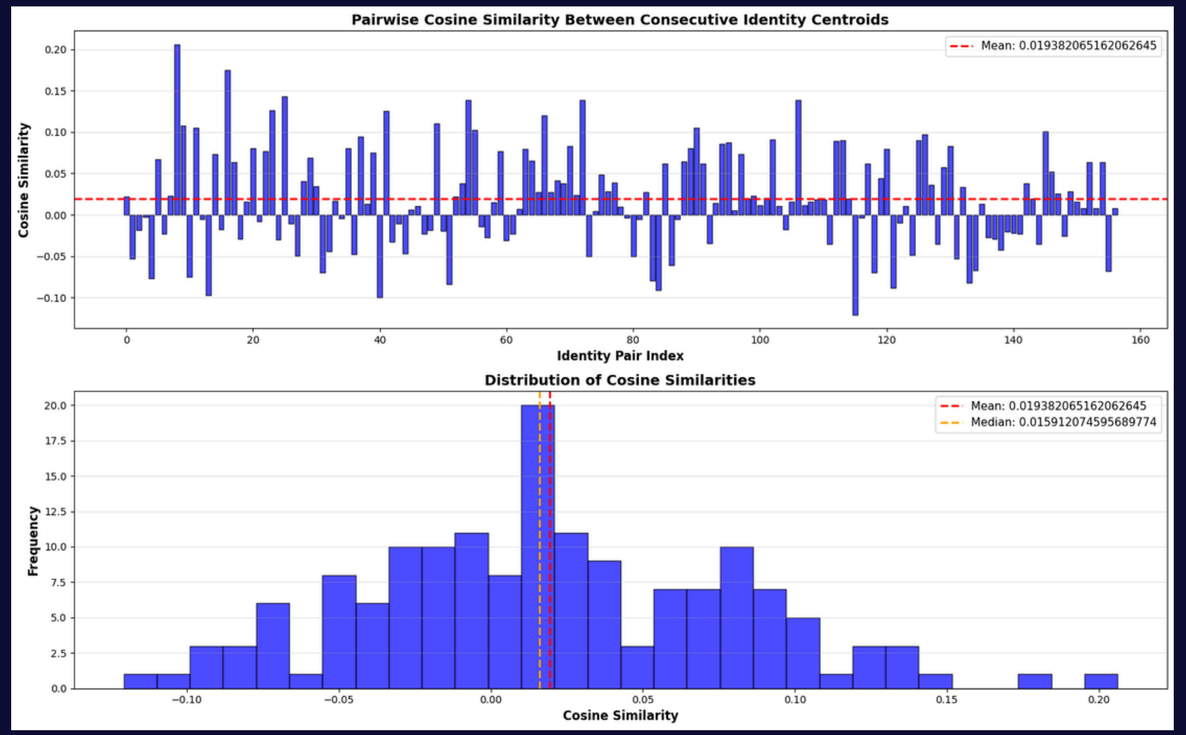
Metric	George W Bush	Colin Powell	Laura Bush
Number of Images	530	236	41
Radius (512D)	19.331116	19.149479	18.458601
Dispersion (512D)	1.6086893	1.5028448	1.6061269
Intrinsic Dimension	23	25	32
Radius (2D PCA)	6.788703	4.546931	2.5075235

FRGC Filter identities >= 100 imgs

FRGC

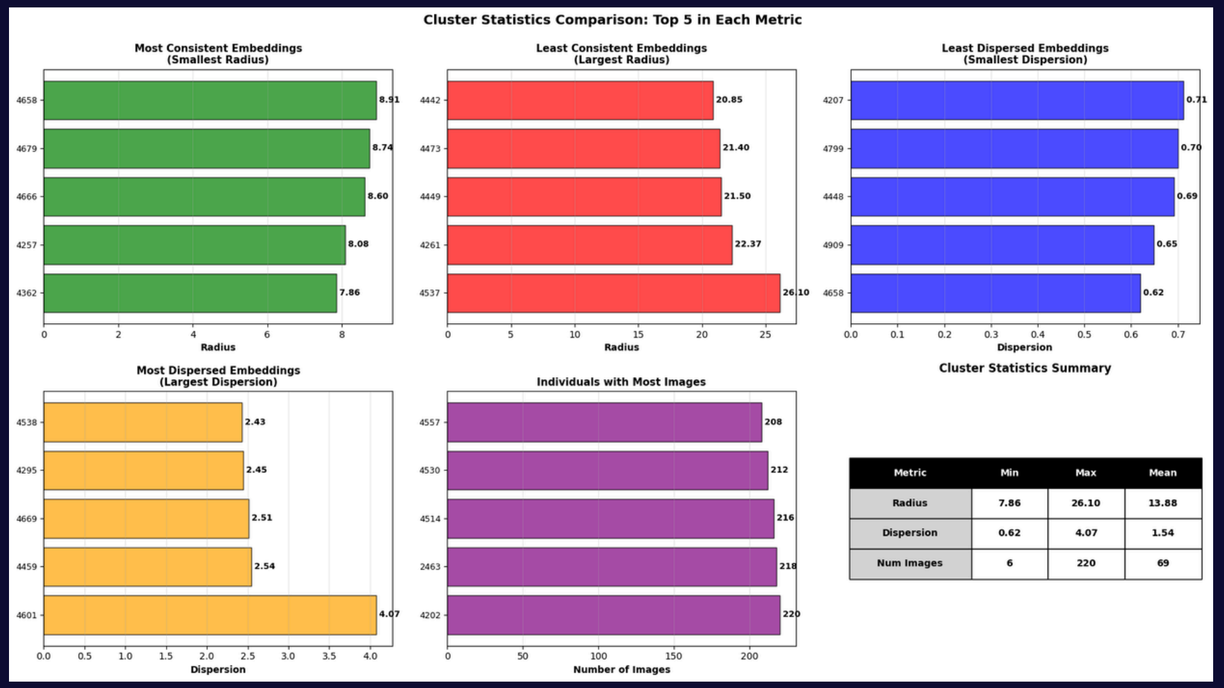


LFW



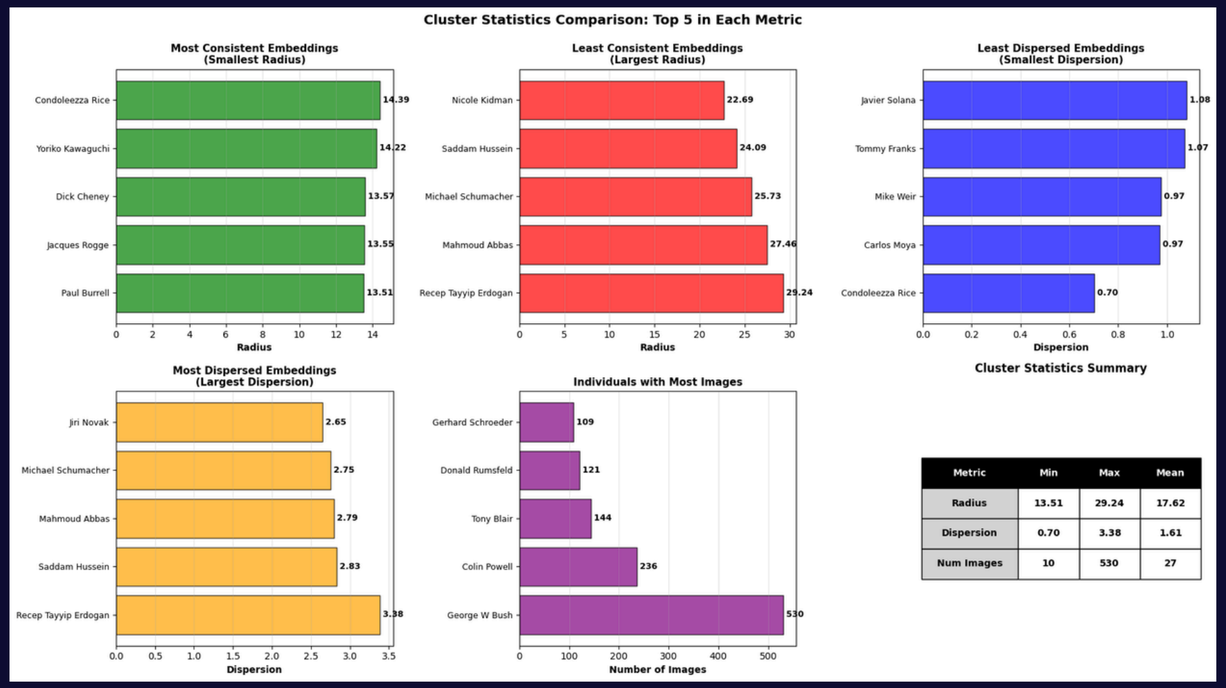
FRGC Filter identities >= 100 imgs

FRGC



Metric	Min	Max	Mean
Radius	7.86	26.10	13.88
Dispersion	0.62	4.07	1.54
Num Images	6	220	69

