MLOps Project Report

3D Scene Reconstruction from Multi-View Images

ME22B214 — Yash Purswani
IIT Madras | MLOps | 2024–25

April 2025

Table of Contents

Table of Contents 2
Executive Summary 4
1. Business Understanding 5
1.1 Problem Statement 5
1.2 Use Cases 5
1.3 ML and Operational Metrics 5
2. Data Sources 6
2.1 Dataset Summary 6
2.2 IMC 2025 Training Dataset 6
2.3 Data Versioning 6
2.4 Known Biases and Limitations 6
3. System Architecture 7
3.1 High-Level Overview 7
3.2 Component Breakdown 7
3.3 Ray Serve Deployment Graph 7
3.4 Networking 8
3.5 Service Port Map 8
4. ML Pipeline 9
4.1 DVC Pipeline Overview 9
4.2 Shortlist Generation (Image Pair Selection) 9
4.3 Feature Extraction and Matching (MASt3R Hybrid) 10
4.4 COLMAP Incremental Structure-from-Motion 10
4.5 Online Inference Pipeline 10
4.6 Model Selection and Promotion 10
5. API Reference 11
5.1 Job Stage Values 11
5.2 Example Workflow 12
6. Security & Compliance 13
6.1 Security Controls 13
6.2 Docker Secrets 13
6.3 Protected API Endpoints 13
6.4 CI/CD Security Workflow 13
7. MLOps Stack 14
7.1 DVC — Data and Pipeline Versioning 14
7.2 MLflow — Experiment Tracking 14

7.3 Airflow — Orchestration 14
7.4 Prometheus and Grafana — Monitoring 14
7.5 Alertmanager — Automated Retraining Trigger 15
8. Installation and Setup 16
8.1 Prerequisites 16
8.2 Docker Setup (Recommended) 16
8.3 Key Environment Variables 16
9. User Interface Guide 17
9.1 Dashboard Layout 17
9.2 Login 17
9.3 Upload Flow 17
9.4 Processing Stages (Stage Tracker) 17
9.5 3D Model Viewer 17
9.6 Stats Table 18
9.7 Downloads 18
9.8 Drift Warning Banner 18
10. Conclusions 19
10.1 Summary of Achievements 19
10.2 Known Limitations 19

Executive Summary

This report documents the design, implementation, and evaluation of a production-grade MLOps system for 3D scene reconstruction from unordered multi-view image collections. The system accepts a ZIP archive of photographs and returns a sparse 3D point cloud together with per-camera pose estimates (rotation matrix `R` and translation vector `t`).

The core inference is powered by MASt3R, a state-of-the-art transformer-based matcher, complemented by ALIKED and SuperPoint local feature detectors, and COLMAP for incremental Structure-from-Motion. The full stack is containerised with Docker, orchestrated by Apache Airflow, tracked with DVC and MLflow, and monitored via Prometheus and Grafana.

Metric	Target	Status
Number of Images	-	Tracked via MLflow
Registration Rate	≥ 90%	Monitored in Grafana
Inference Latency (50 images)	≤ 5 min	Prometheus histogram
API /health response time	≤ 200 ms	Prometheus scrape

1. Business Understanding

1.1 Problem Statement

Given a set of unordered, unstructured multi-view images captured by handheld phones, warehouse drones, or vehicle-mounted cameras, the system reconstructs the 3D environment by estimating the camera pose — rotation matrix `R` and translation vector `t` — for each image. This is the core computer vision task of Structure-from-Motion (`SfM`).

Traditional approaches rely on hand-crafted features `(SIFT, ORB)` and geometric verification `(RANSAC)`. This system replaces the feature extraction and matching steps with state-of-the-art neural networks to achieve significantly higher accuracy on challenging real-world scenes, including stairways, heritage monuments, and outdoor environments.

