

MTP Thesis: Restoration Site Identification

Apoorva Dewan `Apoorva.Dewan.cs521@cse.iitd.ac.in`

Ojasvi Bansal `Ojasvi.Bansal.cs521@cse.iitd.ac.in`

23rd June 2026

Contents

1	Introduction	4
1.1	Motivation and Problem Statement	4
1.2	Our Approach	5
1.3	Scope and Limitations	5
1.4	Thesis Structure	5
2	GEE Applications	7
2.1	Introduction	7
2.1.1	What is the Environmental Similarity App?	7
2.1.2	Use Cases	7
2.1.3	Example: Pandalgudi Trial	8
2.1.4	What You Can Do	8
2.2	Main App: Build rules from scratch	8
2.2.1	Interface Overview	8
2.2.2	Step-by-Step Workflow	9
2.2.3	Tips for Best Results	22
2.3	Accompanying App: Example Restoration Sites	22
2.3.1	Purpose and Relationship to the Main App	22
2.3.2	Predefined Restoration Sites	23
2.3.3	Interface Design	23
2.3.4	Workflow	24
2.4	Additional Information	25
2.4.1	Data Sources	25
2.4.2	Quick Workflow Summary	26
2.4.3	Future Enhancements	26
2.4.4	Contact & Feedback	26
2.4.5	Acknowledgements	27
3	Spatial Clusters	28
3.1	Introduction	28
3.2	Methodology	28
3.2.1	Generation of S2 Level-13 Grids over India	28
3.2.2	Creation of Major Land-Cover Layers	29
3.2.3	Merging GeoTIFF Tiles	30

3.2.4	Computation of Land-Cover Fractions per S2 Grid	30
3.2.5	Determine the Optimal Spatial Cluster Count	32
3.2.6	Spatial Cluster Generation	33
3.2.7	Spatial Cluster Refining	33
3.2.8	Rasterization of Spatial Clusters	33
3.3	Spatial Cluster Classification	34
3.3.1	Cluster 0 : Mostly Trees	34
3.3.2	Cluster 1 : Intensive croplands	34
3.3.3	Cluster 2 : Mostly shrublands	35
3.3.4	Cluster 3 : Himalayan areas	35
3.3.5	Cluster 4 : Mostly wetlands and riverine areas	35
3.3.6	Cluster 5 : Urban and adjoining agriculture areas	36
3.3.7	Cluster 6 : Crops and trees	36
3.3.8	Cluster 7 : Bare and shrub areas	36
3.3.9	Cluster 8 : Crops and shrubs	37
3.3.10	Cluster 9 : Bare and shrub areas	37
4	Phenology Clusters	38
4.1	Introduction	38
4.2	Methodology	38
4.2.1	Constructing a Harmonized Landsat NDVI Time Series	39
4.2.2	Harmonic Regression on the NDVI Time Series	41
4.2.3	Sampling and Clustering by Land-Cover Class	42
4.2.4	Curve Reconstruction and Cluster Validation	44
4.3	Year-on-Year Cluster Assignment and Stability	45
4.3.1	Per-window Harmonic Coefficient Images	45
4.3.2	Nearest-Centroid Cluster Assignment per Pixel	46
4.3.3	Per-grid Majority Vote and Export	47
4.4	Summary	48
5	Case Study: Auroville Botanical Gardens	49
5.1	Introduction	49
5.1.1	Background	49
5.2	Building the Rule Set for Pandalgudi	49
5.2.1	Reference Site and Parameter Selection	50
5.2.2	Exported Rule Set	50
5.2.3	Generated Report	51
5.3	Results: Identifying Candidate Reference Sites	52
5.3.1	Within the Pandalgudi Ecoregion	52
5.3.2	Extending to Surrounding Ecoregions	53
5.4	Current Status: Ground Verification	53
5.5	Next Steps	54
A	Technical Guide for the Apps	55
A.1	Overview	55

A.2	Code Organisation	55
A.3	How the Rule Engine Works	56
A.4	Adding a New Environmental Layer	57
A.4.1	Step 1: Create the Layer Module	57
A.4.2	Step 2: Register the Layer in <code>main.js</code>	59
A.4.3	Step 3: Include the Layer in Rule Evaluation	60
A.4.4	Step 4: Extend JSON Export	60
A.4.5	Step 5: Extend JSON Import	61
A.4.6	Step 6: Test the Layer	61
A.5	Adding a New Masking Layer	61
A.6	Adding a New Predefined Site to the Accompanying App	62
A.6.1	The <code>projectRegions</code> Array Structure	62
A.6.2	Step-by-Step Instructions to Add a New Site	63
A.7	GEE-specific Notes for Developers	63

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Ecological degradation is one of the most pressing environmental challenges of our time. Across India and the broader Global South, land that once supported diverse and productive ecosystems has been transformed by centuries of agricultural expansion, uncontrolled grazing, urbanisation, and industrial activity. The resulting loss of biodiversity, soil fertility, and water-holding capacity has cascading consequences for the communities that depend on these landscapes for their livelihoods and wellbeing.

Ecological restoration - the deliberate process of assisting the recovery of a degraded, damaged, or destroyed ecosystem - has emerged as a central strategy in global conservation and climate commitments. However, translating these commitments into effective, on-the-ground action requires answering a deceptively difficult question: *where should restoration happen, and what should it look like?*

Answering this question at scale is not straightforward. Restoration practitioners and policymakers need two things simultaneously. First, they need *candidate sites* for intervention: locations that are currently degraded but retain sufficient ecological potential for recovery. Second, they need *reference sites*: places that represent the healthy, intact end-state that a restoration project is working towards. Without reference sites, restoration has no ecological target. Without a systematic method for identifying candidate sites, restoration remains ad hoc, dependent on individual knowledge and field access.

Existing approaches to site identification are largely manual. Ecologists rely on local expertise, field surveys, and coarse-resolution national datasets to identify where degradation is occurring and where intact analogues might be found. These approaches are time-consuming, difficult to scale, and inherently limited by the geographic reach of any single team. There is a clear need for a data-driven, reproducible, and scalable tool that can assist restoration practitioners in identifying both reference and candidate sites across large landscapes.

1.2 Our Approach

This thesis presents the design, development, and evaluation of the **Environmental Similarity App** - an interactive, rule-based decision-support tool built on the **Google Earth Engine (GEE)** platform. The core idea is straightforward: if we can characterise the ecological fingerprint of a known restoration site in terms of its rainfall, temperature, elevation, soil type, terrain, fire history, and land cover change trajectory, we can then search for other locations across a landscape that share that fingerprint.

The tool implements this as a configurable rule-based similarity search. Users define a reference point, set thresholds across multiple environmental datasets, and run an inference step that highlights pixels across a selected ecoregion that satisfy all specified conditions simultaneously. The result is a spatially explicit map of locations that are ecologically similar to the reference site - which can be interpreted either as candidate sites for restoration or as intact reference sites, depending on the use case.

A key design decision was to make the tool accessible to two distinct user groups: ecologists and researchers who need full configurability over every parameter, and field practitioners who need quick, pre-configured results they can explore without deep technical knowledge of the underlying datasets. This led to the development of two complementary applications - a Main App for custom rule-building and an Accompanying App populated with pre-configured rule sets for a curated set of restoration sites across India.

In addition to the similarity search tool, this thesis presents two analytical methods that provide deeper ecological context. The first is **spatial clustering**, which groups regions with similar multi-dimensional environmental profiles into discrete ecological zones. The second is **phenology clustering**, which analyses time-series vegetation data to identify locations that share the same seasonal growth patterns as a reference site - a dimension of ecological similarity that static environmental layers cannot capture.

