Visual Debugger for Dataset Bias and Quality

A Project Report Submitted in Partial Fulfilment of the Requirements for the Course

DES646: AI/ML for Designers

Submitted by:

Debarpita Dash (220328)

Nimmala Rithika Reddy (220716)

Sanapala Jaswanth (220955)

Sandela Sai Amruta Varshini (220960)

Shrimi Agrawal (221036)

Under the Guidance of:

Prof. Amar Behera

Department of Design

Indian Institute of Technology Kanpur

November 2025

Contents

1. Abstract

This project presents an interactive, visual tool for detecting and mitigating dataset bias and quality issues before model training. Designed using Streamlit and Python, the system leverages pretrained models, influence analysis, and visual diagnostics to make dataset debugging intuitive and interpretable. The tool helps users identify duplicate, imbalanced, and low-quality samples, visualize embeddings, and explore the dataset.

2. Introduction

This project is a Visual Debugger for Image Datasets, designed to help users inspect, analyze, and improve the quality of datasets before training machine learning models. It automatically detects duplicates, class imbalances, outliers, and mislabeled or bias-conflicting samples, and provides interactive visualizations of embeddings and dataset statistics.

For designers, this tool is especially useful because it allows them to visually explore their datasets without needing deep technical knowledge. Designers can quickly understand which images may cause biases or inconsistencies in AI models, identify patterns or unintended correlations, and make informed decisions about relabeling, filtering, or curating their datasets to ensure fair and high-quality outputs in creative AI applications.

3. Background and Literature Review

Many existing bias analysis tools focus on post-hoc model explainability, but dataset-level bias detection remains underexplored. Key inspiration comes from:

- Koh and Liang (2017): Influence functions to estimate sample impact on model performance.
- Ribeiro et al. (2016): LIME for local interpretability.
- Lundberg and Lee (2017): SHAP values for feature importance.

Our contribution lies in integrating these ideas with visual design and usability to create a designer-friendly bias debugging environment.

4. Aim and Objectives

4.1 Aim

To develop a lightweight, interactive tool that enables designers and researchers to detect, visualize, and mitigate dataset bias and quality issues before model training. By combining principles of explainable AI and visual design, the tool seeks to make the process of understanding data problems more transparent, interpretable, and accessible to users who may not come from a deep technical background.

4.2 Objectives

The objectives of the project are as follows:

1. **Data Diagnostics:** The project aims to automatically identify common data issues in image datasets such as duplicates, class imbalance, low-quality samples, and embedding-based outliers. By leveraging pretrained vision models, the tool provides users with a quick and visual summary of their dataset's overall health.

2. **Bias Detection via Self-Influence:** A simplified self-influence computation technique is implemented to detect images that disproportionately affect model performance. These "bias-conflicting" or mislabeled samples can distort learning outcomes and fairness. Highlighting them helps users understand which specific data points may be responsible for unwanted bias in the trained model.

3. **Visual Exploration:** The project builds an interactive Streamlit dashboard where users can:

    - Explore class distributions and duplicate clusters visually.
    - Project embeddings (e.g., UMAP) to identify outliers or spurious correlations.

- Inspect bias-conflicting images ranked by influence scores.

4. **Actionable Feedback:** The tool provides data-level recommendations such as relabeling, reweighting, or removing problematic samples. It also visualizes how fairness or accuracy metrics change after applying these corrective actions.

5. **Optional Label Completion:** For partially labeled datasets, the tool integrates a semi-supervised labeling module that suggests probable class labels for unlabeled images using pretrained models or nearest-neighbor embedding similarity, enabling labeling before bias analysis.

5. Motivation

Current data auditing and bias detection workflows in machine learning primarily rely on post-training diagnostics, where model gradients, prediction errors, or fairness metrics are analyzed after a model has already been built. These methods demand significant computational resources, long development cycles, and specialized expertise in fairness evaluation and model interpretability. Moreover, existing tools are often limited to narrow use cases, such as outlier detection, label noise estimation, or static visualization of embeddings, and do not integrate these capabilities into a unified pipeline.

As a result, developers cannot easily identify dataset-level biases, low-quality samples, or structural inconsistencies before they influence model behavior. The motivation for this project is to address this technical gap by creating a pre-training dataset debugger that combines influence estimation, embedding-based diagnostics, and interactive visualization within a single system. Unlike existing systems that operate only after model training or rely on isolated analytical modules, this system performs proactive bias detection, duplicate identification, imbalance analysis, and label quality assessment before a model is trained. This early-stage diagnostic capability reduces computational overhead, prevents bias propagation, and provides a modular and interpretable workflow that can be integrated directly into machine learning development pipelines.

6. System Design and Architecture

The **Visual Debugger** system is designed as a modular end-to-end pipeline that facilitates data set diagnostics, bias detection, and visualization. The architecture is organized into clearly defined modules, each responsible for a specific task. This modular design ensures flexibility, reusability, and ease of extension.

6.1 Directory Structure

The project is organized into modular directories as follows:

```
visual-debugger/
  data_utils/       # Dataset loading, metadata extraction
    loader.py       # Load dataset, extract paths, labels, image stats
    labelling.py     # Semi-supervised label suggestions
    quality.py      # Detect corrupt, low-res, or low-contrast images
    visualize.py    # visualization of data

  embedding/        # Embedding extraction and analysis
    extract.py      # Extract embeddings using ResNet18 / CLIP
    indexer.py      # Build FAISS / kNN indices for fast retrieval
    visualize.py    # UMAP / t-SNE visualization of embeddings

  diagnostics/      # Dataset quality analysis
    duplicates.py   # Detect duplicate images using cosine similarity
    imbalance.py    # Class distribution & imbalance metrics
    outliers.py     # Identify outliers based on embedding distance
    diversity.py    # Diversity index (entropy / pairwise distance)
    fix_dataset.py  # Fix dataset

  influence/        # Bias and influence analysis
    influence.py      # Compute self-influence scores
    sensitive_attr.py   # considering sensitive attributes

  dashboard/        # Streamlit interactive UI
    app.py          # Main entrypoint with Label / Analyze options
```

```
testing/              # Unit tests
outputs/              # Outputs
requirements.txt   # Python dependencies
README.md          # Project documentation
```

6.2 System Workflow

The overall system workflow can be divided into the following steps:

1. **Dataset Input:** Users can upload a fully labeled dataset or a partially labeled dataset. If the dataset is partially labeled, the semi-supervised labeling module predicts missing labels using pretrained embeddings (ResNet18/CLIP) and nearest-neighbor similarity.