1.2 Use Cases

Domain	Application
AR / VR	Reconstruct real environments for immersive mixed-reality experiences
Robotics	Generate scene maps for autonomous navigation and pick-and-place tasks
Autonomous Driving	3D mapping and localisation from dashcam imagery
Cultural Heritage	Digital preservation of historical monuments and artefacts
Surveying & Topography	Georeferenced point clouds from drone surveys
Structural Inspection	Infrastructure inspection from mobile photo collections

1.3 ML and Operational Metrics

The primary quality metric is the harmonic mean of `mAA` (mean Average Accuracy) and the clusterness score, where `mAA` measures the fraction of camera poses registered within per-scene angular and translation error thresholds. A pose is counted as accurate when both its rotation error (degrees) and normalised translation error fall below scene-specific thresholds defined in train_thresholds.csv.

Operational Metric	Target	Monitoring Tool
End-to-end latency per scene	≤ 5 minutes	Prometheus histogram
API /health response time	≤ 200 ms	Prometheus scrape
Registration rate	≥ 90%	Grafana dashboard
Score overall	≥ 50%	MLflow tracking

2. Data Sources

2.1 Dataset Summary

Dataset	Source	License	Known Bias / Notes
IMC25 train	Kaggle / CVG	CC BY	Outdoor, heritage sites; high-quality imagery
IMC25 test	Kaggle / CVG	CC BY	Stairs and ET-type scenes; challenging geometry

2.2 IMC 2025 Training Dataset

The primary dataset is from the Image Matching Challenge 2025 (Kaggle / CVG Group). The training split covers 34 dataset/scene pairs including ancient monuments (`dioscuri`, `baalshamin`, `cyprus`), iconic landmarks (`taj_mahal`, `sacre_coeur`, `piazza_san_marco`), and deliberately challenging geometry (stairs, ETs). Ground-truth camera poses are provided in train_labels.csv as semicolon-separated rotation matrices (9 floats) and translation vectors (3 floats).

2.3 Data Versioning

All datasets are tracked with DVC. Key DVC pointers:

- data/train/ → data/train.dvc
- data/test/ → data/test.dvc
- data/train_labels.csv → tracked with md5: 65d4ed3b1e8...
- data/train_thresholds.csv → tracked with md5: 543f9a7c118...

2.4 Known Biases and Limitations

- Training data is heavily weighted towards outdoor heritage and landmark scenes; performance may degrade on industrial or indoor-dominated datasets.
- All training images are high quality (DSLR or recent smartphone); heavily compressed imagery may reduce accuracy.
- Repetitive structures (stairs, shelving, tiled floors) are systematically harder: shortlist generator may propose incorrect pairs and `COLMAP` may produce disconnected sub-models.

3. System Architecture

3.1 High-Level Overview

The system is composed of two largely independent layers: an offline MLOps pipeline for training and evaluation, and an online inference service for production use.

> Architecture summary: React UI → nginx TLS → Ray Serve API Gateway → Ray RPC → GPU Model Worker MLOps stack: DVC (versioning) | MLflow (tracking) | Airflow (orchestration) | Prometheus + Grafana (monitoring)

3.2 Component Breakdown

Component	Technology	Responsibility
Frontend	React + Three.js + Vite	Upload form, stage tracker, 3D pointcloud viewer, stats table
Reverse Proxy	nginx (TLS 1.3)	HTTPS termination, /api/* forwarding to ray-serve
API Gateway	FastAPI + Ray Serve	Auth, upload, job management, drift, Prometheus metrics
GPU Worker	Ray Serve + PyTorch	MASt3R pipeline, COLMAP SfM, PLY export — loaded once at startup
Experiment Tracking	MLflow	Params, metrics, artifacts per DVC run
Pipeline Versioning	DVC	Stage DAG, data and config hashing, reproducibility
Orchestration	Apache Airflow	DAG scheduling: experiment_pipeline_dag, drift_detection_dag
Metrics	Prometheus + Alertmanager	Scrape, alert rules, webhook retrain trigger
Dashboards	Grafana	Real-time visualisation of reconstruction and drift metrics
Database	PostgreSQL	Airflow metadata store

3.3 Ray Serve Deployment Graph

The backend runs as two Ray Serve deployments within a single container:

- `APIGateway` (CPU-only, 2 cores): wraps `FastAPI` via `@serve.ingress;` handles all HTTP traffic and delegates inference over Ray RPC (zero-copy via object store).

