Documentation: Amini GeoFM Decoding the Field Challenge

By: AI Matrix Team

All Important Files:
https://drive.google.com/drive/folders/1s1BwHftR_hTJBdiUzFzRh2M2FvG7tvQe?usp=sharing
How to run this:
https://drive.google.com/file/d/1t5-6t1dTDleWFMdPL3Z-9AHJQJVlTp89/view?usp=drive_link
Folder link:
https://drive.google.com/drive/folders/1IONtRQyO9abwtYVykUCWu0f-F54YpzxE?usp=sharing
Note books link:
https://drive.google.com/file/d/1r9_9YDs96BkpczpGKVUdyCV3Fk7rV6iS/view?usp=sharing
https://drive.google.com/file/d/1a2qbPw1ryHi-8-PJpB55ACpSRraLTefC/view?usp=sharing

Summary

The Amini GeoFM Decoding the Field Challenge Crop Type Classification Solution predicts plantation crop types (**cocoa, rubber, oil**) from multivariate satellite time-series data. The system fuses **traditional statistical + spectral vegetation features** with **deep time-series embeddings from PatchTST** and feeds the combined representation into an **ensemble of XG Boost, Logistic Regression, and SVM models**. Outputs are class probabilities and final predicted labels, delivered as CSV for downstream analytics and leaderboard submission. Current private leaderboard performance: **0.7473 macro F1**.

1. Overview & Objectives

Purpose

Identify and classify agricultural land use from satellite remote sensing to support monitoring, resource allocation, sustainability efforts, and yield forecasting.

Scope

- Input: Labeled and unlabeled satellite spectral time-series in CSV form.
- Output: Predicted crop type per unique field/ID.
- Operating Environment: Notebook + cloud drive (e.g., Google Drive path `/content/drive/MyDrive/`).

Objectives

- Build a hybrid ML + DL pipeline for crop type classification.
- Engineer robust temporal, spectral, and index-based features from raw time-series.
- Generate deep sequence embeddings via **PatchTST** for representational lift.
- Evaluate models with **Accuracy** and **macro F1**; track **public & private leaderboard** scores.
- Deliver reproducible inference pipeline + exportable predictions.

Satellite Data Extraction with GEE

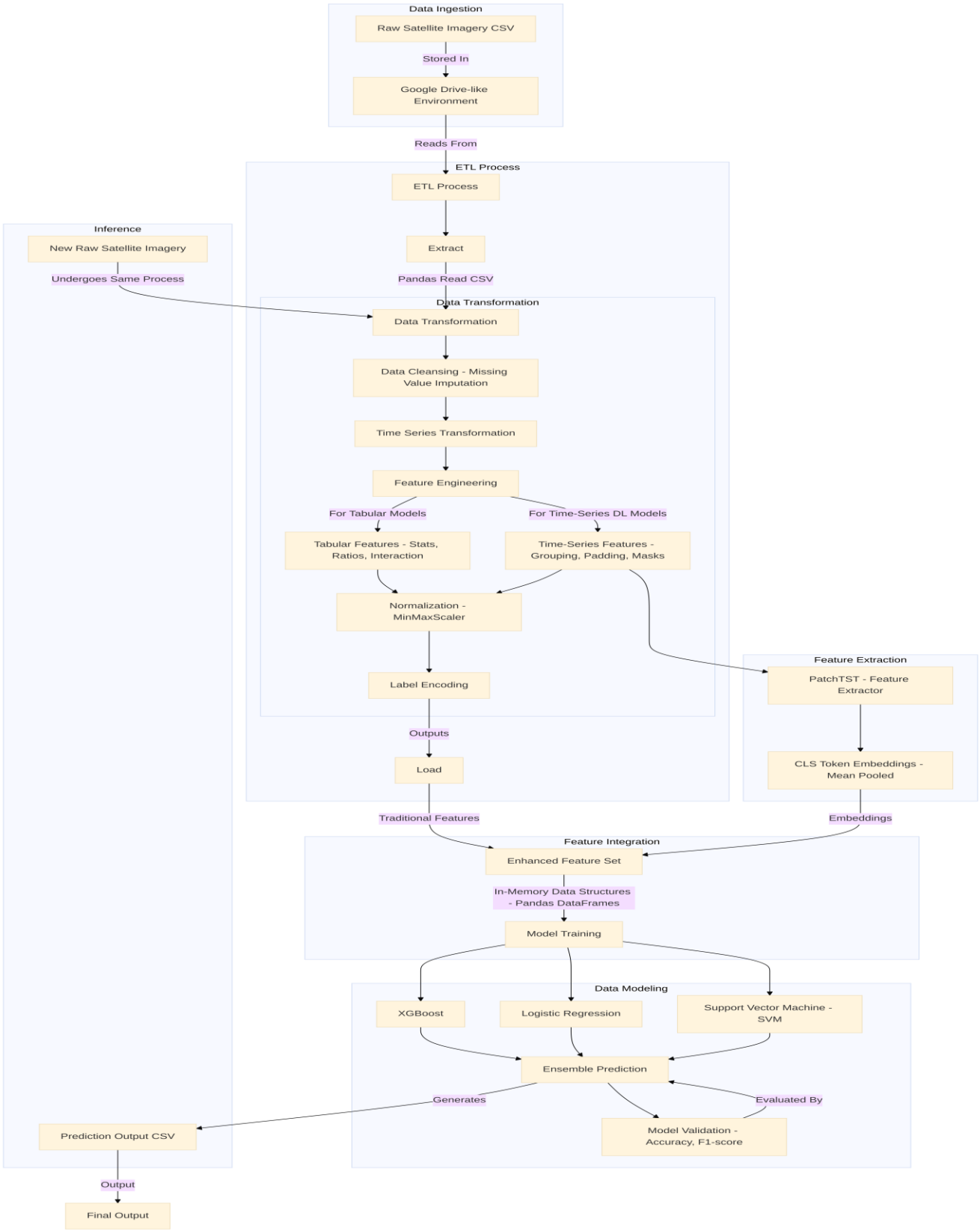
Code Script Link https://code.earthengine.google.com/8e6972c781ae100e5f6608afac8ef3e4

