# Evaluation Report for ML Project

## Step 1: Problem Formulation

- Clarifying Questions: Our project aims to develop a vector-based face recognition system using deep learning and computer vision techniques.
- Use Case and Business Goal: The primary use case is to search and retrieve similar images from a database. This can be used as a part of a fandom webpage (who do you look like in ____ fandom?).
- Requirements: Efficient image retrieval with high accuracy and low latency.
- Scope and Scale: The dataset contains 202,599 images. The scope includes feature extraction, vector database creation, and similarity search.
- Performance Constraints: The system must handle large datasets and provide quick retrieval times.
- Data Sources and Availability: CelebA dataset.
- Assumptions: The image dataset is labeled and preprocessed. There is access to necessary hardware for training and inference.
- ML Objective: The ML task is similarity search using image embeddings.
- ML I/O: Input is an image, output is a list of similar images.
- ML Category: Unsupervised learning (for feature extraction) and similarity search.
- Need for ML: Yes, ML is needed for feature extraction and efficient similarity search.

## Step 2: Metrics (Offline and Online)

- Offline Metrics: F1-Score@k and Mean Average Precision (mAP) are relevant for evaluating the retrieval quality.
- Online Metrics: Latency and retrieval accuracy would be critical for evaluating real-time performance.
- Trade-offs: There may be a trade-off between latency and accuracy, and computational cost during inference.

## Step 3: Architectural Components (MVP Logic)

- High-Level Architecture:
    - Non-ML Components: MongoDB for storing vectors, data preprocessing (DeepFace).
    - ML Components: Feature extraction model (DeepFace FaceNet512), nearest neighbor search (FAISS).
- Modular architecture design:
    - Generator: Generates embeddings from the dataset, uploads them to the database.

    - Search base: Downloads embeddings from database at initialization, creates FAISS index and provides its similarity search capabilities.

## Step 4: Data Collection and Preparation

- Data Needs: Face images.
- Data Sources: Opensource, hence free and accessible.
- Data Storage: MongoDB for vector storage.
- ML Data types: Numerical - embeddings with float values
- Labeling: Doesn't need labels.
- Data Augmentation: Could be applied to improve the robustness of the model.
- Data Generation Pipeline:
    - Download face images locally
    - Generate embeddings

## Step 5: Feature Engineering

- Choosing Features: Reduce embedding's size using PCA.
- Feature Representation: Image embeddings generated by the FaceNet512 model.
- Feature Processing: Preprocessing steps like resizing and normalization are necessary and are implemented in the embedding generation method from DeepFace.
- Missing Values: FaceNet512 model produces embeddings without missing values.
- Feature importance: PCA reduces embedding's size from 512 to 49 without losing variance of data.
- Featurizer: FaceNet512 model.
- Static vs dynamic features: Static, since embeddings are downloaded from the database. Embeddings of input images are computed online, thus such features are dynamic.

## Step 6: Model Development and Offline Evaluation

- Model Selection:
    - Heuristics: How similar visually input image is to the retrieved images.
    - First model choice: Google ViT base (86.4M params) model pretrained on ImageNet-21k. Retrieved images were far from input.
    - Second model choice: Google ViT huge (632M params) model pretrained on ImageNet-21k. Retrieved images were far from input, inference time was higher.
    - Third model choice: Analysis of models from DeepFace repository. FaceNet512 was selected for excellent performance and relatively low latency.
    - FaceNet512 showed ideal results.
    - No need in ensemble, since DL model is used for feature extraction.
- Dataset:
    - Sampling: Since the dataset is relatively small, no sampling is needed.

    - Data splits: Since no training happens, no train-test-val split is needed.
    - Class imbalance: Since no training happens, we don't consider class imbalance. We care about most similar images, not about who are pictured on them.
- Training: Pretrained FaceNet512 doesn't require fine-tuning.
- Evaluation: Offline evaluation metrics such as mAP and precision/recall for retrieved results.

## Step 7: Prediction Service

- Data Processing and Verification: Putting image in the same directory as project, and inserting the directory of image as input.
- Web app and serving system: No frontend is developed in this project, hence nothing to write here.
- Prediction service: Jupyter Notebook
- Batch vs Online prediction: The project involves both batch and online prediction. Embeddings are precomputed and stored (batch), and queries are processed in real-time (online).
- Nearest Neighbor Service: Excessive, since dataset size is small.
- ML on the Edge (on-device AI): Our project can be considered as ML on the Edge, since we run it on our local machines, local runs are free and fast, due to discrete GPU modules in our laptops.

