SDP 2025

User Design Document

Project Name	Unified Deep Learning Benchmark for Satellite Image Restoration and Generation
Faculty Supervisor Name	Ismayil Shahaliyev
Team Lead Name & ID	Royana Huseynova 11270
Team member #1 Name & ID • What sections did this person contribute to?	Royana Huseynova 11270 • Section 2: Class diagram
Team member #2 Name & ID • What sections did this person contribute to?	Pasha Zulfugarli 13731 • Section 1
Team member #3 Name & ID What sections did this person contribute to?	Nijat Alisoy 17366 • Section 2
Team member #4 Name & ID • What sections did this person contribute to?	Huseyn Sadatkhanov 13770 • Section 1

Guidelines:

1. In Section 1 you should create a journey map in Figma, for **each of** the **four** most important use-cases from the research document. Do not worry about trivial use cases like user sign-up and user password reset.
2. In Section 2 (Database Design), discuss the data model you propose to support your user journey maps. The database design may be relational (using ERDs) or document storage (using document collections)
3. Page Count: 15 Pages Maximum

Section 1: Use Cases

The full set of use-cases can be accessed in our Figma workspace: https://www.figma.com/design/rEF6XX0suEK3RX6ZDPE7dN/SDP-2026?node-id=262-2&t=z6hBXMo5nYjN9T3F-1

Use-Case 1: Evaluate Cloud-Removal Models

Cloud-removal quality cannot be judged from metrics alone because clouds create complex spatial and spectral artifacts. A researcher needs consistent previews to verify model performance on AllClear and other benchmark datasets. Goal is to automatically generate:

- cloudy → predicted → clear triplets
- pixel-difference overlays
- per-scene PSNR/SSIM/SAM summaries

Use-Case 2: Compare Cloud-Removal Models

Researchers need a fair way to compare multiple CR models under identical settings. Numerical differences alone do not show where one model fails or succeeds. Goal is to automatically generate:

- side-by-side model outputs for each scene
- comparison triplets
- per-model metric tables
- visual difference maps showing where models diverge

Use-Case 3: Benchmark Super-Resolution Models

Super-resolution model performance varies by scale factor, and metrics often miss texture inconsistencies. Researchers need Low-Resolution (LR) → Super-Resolved (SR) → High-Resolution (HR) comparisons to see where detail is gained or hallucinated. Goal is to automatically generate:

- LR → SR → HR panels
- scale-factor-specific metric results (PSNR/SSIM/ERGAS)
- error heatmaps
- saved SR samples for quick inspection

Use-Case 4: Evaluate Super-Resolution Models

Dataset scenes usually contain diverse landscapes, making it hard to judge SR quality from metrics alone. Researchers need high-quality visual comparisons across locations. Goal is to automatically generate:

- LR → SR → HR triplets for each tile
- zoom-in crops for fine detail
- per-tile metric summaries
- difference overlays to show artifacts

Use-Case 5: Evaluate HD Generative Models

HD generative models have no ground truth, so visual inspection is essential for understanding realism and distortions. Researchers need grids comparing generated samples to real references. Goal is to automatically generate:

- generated vs. real grids
- FID/QNR summaries
- spectral–spatial distortion overlays

Use-Case 6: Normalize Pixel Ranges and Harmonize Spectral Bands

Datasets use different pixel scales and band conventions, making preprocessing crucial for fair evaluation. Researchers need automatic normalization to avoid inconsistent inputs. Goal is to automatically perform:

- min-max or standard normalization
- band alignment and reordering
- scaling to consistent ranges
- visual checks showing before/after normalization

Use-Case 7: Generate Visual Inspection Samples

Metrics cannot fully capture visual quality or artifacts. Researchers need quick previews showing model strengths and weaknesses. Goal is to automatically generate:

- CR: cloudy → predicted → clear
- SR: LR → SR → HR
- HD: generated vs. real

- optional overlays (error maps, pixel differences)

Use-Case 8: Ensure Reproducible Evaluation Results

Experiments must be repeated across models, datasets, and runs. Researchers need consistent logging of configurations and seeds. Goal is to automatically generate:

- JSON configs with all parameters
- stored seeds and preprocessing options
- logs linking outputs to configurations
- summary files for reproducibility

Use-Case 9: Validate Dataset Integrity Before

Misaligned or corrupted files can produce incorrect results. Researchers need automatic dataset checks before running models. Goal is to automatically perform:

- missing-file detection
- band count/shape validation
- cloudy–clear pairing checks
- sample previews for verification

Use-Case 10: Analyze Model Sensitivity to Preprocessing Variations

Small preprocessing changes (normalization, band selection, scaling) can significantly change results. Researchers need visual and numerical comparisons across variations. Goal is to automatically generate:

- outputs for multiple preprocessing configurations
- metric comparisons across settings
- side-by-side visual panels
- analysis of sensitivity patterns

Section 2: Database Design

In our project, the core of the software design is the *EvaluationPipeline* class, which coordinates the whole evaluation process for satellite image models. It connects the data side (*DatasetFactory, Dataset, DataLoader*) with the model side (*Model, ModelEvaluato*r) and the analysis side (*Metric, PipelineResult*). Given a configuration, the pipeline loads the selected dataset, feeds batches through a chosen model, computes the requested metrics,

and stores the final scores. This structure lets us plug in new datasets, models, and metrics without changing the main evaluation logic. This design is necessary because satellite tiles are large, diverse, and often multispectral, so the pipeline must standardize preprocessing and evaluation across datasets.

On the data side, the system is organized around *DatasetFactory*, *Dataset*, and *DataLoader*. *DatasetFactory* is responsible for constructing dataset objects for different sources we use in the project, such as *WorldStrat, AllClear,* and *SEN12MS-CR-TS*. Each Dataset instance stores the image paths, metadata, and any preprocessing steps like normalization or tiling. It can load a single item and return both the input and ground-truth samples. The *DataLoader* then takes this dataset and turns it into iterable batches, handling details such as batch size, shuffling, and parallel loading. Since datasets like *SEN12MS-CR-TS* contain multispectral bands and *WorldStrat* uses RGB with varying resolutions, each Dataset implementation also handles band selection and consistent resizing. This modular setup keeps the data pipeline flexible and ensures that new datasets can be added with minimal changes to the rest of the system.

On the model side, the architecture revolves around the *Model, ModelEvaluator, Metric*, and *PipelineConfig* classes. The Model class wraps a trained network and its weight file, exposing a simple forward method for inference. *ModelEvaluator* takes this model and runs it on each batch provided by the data loader, producing predictions in a consistent format. Metrics are implemented through the abstract Metric class, which defines common methods like update, reset, and compute. This allows us to plug in different evaluation metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), Spectral Angle Mapper (SAM), Relative Global Dimensional Error (ERGAS), or Quality with No Reference (QNR) without changing the pipeline logic. These metrics are important because satellite super-resolution (SR) and cloud-removal (CR) models must be judged not only on spatial accuracy (PSNR/SSIM) but also spectral consistency (SAM/QNR), which strongly affects downstream remote-sensing tasks. The *PipelineConfig* keeps all evaluation settings in one place, including dataset paths, batch sizes, device selection, and selected metrics, ensuring that each run is reproducible and easy to modify.

The repository is organized so that each component of the system has a clear place. Dataset loaders are stored under a dedicated datasets directory, and each dataset type (Cloud Removal, Super-Resolution, High-Dimensional Image Generation) has its own module with its specific preprocessing steps. Models and their checkpoints are stored under models, while all metric implementations sit inside metrics. The *pipeline/* folder contains core classes like *EvaluationPipeline, ModelEvaluator*, and *PipelineConfig*. Finally, configuration files and experiment scripts are stored in configs/ and scripts/ to keep evaluation runs

organized. This structure makes it easy to extend the project with new datasets, new evaluation modes, or additional metrics as we continue developing the system. Since we use publicly available datasets instead of maintaining our own database,

The links and instructions for accessing the datasets will be available in the README file of the project. In addition, the *scripts/* directory will contain helper utilities for dataset visualization, batch metric computation, and model comparison.

UML Class Diagram