This script extracts a cloud-free Sentinel-2 time-series dataset for crop classification, mapped to uniquely identified training points:

1. **Region of Interest (ROI)**:
   Defined as a rectangular polygon bounding the area of interest in West Africa.
2. **Training Data Enhancement**:
   Each training point is assigned a unique ID (`unique_id`) for traceability across time-series samples.
3. **Sentinel-2 SR Collection**:
   Filters applied:
   - Spatially within the ROI
   - Temporally from Jan 2021 to Dec 2022
   - Cloud coverage < 20%
4. **Cloud Masking**:
   Excludes pixels labeled as clouds, shadows, or cirrus using the Scene Classification Layer (SCL). Only relevant spectral bands (B2–B12) are retained.
5. **Time-Series Sampling**:
   For each cleaned image:
   - Spectral values are sampled at training point locations
   - The sampling date is recorded as a new feature (`date`)
   - All samples are aggregated into a single time-series dataset
6. **Export**:
   The final flattened time-series table is exported to Google Drive in CSV format, titled: `Sentinel2_CropTimeSeries_withUID`.

2. Solution Architecture

The end-to-end workflow is an orchestrated, modular ML pipeline.


Data Ingestion
Raw Satellite Imagery CSV
Stored In
Google Drive-like Environment
Reads From
ETL Process
ETL Process
Extract
Pandas Read CSV
Data Transformation
Data Transformation
Data Cleansing - Missing Value Imputation
Time Series Transformation
Feature Engineering
For Tabular Models
For Time-Series DL Models
Tabular Features - Stats, Ratios, Interaction
Time-Series Features - Grouping, Padding, Masks
Normalization - MinMaxScaler
Label Encoding
Outputs
Load
Inference
New Raw Satellite Imagery
Undergoes Same Process
Feature Extraction
PatchTST - Feature Extractor
CLS Token Embeddings - Mean Pooled
Traditional Features
Embeddings
Feature Integration
Enhanced Feature Set
In-Memory Data Structures - Pandas DataFrames
Model Training
Data Modeling
XGBoost
Logistic Regression
Support Vector Machine - SVM
Ensemble Prediction
Generates
Evaluated By
Model Validation - Accuracy, F1-score
Prediction Output CSV
Output
Final Output

3. ETL Process

3.1 Extract

- **Data Sources:** `train.csv`, `Train.csv`, `test.csv`, `Test.csv` in `/content/drive/MyDrive/`.
- **Format:** Comma-separated; each row corresponds to an observation at a timestamp for a given `unique_id` across multiple spectral bands.
- **Method:** `pandas.read_csv()` loads directly into memory DataFrames.