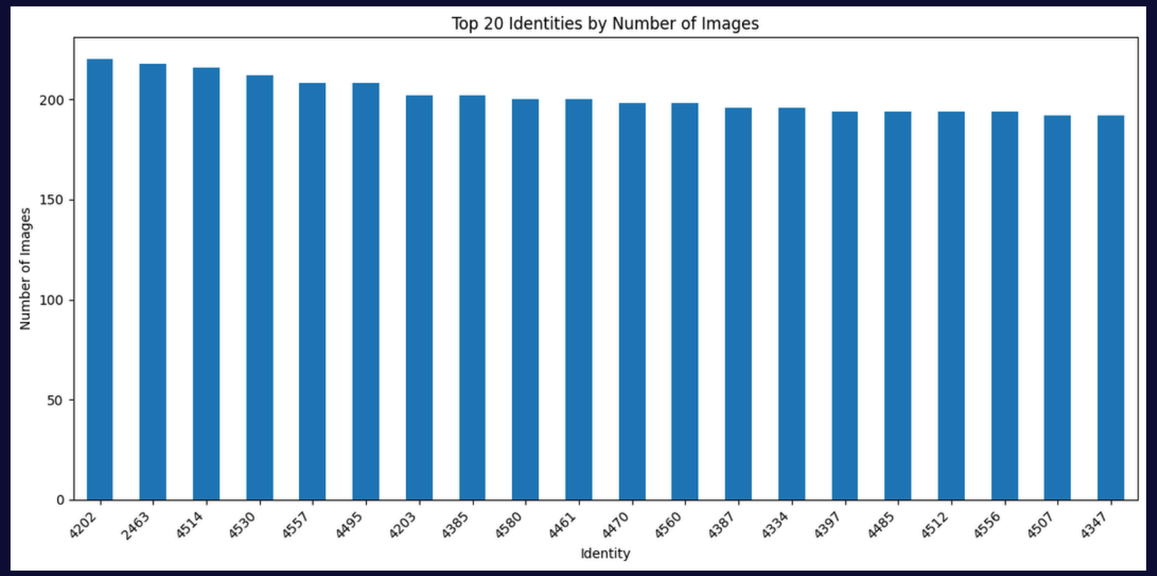
LFW



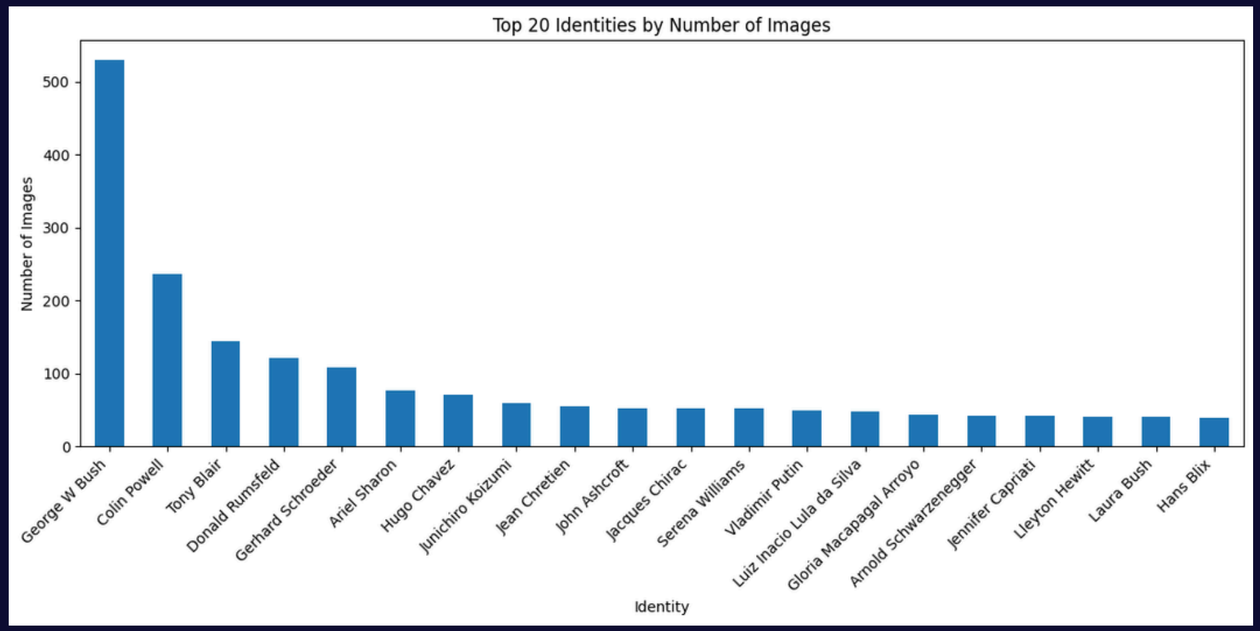
Metric	Min	Max	Mean
Radius	13.51	29.24	17.62
Dispersion	0.70	3.38	1.61
Num Images	10	530	27