1.3 Scope and Limitations

The current version of the tools is designed and validated for use within India, making use of India-specific datasets for land use, land cover, and land degradation. The environmental layers used for the similarity search (precipitation, temperature, elevation, soil, terrain, fire) are global datasets and could in principle support extension to other geographies.

It is important to note that this system is **experimental and intended as a decision-support tool**, not a replacement for field expertise. Results should always be validated with on-the-ground knowledge before informing restoration planning. The tool is designed to narrow the search space, generate hypotheses for field investigation, and support evidence-based decision-making - not to prescribe restoration interventions autonomously.

1.4 Thesis Structure

The remainder of this thesis is organised as follows:

Chapter 2: GEE Applications describes the Environmental Similarity App in detail. It covers the conceptual framework, the two-application architecture (Main App and Accompanying App), the full step-by-step workflow, all environmental and masking layers, data sources, and guidance on interpreting results.

Chapter 3: Spatial Clusters presents a complementary analytical approach to site identification using clustering of spatial environmental features derived from multi-layer geospatial data. The methodology groups regions with similar environmental and land-cover characteristics, enabling the identification of distinct spatial patterns. The chapter describes the clustering approach, selection of algorithms, and how resulting spatial cluster maps can be used alongside the similarity app to filter, interpret, validate, and contextualise environmentally similar sites.

Chapter 4: Phenology Clusters introduces a time-series-based approach to ecological similarity that captures seasonal vegetation dynamics. It describes how NDVI time-series data are extracted and clustered to identify locations with phenology patterns similar to those of a reference site. This offers a complementary and in some cases more sensitive indicator of ecological similarity than static environmental variables alone.

Appendix A: Technical Guide for the Apps is aimed at developers and researchers who wish to understand, modify, or extend the applications. It explains how Google Earth Engine scripts are organised across the two application folders (`gee-app` and `gee-app2`), describes the role of each code module, and provides step-by-step instructions for common modifications, such as adding a new environmental layer or changing the masking logic. All source code is openly available, and links to the relevant GEE scripts and GitHub repository are provided throughout.

Chapter 2

GEE Applications

2.1 Introduction

2.1.1 What is the Environmental Similarity App?

This chapter describes the Environmental Similarity App, an interactive decision-support tool built on the *Google Earth Engine (GEE)* platform. It performs a **similarity search for environmental conditions**: identifying regions whose ecological characteristics match those of a specified area of interest, either today or at a point in the past, using historic data. The app is available in two versions:

- **Main App**- build your own rule set from scratch for any restoration site by configuring environmental layers, temporal ranges, and masking parameters step by step.
- **Accompanying App**- explore precomputed rule sets for a curated set of reference restoration sites across India, with all parameters already configured and ready to run.

The app integrates multiple datasets (terrain, precipitation, fire occurrence, temperature, elevation, land degradation, soil cover, and land use/land cover (LULC)) to build rule-based models that detect sites with similar ecological signatures.

2.1.2 Use Cases

At its core, the app always performs the same three steps:

1. **Define** a starting area of interest (your reference polygon or point).
2. **Find** regions with similar ecological conditions today, or that were similar in the past using historic data.
3. **Overlay** additional datasets on the results to refine and enhance them for your specific objective.

This three-step flow makes it a powerful multi-purpose tool. Common use cases include:

- **Finding reference sites:** identify high-intactness areas within the ecological region of a restoration project to understand what a healthy end-state looks like. For example, overlay a dataset like *Forests of the World 2020* on the similarity results to locate intact forest areas of the same ecosystem type, which can then be visited for field studies.
- **Scaling up successful restoration:** find areas that today look like what a successful restoration site looked like at the *start* of restoration. This helps inventorise locations with high potential for reproducing that restoration success at scale.
- **Identifying habitat corridors:** characterise the key environmental conditions of a specific habitat (e.g. for a particular species) and use the similarity search to identify the best candidate areas for creating habitat corridors.

2.1.3 Example: Pandalgudi Trial

The following illustrates how the three-step flow was applied in a real trial:

1. **Area of interest:** The restoration project boundary at Pandalgudi.
2. **Similarity search:** The tool identified regions with similar rainfall, temperature, elevation, soil type, and other ecological conditions.
3. **Overlay and refinement:** The *Forests of the World 2020* dataset was overlaid to filter results and identify likely reference sites for follow-up field studies.



Note

This app is experimental and under active development. Results should be treated as indicative and validated with field knowledge before planning restoration interventions.

2.1.4 What You Can Do

- Select a reference restoration site anywhere within an ecoregion
- Define its ecological fingerprint using multiple environmental datasets
- Detect areas that have undergone land cover change (degradation or recovery)
- Find similar sites across the landscape that match your reference site's conditions
- Refine and filter results using masking layers and post-processing tools

2.2 Main App: Build rules from scratch

2.2.1 Interface Overview

The app interface is divided into three main areas:

Interface Area	Purpose
Left Map Panel	Displays reference site rules and validation pixels (shows the ecological signature of your chosen site).
Right Map Panel	Displays inference results (the areas across the ecoregion that match your reference site's conditions).
Control Panel	Step-by-step workflow panel where you configure all analysis parameters, from ecoregion selection through to fine-tuning.

Table 2.1: Overview of the app interface areas

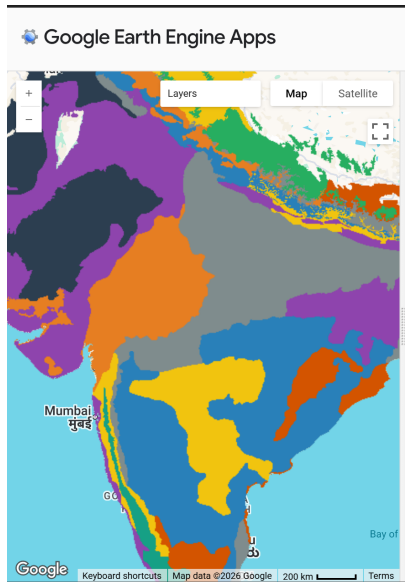


Figure 2.1: Left Map Panel



Figure 2.2: Right Map Panel

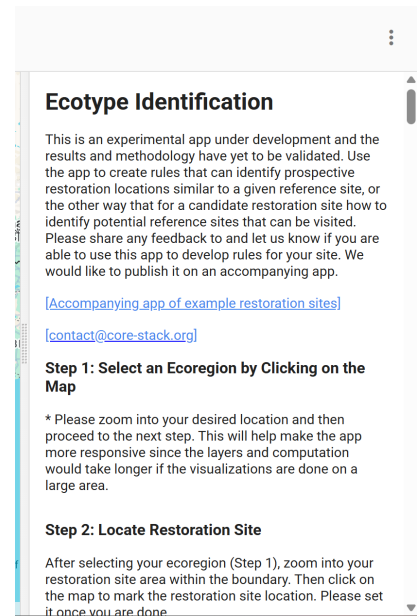


Figure 2.3: Control Panel

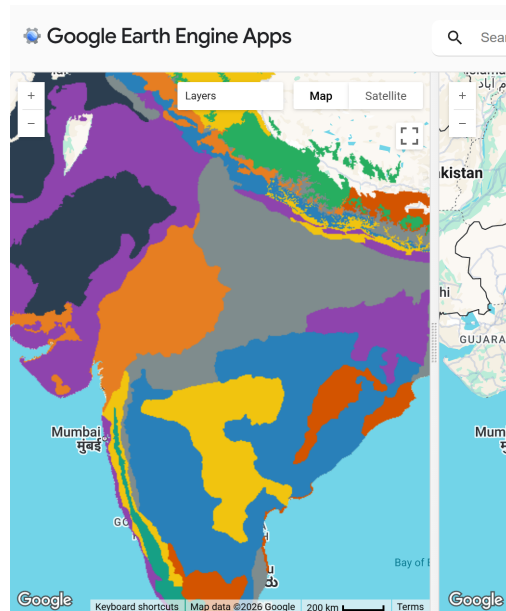
2.2.2 Step-by-Step Workflow

The workflow consists of **9 sequential steps**. Complete them in order (each step builds on the previous one).