2. **Feature Extraction and Embedding:** The system computes image embeddings using pretrained backbones and applies dimensionality reduction (UMAP/PCA) to facilitate visualization of class clusters and outliers.

3. **Data Diagnostics:** Duplicate images, class imbalance, and low-quality or anomalous samples are automatically detected. This step provides a comprehensive overview of dataset health, enabling informed decisions.

4. **Bias and Self-Influence Analysis:** A simplified self-influence computation identifies bias-conflicting or mislabeled samples that disproportionately affect model performance and fairness. These samples are highlighted for user inspection.

5. **Interactive Visualization:** The Streamlit dashboard visualizes class distributions, embedding projections, duplicates, outliers, and influence-ranked samples. Users can explore data interactively and obtain actionable insights.

6. **Actionable Feedback and Dataset Correction:** Based on diagnostics, the system provides recommendations such as relabeling, reweighting, or removing problematic samples.

6.3 System Flow Diagram

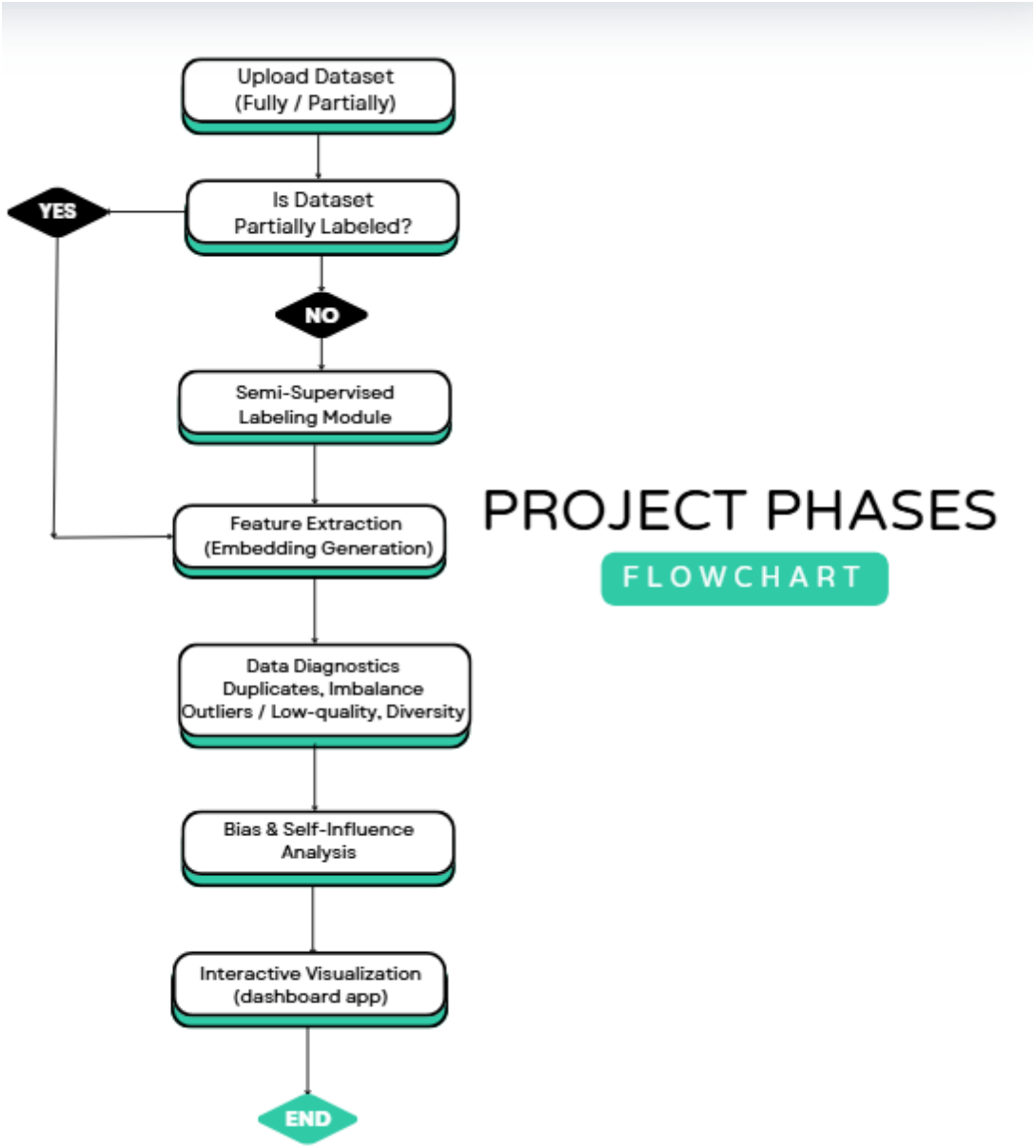

Figure 1: Proposed system flow of the Visual Debugger pipeline from dataset input to actionable feedback.

This design ensures a transparent, end-to-end pipeline that integrates dataset analysis, bias detection, and interactive visualization, enabling both technical and non-technical users to evaluate and improve dataset quality before model training.

7. Detailed Methodology

7.1 Data Utilities (`data_utils/`)

7.1.1 Dataset Loading and Metadata Extraction (`loader.py`)

This module provides foundational utilities for managing image datasets. It includes functions to load datasets, extract metadata, and organize them for analysis. The `load_dataset()` function supports both structured datasets—where images are arranged in folders corresponding to class labels—and unstructured datasets with separate label files.

Each image record includes file path, label, and metadata such as width, height, aspect ratio, brightness, and color channel information. Metadata extraction is implemented using the Pillow library for image I/O and NumPy for numerical computation, with progress tracking via `tqdm`.

The `summarize_dataset()` function aggregates dataset-level insights, including the number of labeled and unlabeled samples, class distribution, and image resolution statistics. This module ensures the dataset is well-structured and ready for downstream embedding and diagnostic analysis.

7.1.2 Semi-Supervised Labeling (`labeling.py`)

This component performs semi-supervised labeling using a pretrained ResNet-18 network as the feature extractor. The network generates embeddings for all images, and label propagation is achieved by computing nearest neighbors between labeled and unlabeled data.

For efficient retrieval, the module employs FAISS (Facebook AI Similarity Search) when available, or scikit-learn's `NearestNeighbors` as a fallback. Label propagation is based on cosine similarity, and pseudo-labels are filtered using a confidence threshold to retain only reliable assignments.