FRGC Filter identities >= 100 imgs

FRGC

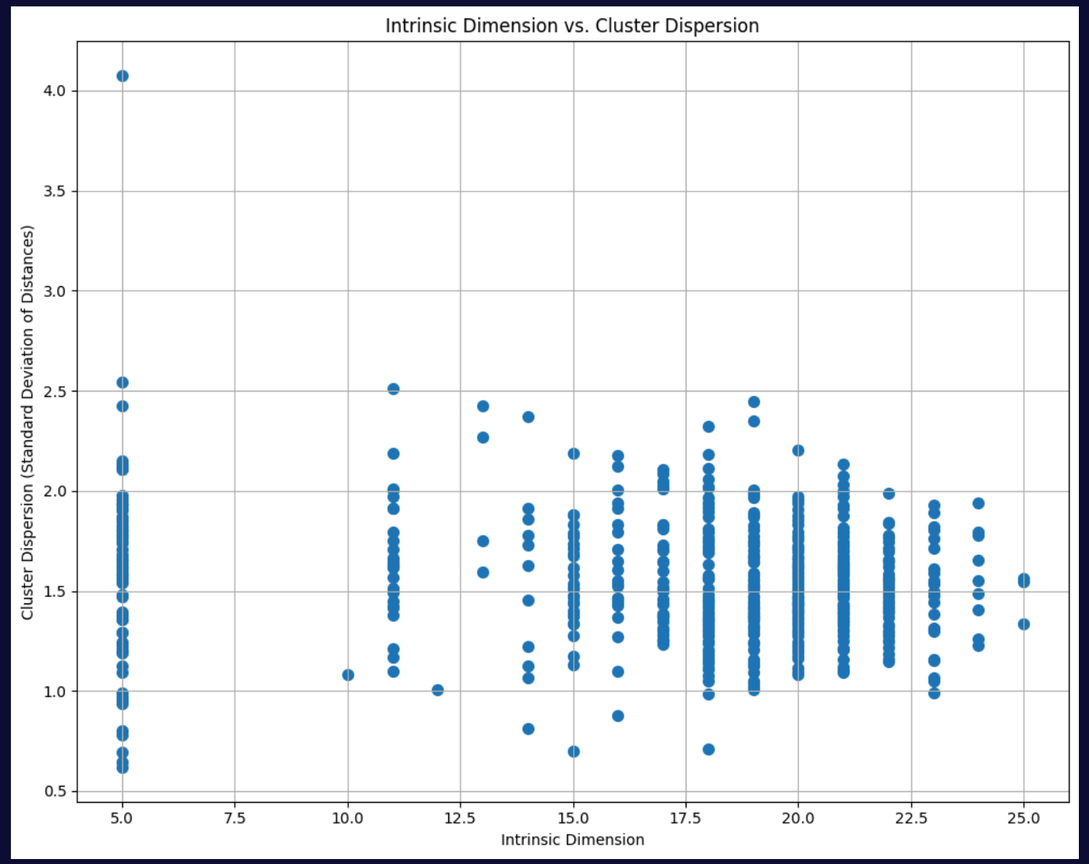


LFW

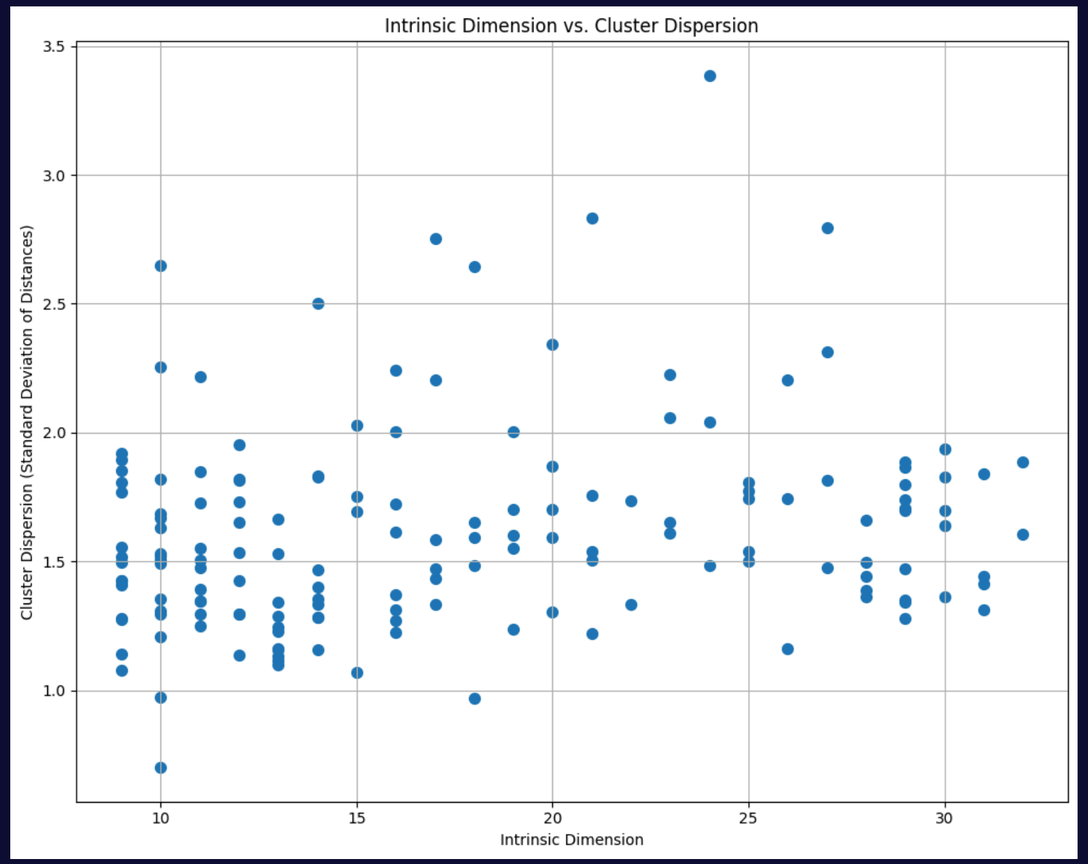


FRGC Filter identities >= 100 imgs

FRGC

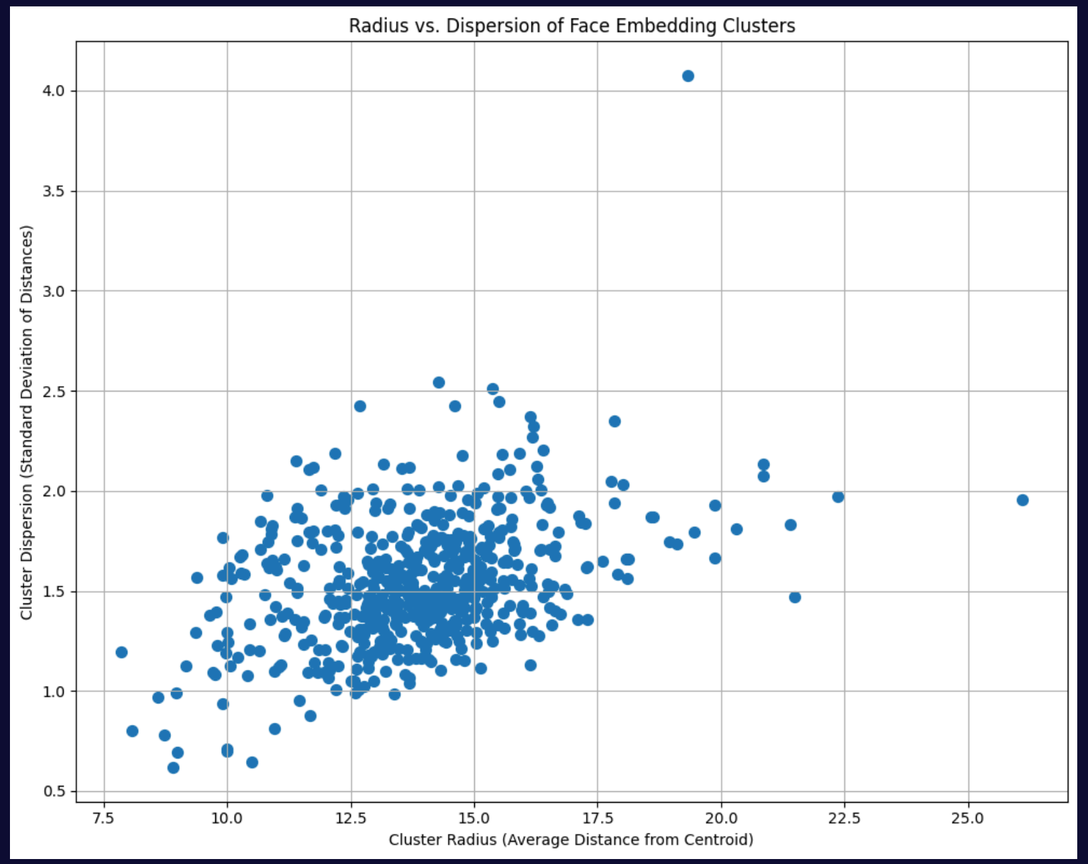


LFW

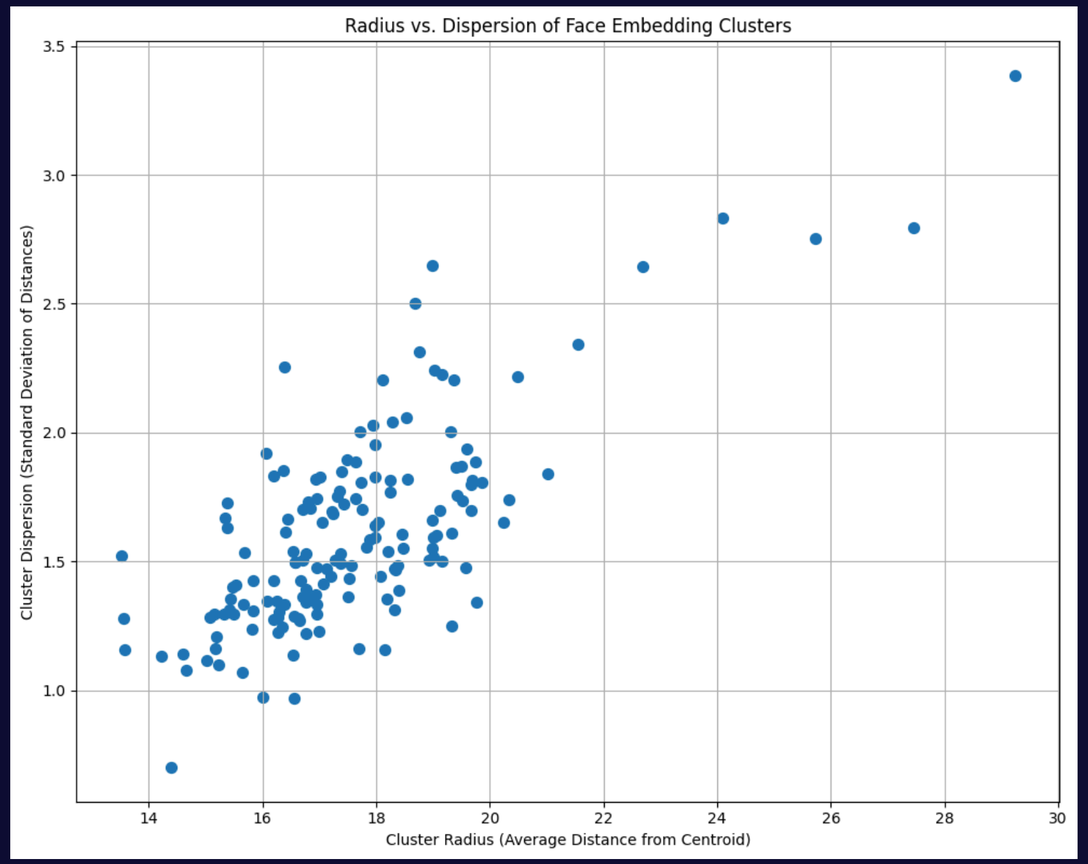


FRGC Filter identities >= 100 imgs

FRGC

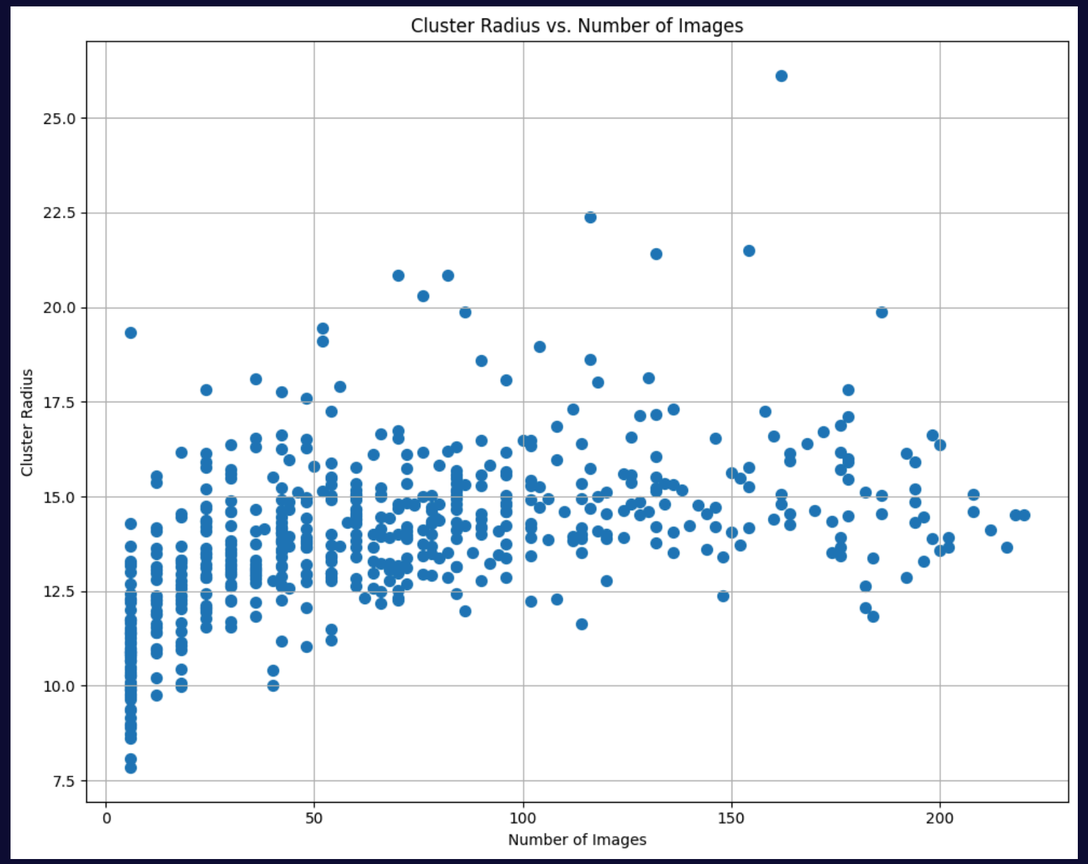


LFW

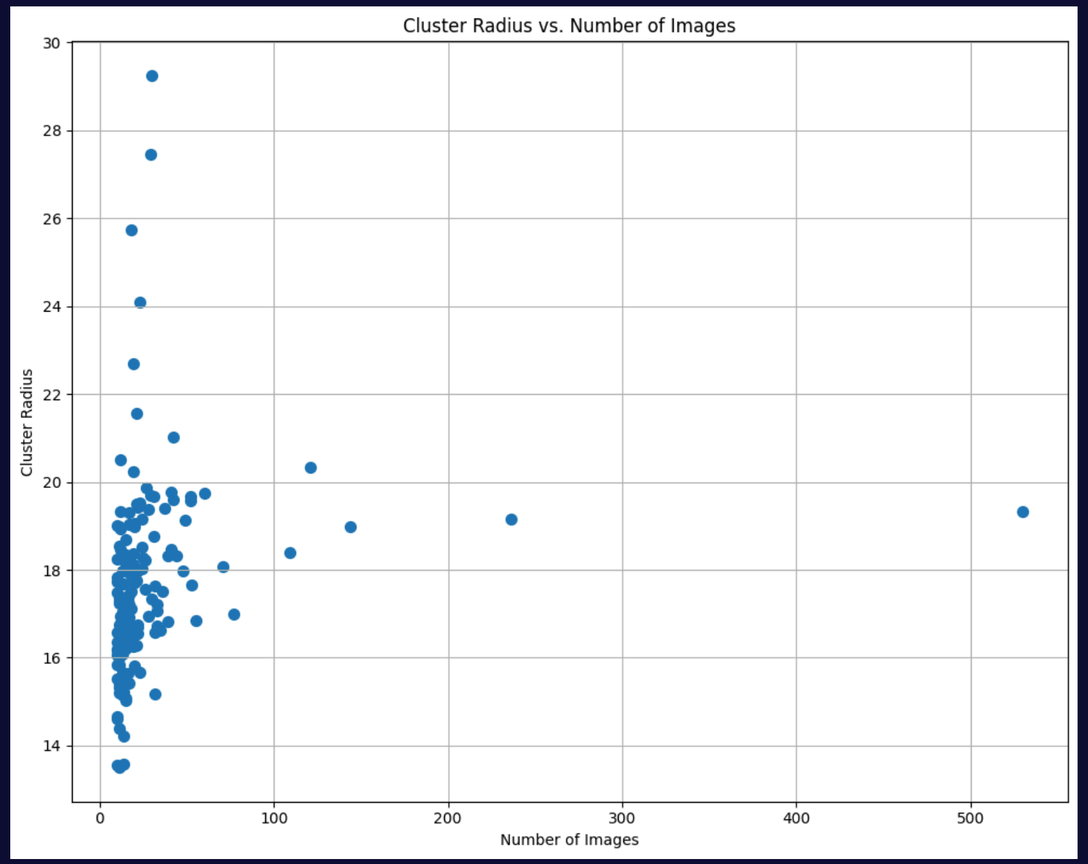


FRGC Filter identities >= 100 imgs

FRGC



LFW



FRGC Filter identities >= 100 imgs

FRGC

Identities with the 10 highest intrinsic dimensions:

	intrinsic_dimension
4397	25
4758	25
4711	25
4433	24
4511	24
4327	24
4312	24
4708	24
4839	24
4493	24

Identities with the 10 lowest intrinsic dimensions:

	intrinsic_dimension
4448	5
4771	5
4837	5
4459	5
4583	5
4694	5
4586	5
4391	5
4835	5
4306	5

LFW

Identities with the 10 highest intrinsic dimensions:

	intrinsic_dimension
John Ashcroft	32
Laura Bush	32
Hugo Chavez	31
Gloria Macapagal Arroyo	31
Arnold Schwarzenegger	31
Tom Ridge	31
Vladimir Putin	30
Luiz Inacio Lula da Silva	30
Jennifer Capriati	30
Andre Agassi	30

Identities with the 10 lowest intrinsic dimensions:

	intrinsic_dimension
Bill McBride	9
Jacques Rogge	9
Jason Kidd	9
Ian Thorpe	9
Jean-David Levitte	9
Javier Solana	9
Paul Wolfowitz	9
Paradorn Srichaphan	9
Richard Gere	9
Mohammad Khatami	9

FRGC Filter identities >= 100 imgs

FRGC

Top 10 Identities with Highest Intrinsic Dimensions

Identity	Intrinsic Dim	Radius	Dispersion	Num Images
4397	25	15.92	1.34	194
4711	25	14.40	1.55	90
4758	25	14.35	1.56	78
4312	24	13.87	1.48	106
4327	24	16.70	1.79	172
4433	24	14.21	1.26	90
4476	24	15.96	1.40	108
4493	24	15.58	1.55	96
4511	24	14.61	1.65	110
4572	24	12.32	1.23	62

Cluster Statistics Summary

Metric	Max	Min	Mean
Radius (512D)	26.10	7.86	13.88
Dispersion	4.07	0.62	1.54
Num Images	220	6	69