- `GPUModelWorker` (1 GPU, 4 cores): loads `MASt3R` pipeline once at startup; exposes `reconstruct()` and `ping()` methods; never reloads weights between requests.

A semaphore limits concurrent jobs to `MAX_CONCURRENT_JOBS = 1`, preventing GPU OOM errors.

3.4 Networking

All services run on a Docker bridge network named `mlops_net`. Service hostnames (e.g., `mlflow`, `airflow-apiserver`, `prometheus`) resolve internally. The network name is pinned in docker-`compose.yaml` to ensure `DockerOperator` containers spawned by Airflow can reach `mlflow:5000`.

3.5 Service Port Map

Service	Port	Description
`scene3d-ui`	5173 / 443	Frontend (HTTP / HTTPS)
`ray-serve`	8000 / 8265	API + Ray dashboard
`mlflow`	5000	Experiment tracking UI
`airflow-apiserver`	8080	Airflow web UI + REST API
`prometheus`	9090	Metrics storage
`alertmanager`	9093	Alert routing
`grafana`	3001	Dashboards
`postgres`	internal	Airflow metadata DB

4. ML Pipeline

4.1 DVC Pipeline Overview

The offline pipeline is defined in `dvc.yaml` as a six-stage DAG. DVC tracks content hashes at each stage boundary so only changed stages re-run. Every run creates a parent `MLflow` run with child runs per stage.

Stage	Script	Key Inputs	Key Outputs
validate	validate_data.py	train_labels.csv, data/train/	validation_metrics.json
eda_baselines	eda_baselines.py	train_labels.csv, data/train/	eda_baselines.json (drift baseline)
image_preprocess	image_processing.py	conf/preprocess.yaml, data/train/	data/processed/images/
prepare	prepare_submission.py	processed images, train_labels.csv	prepared_input.csv
run_pipeline	reconstruct_scenes.py	conf/mast3r.yaml, prepared_input.csv	eval_prediction.csv, reconstruction_metrics.json
evaluate	evaluate.py	eval_prediction.csv, train_labels.csv	metrics.json (mAA, clusterness, registration rate)

4.2 Shortlist Generation (Image Pair Selection)

Matching all `N×N` image pairs is computationally infeasible. An ensemble of four global descriptor retrievers selects the most promising pairs:

- `MASt3R-ASMK` (primary): vocabulary-tree retrieval using MASt3R dense descriptors with the ASMK aggregation method. Configured with `fps_n=10, fps_k=25`.
- `MASt3R-SPoC`: alternative global descriptor from the MASt3R retrieval head. `top-k=10` per query.
- `DINOv2` (DINOV2_BASE): general-purpose vision transformer; cross-domain robustness. `top-k=10`.
- `ISC` (isc_ft_v107): descriptor trained for image copy detection; effective for repeated structures. `top-k=10`.

The union of all retriever proposals forms the final shortlist. An all-pairs fallback fires when scene size is below threshold.

4.3 Feature Extraction and Matching (MASt3R Hybrid)

For each shortlisted pair the MASt3R Hybrid Matcher runs two complementary matching modes:

- Dense matching: `MASt3R` end-to-end network at 512 px resolution. Minimum 15 matches required; subsampling stride 8; pixel tolerance 5; match threshold 1.001.
- Sparse — `ALIKED` + `LightGlue`: learned keypoint detector, max 4096 keypoints at 1280 px.
- Sparse — `MagicLeap SuperPoint`: classical-style detector, max 4096 keypoints at 1600 `px, nms_radius=4, keypoint_threshold=0.0005`.

Dense and sparse matches are fused late in the pipeline (`fuse_matching_sets_late`) to maximise coverage without double-counting.

4.4 COLMAP Incremental Structure-from-Motion

Fused matches are imported into a COLMAP database. Key mapper settings:

- `mapper_min_model_size: 3` — minimum images to form a valid sub-model.
- `mapper_max_num_models: 25` — maximum disconnected sub-models allowed.
- `num_threads: 1` — avoids CPU contention with the GPU worker.

COLMAP registers images one by one via PnP, runs bundle adjustment after each batch, and filters outliers by reprojection error. Images that cannot be registered receive NaN poses in the output CSV.