Step 1: Select an Ecoregion

Action

Choose the broad ecological region for your analysis by clicking on the map.



- **How to Do It**

- Zoom to your area of interest on the left map panel.
- Click directly on the map to select the ecoregion at that point.
- The app uses the **RESOLVE Ecoregions 2017** dataset. The ecoregions are shown in different colours.
- Once selected, the ecoregion boundary becomes the **Region of Interest (ROI)** for all further analysis.

Step 2: Locate Your Restoration Site

Action

Click on a specific reference point within the ecoregion to mark your restoration site.

- **How to Do It**

- Zoom in further to your chosen restoration site location.
- Click once on the map at that point.
- A **red circular boundary** will appear around the selected location, confirming your selection.
- Click **‘Set Restoration Location’**. To start over, use **‘Clear & Set Again’**.



Figure 2.4: Ecoregion set

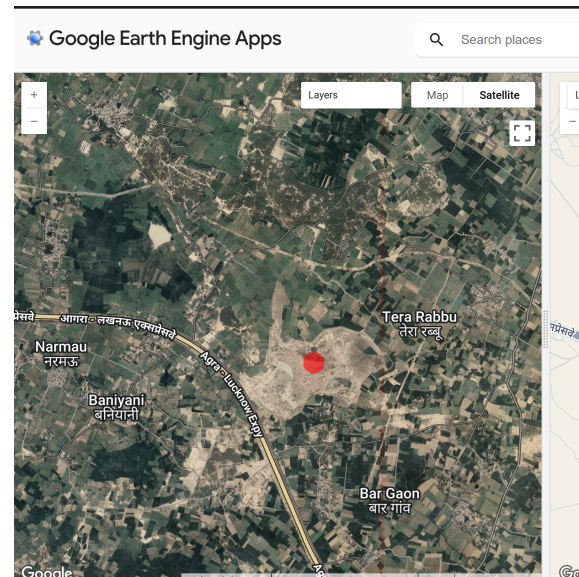


Figure 2.5: Red boundary at chosen location within the ecoregion

• What Happens Automatically

The app queries each environmental dataset at your clicked coordinates and extracts:

- Annual precipitation
- Annual Mean Temperature
- Elevation
- Soil constraints
- Terrain class
- Fire occurrence count

This data forms the **baseline** for the rules you will build in the next steps.

Step 3: Select Environment Layers

Action

Find the best combination of these inputs that helps you isolate the reference site from surrounding areas.

Note: To re-load any layer, click **Clear Map** and then load it again.

The following environmental layers are available. Set thresholds or select classes for each layer relevant to your reference site:

- **Annual Precipitation (WorldClim BIO Variables V1)** Provide a range (in mm/year) corresponding to the precipitation conditions at your site.

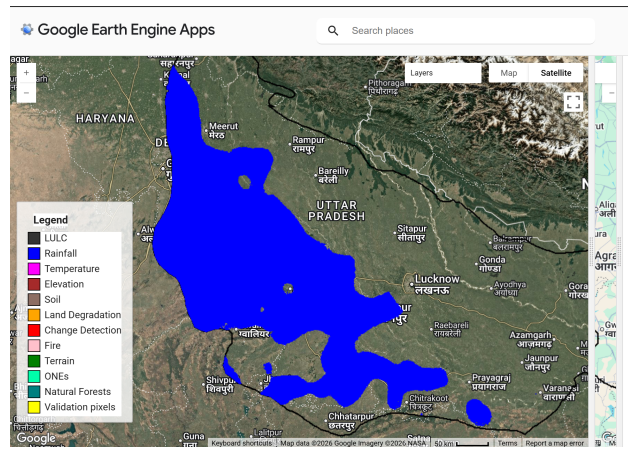


Figure 2.6: Rainfall layer loaded on left map

- **Annual Mean Temperature (WorldClim BIO01)** Provide a temperature range in °C corresponding to the conditions at your site.
- **Elevation** Provide an elevation range (in metres) corresponding to the area.

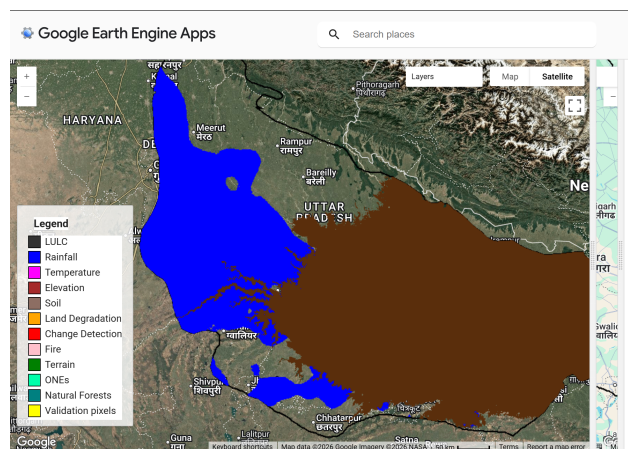


Figure 2.7: Elevation layer loaded on top of rainfall layer

- **Soil Constraints (HWSD)**
 - **Topsoil Texture** — select one or more applicable classes:
 - * Clay (heavy), Silt clay, Clay (light), Silty clay loam, Clay loam
 - * Silt, Silt loam, Sandy clay, Loam, Sandy clay loam
 - * Sandy loam, Loamy sand, Sand
 - **Soil Drainage** — select one or more applicable classes:

- * Excessively drained, Somewhat excessively drained, Well drained
- * Moderately well drained, Imperfectly drained, Poorly drained
- **Topsoil pH** — select one or more applicable classes:
 - * Strongly Acidic, Moderately Acidic–Neutral
 - * Slightly Alkaline, Moderately Alkaline
- **Terrain (CoRE Stack)** Select the terrain classes that best characterise your reference site area:
 - V-shape river valleys, Deep narrow canyons
 - Lateral midslope incised drainages, Local valleys in plains
 - Upland incised drainages, Stream headwaters
 - U-shape valleys
 - Broad flat areas
 - Broad open slopes
 - Mesa tops
 - Upper slopes
 - Local ridge/hilltops within broad valleys
 - Lateral midslope drainage divides, Local ridges in plains
 - Mountain tops, High ridges

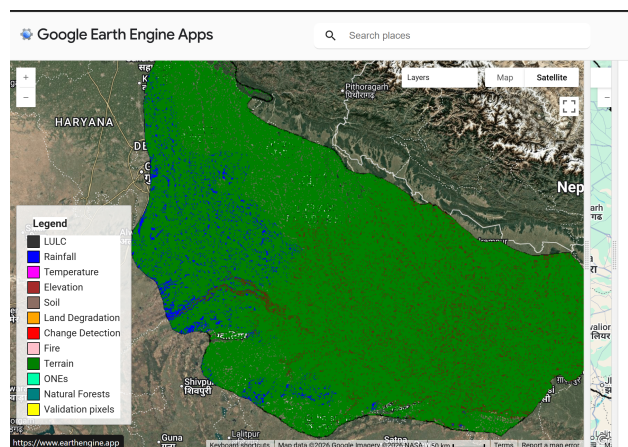


Figure 2.8: Terrain layer loaded on top of rainfall and elevation

 Tip

Internally, the app creates a Boolean mask for each criterion and combines them using **AND** logic (only pixels satisfying *all* conditions are selected). Start with broader ranges and narrow down iteratively in Step 5.