LFW

Top 10 Identities with Highest Intrinsic Dimensions

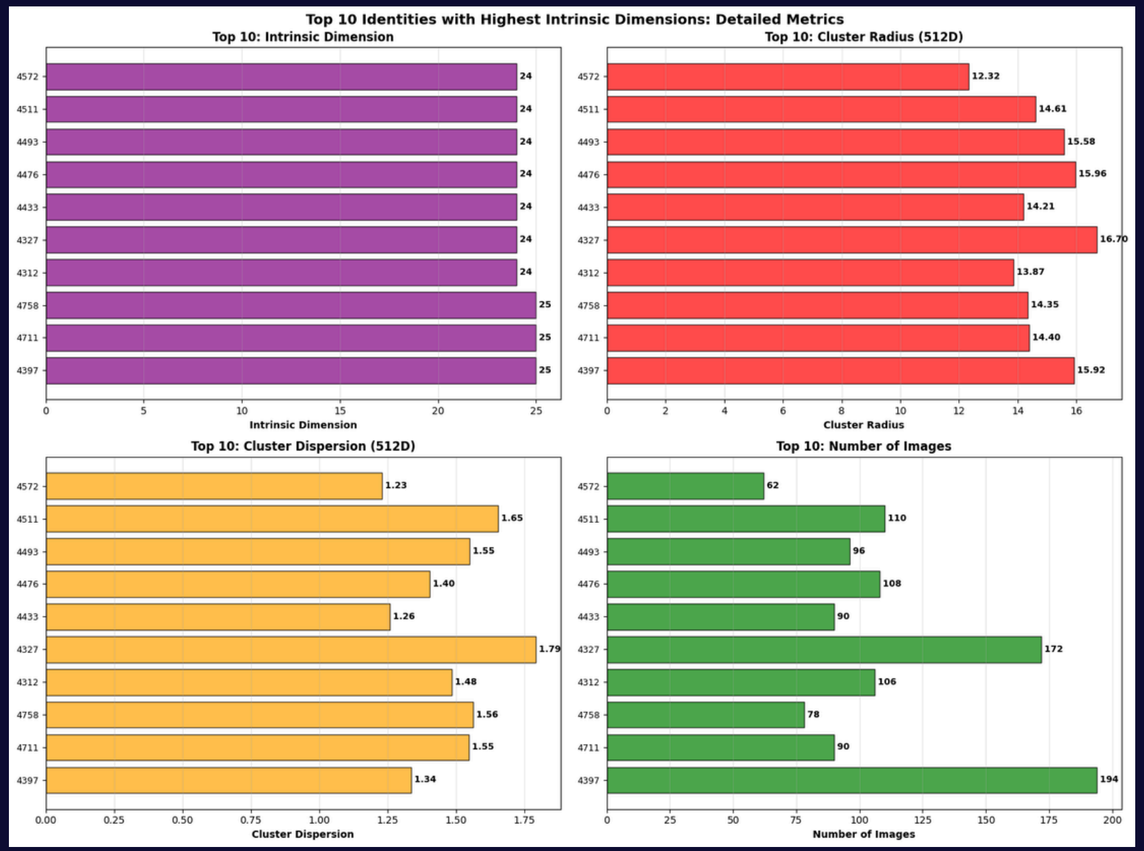
Identity	Intrinsic Dim	Radius	Dispersion	Num Images
John Ashcroft	32	17.65	1.89	53
Laura Bush	32	18.46	1.61	41
Arnold Schwarzenegger	31	21.02	1.84	42
Gloria Macapagal Arroyo	31	18.31	1.31	44
Hugo Chavez	31	18.07	1.44	71
Tom Ridge	31	17.06	1.42	33
Andre Agassi	30	17.51	1.36	36
Ariel Sharon	30	17.01	1.83	77
Jennifer Capriati	30	19.60	1.93	42
Luiz Inacio Lula da Silva	30	17.98	1.64	48

Cluster Statistics Summary

Metric	Max	Min	Mean
Radius (512D)	29.24	13.51	17.62
Dispersion	3.38	0.70	1.61
Num Images	530	10	27

FRGC Filter identities >= 100 imgs

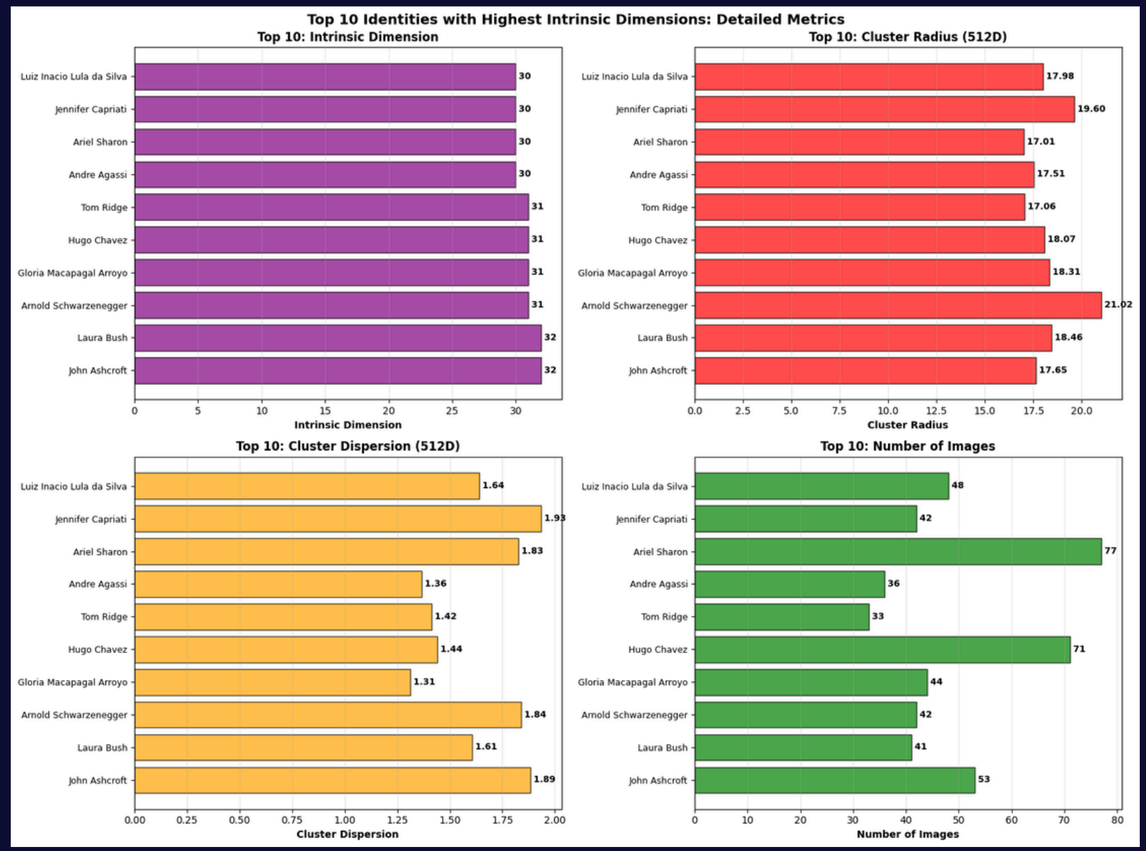
FRGC



Cluster Statistics Summary

Metric	Max	Min	Mean
Radius (512D)	26.10	7.86	13.88
Dispersion	4.07	0.62	1.54
Num Images	220	6	69

LFW



Cluster Statistics Summary

Metric	Max	Min	Mean
Radius (512D)	29.24	13.51	17.62
Dispersion	3.38	0.70	1.61
Num Images	530	10	27

Manifold distances
FRGC dataset – all imgs

Manifold Distances Paper

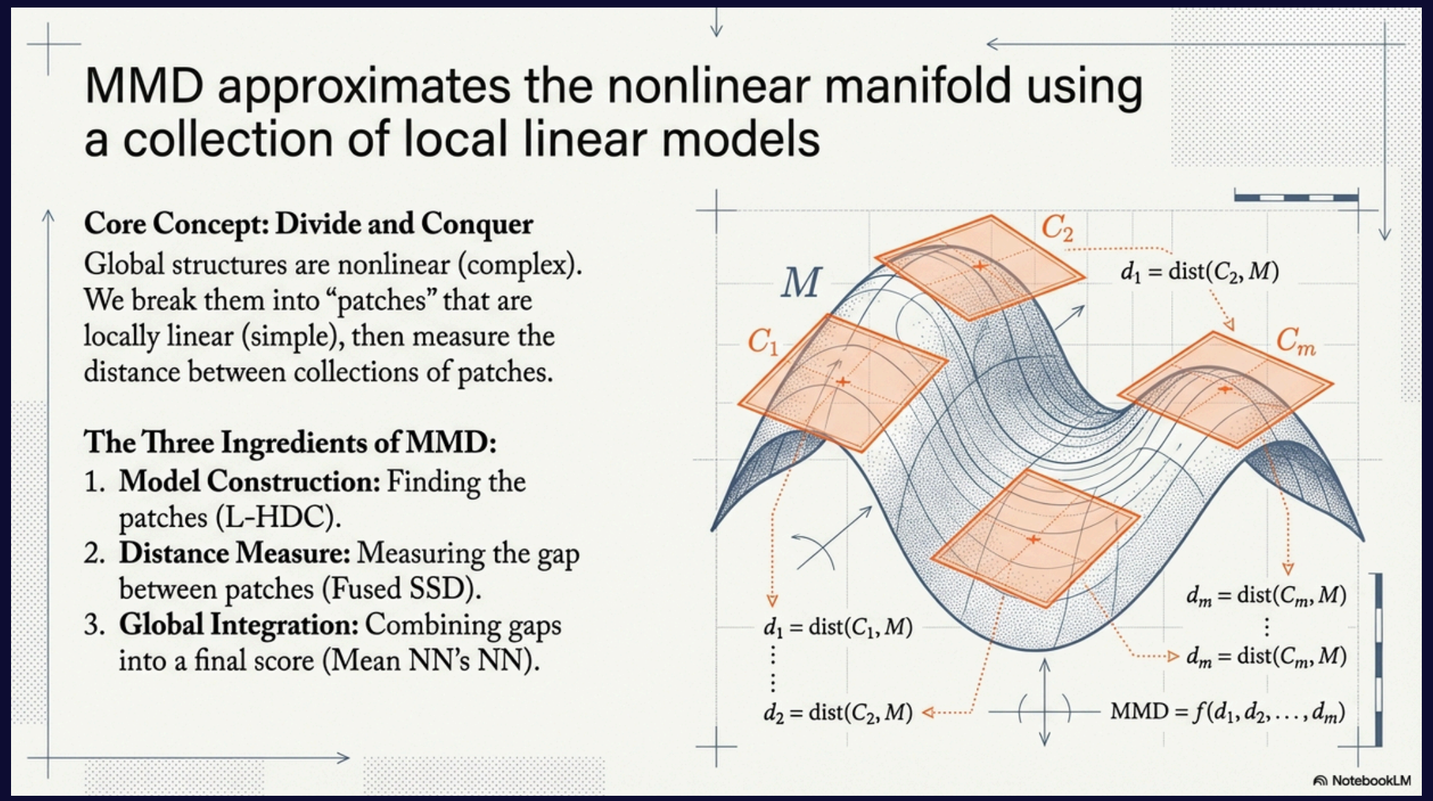
Manifold–Manifold Distance and Its Application to Face Recognition With Image Sets

Ruiping Wang, *Member, IEEE*, Shiguang Shan, *Member, IEEE*, Xilin Chen, *Senior Member, IEEE*, Qionghai Dai, *Senior Member, IEEE*, and Wen Gao, *Fellow, IEEE*

- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6226465
  - https://www.robots.ox.ac.uk/~az/lectures/ml/tenenbaum-isomap-Science2000.pdf

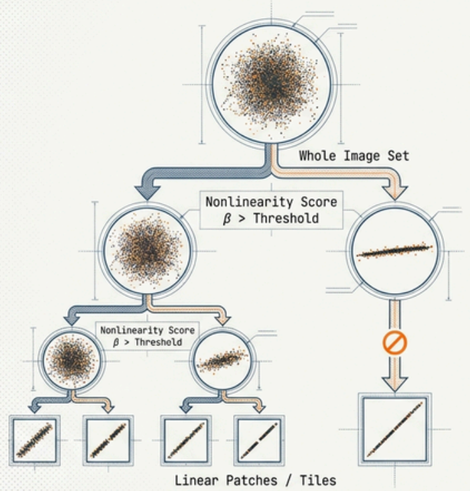
```python
import numpy as np
from typing import List, Tuple

from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import PCA
from sklearn.metrics import pairwise_distances

from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import shortest_path
```

- Each identity = a set of embeddings of a face 512D = manifold
- Manifold it approximates to many local subspaces
- Distance between 2 identities = distance combination between those subspaces



Manifold Distances Papers

Ingredient 1: Linearity-constrained Hierarchical Divisive Clustering (L-HDC)



The Challenge:
Standard clustering (K-Means) requires guessing "k" and doesn't guarantee linearity.