3.2 Transform

Core Transformation Steps

- **Time Series Conversion:** Convert `time` column to `datetime` dtype.
- **Label Encoding:** Convert `crop_type` categorical labels → integer codes (train only; map stored for inference).
- **Missing Value Imputation:** Column-wise mean imputation for numeric fields (alt strategies TBD).
- **Traditional Statistical Aggregations (grouped by `unique_id`):** mean, std, min, max, median, count, sum, variance across spectral bands.
- **Temporal Derived Features:** month, day_of_year (DOY), time_since_first_observation.
- **Vegetation Indices:** NDVI, NDRE, GCI, SAVI (per timestamp; optional aggregation to summary stats).
- **Interaction Features:** Pairwise and/or ratio/contrast features across bands (configurable).

PatchTST Embedding Generation

- Raw time-series (per `unique_id` sequence) is passed to a **PatchTST** model.
- Extract **CLS token** representation; mean-pool if multi-segment.
- Produces dense temporal embeddings capturing seasonal dynamics.

Feature Integration & Scaling

- Join: `[stats + temporal + veg idx + interactions]` ⊕ `[PatchTST embeddings]`.
- Scale all numeric features with **MinMaxScaler** fit on training data; apply same scaler to validation/test/inference.

3.3 Load

- Final engineered + embedded features held in memory as `pandas.DataFrame` objects.
- Intermediate artifacts (aggregated features, embeddings) optionally cached to CSV/parquet for reproducibility.

4. Data Modeling

4.1 Model Roles

Component	Role	Notes
PatchTST	Feature extractor	Produces sequence embeddings; not used as a standalone classifier in current version.
Random Forest	Nonlinear ensemble tree classifier	Hyperparameter tuned (GridSearchCV). Key contributor in ensemble.
Logistic Regression	Linear baseline	Interpretable weights; stable probability estimates.
Support Vector Machine (SVM)	Margin-based classifier	Kernelized or linear (configurable); probabilities via calibration or Platt scaling.

4.2 Feature Set Summary

- Statistical aggregates per field.
- Temporal signals (month, DOY, duration).
- Vegetation indices & ratios (NDVI, NDRE, GCI, SAVI).
- Interaction features across bands.
- PatchTST CLS embeddings.
- All scaled via MinMaxScaler.

4.3 Training Configuration

Random Forest

- Tuned parameters: `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`, (optional) `max_features`.
- GridSearchCV or RandomizedSearchCV over defined grid.

Logistic Regression

- Solver: `lbfgs` / `saga` (depending on data scale & regularization).
- Regularization: `C`, `penalty` (L2 typical).

SVM

- Kernel: linear or RBF (most common).
- Key params: `C`, `gamma` (if RBF).
- Probability estimates enabled (`probability=True`) or externally calibrated.

Training Inputs: Combined scaled feature matrix; stratified splits recommended.

4.4 Validation Strategy

- **Cross-Validation:** Used during hyperparameter tuning (GridSearchCV for RF; can extend to LR/SVM).
- **Holdout/Test Evaluation:** Independent set to assess generalization pre-ensemble.

- **Leaderboard Monitoring:** Public vs private leaderboard for competition performance.

5. Ensemble Prediction & Inference

5.1 Inference Data Flow

1. Ingest new CSV data.
2. Apply full ETL + feature build pipeline (must reuse training encoders/scalers).
3. Generate PatchTST embeddings for each `unique_id`.
4. Merge embeddings with engineered features.
5. Score with each trained base model (RF, LR, SVM) → per-class probabilities.
6. Combine via **soft voting / weighted averaging** (current approach).
7. Argmax → Final predicted crop label.

5.2 Output Artifacts

- **Per-record probabilities** (optional diagnostic export).
- **Final label predictions** → `ensemble_predictions_final.csv`.
- **Submission-ready file** (competition formatted).

5.3 Interpretation Notes

• Highest probability = predicted crop.
• Compare class probabilities across models to diagnose disagreement.