Step 4: Select Temporal Range for Time-Based Layers

Action

Set the temporal range to filter fire incidence and land cover change detection layers. You can skip this step if you are using the tool for the first time.

- **Setting the Temporal Range**

Two year inputs are required:

- **Base Year:** the historical reference year, commonly **1985**, representing the ecological baseline before degradation.
- **Restoration Start Year:** the year when restoration activities began (or are planned to begin).

The time-based layers (fire and change detection) are evaluated over the period **between the base year and the restoration start year**.

- **Which Case Applies to You?**

	Case 1: Finding candidate restoration sites	Case 2: Finding reference sites	Case 3: Characterising a specific habitat
Situation	You have a reference site and want to find other locations that can be similarly restored.	You have a candidate site and want to find similar sites that are already restored.	You want to identify areas sharing the key environmental characteristics of a specific habitat type.
Restoration Start Year	The year you initiated restoration at your reference site.	Set to something recent, e.g. 2024 .	Set to the year that defines the habitat state of interest, e.g. 2000 if looking for areas forested since then.
Base Year	1985	The year restoration started at the reference site, or 1985 if seeking pristine sites unchanged since then.	1985 (e.g. to find areas that have been continuously forested, set base year to 1985 and restoration start to 2024 and select Forest at both time points).

Table 2.2: Temporal range settings across the three use cases

- **Fire Occurrences (MODIS MCD64A1)**

The app uses the **MODIS MCD64A1 Burned Area Product**, which provides monthly global burned area data at 500 m resolution. For each monthly image, a pixel is marked as *burned* if its **BurnDate** band has a value greater than zero, indicating that burning was detected in that pixel during that month. These monthly burned flags are then summed over the selected period (between the base year and restoration start year) to produce a **total count of burned months** per pixel.

- **Minimum Fire Occurrences:** enter a positive integer threshold for the minimum number of months in which a pixel must have recorded burning. For example, a value of **3** retains only pixels that were detected as burned in at least 3 separate months over the selected period.

Note

This metric counts the *number of months with detected burning*, not fire intensity. It does not distinguish between large and small fires (a pixel scorched briefly in one month counts the same as one affected by a large fire). Use this layer to characterise areas with *recurring* fire disturbance rather than fire severity.

• Spatial Cluster Assets

Select the spatial cluster classes that best characterise your site at each of the two time points:

- **Classes at the Base Year:** what the land looked like before degradation
- **Classes at the Restoration Start Year:** what the land looked like when restoration began:

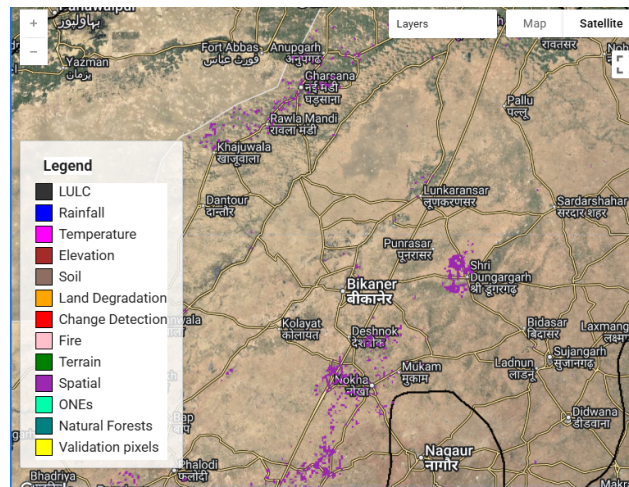


Figure 2.9: Spatial layer loaded

Change Detection (GLC-FCS30D)

Select the land cover classes that best characterise your site at each of the two time points:

- **Classes at the Base Year:** what the land looked like before degradation
- **Classes at the Restoration Start Year:** what the land looked like when restoration began:

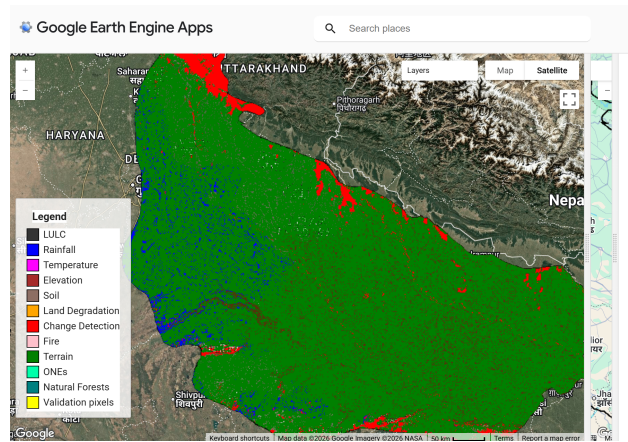


Figure 2.10: Change detection layer loaded

Step 5: Evaluate Rules

Action

Run the rules and visually inspect whether they correctly isolate your reference site.

• How to Do It

- Click ‘**Run Step 5**’ in the control panel.
- The **left map panel** updates to show **validation pixels** in yellow i.e., the pixels within your ROI that satisfy all the conditions you set.

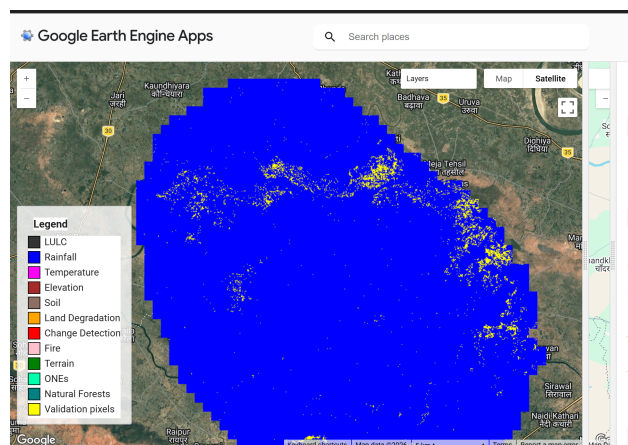


Figure 2.11: Validation pixels appear in yellow on the left map

• What to Look For

- Highlighted pixels should **cluster around your reference site** location.
- If *too many* pixels are highlighted across the entire ecoregion, tighten your ranges (e.g., narrow the precipitation range).

- If *too few* (or no) pixels are highlighted, widen the ranges or reconsider the land cover classes.

Step 6 & 7: Enter Year and Apply Masking Layers

Action

Specify the current year to enable finer filtering of potential restoration or reference sites based on present-day land cover conditions.

This step is particularly relevant for **Case 1** (identifying potential restoration sites), but also provides useful filtering for **Case 2** (finding reference sites).

- **Input Required**

- **Current Year:** set this to a recent year, e.g. **2024**.