4.5 Online Inference Pipeline

The API-triggered inference path mirrors the DVC pipeline but runs directly (no DVC overhead):

- 1. ZIP extraction to temporary workspace
- 2. `MASt3R` hybrid matching on `GPUModelWorker`
- 3. `COLMAP SfM` in same workspace (`pycolmap.incremental_mapping`)
- 4. PLY export via `pycolmap.Reconstruction.export_PLY()`
- 5. Voxel downsampling (`voxel_size=0.02, max_points=500,000`)
- 6. Results persisted to `/app/results/`

Configuration is loaded from `conf/best_config.yaml` (production winner) with fallback to conf/mast3r.yaml.

4.6 Model Selection and Promotion

After each DVC experiment, scripts/select_best_run.py selects the run with the highest `score_overall` from the `scene_reconstruction_dvc` experiment and saves its

YAML config to `conf/best_config.yaml`. The ray-serve container loads this file at startup, closing the experiment-to-production loop.

5. API Reference

The API Gateway (`FastAPI` + Ray Serve) listens on port 8000. All data endpoints require a `JWT` Bearer token obtained from POST /auth/token. Infrastructure probes (`/health, /ready, /metrics`) are unauthenticated.

Endpoint	Method	Auth	Description
/auth/token	POST	No	Obtain JWT. Body: {username, password}. Returns {access_token, token_type, expires_in:900}
/health	GET	No	Liveness probe. Returns {status, version, timestamp}
/ready	GET	No	GPU worker readiness. Returns {status, device}. 503 if loading
/metrics	GET	No	Prometheus text metrics scraped every 10s
/upload	POST	Yes	Upload ZIP (multipart). Query: dataset_name, scene_name. Returns {job_id}
/jobs/{job_id}	GET	Yes	Job status: stage, progress 0-100, registration_rate, has_drift, download_url
/clusters/{job_id}	GET	Yes	Per-cluster point counts and PLY filenames
/jobs/{id}/insights	GET	Yes	Drift report + recommendation text for a completed job
/download/jobs/{id}	GET	Yes	Download all PLY files as ZIP archive
/download/jobs/{id}/csv	GET	Yes	Download IMC2025-format submission CSV
/drift	POST	Yes	Drift-only check on a ZIP — no reconstruction started
/drift/trigger-retrain	POST	Yes	Manually trigger Airflow experiment_pipeline_dag

5.1 Job Stage Values

Stage	Progress	What is happening
queued	0%	Waiting for pipeline semaphore (max 1 concurrent job)
extracting	10%	Unpacking ZIP archive to temporary workspace
matching	30%	MASt3R + ALIKED + SuperPoint running on GPU
triangulating	70%	COLMAP incremental SfM in progress

Stage	Progress	What is happening
decimating	85%	Voxel downsampling point cloud to ≤500k points
success	100%	Reconstruction complete — download available
failed	0%	Unrecoverable error — see error field in response