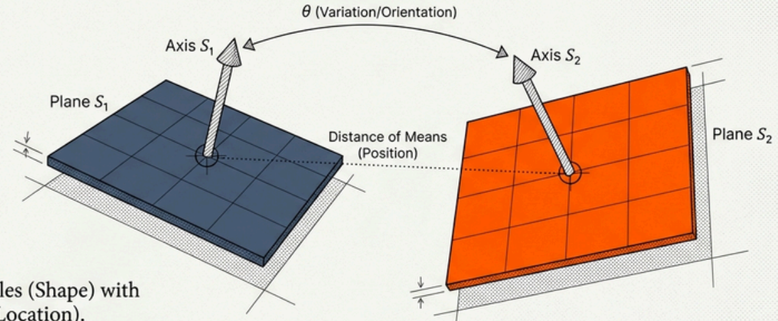
The L-HDC Solution:

1. Recursive top-down splitting.
2. Splits continue until the patch is sufficiently flat.
3. Geometry dictates structure, not parameters.

NotebookLM

MMD

Ingredient 2: A robust distance metric accounts for both orientation and position



The Innovation:
Fusing Principal Angles (Shape) with Exemplar Distance (Location).

The Formula:

$$d(S_1, S_2) = \sqrt{\sin^2\theta_0 + \frac{1}{r}\sum \sin^2\theta_k}$$

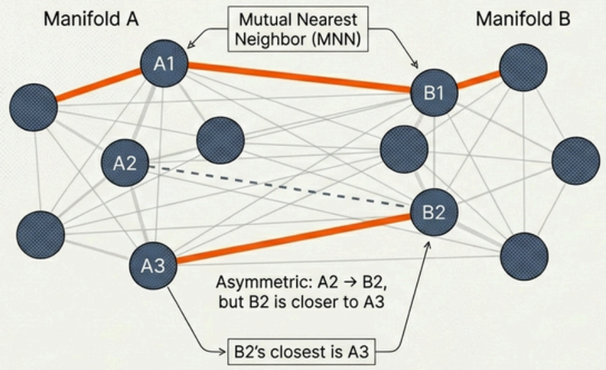
Sample Mean Distance

Variation Modes (Principal Angles)

NotebookLM

MMD

Ingredient 3: The 'Mutual Nearest Neighbor' integration strategy filters out noise



The Options Evaluated:

1. **Min NN:**
   Unstable. Too sensitive to outliers.
2. **Mean NN:**
   Generic. Ignores distribution.
3. **Mean NN's NN (The Winner):**
   Focuses on mutual nearest neighbors. It transfers weight to the "closest" common views between two video sets, ignoring the noise.

NotebookLM

Papers connection

Isomap Algorithm

Firstly, the pairwise Euclidean distance matrix $\boldsymbol{D}_E$ and geodesic distance matrix $\boldsymbol{D}_G$, based on k-NN graph, are computed as in [28]. Then a matrix holding distance ratios is obtained as: $\boldsymbol{R}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{D}_G(\boldsymbol{x}_i, \boldsymbol{x}_j)/\boldsymbol{D}_E(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Clearly, these three matrices are all of size $N \times N$. Since geodesic distance is always not smaller than Euclidean distance, $\boldsymbol{R}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 1$ holds for any entry of $\boldsymbol{R}$. Besides, another matrix $\boldsymbol{H}$ of size $k \times N$ is also constructed, each column $\boldsymbol{H}(:, j)(j = 1, \ldots, N)$ holding the indices of k nearest neighbors of the data point $\boldsymbol{x}_j$. Note that, as a byproduct of the computation of $\boldsymbol{D}_E$ and $\boldsymbol{D}_G$, the construction of $\boldsymbol{H}$ requires no extra computation. To measure the nonlinearity degree of an MLP, $\boldsymbol{X}^{(i)}$ $(i = 1, 2, \ldots, m)$, we define the following *nonlinearity score function*:

$$\beta^{(i)} = \frac{1}{N_i^2} \sum_{p=1}^{N_i} \sum_{q=1}^{N_i} \boldsymbol{R}\left(\boldsymbol{x}_p^{(i)}, \boldsymbol{x}_q^{(i)}\right). \tag{8}$$

[28] J. Tenenbaum, V. Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 22, pp. 2319–2323, Dec. 2000.

A Global Geometric Framework for Nonlinear Dimensionality Reduction

- The isomap algorithm

Table 1. The Isomap algorithm takes as input the distances $d_X(i,j)$ between all pairs i,j from N data points in the high-dimensional input space X, measured either in the standard Euclidean metric (as in Fig. 1A) or in some domain-specific metric (as in Fig. 1B). The algorithm outputs coordinate vectors $\mathbf{y}_i$ in a d-dimensional Euclidean space Y that (according to Eq. 1) best represent the intrinsic geometry of the data. The only free parameter (ϵ or K) appears in Step 1.

Step		
1	Construct neighborhood graph	Define the graph G over all data points by connecting points i and j if [as measured by $d_X(i,j)$] they are closer than ϵ (ϵ-Isomap), or if i is one of the K nearest neighbors of j (K-Isomap). Set edge lengths equal to $d_X(i,j)$.
2	Compute shortest paths	Initialize $d_G(i,j) = d_X(i,j)$ if i,j are linked by an edge; $d_G(i,j) = \infty$ otherwise. Then for each value of $k = 1, 2, \ldots, N$ in turn, replace all entries $d_G(i,j)$ by $\min(d_G(i,j), d_G(i,k) + d_G(k,j))$. The matrix of final values $D_G = \{d_G(i,j)\}$ will contain the shortest path distances between all pairs of points in G (16, 19).
3	Construct d-dimensional embedding	Let λ_p be the p-th eigenvalue (in decreasing order) of the matrix $\tau(D_G)$ (17), and v_p^i be the i-th component of the p-th eigenvector. Then set the p-th component of the d-dimensional coordinate vector $\mathbf{y}_i$ equal to $\sqrt{\lambda_p} v_p^i$.

Papers

Isomap Algorithm

DREAMS: Preserving both Local and Global Structure in Dimensionality Reduction

- why I use k=10

A Global Geometric Framework for Nonlinear Dimensionality Reduction

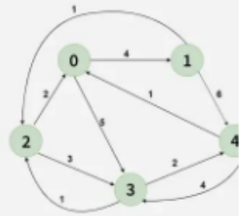
- The isomap algorithm
- Floyd warshall algoritmhm

Table 1. The Isomap algorithm takes as input the distances $d_X(i,j)$ between all pairs i,j from N data points in the high-dimensional input space X, measured either in the standard Euclidean metric (as in Fig. 1A) or in some domain-specific metric (as in Fig. 1B). The algorithm outputs coordinate vectors $\mathbf{y}_i$ in a d-dimensional Euclidean space Y that (according to Eq. 1) best represent the intrinsic geometry of the data. The only free parameter (ϵ or K) appears in Step 1.

Step		
1	Construct neighborhood graph	Define the graph G over all data points by connecting points i and j if [as measured by $d_X(i,j)$] they are closer than ϵ (ϵ-Isomap), or if i is one of the K nearest neighbors of j (K-Isomap). Set edge lengths equal to $d_X(i,j)$.
2	Compute shortest paths	Initialize $d_G(i,j) = d_X(i,j)$ if i,j are linked by an edge; $d_G(i,j) = \infty$ otherwise. Then for each value of $k = 1, 2, \ldots, N$ in turn, replace all entries $d_G(i,j)$ by $\min\{d_G(i,j), d_G(i,k) + d_G(k,j)\}$. The matrix of final values $D_G = \{d_G(i,j)\}$ will contain the shortest path distances between all pairs of points in G (16, 19).
3	Construct d-dimensional embedding	Let λ_p be the p-th eigenvalue (in decreasing order) of the matrix $\tau(D_G)$ (17), and v_p^i be the i-th component of the p-th eigenvector. Then set the p-th component of the d-dimensional coordinate vector $\mathbf{y}_i$ equal to $\sqrt{\lambda_p} v_p^i$.

KNN The k-nearest neighbor recall (KNN) is the fraction of k-nearest neighbors in the high-dimensional data that are preserved as k-nearest neighbors in the low-dimensional embedding. We used $k = 10$ throughout all experiments (for results with different choices of k, see Figure S5). The final metric is given as the average across all n data points. KNN quantifies the preservation of local structure in the embedding.

16. This procedure, known as Floyd's algorithm, requires $O(N^3)$ operations. More efficient algorithms exploiting the sparse structure of the neighborhood graph can be found in (38).



Floyd Warshall Algorithm

Your All-in-One Learning Portal: GeeksforGeeks is a comprehensive educational platform that empowers learners across domains-spanning computer science and...

GeeksforGeeks / Jul 23, 2025

Manifold–Manifold Distance and Its Application to Face Recognition With Image Sets

Isomap Algorithm

```python
#ISOMAP ALGORITHM (Paper: "Global Geometric Framework for Nonlinear Dimensionality Reduction", Tenenbaum et al. 2000)
#K-NN GRAPH( 1. Construct neightborhood graph) ----------
def build_knn_graph(X: np.ndarray, k: int = 10, metric: str = 'euclidean') -> csr_matrix:
    # X is (n,D), n points in D dimensions, k neighbors, metric for distance euclidean, csr_matrix is sparse matrix
    #use k=10 because it is a good balance between local and global structure (DREAMS paper)
    n = X.shape[0]

    #'i is one of the K nearest neighbors of j (K-Isomap)' - found k neighbors more close to each point
    neighbors = NearestNeighbors(n_neighbors=min(k + 1, n), metric=metric).fit(X)
    distances, indexs = neighbors.kneighbors(X)
    #take out 1st neighbor. because it is the point itself (distance 0)
    distances = distances[:, 1:]                        #(n, k)
    indexs = indexs[:, 1:]                              #(n, k)
    #matrix of connectivity G, row origin to column destination neighbors, data is the distance between them
    rows = np.repeat(np.arange(n), indexs.shape[1])     # arrange(n) create each point index, indexs.shape[1] each point have 2 neighbors, repeat create row for each neighbor
    cols = indexs.flatten()                             # flatten create column for each neighbor

    #'set edge lengths equal to dX(i,j)'
    data = distances.flatten()
    G = csr_matrix((data, (rows, cols)), shape=(n, n))  # G[i,j] = distance if j is neighbor of i, 0 otherwise
    G =(G + G.T)/2                                      #symmetric matrix to ensure undirected graph, is the average of both directions so they have same weight
    return G

#GEODESIC DISTANCES ( 2. Compute shortest paths) ----------
def geodesic_distance_matrix(X: np.ndarray, k: int = 10) -> np.ndarray:
    G = build_knn_graph(X, k=k)                         #'Initialize dG(i,j) = dX(i,j) if linked'
    D = shortest_path(G, method='FW', directed=False)   #'Then for each value of k =1,2,...,N in turn, replace all entries dG(i, j) by min{dG(i, j),dG(i,k)+dG(k, j)}.' - this is Floyd-Warshall algorithm(
    return D                                            #'shortest path distances between all pairs of points in G'

# Step 3 it is not necessary for our purposes (embedding in lower dimension)
#----------
```