- **How It Is Used**

Variable	Case 1 (Find candidate sites)	Case 2 (Find reference sites)	Case 3 (Characterising a specific habitat)
What the Current Year Controls	Identifies locations that <i>today</i> look like what the reference site looked like when restoration was initiated there (helping pinpoint sites that are currently in a restorable condition).	Provides an additional temporal filter to refine which reference sites are selected based on their current land cover state.	Identifies areas that currently exhibit environmental characteristics similar to the selected habitat type.

Table 2.3: Role of the current year input across the three use cases

Four masking layers are available. You can use them individually or in combination: each is ANDed with the inference map so only pixels satisfying all selected classes are retained.

- **LULC (IndiaSAT v3)**

Select the land use/land cover classes where you feel restoration activities might be feasible. This layer shows all pixels falling under the selected classes in the current year (as set in Step 6).

Available LULC classes:

- Built-up
- Kharif water

- Kharif and Rabi water
- Kharif, Rabi and Zaid water
- Tree / Forests
- Barren lands
- Single Kharif Cropping
- Single Non-Kharif Cropping
- Double Cropping
- Triple Cropping
- Shrubs / Scrubs

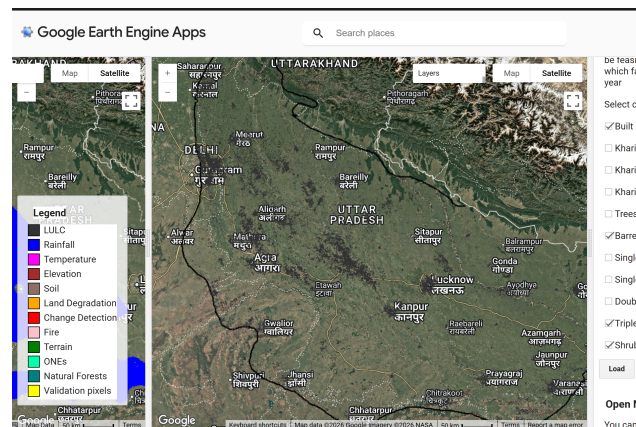


Figure 2.12: LULC layer loaded on right map

• Open Natural Ecosystems (ONEs)

You can use ONEs classes in addition to, or instead of, the IndiaSAT LULC classes above for finer ecological filtering:

- Open Agriculture
- Bare / Rocky / Sparsely Vegetated
- Built-Up
- Cultivated Trees
- Dune
- Forest
- Saline Flat
- Savanna Grassland
- Savanna Shrubland

- Savanna Woodland
- Water / Wetland

• Degradation Types (India WRIS)

Select the degradation types that match the conditions at your reference site. The available classes are grouped below by degradation process:

- **Water Erosion:** Sheet erosion (Slight / Moderate / Severe), Rills, Gullies, Ravines (Shallow / Deep)
- **Wind Erosion:** Sheet erosion (Slight / Moderate / Severe), Partially Stabilised Dunes, Unstabilised Dunes
- **Salinisation / Alkalisiation:** Saline (Slight / Moderate / Severe), Sodic (Slight / Moderate / Severe), Saline-Sodic (Slight / Moderate / Severe), Rann
- **Waterlogging:** Surface ponding — Permanent, Surface ponding — Seasonal, Subsurface waterlogged
- **Anthropogenic:** Mining and dump areas, Industrial effluent affected areas, Brick kiln
- **Acidification:** Moderate, Severe
- **Glacial:** Frost shattering, Frost heaving
- **Others:** Riverine sands / Sea ingress, Barren rocky / Stony waste, Mass movement / Mass wastage
- **Non-degraded:** Non Degraded Land / None of the Above

• Natural Forests (Global 2020)

This layer provides a **probability map of natural forests** at 10 m resolution. Apply a threshold to convert the probability map into a binary forest mask.

- **Threshold (0–250):** Sets the minimum probability required to classify a pixel as natural forest. Higher values produce stricter, more confident forest mapping.

Note

Each selected mask is converted to a binary image and ANDed with the rule-based inference map. Only pixels satisfying both the ecological rules from Steps 3–4 *and* the feasibility classes selected here are retained in the final output.

Step 8: Apply Rules for Final Inference

Action

Run the complete analysis to generate the ecotype similarity map.

- **What Happens**

- The app reuses the validated rules from Step 5.
- Temporal layers are substituted with the inference years from Step 6.
- All masking layers from Step 7 are applied simultaneously.
- The final output shows pixels satisfying **all** spatial, temporal, and masking constraints.

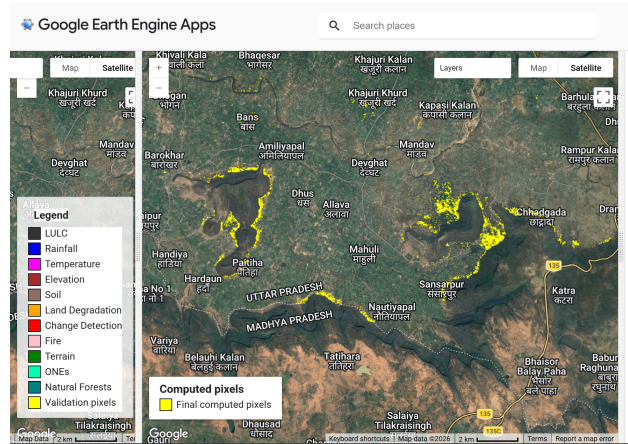


Figure 2.13: Final inference pixels shown in yellow on right map

- **Reading the Output**

- **Yellow-highlighted regions** on the right map panel are the result, i.e., locations with ecological characteristics similar to your reference site.
- These are **candidate sites** for ecological restoration interventions.

Step 9: Fine-tuning and Post-processing

Action

Refine the output to reduce noise and produce cleaner, more actionable results.

- **Available Tools**

- **JSON Rule Export and Import:** The app allows users to export all configured filtering rules as a structured JSON file at the end of the workflow. This JSON captures constraints across environmental variables and temporal settings. Users can also load a previously saved JSON file at the beginning of the workflow to automatically restore all parameter selections, enabling reproducibility and efficient reuse of configurations.

- **Size-based filtering:** Removes small isolated pixel clusters that are unlikely to be meaningful restoration sites. This reduces noise and emphasises larger, contiguous patches.
- **SNIC Segmentation** (*planned*): Groups adjacent pixels with similar spectral and environmental properties into polygons, allowing for more precise boundary delineation.

2.2.3 Tips for Best Results

- Use a well-known reference site
- Iterate Step 5 multiple times
- Zoom in before selecting locations
- Use masking layers to reduce false positives
- Apply size filtering for cleaner results

2.3 Accompanying App: Example Restoration Sites

Overview

The accompanying app provides a simplified entry point into the similarity search workflow. Rule-building has already been completed for a curated set of restoration sites, and users can explore precomputed results immediately, without configuring any parameters from scratch.

2.3.1 Purpose and Relationship to the Main App

The two applications form a complementary pair designed to support different stages of the restoration-planning workflow:

	Main App	Accompanying App
Purpose	Build new rule sets for any restoration site from scratch.	Explore precomputed rule sets for a curated set of reference sites.
Who it is for	Researchers and advanced users configuring a new site.	Field teams and new users wanting a quick overview.
Interface	Two synchronised map panels with a full 9-step control panel.	Single map panel with a simplified control panel.
Rules	Built interactively by the user in Steps 3–7.	Pre-loaded; can be modified by the user if needed.
Typical use	Detailed configuration and analysis for a new site.	Quick scanning and exploration before moving to the main app.