This approach enables label expansion in partially annotated datasets, ensuring a richer labeled dataset for subsequent analysis and visualization.

7.1.3 Image Quality Assessment (`quality.py`)

The image quality module assesses visual clarity using the Laplacian variance technique to detect blurry or low-focus images. Each image's sharpness score is computed via OpenCV's Laplacian operator, and samples below a specified variance threshold are flagged as low-quality.

The results are stored in a structured DataFrame, and visualization utilities display side-by-side grids of sharp versus blurry images. This ensures that only high-quality, focused samples proceed to embedding and model training stages.

7.1.4 Dataset Visualization (`visualize.py`)

Visualization utilities enable exploration of dataset properties through statistical plots. Functions include:

- `plot_class_distribution()` — Displays class frequency as a bar chart to detect imbalance.
- `plot_brightness_histogram()` — Shows distribution of mean brightness across samples.
- `plot_resolution_scatter()` — Plots width vs. height to identify irregular image resolutions.

These tools provide early visual diagnostics that support data cleaning and balancing.

7.2 Embedding Module (`embedding/`)

7.2.1 Embedding Extraction (`extract.py`)

This module computes high-dimensional embeddings from images using pretrained neural networks such as ResNet-18 or CLIP. These embeddings encapsulate semantic information, allowing visual similarity comparisons between images.

The `extract_embeddings()` function performs standardized preprocessing—resizing, normalization, and tensor conversion—before feeding data in mini-batches through the

model. Embeddings are extracted from the penultimate layer and stored as NumPy arrays or PyTorch tensors.

GPU acceleration ensures efficient computation even on large datasets. These embeddings serve as the foundation for duplicate detection, clustering, and bias analysis.

7.2.2 Index Building (`indexer.py`)

This submodule constructs fast retrieval indices using FAISS or scikit-learn for approximate nearest-neighbor search. Embeddings are normalized to unit length to enable cosine similarity-based comparison.

FAISS indices (e.g., `IndexFlatIP`, `IndexIVFFlat`) allow scalable similarity queries like "find top-k similar samples." These indices are later leveraged by duplicate, outlier, and influence detection modules for efficient large-scale analysis.

7.2.3 Embedding Visualization (`visualize.py`)

To visualize high-dimensional embeddings, dimensionality reduction methods such as PCA and UMAP are applied. The `plot_embedding_projection()` function projects embeddings into 2D or 3D and colors them based on class labels or cluster assignments.

These scatter plots, generated via Matplotlib and Seaborn, reveal class separability, potential outliers, and overlapping classes, offering intuitive insight into dataset structure.

7.3 Diagnostics Module (`diagnostics/`)

7.3.1 Duplicate Detection (`duplicates.py`)

Duplicate detection identifies visually identical or near-identical images by computing cosine similarity between embeddings. The `find_duplicates()` function flags pairs exceeding a predefined similarity threshold (typically > 0.95).

The output includes duplicate group mappings and representative images, helping maintain dataset diversity and reduce redundancy.

7.3.2 Class Imbalance Analysis (`imbalance.py`)

This module examines dataset balance by computing per-class sample counts and visualizing them as bar charts. Additional quantitative measures such as Gini Index and class entropy are computed to assess imbalance severity.

These insights inform rebalancing strategies, including oversampling or weighted loss adjustments, improving downstream model fairness and stability.

7.3.3 Outlier Detection (`outliers.py`)

Outlier analysis identifies samples that deviate significantly from class norms. Using distances from each sample to its class centroid, points beyond a statistical threshold are flagged as potential outliers.

Visualizations of outlier positions in embedding space assist analysts in deciding whether to correct labels or remove anomalous samples.

7.4 Influence Module (`influence/`)

7.4.1 Self-Influence Analysis (`influence.py`)

This module computes self-influence scores that estimate how individual samples affect model behavior. Approximate influence functions are computed in embedding space to identify samples that disproportionately shift decision boundaries or degrade fairness metrics.

Such points often correspond to mislabeled or bias-inducing data. Identifying and correcting them improves both model generalization and equity in performance across demographic subgroups.

7.5 Dashboard Module (`dashboard/`)

7.5.1 Interactive Interface (`app.py`)

The main Streamlit-based interface connects all analytical modules. Users can either:

1. Upload a semi-labeled dataset for preprocessing and embedding generation, or

2. Run diagnostics to identify duplicates, imbalances, and bias-inducing samples.

The dashboard integrates visual plots, metrics, and summaries into an intuitive interface that enables real-time inspection of dataset health.

8. Results and Analysis

The Visual Debugger successfully identified structural issues within the dataset, including imbalance, potential outliers, and bias-conflicting samples. The system also provided clear embedding visualizations, duplicate checks, and fairness indicators, enabling users to inspect and repair data before model training.

Semi-Supervised Labeling Results

To evaluate the effectiveness of the semi-supervised labeling module, a partially labeled dataset was created from Fashion-MNIST by masking 30% of the ground-truth labels. The system achieved a label propagation accuracy of approximately 61.6%, demonstrating its ability to infer missing annotations using pretrained embedding similarity. Confidence scores were generated for all predicted labels, allowing users to prioritize high-certainty corrections.