Papers connection

Beta No Linear

as in [28]. Then a matrix holding distance ratios is obtained as: $\boldsymbol{R}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{D}_G(\boldsymbol{x}_i, \boldsymbol{x}_j)/\boldsymbol{D}_E(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Clearly, these three matrices are all of size $N \times N$. Since geodesic distance is always not smaller than Euclidean distance, $\boldsymbol{R}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 1$ holds for any entry of $\boldsymbol{R}$. Besides, another matrix $\boldsymbol{H}$ of size $k \times N$ is also constructed, each column $\boldsymbol{H}(:, j)(j = 1, \ldots, N)$ holding the indices of k nearest neighbors of the data point $\boldsymbol{x}_j$. Note that, as a byproduct of the computation of $\boldsymbol{D}_E$ and $\boldsymbol{D}_G$, the construction of $\boldsymbol{H}$ requires no extra computation. To measure the nonlinearity degree of an MLP, $\boldsymbol{X}^{(i)}$ $(i = 1, 2, \ldots, m)$, we define the following *nonlinearity score function*:

$$\beta^{(i)} = \frac{1}{N_i^2} \sum_{p=1}^{N_i} \sum_{q=1}^{N_i} \boldsymbol{R}\left(\boldsymbol{x}_p^{(i)}, \boldsymbol{x}_q^{(i)}\right). \tag{8}$$

```python
#BETA no linear
def estimate_beta(X: np.ndarray, k: int = 10) -> float:
    n = X.shape[0]
    if n < 2: # there is no, no linear structure
        return 1.0
    #'euclidean pairwise distances D_E'=||x_p - x_q||
    De = pairwise_distances(X, metric='euclidean')
    #'geodesic distance matrix D_G'
    Dg = geodesic_distance_matrix(X, k=k)

    # use upper triangle pairs where De > EPS and Dg fin
    valid = (De > EPS) & np.isfinite(Dg) #avoid division
    if not np.any(valid):
        return 1.0

    #B^(i) = (1/N_i^2) * sum(R(x_p, x_q))= mean(R(x_p^(i
    ratios = (Dg[valid] / (De[valid] + EPS))
    return float(np.mean(ratios))
# B=1, lineal cluster
# B>1, no lineal cluster, B=1.5 Shape of "S" cluster, B=
```

Papers connection

Algorithm 1: L-HDC

Algorithm 1 Linearity-Constrained HDC (L-HDC)

1 Initialization: $\boldsymbol{X}^{(1)} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, $m = 1$.
Compute the nonlinearity score $\beta^{(1)}$ according to (8).

2 Choose $\boldsymbol{X}^{(i)}$ ($i \in \{1, 2, \ldots, m\}$) with the largest score $\beta^{(i)}$ as the *parent* cluster. Split $\boldsymbol{X}^{(i)}$ as follows:

2.1 According to geodesic distance $\mathbf{D}_G$, select two furthest seed points, $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ from $\boldsymbol{X}^{(i)}$.

2.2 Initialize two *child* clusters: $\boldsymbol{X}_l^{(i)} = \{\boldsymbol{x}_l\}$, $\boldsymbol{X}_r^{(i)} = \{\boldsymbol{x}_r\}$. Update: $\boldsymbol{X}^{(i)} \leftarrow \boldsymbol{X}^{(i)} \backslash \{\boldsymbol{x}_l, \boldsymbol{x}_r\}$.

2.3 while ($\boldsymbol{X}^{(i)} \neq \emptyset$) do

2.3.1 For current $\boldsymbol{X}_l^{(i)}$, construct its neighbor points set, denote by P_l. According to $\boldsymbol{H}$, P_l gathers the k-NN samples of all the points in $\boldsymbol{X}_l^{(i)}$.

2.3.2 For current $\boldsymbol{X}_r^{(i)}$, construct its neighbor points set P_r in the similar way to step 2.3.1.

2.3.3 Sequentially update:

$$\boldsymbol{X}_l^{(i)} \leftarrow \boldsymbol{X}_l^{(i)} \cup (P_l \cap \boldsymbol{X}^{(i)}),\ \boldsymbol{X}^{(i)} \leftarrow \boldsymbol{X}^{(i)} \backslash (P_l \cap \boldsymbol{X}^{(i)});$$
$$\boldsymbol{X}_r^{(i)} \leftarrow \boldsymbol{X}_r^{(i)} \cup (P_r \cap \boldsymbol{X}^{(i)}),\ \boldsymbol{X}^{(i)} \leftarrow \boldsymbol{X}^{(i)} \backslash (P_r \cap \boldsymbol{X}^{(i)}).$$

2.4 $\boldsymbol{X}^{(i)}$ is split into two smaller ones: $\boldsymbol{X}_l^{(i)}$ and $\boldsymbol{X}_r^{(i)}$. Update: $m \leftarrow m + 1$, compute $\beta_l^{(i)}$ and $\beta_r^{(i)}$.

3 The splitting procedure continues until the nonlinearity score $\beta^{(i)}$ in step 2 is less than a *threshold* δ.

```python
#ALGORITHM 1, Linearity-Constrained HDC (L-HDC)
def recursive_split(X: np.ndarray, delta_thresh: float = 1.1, k: int = 10) -> List[np.ndarray]:
    # delta_thresh: threshold for beta to stop splitting = 1.1 (slightly non-linear it starts f

    n = X.shape[0]

    # "1. Initialization: X(1) = {x1, x2, ..., xN}, m = 1"
    X_full_indices = np.arange(n)
    # "1. ....Compute the nonlinearity score B(1) according to (8)"
    beta_full = estimate_beta(X, k=k)
    queue = [(X_full_indices, beta_full)] #list of (cluster_indices, beta_score)
    final_patches = []

    #matrix H used in 2.3.1, 2.3.2 -> H: k×N, each column contains k nearest neighbor indices f
    neighbors = NearestNeighbors(n_neighbors=min(k + 1, n), metric='euclidean').fit(X) #neighbo
    _, H = neighbors.kneighbors(X)  #get k+1 neighbors (including self) (n,k+1)
    H = H[:, 1:]                    #discard 1st column (auto neighbor)"..,each column H(:, j)
    H = H.T                         #transpose to (k, n): so each column j, H[:, j] = k neighbo


    while queue:
        # "2. Choose X(i) with the largest score B(i) as the parent cluster. Split X(i) as follo
        idx_max = np.argmax([beta for _, beta in queue])
        X_i_indices, beta_i = queue.pop(idx_max)

        # "3. The splitting procedure continues until the nonlinearity score B(i) < threshold D
        if beta_i < delta_thresh: # If cluster too small or sufficiently linear, accept as fina
            final_patches.append(X_i_indices)
            continue

        # "2.1. According to geodesic distance Dg, select two furthest seed points x1 and xr"
        X_sub = X[X_i_indices]
        Dg = geodesic_distance_matrix(X_sub, k=k) #geodesic distance matrix for this cluster
        max_dist = -1 #2 point with max geodesic distance (two furthest points = diameter of ma
        seed_l_local = 0
        seed_r_local = 0
        for i in range(len(X_i_indices)):
            for j in range(i + 1, len(X_i_indices)):
                if np.isfinite(Dg[i, j]) and Dg[i, j] > max_dist: #isfinite to avoid disconnect
                    max_dist = Dg[i, j]
                    seed_l_local = i
                    seed_r_local = j
        seed_l_global = X_i_indices[seed_l_local] #local to global indices
        seed_r_global = X_i_indices[seed_r_local]
```

..... it goes on the code

Papers connection

Patch representation

The extracted clusters (i.e., $\boldsymbol{X}^{(i)}$'s) are then represented by linear subspaces to obtain the final local models. Principal component analysis (PCA) is employed for its simplicity and efficiency. For each local model C_i, we denote its sample mean (i.e., exemplar) by $\boldsymbol{e}_i$ and corresponding principal component matrix by $\boldsymbol{P}_i \in \mathrm{R}^{D \times d_i}$ that is computed as the leading eigenvectors of the covariance matrix and forms a set of orthonormal basis of the subspace. Here d_i denotes the PCA subspace dimension. Since the subspace (or local model) is spanned by a set of samples, $\boldsymbol{e}_i$ and $\boldsymbol{P}_i$ play different roles to jointly describe the local model: the former characterizes the data sample itself, and the latter characterizes the data variation modes.

```python
#PATCH REPRESENTATION (e_i, P_i) via PCA p. 4470

def compute_patch_representation(X: np.ndarray, indices: np.ndarray, var_threshold: float = 0.95) -> Tuple[np.ndarray, np.ndarray]:
    #var_threshold: 95% so it capture the 1st d eigenvectors that explain 95% of variance

    Xp = X[indices]                         # extract patch samples
    # "For each local model Ci, we denote its sample mean (i.e., exemplar) by ei"
    e = np.mean(Xp, axis=0)
    e_norm = e / (np.linalg.norm(e) + EPS)  # normalize to unit vector
    Xc = Xp - e                             # centered data, xp is the samples in the patch - e is the mean of the patch

    if Xc.shape[0] <= 1:                    # not enough samples to compute PCA
        P = np.zeros((X.shape[1], 0))       # empty principal component matrix
        return e_norm, P

    # "and corresponding principal component matrix by Pi ∈ R^D×di that is computed as the leading eigenvectors of the covariance mat
    pca = PCA(n_components=min(Xc.shape[0], Xc.shape[1]))  #xc.shape[0] number of samples in patch, Xc.shape[1] dimension of data / m
    pca.fit(Xc)                                            # fit PCA on centered data, has the eigenvectors(PC) and eigenvalues of X
    cumvar = np.cumsum(pca.explained_variance_ratio_)      # cumulative variance explained by each principal component
    d = int(np.searchsorted(cumvar, var_threshold) + 1)    # number of components to reach var_threshold, +1 to have the count of co
    d = min(d, pca.components_.shape[0])                   # ensure d does not exceed available components
    P = pca.components_[:d].T                              # PC as columns (D, d)
    return e_norm, P                                       # "subspace (or local model) is spanned by a set of samples, ei(describe
```

Papers connection

SSD Distance between 2 patches - Local Model Distance measure

B. Local Model Distance Measure

With the local models constructed above, we can use SSD to measure their distance. Intuitively, a reasonable and complete SSD should take into account both the principal axes $\boldsymbol{P}_i$ and the sample mean $\boldsymbol{e}_i$. As shown in Fig. 5, $\boldsymbol{e}_i$ tells the position of the subspace located in the global observation data space, and $\boldsymbol{P}_i$ tells the spanning directions of the subspace. However, the most commonly exploited SSD, i.e. principal angles, is only

To calculate the principal angles, a numerically stable algorithm proposed in [18] is based on Singular Value Decomposition (SVD). In the method, the SVD of $\boldsymbol{P}_1^T\boldsymbol{P}_2$ is first computed as follows:

$$\boldsymbol{P}_1^T\boldsymbol{P}_2 = \boldsymbol{Q}_1\boldsymbol{\Lambda}\boldsymbol{Q}_2^T \tag{10}$$

where $\boldsymbol{\Lambda} = diag(\sigma_1, \ldots, \sigma_r)$, $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are two orthogonal matrices. The singular values $\sigma_1, \ldots, \sigma_r$ are just the cosines of the principal angles, i.e. the so-called canonical correlations:

$$\cos\theta_k = \sigma_k, \qquad k = 1, 2, \ldots, r. \tag{11}$$

The associated canonical vectors are $\boldsymbol{U} = \boldsymbol{P}_1\boldsymbol{Q}_1 = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{d_1}]$ and $\boldsymbol{V} = \boldsymbol{P}_2\boldsymbol{Q}_2 = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{d_2}]$, which are obtained by aligning the two principal axes $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ respectively through an orthogonal transformation, as shown in Fig. 5. One may find that a previous work [29] has also employed PCA based local models and then globally aligned the local PCA subspaces. However, their alignment is for single manifold dimensionality reduction, while ours is for comparing a pair of local models from two manifolds.