Table 2.4: Comparison of the main app and the accompanying app

2.3.2 Predefined Restoration Sites

A representative set of restoration sites across different ecological contexts has been included. Sites were selected based on ongoing restoration activities, availability of ground-truth knowledge, or ecological significance:

- **Sahyadri hilltops**
- **Shola grasslands**
- **Kedarnath Bugyal Conservation Project**
- **Aravalli Biodiversity Park, Gurugram**
- **Aminagal Shola-grassland Restoration**
- **Darjeeling Hills / Project SERVE**
- **Rekhalgere site / Medius Earth**

For each site, the full ecological signature was extracted in advance using the detailed workflow of the main app, and the resulting rule sets and inferred pixels are stored and ready to load.

2.3.3 Interface Design

The graphical interface prioritises clarity and rapid comprehension:

- **Single central map panel:** displays both the reference site boundary and the inferred matching areas on one map.

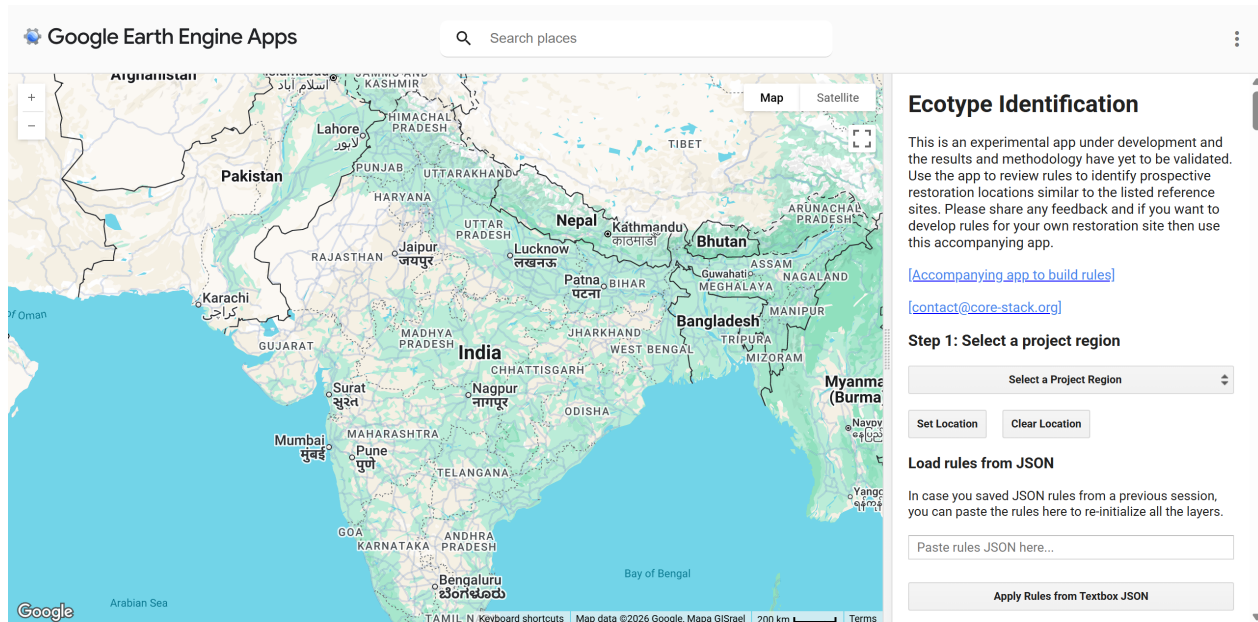


Figure 2.14: Accompanying app

- **Simplified control panel:** contains only the essential interactive elements: site selection dropdown, layer controls, and year inputs.
- **JSON rule loading:** a text box allows users to paste saved rules from a previous session to re-initialise all layers instantly.
- **Dynamic legend:** displayed on the map once you select a project region to explain the meaning of each visible layer.

2.3.4 Workflow

The workflow is deliberately simplified to improve accessibility:

1. **Select a project region:** choose from the dropdown list of predefined restoration sites. Once a region is selected, all layers for that region are loaded onto the map, along with the final 'AND of the layers'. Users can view each of these using the toggle on the map.
2. **Load rules from JSON:** the app allows you to paste rules from a previous session to re-initialise all layers.
3. **Select environment layers:** the same environmental layers as the main app are available and are pre-configured with the site's rule thresholds. Users can adjust these if needed.
4. **Apply time-based layers:** fire occurrence and change detection layers are pre-set with the site's temporal parameters. The restoration start year and base year are already filled in.

5. **Select year and apply masking layers:** LULC, ONEs, degradation type, and natural forest masks (whichever needed) are pre-selected for the site. The current year is pre-set.
6. **Compute AND of all loaded layers:** combines all rules and masks from Steps 2–6 using AND logic to produce the final output. Yellow pixels on the map represent locations satisfying all conditions simultaneously. Use **Size Filtering** to remove small isolated pixel clusters. Click **Print Final Rule JSON** to print your rule set for reuse in future sessions. You can use the **Export AND as Polygons** button to download a KML file of the final computed pixels. The **Show Report** gives a summary table of all active parameters and their values, including the total area of the computed pixels in hectares and km².

Layer	Values
rainfall	500 - 900 mm
elevation	60 - 120 m
terrain	Broad Flat Areas, Broad open slopes
lulc	Trees
change_detection	From: trees To: trees

Figure 2.15: **Example of report generated**

Note

The accompanying app covers India only, using the same datasets as the main app (IndiaSAT LULC, India WRIS degradation, RESOLVE Ecoregions). The predefined rule sets reflect the ecological conditions at the time they were generated and may need updating as newer dataset versions become available.

2.4 Additional Information

2.4.1 Data Sources

- Ecoregion boundaries- RESOLVE Ecoregions 2017
- Precipitation data- WorldClim BIO12
- Elevation data- NASA SRTM DEM
- Temperature data- WorldClim BIO Variables V1
- Soil data- Harmonized World Soil Database v1.2
- Fire data- MODIS MCD64A1

- Terrain data- Derived from NASA SRTM DEM
- Change detection- GLC-FCS30D (1985–2022)
- LULC data- India SAT LULC
- Land degradation- India WRIS
- Open Natural Ecosystems (ONEs) data- India’s ONE
- Natural Forests (Global 2020)-Natural Forests of the World
- Spatial Cluster Assets- Derived from GLC-FCS30D (2000–2022)

2.4.2 Quick Workflow Summary

1. Select Ecoregion
2. Locate Restoration Site
3. Set Temporal Inputs (Rules)
4. Build Rules
5. Evaluate Rules
6. Set Temporal Inputs (Inference)
7. Apply Masking Layers
8. Run Final Inference
9. Fine-tune Results

2.4.3 Future Enhancements

- NDVI/EVI integration
- Automated segmentation
- GeoTIFF/Shapefile export
- Integration with restoration site database

2.4.4 Contact & Feedback

This app is under active development. Please share feedback through the app interface or reach out via email.