## Step 8: Online Testing and Model Deployment

- Not applicable, since no deployment is done, and no deployment will be done in the future.

## Step 9: Scaling, Monitoring, and Updates

- Not applicable, since no deployment is done, and no deployment will be done in the future.

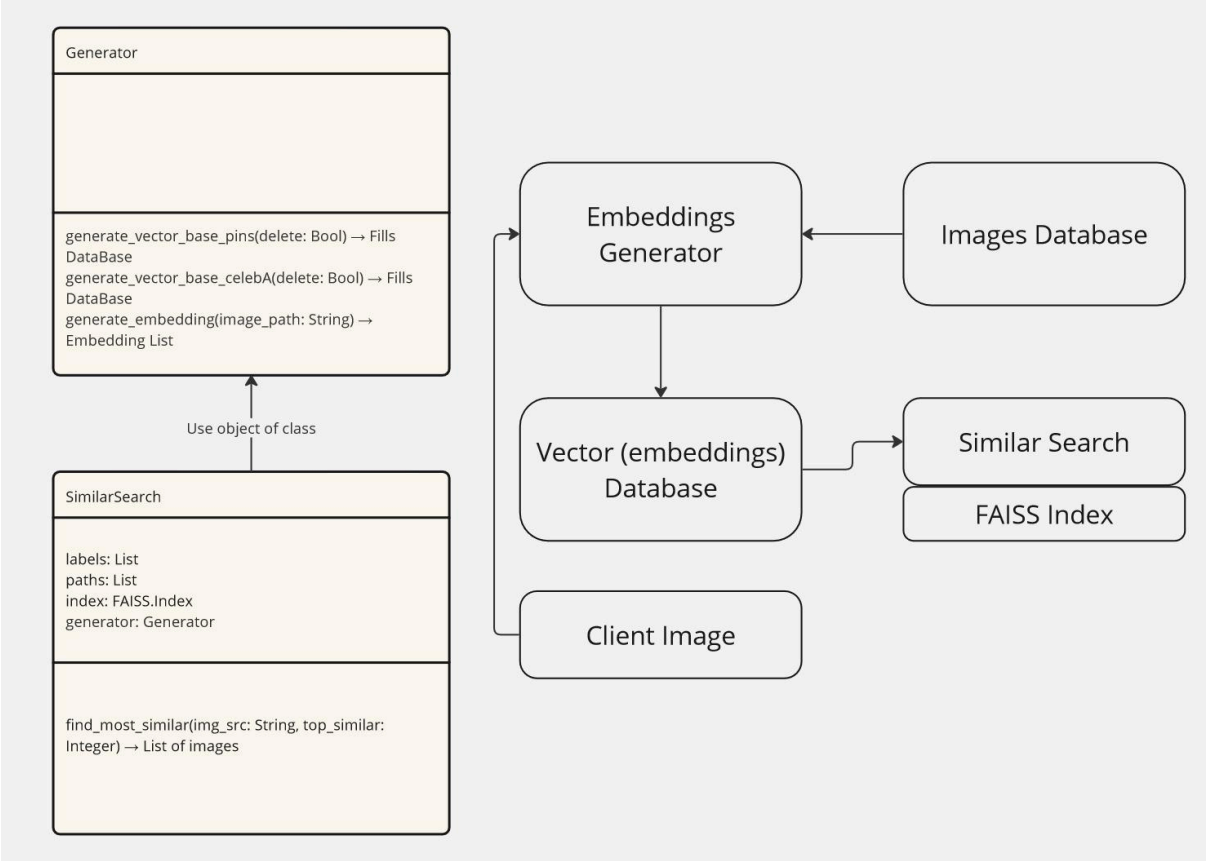
## Conclusion

The project can benefit from further enhancements such as:

- Implementing a frontend for better user interaction.
- Considering deployment strategies for real-world application use.

By following these recommendations, our project can be improved to better meet the requirements of a system design, ensuring its effectiveness and usability in practical scenarios.

# UML-diagram


Generator
generate_vector_base_pins(delete: Bool) → Fills DataBase
generate_vector_base_celebA(delete: Bool) → Fills DataBase
generate_embedding(image_path: String) → Embedding List
Use object of class
SimilarSearch
labels: List
paths: List
index: FAISS.Index
generator: Generator
find_most_similar(img_src: String, top_similar: Integer) → List of images
Embeddings Generator
Images Database
Vector (embeddings) Database
Similar Search
FAISS Index
Client Image