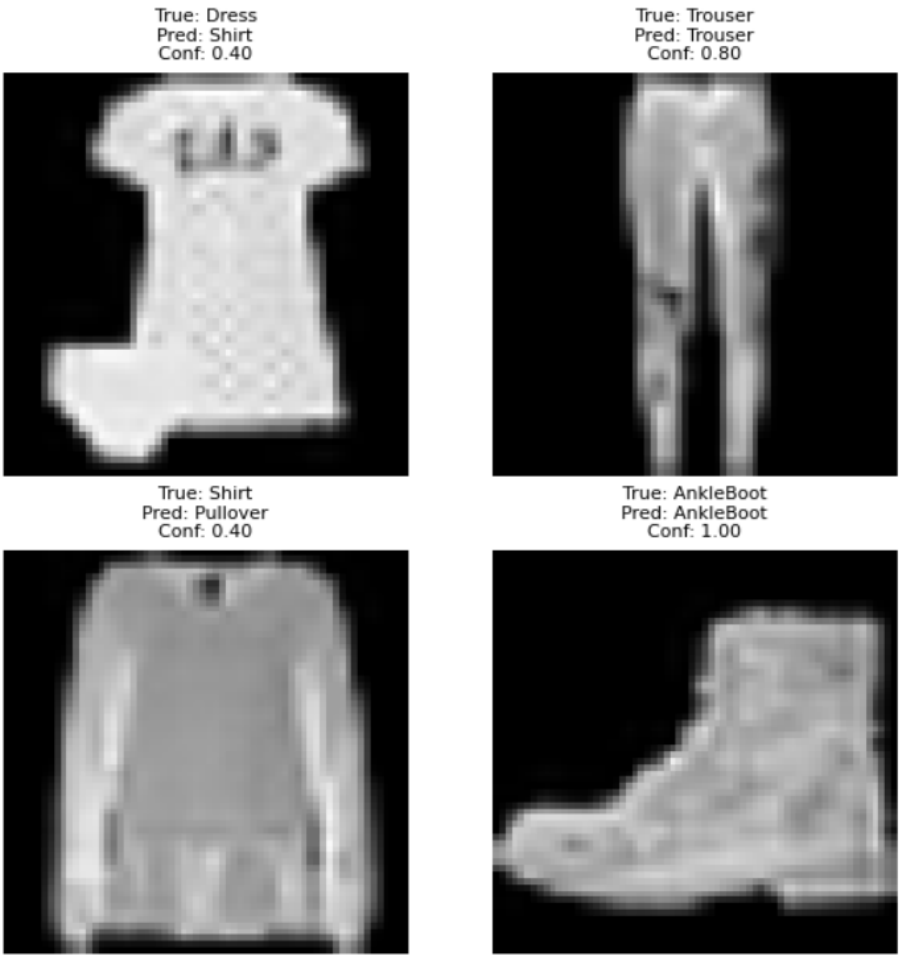

Figure 2: Semi-supervised label recovery showing true labels, predicted labels, and confidence scores for originally unlabeled samples.

Dataset Diagnostics

The diagnostics dashboard initiates the complete data inspection workflow. Users begin by uploading a dataset and extracting embeddings using pretrained backbones. These embeddings form the basis for later analyses such as duplicate detection, class imbalance evaluation, and outlier identification.

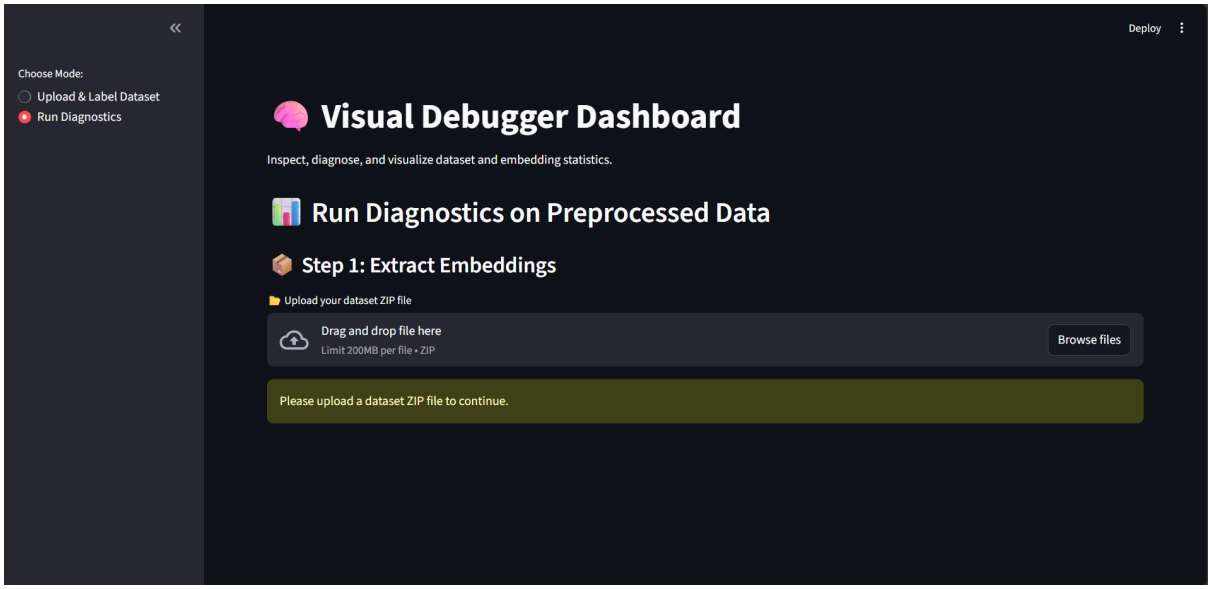

Figure 3: Main dashboard interface for running dataset diagnostics.

Once the dataset is loaded, users select a backbone model such as CLIP or ResNet18. The system extracts high-dimensional embeddings for all samples and reports the embedding count and dimensionality. This embedding step enables similarity-based diagnostics without requiring a trained classifier.