Contact Emails:

- Apoorva Dewan: Apoorva.Dewan.cs521@cse.iitd.ac.in
- Ojasvi Bansal: Ojasvi.Bansal.cs521@cse.iitd.ac.in

- Prof. Aaditeshwar Seth: aseth@cse.iitd.ac.in

Developed by: Apoorva Dewan & Ojasvi Bansal

Supervisor: Prof. Aaditeshwar Seth

2.4.5 Acknowledgements

We thank the following individuals for their valuable feedback, field insights, and support in developing and testing the apps:

- Anand Kr. Pandey, AVBG
- Kiran Baldwin, AVBG
- Shashank Bhardwaj, ABVG
- Madhura Nipadhkar, WCT

Chapter 3

Spatial Clusters

3.1 Introduction

While the rule-based similarity search engine provides a highly configurable framework for locating specific environmental profiles, it operates on user-defined local parameters that may not capture broader, macro-level ecological trends. To complement this localized approach, this chapter introduces a systematic spatial clustering framework designed to partition the entire geographical extent of India into discrete, homogenous ecological zones based on multi-decadal land-cover dynamics. By aggregating more than two million high-resolution spatial grid cells according to their fractional land-cover composition, we can map regional environmental signatures without a prior selection of a reference point. This pan-India partitioning provides necessary macro-ecological context, serving as an empirical baseline to filter, validate, and contextualize candidate restoration sites identified by the interactive similarity applications.

3.2 Methodology

3.2.1 Generation of S2 Level-13 Grids over India

The first step of the pipeline involves generating pan-India spatial grids using the S2 Geometry system. India was partitioned into S2 Level-13 cells, resulting in a total of 2,256,839 grids. Each grid cell was represented using:

- **S2 Cell ID:** stored as a string identifier of the form `id_{s2 cell id}`
- **WKT Polygon Geometry:** stores the polygon representing the S2 grid cell

We use Indian Boundary as ROI and get list of all polygons in it. These polygons are converted into s2 cells with ids using `polygon_to_s2_cells()`. We store each grid with its s2_id and polygon information using `s2cell_to_ee_polygon()`.

```
1 def polygon_to_s2_cells(poly, level=13):  
2     minx, miny, maxx, maxy = poly.bounds
```

```

3     rect = LatLngRect.from_point_pair(
4         LatLng.from_degrees(miny, minx),
5         LatLng.from_degrees(maxy, maxx)
6     )
7     coverer = RegionCoverer()
8     coverer.min_level = level
9     coverer.max_level = level
10    coverer.max_cells = 100000
11    return coverer.get_covering(rect)
12
13 def s2cell_to_ee_polygon(cell_id_int):
14     cell = Cell(CellId(cell_id_int))
15     coords = []
16     for i in range(4):
17         v = cell.get_vertex(i)
18         ll = LatLng.from_point(v)
19         coords.append([ll.lng().degrees, ll.lat().degrees])
20     coords.append(coords[0])
21     return ee.Feature(ee.Geometry.Polygon([coords]))

```

Listing 3.1: Convert polygon to s2 cells and s2 cells to ee polygon

The generated grid dataset was stored as a Google Earth Engine asset:

projects/ee-ojasvibansal/assets/s2_level13_india

3.2.2 Creation of Major Land-Cover Layers

Annual land-cover datasets from 2000 to 2022 were processed to generate simplified major land-cover classes for the entire Indian region. The original land-cover classes were grouped into eight major categories:

1. Cropland
2. Tree Cover
3. Shrub/Grassland
4. Bare Land
5. Water Bodies
6. Built-up Area
7. Wetlands
8. Snow/Ice

The methodology involved the following steps:

- Selecting the annual land-cover band corresponding to a specific year.

- Remapping detailed land-cover labels into the eight major classes using predefined class mappings.
- Clipping the remapped raster to the India boundary.
- Exporting the resulting raster at 30m spatial resolution for pan-India coverage.

Due to the very large spatial extent of India, each yearly export was generated as 4 GeoTIFF tiles for each year.

3.2.3 Merging GeoTIFF Tiles

For every year from 2000 to 2022, the exported raster tiles were merged into a single pan-India `.tif` file using QGIS. This produced one final raster layer per year representing the major land-cover classes across the entire country. The merged rasters were stored locally in `.tif` format and used for subsequent spatial fraction computation.

3.2.4 Computation of Land-Cover Fractions per S2 Grid

For each S2 Level-13 grid cell, fractional coverage of all eight major land-cover classes was computed. The workflow followed was:

1. Convert the S2 Cell ID into its corresponding polygon geometry.
2. Extract the raster window overlapping the polygon from the yearly land-cover raster.
3. Generate a polygon mask so that only pixels lying inside the S2 cell are considered.
4. Remove nodata values and invalid pixels.
5. Count the number of pixels belonging to each major land-cover class.
6. Compute normalized fractions by dividing class-wise pixel counts by the total number of valid pixels inside the grid.

To efficiently process more than 2 million grids, multiprocessing and batch-based execution were used. For each year, the output dataset contained:

- S2 Grid ID
- Fractional coverage for each major land-cover class

We use `compute_fractions()` function to compute each GLC fractions for each s2 level-13 grid per year. We use 7 workers to further reduce computation time.

```

1 def compute_fractions(args):
2     row, tif_path = args
3
4     try:
5         dataset = rasterio.open(tif_path)
6
7         poly = s2cell_to_polygon(row["s2_id"])

```

```

8
9     window = geometry_window(dataset, [mapping(poly)])
10    transform = dataset.window_transform(window)
11
12    arr = dataset.read(1, window=window)
13
14    mask_arr = geometry_mask(
15        [mapping(poly)],
16        transform=transform,
17        invert=True,
18        out_shape=arr.shape
19    )
20
21    arr = arr[mask_arr]
22
23    if dataset.nodata is not None:
24        arr = arr[arr != dataset.nodata]
25
26    arr = arr[arr > 0]
27
28    if arr.size == 0:
29        fracs = [0]*8
30    else:
31        counts = np.bincount(arr, minlength=9)
32        fracs = (counts[1:9] / arr.size).tolist()
33
34    return {
35        "s2_id": row["s2_id"],
36        "crop_frac": fracs[0],
37        "tree_frac": fracs[1],
38        "shrub_frac": fracs[2],
39        "bare_frac": fracs[3],
40        "water_frac": fracs[4],
41        "built_frac": fracs[5],
42        "wetland_frac": fracs[6],
43        "snow_frac": fracs[7],
44    }
45
46    except:
47        return {
48            "s2_id": row["s2_id"],
49            "crop_frac": 0, "tree_frac": 0, "shrub_frac": 0,
50            "bare_frac": 0, "water_frac": 0, "built_frac": 0,
51            "wetland_frac": 0, "snow_frac": 0,
52        }

```

Listing 3.2: GLC Fraction computation per S2 grid

The resulting datasets for the years 2000 to 2022 were uploaded to Google Earth Engine assets:

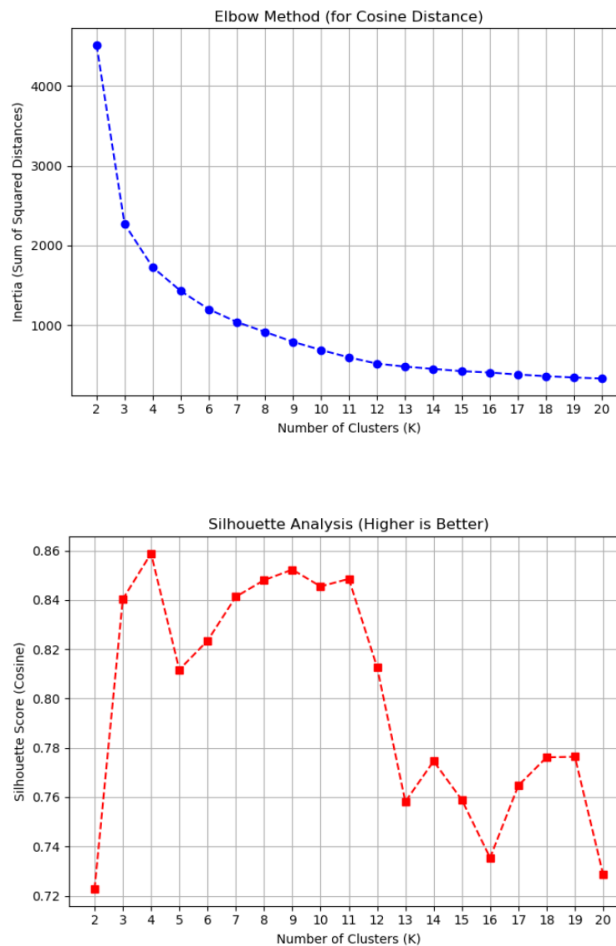
`projects/ee-ojasvibansal/assets/spatial_clusters_cosine_similarity/spatial_{year}`