2) Previous Work on SSD: Based on principal angles, various subspace distances have been defined in the literature. For example, in the pioneering study named Mutual Subspace Method (MSM) [14], only the smallest principal angle θ_1 is used to define a distance called *Max Correlation* as follows:

$$d_{Max}(S_1, S_2) = (1 - \cos^2\theta_1)^{1/2} = \sin\theta_1. \tag{12}$$

In contrast, another distance called *Min Correlation* is similarly defined using the largest principal angle θ_r:

$$d_{Min}(S_1, S_2) = (1 - \cos^2\theta_r)^{1/2} = \sin\theta_r. \tag{13}$$

Both above distances depend highly on the probability distribution of the principal angles and are effective only in some specific cases respectively, as noted in [15]. Consequently, another distance called *Projection metric*, which uses all the principal angles, was proposed as follows [20]:

$$d_P(S_1, S_2) = \left(\sum_{k=1}^{r}\sin^2\theta_k\right)^{1/2} = \left(r - \sum_{k=1}^{r}\cos^2\theta_k\right)^{1/2}. \tag{14}$$

recognition [30]. As shown in Fig. 5, the correlation of the two exemplars $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, is the cosine of their angle θ_0. We then define an *exemplar distance measure*:

$$d_E(S_1, S_2) = (1 - \cos^2\theta_0)^{1/2} = \sin\theta_0,$$
$$\text{where } \cos\theta_0 = \boldsymbol{e}_1^T\boldsymbol{e}_2/||\boldsymbol{e}_1|| \cdot ||\boldsymbol{e}_2||. \tag{15}$$

With the two distance measures in (14) and (15), we can fuse them in a seamless manner and reach the following *formal definition of SSD*:

$$d(S_1, S_2) = \left(\sin^2\theta_0 + \frac{1}{r}\sum_{k=1}^{r}\sin^2\theta_k\right)^{1/2}$$
$$= \left(2 - \cos^2\theta_0 - \frac{1}{r}\sum_{k=1}^{r}\cos^2\theta_k\right)^{1/2}. \tag{16}$$

```python
#SSD DISTANCE BETWEEN TWO PATCHES - B) Local Model Distance Measure (p.4470-4471)
def ssd_distance(patchA: Tuple[np.ndarray, np.ndarray], patchB: Tuple[np.ndarray, np.ndarray]) -> float:
    #SSD: Subspace-to-Subspace Distance between two patches, eq16
    #patchA, patchB: (e, P) where e is normalized exemplar, P is principal component matrix, returns d_ssd (floa

    eA, PA = patchA
    eB, PB = patchB

    # EXEMPLAR DISTANCE MEASURE(Eq. 15, p.4471):
    cos0 = float(np.dot(eA, eB))            #||eA||= ||eB|| = 1, already normalized
    sin2_0 = 1.0 - (cos0**2)

    #if either subspace has no principal components (d=0)
    if PA.shape[1] == 0 or PB.shape[1] == 0:
        return float(np.sqrt(sin2_0))       # d_ssd = sin θ (exemplar distance only)

    # VARIATION DISTANCE via Principal Angles (Eq. 10, p.4470):
    M = PA.T @ PB

    U, S, Vt = np.linalg.svd(M, full_matrices=False) #SVD of M, S contains singular values σ

    sigmas = np.clip(S, -1.0, 1.0)          # (Eq. 11) σ = cos θ,-1-1 because it is cosine values
    sin2 = 1.0 - (sigmas ** 2)              # (Eq. 13) Min correlation
    r = min(PA.shape[1], PB.shape[1])       # r is min dimension of the two subspaces (write after Eq. 9) here w

    # PROJECTION METRIC (Eq. 14, p.4471):
    sin2_mean = float(np.sum(sin2[:r]) / (r + EPS))

    # FORMAL DEFINITION OF SSD (Eq. 16, p.4471):
    # This fuses exemplar distance (sin2_0) and variation distance (sin2_mean)
    d_ssd = float(np.sqrt(sin2_0 + sin2_mean))
    return d_ssd
```

Papers connection

SSD Distance between 2 patches - Local Model Distance measure

B. Local Model Distance Measure

With the local models constructed above, we can use SSD to measure their distance. Intuitively, a reasonable and complete SSD should take into account both the principal axes $\boldsymbol{P}_i$ and the sample mean $\boldsymbol{e}_i$. As shown in Fig. 5, $\boldsymbol{e}_i$ tells the position of the subspace located in the global observation data space, and $\boldsymbol{P}_i$ tells the spanning directions of the subspace. However, the most commonly exploited SSD, i.e. principal angles, is only

To calculate the principal angles, a numerically stable algorithm proposed in [18] is based on Singular Value Decomposition (SVD). In the method, the SVD of $\boldsymbol{P}_1^T\boldsymbol{P}_2$ is first computed as follows:

$$\boldsymbol{P}_1^T\boldsymbol{P}_2 = \boldsymbol{Q}_1\boldsymbol{\Lambda}\boldsymbol{Q}_2^T \tag{10}$$

where $\boldsymbol{\Lambda} = diag(\sigma_1, \ldots, \sigma_r)$, $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are two orthogonal matrices. The singular values $\sigma_1, \ldots, \sigma_r$ are just the cosines of the principal angles, i.e. the so-called canonical correlations:

$$\cos\theta_k = \sigma_k, \qquad k = 1, 2, \ldots, r. \tag{11}$$

The associated canonical vectors are $\boldsymbol{U} = \boldsymbol{P}_1\boldsymbol{Q}_1 = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{d_1}]$ and $\boldsymbol{V} = \boldsymbol{P}_2\boldsymbol{Q}_2 = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{d_2}]$, which are obtained by aligning the two principal axes $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ respectively through an orthogonal transformation, as shown in Fig. 5. One may find that a previous work [29] has also employed PCA based local models and then globally aligned the local PCA subspaces. However, their alignment is for single manifold dimensionality reduction, while ours is for comparing a pair of local models from two manifolds.

2) Previous Work on SSD: Based on principal angles, various subspace distances have been defined in the literature. For example, in the pioneering study named Mutual Subspace Method (MSM) [14], only the smallest principal angle θ_1 is used to define a distance called *Max Correlation* as follows:

$$d_{Max}(S_1, S_2) = (1 - \cos^2\theta_1)^{1/2} = \sin\theta_1. \tag{12}$$

In contrast, another distance called *Min Correlation* is similarly defined using the largest principal angle θ_r:

$$d_{Min}(S_1, S_2) = (1 - \cos^2\theta_r)^{1/2} = \sin\theta_r. \tag{13}$$

Both above distances depend highly on the probability distribution of the principal angles and are effective only in some specific cases respectively, as noted in [15]. Consequently, another distance called *Projection metric*, which uses all the principal angles, was proposed as follows [20]:

$$d_P(S_1, S_2) = \left(\sum_{k=1}^{r}\sin^2\theta_k\right)^{1/2} = \left(r - \sum_{k=1}^{r}\cos^2\theta_k\right)^{1/2}. \tag{14}$$

recognition [30]. As shown in Fig. 5, the correlation of the two exemplars $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, is the cosine of their angle θ_0. We then define an *exemplar distance measure*:

$$d_E(S_1, S_2) = (1 - \cos^2\theta_0)^{1/2} = \sin\theta_0,$$
$$\text{where } \cos\theta_0 = \boldsymbol{e}_1^T\boldsymbol{e}_2/||\boldsymbol{e}_1|| \cdot ||\boldsymbol{e}_2||. \tag{15}$$

With the two distance measures in (14) and (15), we can fuse them in a seamless manner and reach the following *formal definition of SSD*:

$$d(S_1, S_2) = \left(\sin^2\theta_0 + \frac{1}{r}\sum_{k=1}^{r}\sin^2\theta_k\right)^{1/2}$$
$$= \left(2 - \cos^2\theta_0 - \frac{1}{r}\sum_{k=1}^{r}\cos^2\theta_k\right)^{1/2}. \tag{16}$$

```python
#SSD DISTANCE BETWEEN TWO PATCHES - B) Local Model Distance Measure (p.4470-4471)
def ssd_distance(patchA: Tuple[np.ndarray, np.ndarray], patchB: Tuple[np.ndarray, np.ndarray]) -> float:
    #SSD: Subspace-to-Subspace Distance between two patches, eq16
    #patchA, patchB: (e, P) where e is normalized exemplar, P is principal component matrix, returns d_ssd (floa

    eA, PA = patchA
    eB, PB = patchB

    # EXEMPLAR DISTANCE MEASURE(Eq. 15, p.4471):
    cos0 = float(np.dot(eA, eB))            #||eA||= ||eB|| = 1, already normalized
    sin2_0 = 1.0 - (cos0**2)

    #if either subspace has no principal components (d=0)
    if PA.shape[1] == 0 or PB.shape[1] == 0:
        return float(np.sqrt(sin2_0))       # d_ssd = sin θ (exemplar distance only)

    # VARIATION DISTANCE via Principal Angles (Eq. 10, p.4470):
    M = PA.T @ PB

    U, S, Vt = np.linalg.svd(M, full_matrices=False) #SVD of M, S contains singular values σ

    sigmas = np.clip(S, -1.0, 1.0)          # (Eq. 11) σ = cos θ,-1-1 because it is cosine values
    sin2 = 1.0 - (sigmas ** 2)              # (Eq. 13) Min correlation
    r = min(PA.shape[1], PB.shape[1])       # r is min dimension of the two subspaces (write after Eq. 9) here w

    # PROJECTION METRIC (Eq. 14, p.4471):
    sin2_mean = float(np.sum(sin2[:r]) / (r + EPS))

    # FORMAL DEFINITION OF SSD (Eq. 16, p.4471):
    # This fuses exemplar distance (sin2_0) and variation distance (sin2_mean)
    d_ssd = float(np.sqrt(sin2_0 + sin2_mean))
    return d_ssd
```

Papers connection

Build the patch set and matrix

```python
#-----------------------------------------------------------------------------------------------------------------------

#BUILD PATCH SET
def build_patch_set(X: np.ndarray, indices_list: List[np.ndarray], var_threshold: float = 0.95) -> List[Tuple[np.ndarray, np.ndarray
    #X is (N,D), indices_list is list of arrays of integer indices into X, returns list of patches (e,P)
    #var_threshold: PCA variance threshold
    patches = []
    for inds in indices_list:
        e, P = compute_patch_representation(X, inds, var_threshold=var_threshold)
        patches.append((e, P))
    return patches

#-----------------------------------------------------------------------------------------------------------------------

#BUILD DISTANCE MATRIX BETWEEN TWO PATCH SETS
def build_distance_matrix(patchesA: List[Tuple[np.ndarray, np.ndarray]], patchesB: List[Tuple[np.ndarray, np.ndarray]]) -> np.ndarra
    m = len(patchesA)                   #m is number of patches in A
    n = len(patchesB)                   #n is number of patches in B
    D = np.zeros((m, n), dtype=float)   #distance matrix (m x n)
    for i in range(m):
        for j in range(n):
            D[i, j] = ssd_distance(patchesA[i], patchesB[j]) #compute d_ssd between patch i in A and patch j in B
    return D

#-----------------------------------------------------------------------------------------------------------------------
```

Papers connection

Option 3: Mean N4, NN's NN

3) Option-3: Mean NN's NN (N^4): A more general intuition is that the *smaller* the distance of the pair is, the *larger* its weight should be. This motivates our third option to transfer some weights from the further pairs to those closer ones. Fig. 7(c) demonstrates the idea.