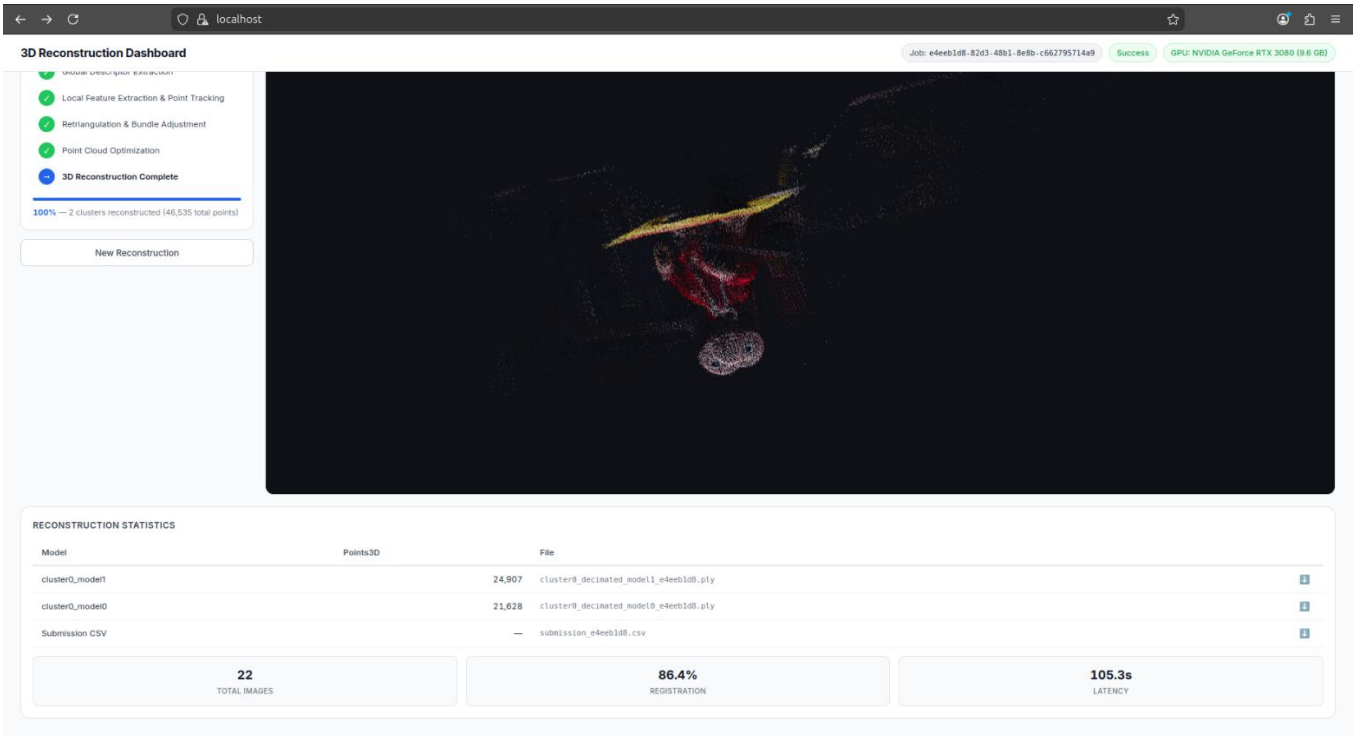
5.2 Example Workflow

```
# 1. Get token
TOKEN=$(curl -s -X POST http://localhost:8000/auth/token \
  -H 'Content-Type: application/json' \
  -d '{"username":"admin","password":"admin"}' | python3 -c \
  "import sys,json; print(json.load(sys.stdin)['access_token'])")
```

```
# 2. Upload images
JOB=$(curl -s -X POST http://localhost:8000/upload \
  -H "Authorization: Bearer $TOKEN" -F 'file=@photos.zip' \
  | python3 -c "import sys,json; print(json.load(sys.stdin)['job_id'])")
```

```
# 3. Poll until success
curl http://localhost:8000/jobs/$JOB -H "Authorization: Bearer $TOKEN"
```

```
# 4. Download PLY archive
curl -O http://localhost:8000/download/jobs/$JOB \
  -H "Authorization: Bearer $TOKEN"
```



6. Security & Compliance

6.1 Security Controls

Control	Implementation
Secrets management	Docker Secrets (file-based, /run/secrets/) — no plaintext passwords in env vars or source code
API authentication	JWT Bearer tokens, HS256, 15-minute expiry; hmac.compare_digest prevents timing attacks
Access logging	Structured JSON logs per request; IPv4 last octet anonymised; X-Request-ID header
Transport security	TLS 1.3 via nginx external termination; port 80 redirects to 443
Container scanning	Trivy on every push; fails CI on CRITICAL/HIGH; SARIF to GitHub Security tab
Dependency auditing	pip-audit on every push for both pyproject.toml files

6.2 Docker Secrets

Two secrets are provisioned:

- `jwt_secret` → `/run/secrets/jwt_secret` (used by ray-serve for token signing)
- `grafana_admin_password` → `/run/secrets/grafana_admin_password` (Grafana admin)

> Generate secrets before first deploy: ./generate_secrets.sh && chmod 644 ./secrets/* The secrets/ directory is in .gitignore and is never committed.