6. Runtime Characteristics

Stage	Observed Runtime*
Preprocessing Notebook	~38 seconds
Training Notebook	~14 min 23 sec

*Runtimes observed on the reported notebook environment; hardware, data size, and I/O bandwidth will affect totals.

7. Performance Metrics

7.1 Core Evaluation Metrics

- **Accuracy** – Overall correctness across all samples.
- **Macro F1-Score** – Harmonic mean of precision & recall averaged equally across classes (recommended for imbalance).

7.2 Reported Leaderboard Scores

Leaderboard	Macro F1
Public	0.730614224
Private	0.747330083

7.3 Additional Metrics (Not Yet Implemented)

- Per-class precision/recall.
- Calibration error (Brier score, reliability curves).
- Confusion matrix by crop type.

8. Error Handling & Logging

Current State

- Minimal explicit error trapping; relies on notebook cell output.
- No centralized logging or structured error reporting.

Recommended Improvements

- Add try/except wrappers around I/O and model training blocks.
- Implement `logging` module with INFO/WARN/ERROR levels.
- Persist pipeline config + run metadata (timestamps, hyperparams, git commit) for traceability.

9. Maintenance, Monitoring & Scaling

Gaps in Current Implementation

- No automated data drift detection.
- No scheduled retraining or model version registry.
- No MLOps integration (CI/CD, model registry, experiment tracking).

Suggested Roadmap

1. **Versioning:** Track data schema hashes + model artifacts.
2. **Experiment Tracking:** Use MLflow or Weights & Biases.
3. **Batch Scoring API:** Containerize ETL+inference; run on schedule.
4. **Scalability:** Migrate heavy feature generation to distributed compute (e.g., Dask, Spark).
5. **Monitoring Dashboards:** Track class distribution drift and performance decay on new data.

10. Reproducibility Checklist

- Freeze Python + library versions (requirements.txt).
- Store label encoding map.
- Persist MinMaxScaler fit parameters.
- Save trained model artifacts (joblib/pkl).
- Script full ETL so no manual Excel preprocessing is ever required.
- Log random seeds + splits.

11. Known Limitations

- Assumes consistent timestamp frequency; irregular sampling may degrade PatchTST embeddings if not padded/aligned.
- Mean imputation may bias spectral metrics where missingness is non-random.
- Vegetation indices require valid band reflectances; missing or zero-division cases need guards.
- PatchTST currently frozen after feature extraction; fine-tuning could improve accuracy.

12. Future Enhancements

Area	Enhancement	Expected Impact
Data Quality	Advanced gap filling (interpolation, lightGBM impute)	Better index stability
Features	Phenology curve descriptors, temporal Fourier components	Capture seasonal patterns
Modeling	Gradient Boosting (XGBoost/LightGBM/CatBoost)	Stronger tabular baselines
Deep TS	Fine-tune PatchTST or compare with Temporal Fusion Transformer	Richer representations
Ensemble	Meta-learner stacking (logit blend)	Higher leaderboard scores
Ops	Automated pipeline + drift alerts	Production readiness

13. Quickstart (Notebook Workflow)

1. Mount Drive & set data paths.
2. Load train/test CSVs.
3. Run preprocessing cell (datetime, impute, encode).
4. Generate features + vegetation indices.
5. Run PatchTST embedding notebook; export embeddings.
6. Merge all features; scale.

7. Train RF, LR, SVM (with CV tuning where configured).
8. Generate per-model probabilities on validation/test.
9. Ensemble → final predictions.
10. Save to `ensemble_predictions_final.csv`.

14. Appendix

14.1 Key Formulas

NDVI: (NIR − Red) / (NIR + Red)
NDRE: (NIR − RedEdge) / (NIR + RedEdge)
GCI: (NIR / Green) − 1
SAVI: ((NIR − Red) / (NIR + Red + L)) × (1 + L), L≈0.5 (soil brightness correction)

14.2 Minimal Column Expectations

- `unique_id`
- `time`
- Spectral bands: e.g., `red`, `nir`, `green`, `red_edge` (names vary)
- `crop_type` (train only)

14.3 Environment Snippet

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

Contact :

Khushi malik

Khushimalik511263@gmail.com
https://www.linkedin.com/in/khushi-6b972b280/

Aman Deva

devaaman8@gmail.com
https://www.linkedin.com/in/aman-deva/