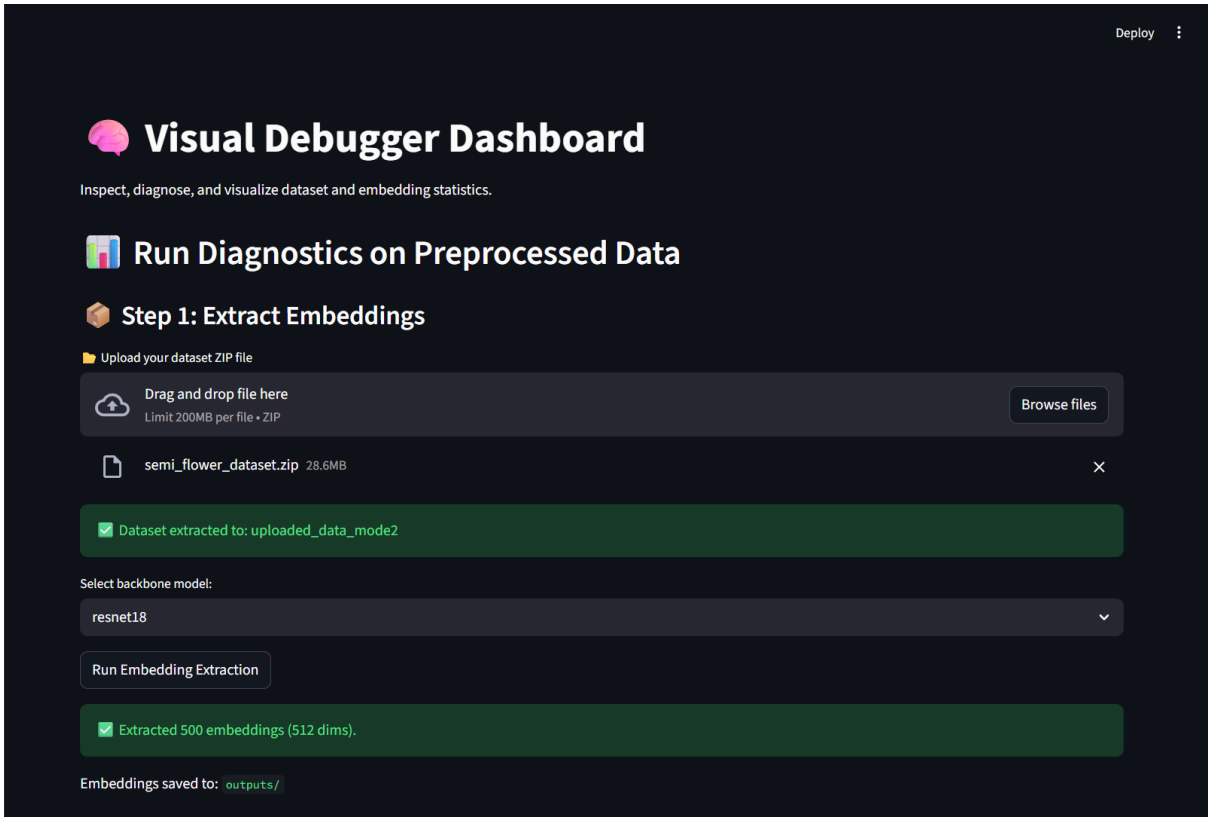

Figure 4: Embedding extraction using CLIP or ResNet18 backbones.

The FAISS index construction step enables fast nearest-neighbor search for duplicate detection. Users may specify a similarity threshold (e.g., cosine similarity ¿ 0.95). In this example, no duplicates were found, indicating that the dataset is free of redundant entries that might otherwise bias training.

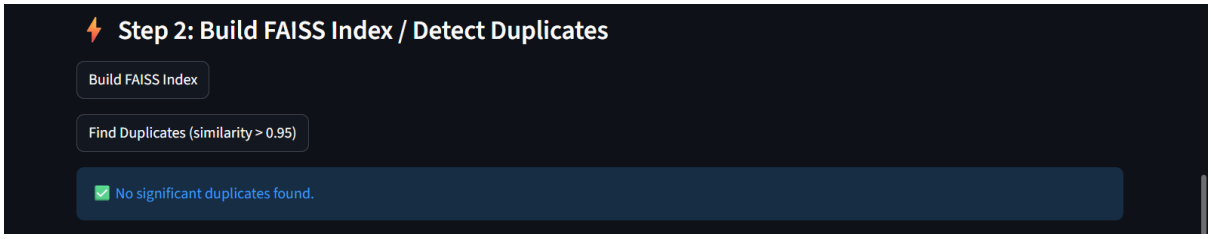

Figure 5: FAISS-based duplicate analysis.

Embedding Visualization

To help users interpret dataset structure, the system supports PCA, t-SNE, and UMAP projections.

PCA Projection: PCA reveals coarse global structure. Distinct clusters indicate well-separated classes; overlapping regions may signal mislabeled or ambiguous samples. The cosine similarity heatmap complements this view by showing global similarity patterns.

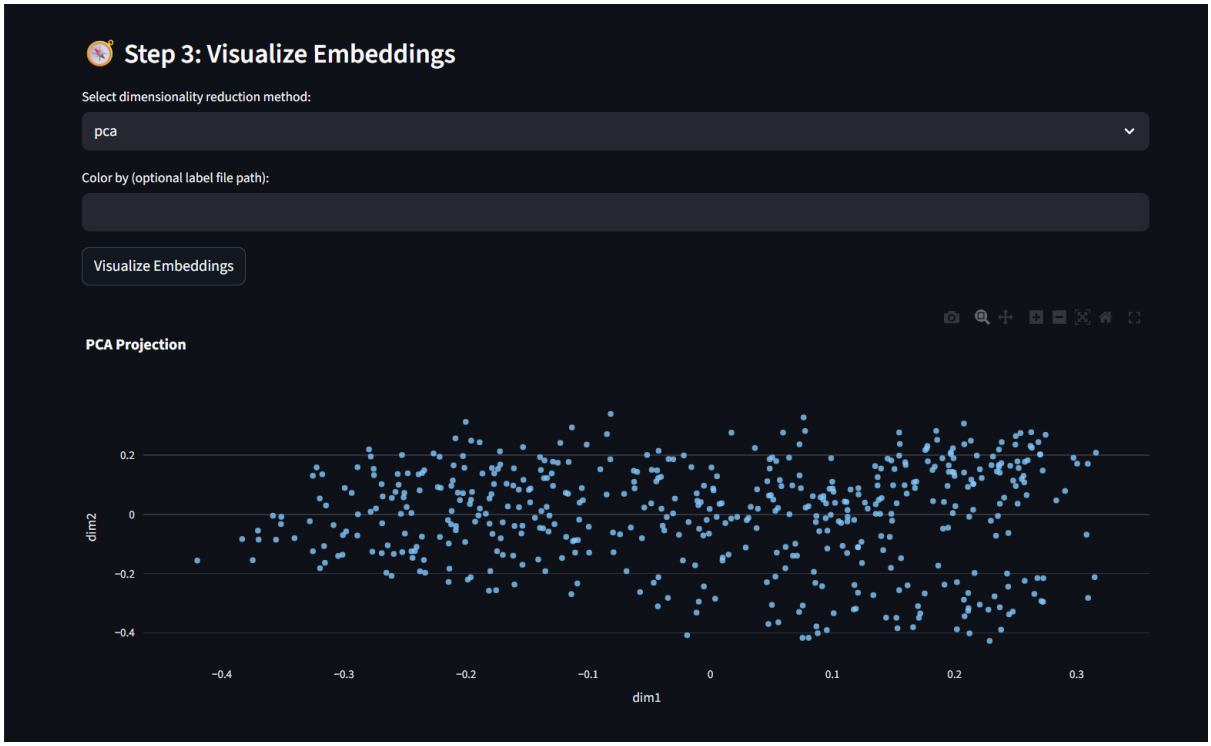

Figure 6: PCA projection of dataset embeddings.