Specifically, for each C_i $(i = 1, 2, \ldots, m)$ in $\mathcal{M}_1$, we find its NN $C'_{N(i)}$ in $\mathcal{M}_2$. Then for $C'_{N(i)}$, we find inversely its NN $C_{N'(N(i))}$ in $\mathcal{M}_1$. Here, we call $C_{N'(N(i))}$ as the NN's NN (N^4) of C_i. By replacing the term $(C_i, C'_{N(i)})$ in (20) with $(C_{N'(N(i))}, C'_{N(i)})$, we then transfer the weight of the former pair to the latter one. In the same manner, for each C'_j $(j = 1, 2, \ldots, n)$ in $\mathcal{M}_2$, we can find its NN $C_{N'(j)}$ and N^4 $C'_{N(N'(j))}$ respectively. Then the term $(C_{N'(j)}, C'_j)$ in (20) is replaced with $(C_{N'(j)}, C'_{N(N'(j))})$. Finally, we derive a more reasonable definition of MMD in the following:

$$d_3(\mathcal{M}_1, \mathcal{M}_2) = \frac{1}{m+n}\left(\sum_{i=1}^{m} d\left(C_{N'(N(i))}, C'_{N(i)}\right) + \sum_{j=1}^{n} d\left(C_{N'(j)}, C'_{N(N'(j))}\right)\right). \quad (21)$$

For more intuitive illustration, let us see the two manifolds in Fig. 7(c). For the local model C_1 (i.e., $i = 1$), its NN and N^4 are $C'_{N(1)} = C'_2$ and $C_{N'(N(1))} = C_2$ respectively. Therefore, the weight of the further pair (C_1, C'_2) (*red dashed arrow* in the figure) will be transferred to the closer pair (C_2, C'_2) (*red solid arrow*). Likewise, for the local model C'_1 (i.e., $j = 1$), the weight of (C_2, C'_1) (*blue dashed arrow*) will also be transferred to (C_2, C'_2) (*blue solid arrow*).

We can see that option-3 combines the merits of option-1 and 2. Compared with option-1, it can guarantee more stable results by using information from more data. Compared with option-2, it can adaptively adjust the weights on different NN pairs in accordance with the real data characteristics more reliably.

```python
#Option 3: Mean N^4 (NN's NN) - (Eq. 21)
# "the smaller the distance of the pair is, the larger its weight should be" - transfer
def compute_mmd_from_distance_matrix(D: np.ndarray) -> float:
    m, n = D.shape
    if m == 0 or n == 0:
        return 0.0

    distances = []
    #NN's NN = go to the neighbor more close in the other C and come back, this way cho
    #For each C_i in M1, transfer weight via NN's NN in M1
    for i in range(m):
        # C'_N(i): Find NN of C_i in M2
        j_N_i = int(np.argmin(D[i, :]))
        # C_N'(N(i)): Find NN of C'_N(i) back in M1 (the NN's NN)
        i_N_prime_N_i = int(np.argmin(D[:, j_N_i]))
        # Use distance d(C_N'(N(i)), C'_N(i)) - transfers weight to closer pair
        distances.append(float(D[i_N_prime_N_i, j_N_i]))
    #For each C'_j in M2, transfer weight via NN's NN in M2
    for j in range(n):
        # C_N'(j): Find NN of C'_j in M1
        i_N_prime_j = int(np.argmin(D[:, j]))
        # C'_N(N'(j)): Find NN of C_N'(j) back in M2 (the NN's NN)
        j_N_N_prime_j = int(np.argmin(D[i_N_prime_j, :]))
        # Use distance d(C_N'(j), C'_N(N'(j))) - transfers weight to closer pair
        distances.append(float(D[i_N_prime_j, j_N_N_prime_j]))

    #final MMD as mean distance across all pairs
    return float(np.mean(distances)) if distances else 0.0
```

Manifold distances
FRGC dataset - all imgs

RESULTS

MMD

en que unidad estan mmd distances

```
MMD between 10011 person_id pairs...
Completed: 10011/10011 pairs computed successfully
```

```
──────────────────────────────────────────────────────────────────────
MMD Distance: min=1.2532, max=1.3816, mean=1.3454
Num Patches: A=2.9±1.0, B=2.8±0.9
Avg Patch Dim: A=36.2±8.7, B=34.8±9.0
Num Matches: mean=1.7 (min=1, max=5)
──────────────────────────────────────────────────────────────────────
```

MMD

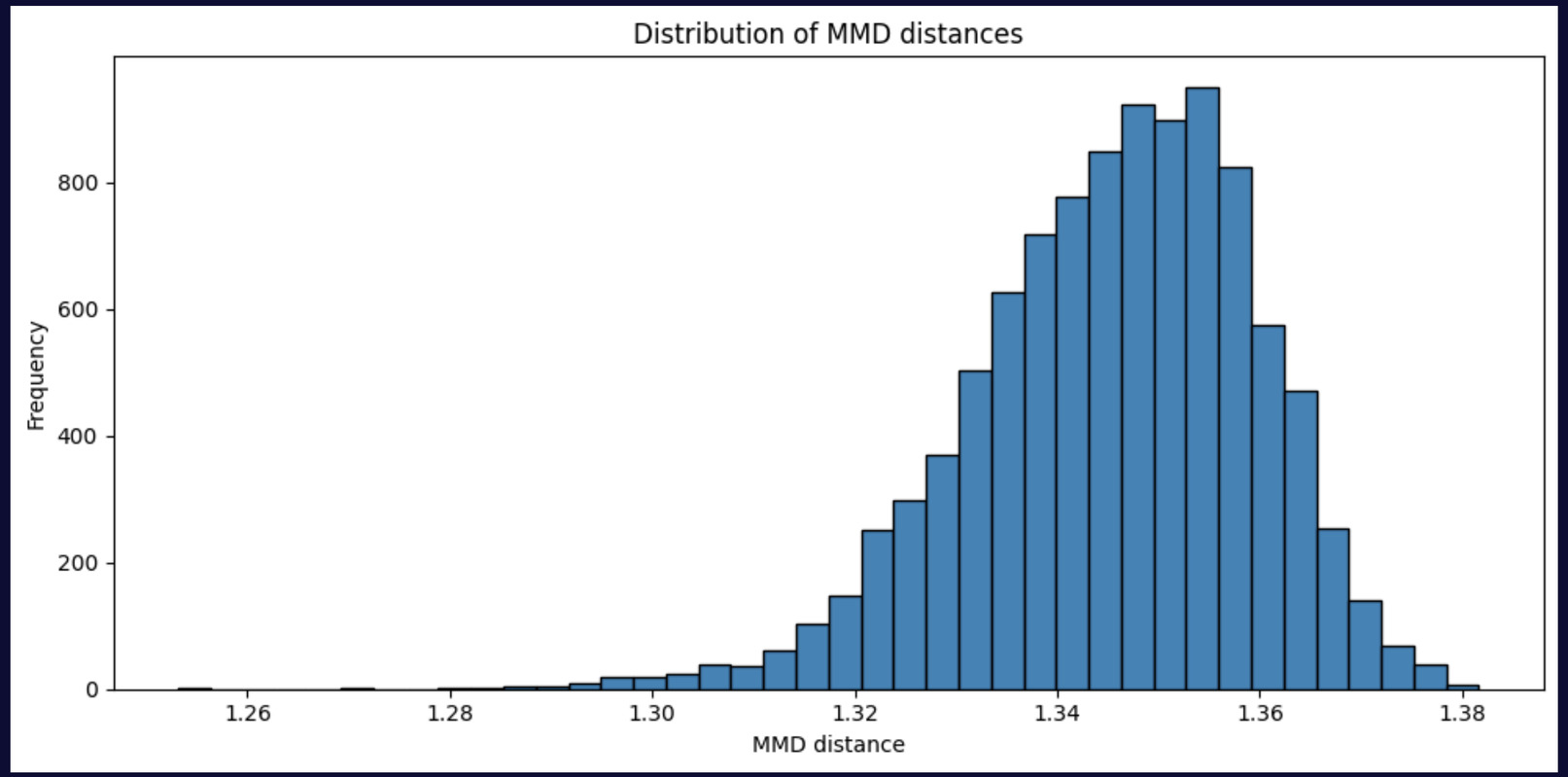
Top 5 closest pairs (smallest MMD):

person_id_A	person_id_B	mmd_distance	num_patches_A	num_patches_B
4217	4557	1.253165	2	3
4221	4633	1.269423	2	3
4221	4559	1.270339	2	2
4575	4708	1.271080	4	1
4429	4575	1.278865	2	4

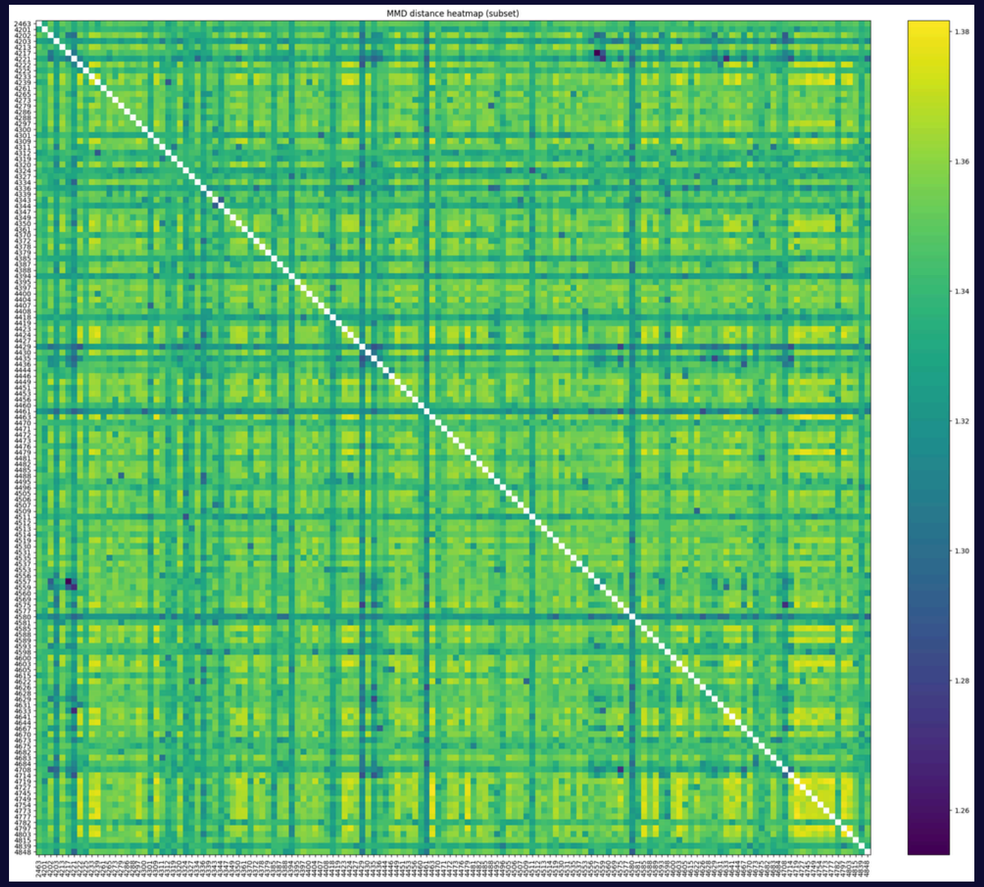
Top 5 farthest pairs (largest MMD):

person_id_A	person_id_B	mmd_distance	num_patches_A	num_patches_B
4463	4745	1.381637	5	3
4463	4727	1.380766	5	3
4719	4777	1.380693	3	3
4797	4803	1.378766	3	4
4719	4773	1.378731	3	3

MMD


Distribution of MMD distances
Frequency
0
200
400
600
800
1.26
1.28
1.30
1.32
1.34
1.36
1.38
MMD distance

MMD



Future Work

- Organize better documentation in LaTEX
- Distribution between manifold
  - dimensionality estimation, distribution angles, angles between subpaces
  - same intrinsic dimension, compare angle distances
  - filter: number of img K, whose intrinsic dimension is K-1
- Distribution embeddings
  - geometru structure
- Compare to face networks
  - Arcface, MagFace, AdaFace, Facenet, Dlib

Thank You