6.3 Protected API Endpoints

The following endpoints require a valid `JWT` token: `/upload`, `/drift`, `/drift/trigger-retrain`, `/status/{job_id}`, `/jobs/{job_id}`, `/download/*`, `/clusters/*`, `/jobs/{job_id}/insights`.

Unauthenticated (infrastructure only): `/health`, `/ready`, `/metrics`, `/auth/token`.

6.4 CI/CD Security Workflow

GitHub Actions workflow `.github/workflows/security.yml` runs on every push to main/develop and every pull request. It scans three Docker images (ray-serve, scene3d-ui) with Trivy and audits Python dependencies with `pip-audit`. The pipeline fails on CRITICAL or HIGH severity findings with available fixes.

7. MLOps Stack

7.1 DVC — Data and Pipeline Versioning

DVC defines the pipeline as a DAG in `dvc.yaml`. Each stage specifies commands, dependencies (deps), and outputs (outs) with content-hash tracking. This ensures full reproducibility: any combination of code, data, and config can be exactly reproduced from git commit + DVC pointers.

Run the full pipeline:

```
PARENT_ID=$(python scripts/start_parent_dvc_run.py | head -n 1)
MLFLOW_PARENT_RUN_ID="$PARENT_ID" dvc repro
python scripts/select_best_run.py
```

7.2 MLflow — Experiment Tracking

Every `dvc repro` invocation creates a parent `MLflow` run with child runs per stage. Each child run logs flattened YAML config parameters, stage metrics (registration rate, score, `mAA`, clusterness per dataset), and artifacts (eval CSV, PLY file, config YAML, git status). The best run's config based on `score_overall` is promoted to `conf/best_config.yaml`.

7.3 Airflow — Orchestration

Two DAGs are deployed:

- `experiment_pipeline_dag`: manually triggered. `FileSensors` wait for required data files; `DockerOperator` runs `dvc repro` in an ephemeral container (`DooD`); `BashOperator` calls `select_best_run.py`; `EmailOperator` notifies on success.
- `drift_detection_dag`: scheduled every 30 minutes. Queries Prometheus for `feature_drift_status` metric; branches to email notification on drift > 0.

7.4 Prometheus and Grafana — Monitoring

Prometheus scrapes the Ray Serve /metrics endpoint every 10 seconds, MLflow every 30 seconds, Airflow every 15 seconds, and node-exporter continuously. Key metrics tracked:

- `api_requests_total` — requests by method, endpoint, status
- `inference_latency_seconds` — histogram with buckets at 10s, 30s, 60s, 120s, 180s, 300s, 600s, 900s
- `registered_images_ratio` — fraction of images placed in model
- `model_server_ready` — `1` if GPU worker is ready

7.5 Alertmanager — Automated Retraining Trigger

Alert rules in `monitoring/alert_rules.yml` fire on `FeatureDriftDetected`, `InputBrightnessDrift`, `InputContrastDrift`, and `PerformanceDecay`. `Alertmanager` routes these to the Airflow API webhook, which triggers `experiment_pipeline_dag` automatically. `ModelServerDown` suppresses drift alerts (they are unreliable when the model server is down).

8. Installation and Setup

8.1 Prerequisites

- Docker >= 24.0 and Docker Compose >= 2.20
- NVIDIA GPU with CUDA 12.6 support + NVIDIA Container Toolkit
- Git and Git LFS
- At least 16 GB GPU VRAM and 32 GB system RAM recommended

8.2 Docker Setup (Recommended)

```
git clone https://github.com/your-org/MLOps-Project-ME22B214.git
cd MLOps-Project-ME22B214 && git lfs pull
# Download IMC2025 dataset
kaggle competitions download -c image-matching-challenge-2025
unzip image-matching-challenge-2025.zip -d data/
```

```
# Configure environment
./setup_env.sh           # interactive .env setup
./generate_secrets.sh    # jwt_secret + grafana_admin_password
chmod 644 ./secrets/*
./generate-certs.sh      # TLS certificates
```

```
# Clone external dependencies
cd extra/ && git clone https://github.com/jenicek/asmk
git clone https://github.com/naver/croco && cd ..
```

```
# Launch full stack
docker compose --profile inference up --build -d
docker compose ps    # verify all services healthy
```