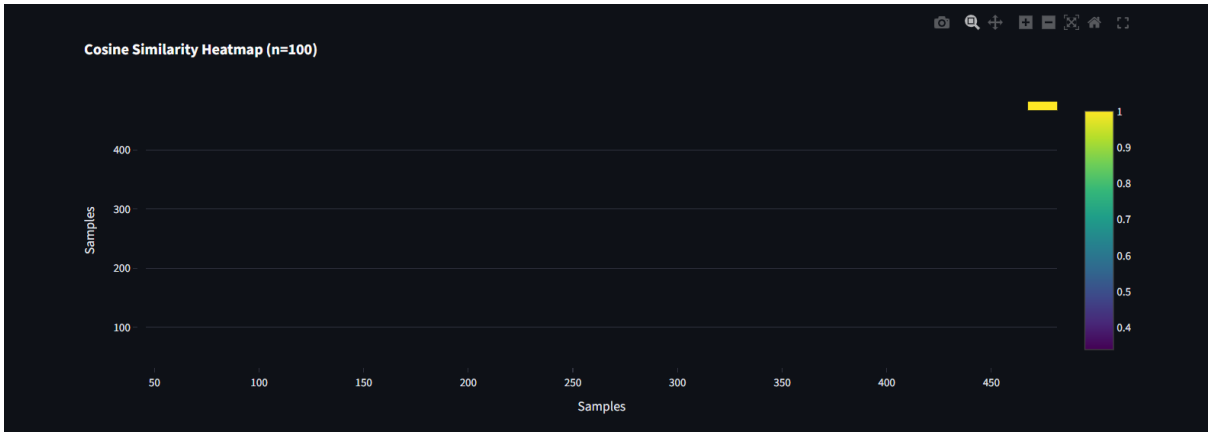

Figure 7: Cosine similarity heatmap for PCA embeddings.

t-SNE Projection: t-SNE emphasizes local relationships and helps detect tight subclusters, rare subclasses, and isolated outliers. The resulting neighborhood structure often reveals mislabeled or atypical examples.

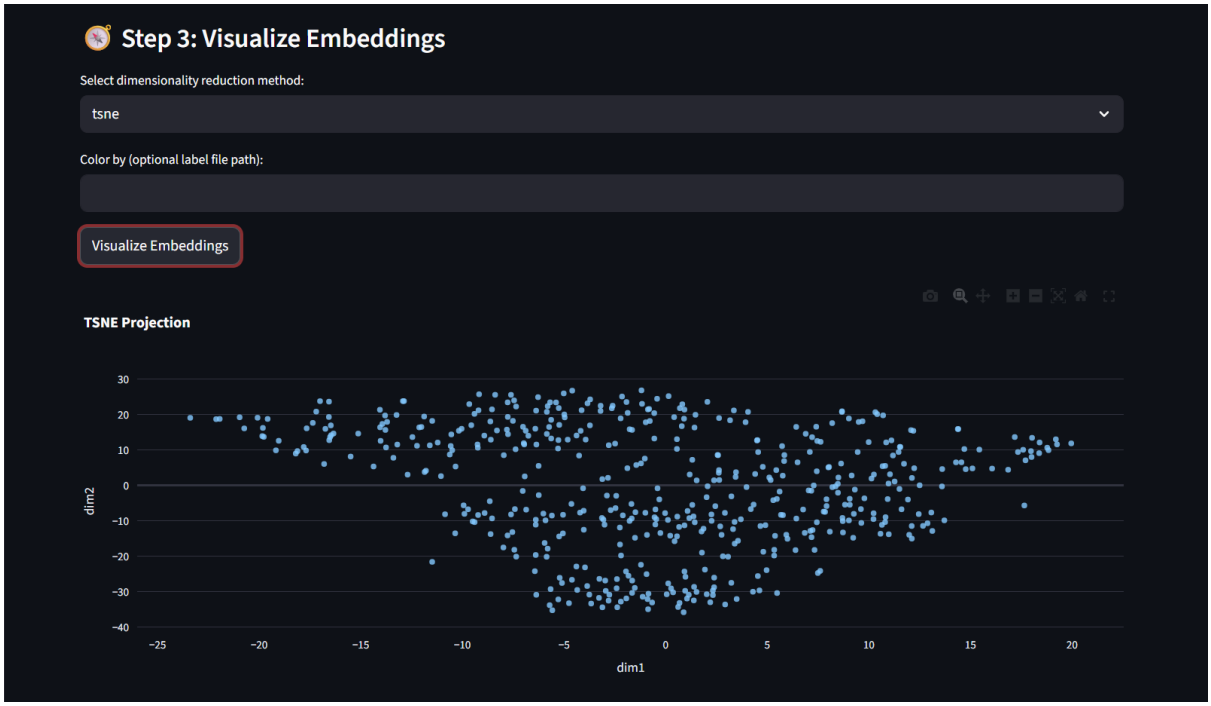

Figure 8: t-SNE embedding visualization.

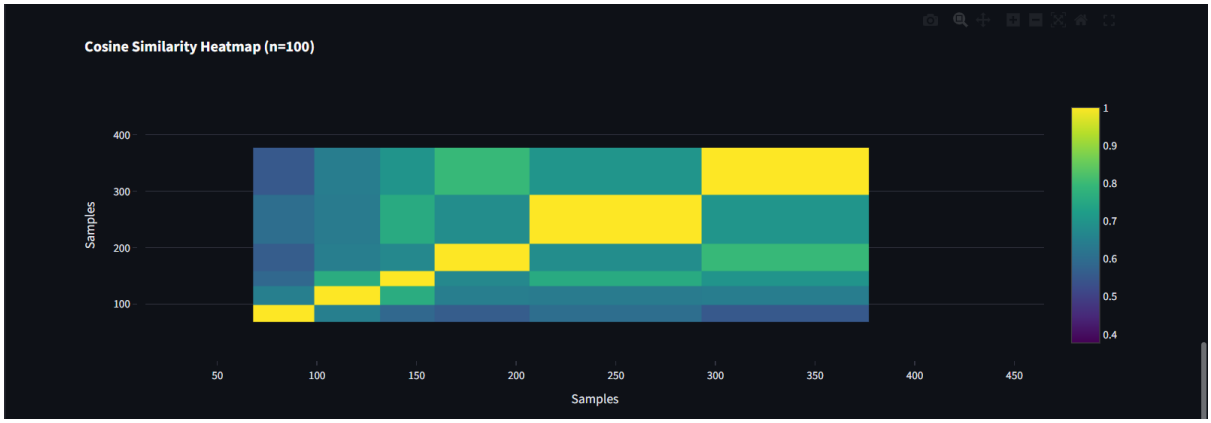

Figure 9: Cosine similarity heatmap for t-SNE embeddings.

UMAP Projection: UMAP captures both local and global manifold structure and often provides the clearest class separation. This makes UMAP especially useful for detecting dispersed clusters or inconsistencies in dataset distribution.

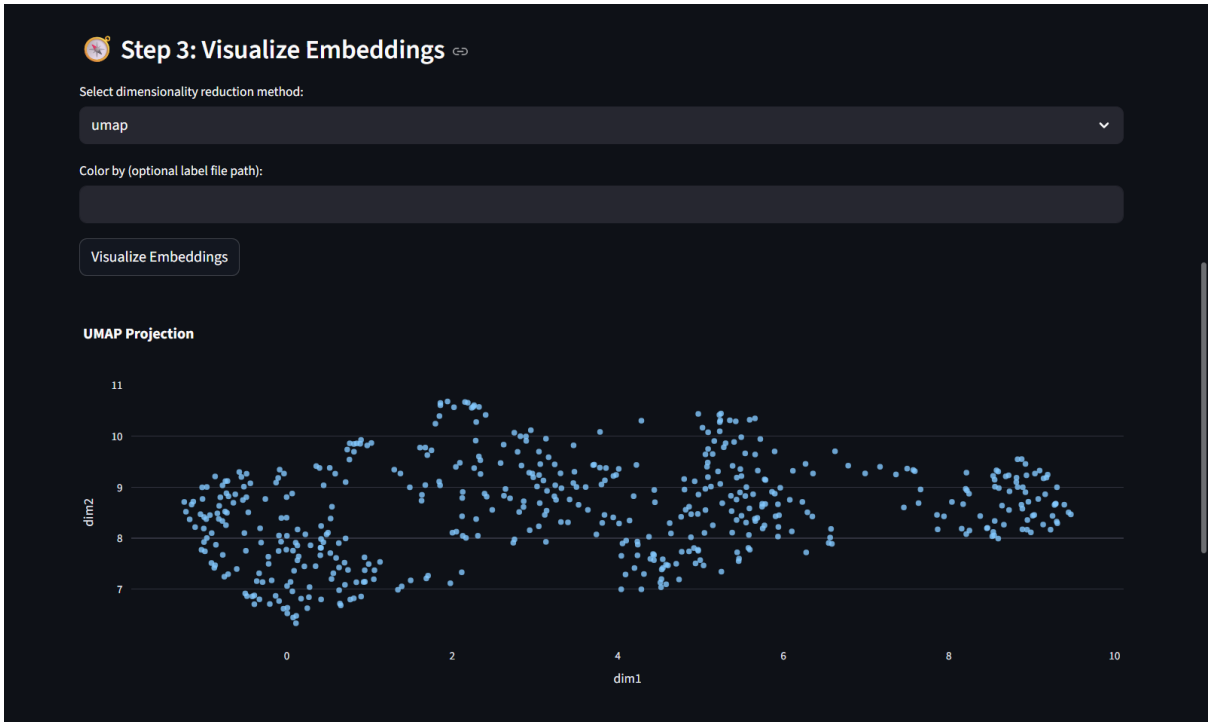

Figure 10: UMAP embedding visualization.

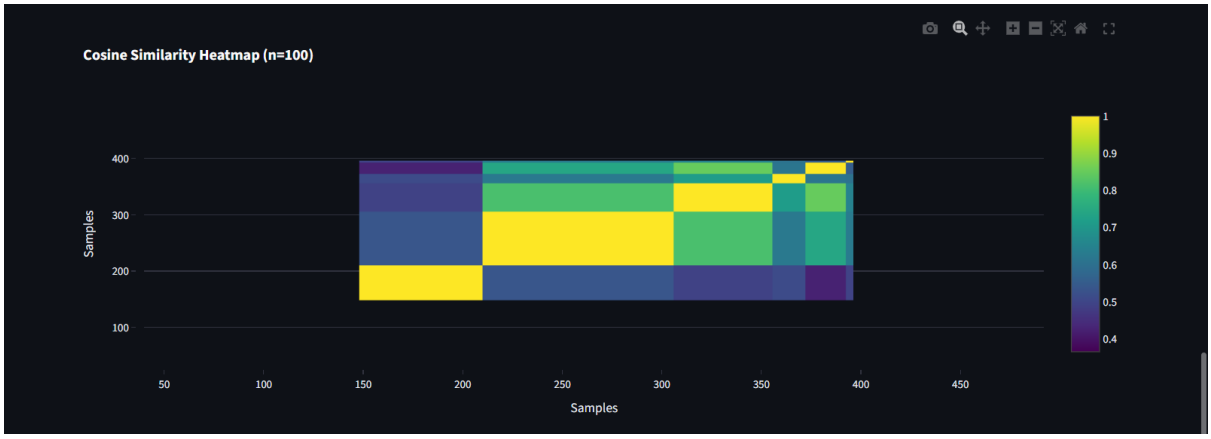

Figure 11: Cosine similarity heatmap for UMAP embeddings.

Duplicate and Imbalance Diagnostics

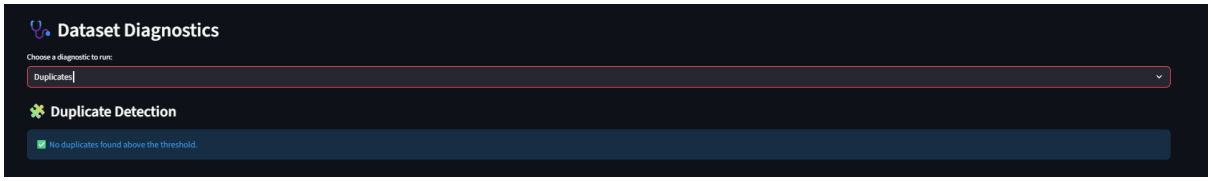

Figure 12: Duplicate analysis results. No duplicates detected.

The imbalance diagnostic computes Shannon entropy, Gini index, and effective number of classes. A Shannon entropy value of 3.3219 and a Gini index of 0.9 indicate a highly uniform distribution. An effective class count of 10 confirms full representation without minority collapse.

Figure 13: Class imbalance diagnostics. All classes are evenly represented.

Outlier Detection

Outlier detection identifies samples whose embeddings deviate from the dataset's main structure. Outliers often correspond to mislabeled, noisy, or ambiguous examples. In this analysis, the system flagged a specific index (1) as an outlier based on its embedding distance.