8.3 Key Environment Variables

Variable	Default	Purpose
AUTH_ENABLED	true	Set false to disable JWT auth (dev only)
JWT_EXPIRY_SECONDS	900	Token lifetime in seconds
SCENE3D_MAX_UPLOAD_MB	500	Maximum upload ZIP size
SCENE3D_VOXEL_SIZE	0.02	Voxel size for point cloud decimation
SCENE3D_MAX_POINTS	500000	Max point cloud size after decimation
PIPELINE_CONFIG	conf/best_config.yaml	Primary pipeline config path
PIPELINE_CONFIG_FALLBACK	conf/mast3r.yaml	Fallback if primary not found

9. User Interface Guide

9.1 Dashboard Layout

The web application (`https://localhost:5173`) presents a four-zone dashboard: Header (title, job ID, stage indicator) | Left Panel (upload form, stage tracker, action buttons) | Right Panel (3D model viewer) | Bottom (stats table).

9.2 Login

A login form accepts username/password and calls POST /auth/token. On success a `JWT` is stored in browser localStorage. Tokens expire after 15 minutes; the app auto-redirects to login on a 401 response.

9.3 Upload Flow

- Select or drag a ZIP archive of images onto the upload zone.
- Optionally set Dataset Name (default: custom) and Scene Name (default: scene_01).
- Click Upload & Reconstruct. The button disables while a job is running.
- The API immediately runs a drift check on the uploaded images and creates a job record.
- The Stage Tracker appears and polls GET `/jobs/{job_id}` every few seconds.

9.4 Processing Stages (Stage Tracker)

Stage	Progress	User-visible message
Extracting	10%	Extracting images from ZIP...
Matching	30%	Running MASt3R feature matching on GPU...
Triangulating	70%	Inference complete — triangulating...
Decimating	85%	Optimising N clusters...
Success	100%	N clusters reconstructed (X,XXX total points)

9.5 3D Model Viewer

Three.js renders the decimated point cloud (≤500k points). Controls: left-click drag = rotate; right-click drag = pan; scroll = zoom; double-click = reset camera. A cluster selector appears when multiple clusters are present. The displayed cloud is the decimated version; full-density PLY files are available for download.

9.6 Stats Table

The bottom panel displays: Total Images, Registered Images, Registration Rate (%), Total 3D Points, Drift Detected (Yes/No), Drift Severity (low/moderate/high), and a per-cluster breakdown with individual download links.

9.7 Downloads

Two download options appear on job success: download .ply — a ZIP archive of per-cluster PLY files (open in `MeshLab`, `CloudCompare`, or Blender); Download CSV — the raw IMC2025 submission-format CSV with rotation matrices and translation vectors (`NaN` for unregistered images).

9.8 Drift Warning Banner

When drift is detected during upload, a yellow warning banner shows severity and which statistics drifted (brightness, contrast, sharpness, aspect ratio). High-severity drift triggers automatic Airflow retraining.

10. Conclusions

10.1 Summary of Achievements

- End-to-end MLOps pipeline from raw images to 3D point cloud with full experiment reproducibility via `DVC` and `MLflow`.
- State-of-the-art MASt3R hybrid matcher integrated with `COLMAP SfM` for high-accuracy pose estimation.
- Production-grade Ray Serve deployment with GPU model worker, `JWT` authentication, structured access logging, and TLS 1.3.
- Automated drift detection with Prometheus alerting and Airflow retraining trigger.
- CI/CD security scanning with Trivy and `pip-audit` on every push.
- React + Three.js frontend with interactive 3D viewer and real-time job progress tracking.

10.2 Known Limitations

- Maximum one concurrent reconstruction job (single GPU semaphore); horizontal scaling would require a job queue with multiple GPU workers.
- `JWT` token expiry (15 min) requires re-authentication for long workflows; a refresh token mechanism would improve UX.
- The drift baseline is computed once from training data; it does not update incrementally as production data accumulates.
- Only sparse `SfM` is implemented; dense depth maps (MVS) are not produced.