Figure 14: Detected outlier sample in the embedding space.

Dataset Repair Suggestions

After running all diagnostics, the system compiles automated repair suggestions summarizing imbalance, duplicates, outliers, and fairness indicators. In this case, the tool reports that the dataset is already balanced, clean, and consistent.

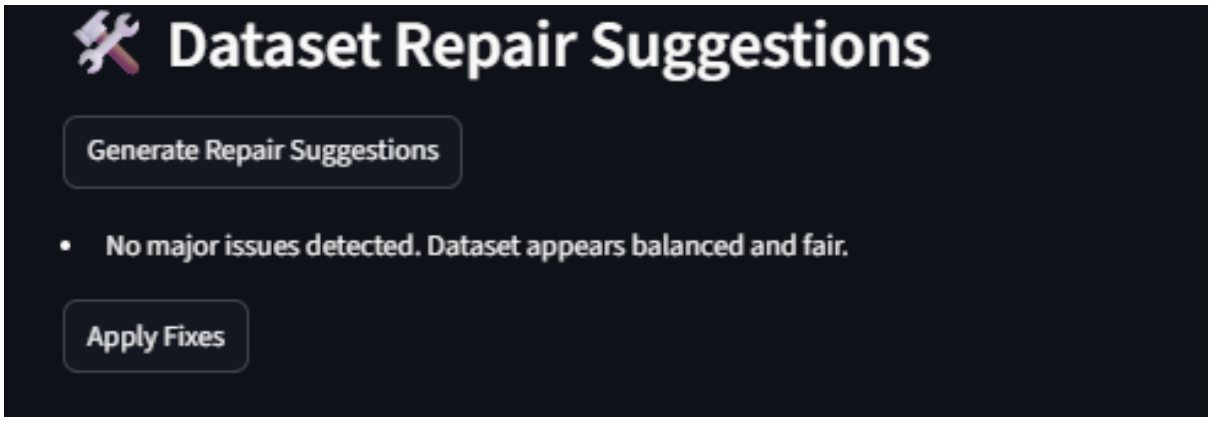

Figure 15: Automated dataset repair suggestions. No corrections required.

Influence and Fairness Analysis

The final module identifies bias-conflicting samples using influence scores. High influence values suggest atypical or mislabeled items, while near-zero values indicate samples that follow the main dataset structure.

The fairness metrics quantify group-level disparities:

- **Demographic parity difference**: values close to 0 indicate equal representation across groups. - **Equalized odds difference**: near-zero values indicate similar error rates across groups.

Large deviations in either metric would signal structural unfairness that may propagate during training.

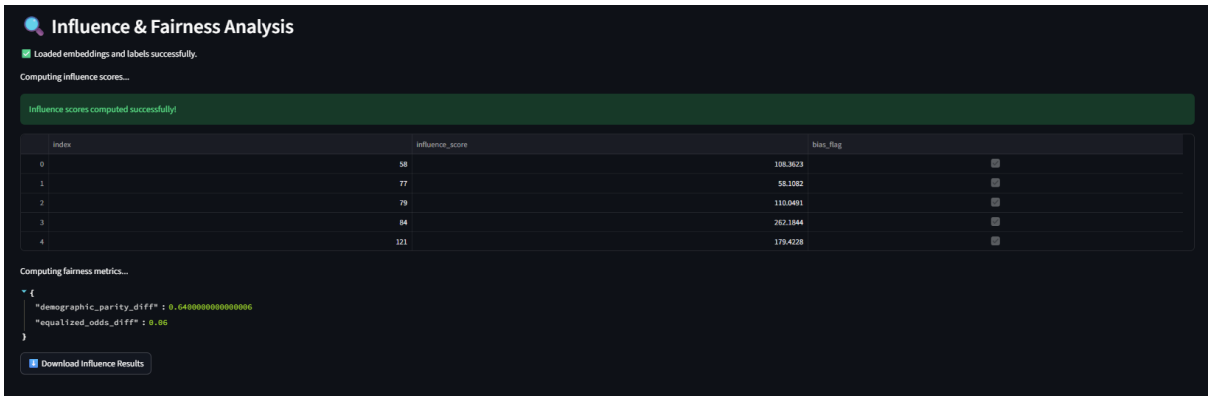

Figure 16: Influence and fairness metrics for bias detection.

8. Discussion and Limitations

The developed system demonstrates a promising step toward interpretable and automated dataset quality assessment. By providing visual insights into dataset bias, redundancy, and imbalance, it allows data scientists to identify issues that would otherwise remain hidden in large-scale datasets. The integration of embeddings and visual analytics enables efficient detection of problematic samples, thereby improving the overall model training pipeline.

However, several limitations remain:

- The system relies heavily on pretrained model embeddings, which may not always

generalize well across different domains or data modalities.

- Computational efficiency is constrained by the embedding generation process, which can be time-intensive for high-resolution or large datasets.
- The current interface primarily supports image datasets and limited numerical attributes, restricting broader applicability to text or multimodal data.
- There is limited automation in suggesting corrective actions — the tool identifies biases but depends on manual interpretation for remediation.

Despite these challenges, the framework provides a solid foundation for future exploration and domain adaptation.

9. Future Scope

Building on the current implementation, several extensions can enhance the system's utility and scalability:

- **Model Interpretability:** Integrate explainability methods such as SHAP and LIME to correlate dataset quality metrics with model decision behavior.
- **Multimodal Expansion:** Extend support for text, audio, and video data, enabling cross-domain bias analysis.
- **Collaborative Debugging:** Incorporate a cloud-based platform allowing multiple users to collaboratively visualize, annotate, and correct dataset issues.
- **Automation:** Develop semi-automated suggestions for data cleaning or augmentation based on detected anomalies.
- **Performance Optimization:** Explore dimensionality reduction and incremental embedding computation for faster scalability.

10. Conclusion

This project bridges a critical gap between dataset curation and explainable AI by providing a *visual, designer-centric debugging platform* for dataset quality assessment. Through intuitive visualizations and bias detection tools, it enhances transparency, fairness awareness, and data-driven decision-making within machine learning workflows.

By transforming abstract numerical embeddings into interpretable visual structures, the system empowers both researchers and practitioners to build more ethical, accountable, and high-performing AI systems. In doing so, it lays the groundwork for a new paradigm in **visual dataset debugging and responsible AI development**.

Appendix

GitHub Repository: https://github.com/DebDDash/des646_project

Each submodule has been documented with inline comments and separate README sections to guide reproducibility and extension.