3.2.5 Determine the Optimal Spatial Cluster Count

The computed land-cover fraction vectors were used as feature representations for clustering. Each S2 grid was represented by an 8-dimensional feature vector:

`[crop, tree, shrub, bare, water, built, wetland, snow]`

To determine the optimal number of spatial clusters, we extracted a random sample of 20,000 grids from the 2022 GLC fractions dataset. The feature vectors were first L_2 -normalized to enforce a cosine similarity metric during clustering rather than standard Euclidean distance. Based on an evaluation of the resulting elbow and silhouette plots, an optimal configuration of 11 clusters was selected for the final spatial asset.



A total of **11 spatial clusters** were generated:

Cluster IDs = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Each cluster represents a characteristic land-cover signature across India.

3.2.6 Spatial Cluster Generation

We load the pre-trained K-Means clustering model to classify historical land cover data from the years 2000 to 2022 into spatial clusters. We read annual GLC fraction data for eight distinct land cover features, apply row-wise L2-normalization to ensure compatibility with a cosine similarity metric, and saves the predicted cluster assignments into individual CSV files per year.

3.2.7 Spatial Cluster Refining

When spatial clusters were viewed on Google Earth Engine, cluster 4 (Mostly wetlands) and cluster 5 (Riverine and coastal areas) had similar areas. So, they were merged. All cluster ids (i) from 5 to 10 were remapped to cluster id (i-1).

Final cluster ids after refining are as follows:

Cluster IDs = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

3.2.8 Rasterization of Spatial Clusters

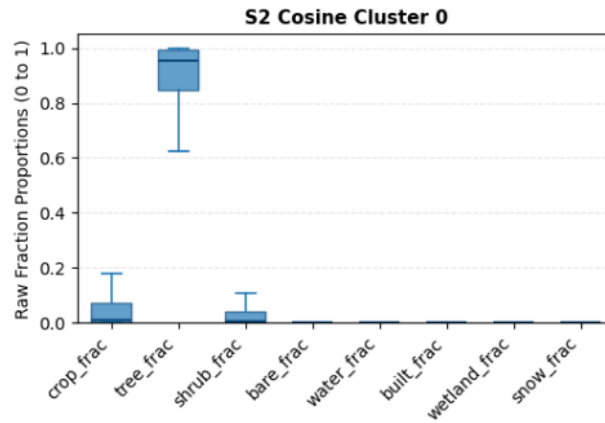
The final cluster assignments were rasterized to generate yearly pan-India cluster rasters. S2 grids for all year from 2000-2022 are csv files which are converted into raster form using its assigned cluster label. The raster outputs were uploaded as Earth Engine assets:

`projects/ee-ojasvibansal/assets/spatial_clusters_cosine_raster/spatial_raster_{year}`

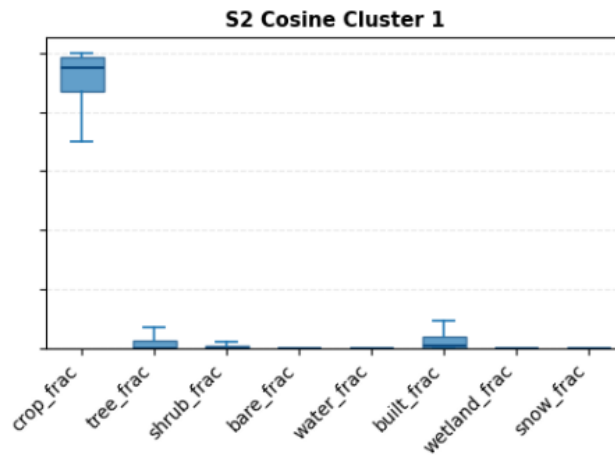
These raster layers enable efficient large-scale visualization and spatial querying inside Google Earth Engine applications.

3.3 Spatial Cluster Classification

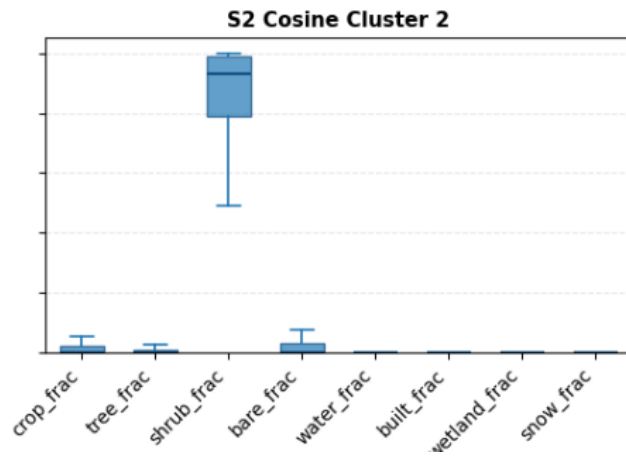
3.3.1 Cluster 0 : Mostly Trees



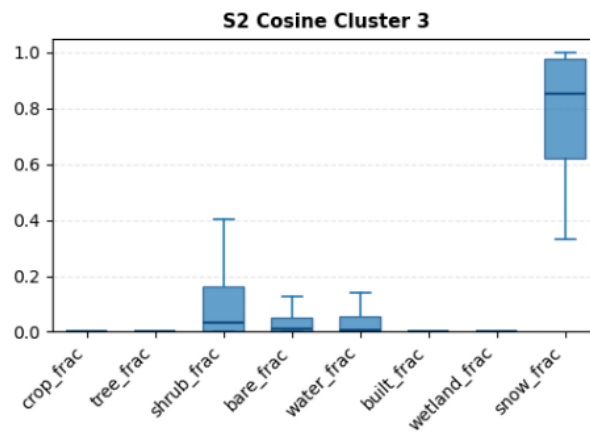
3.3.2 Cluster 1 : Intensive croplands



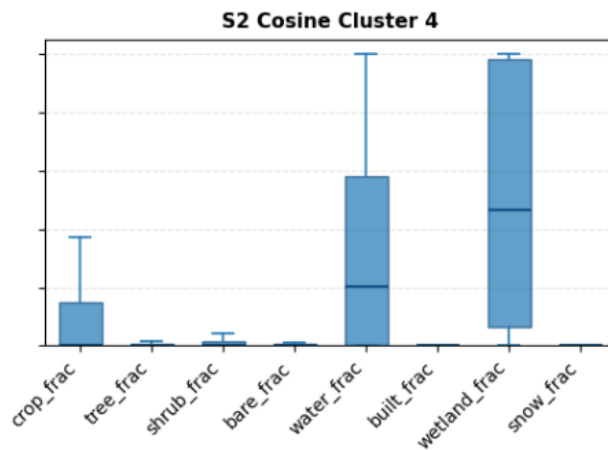
3.3.3 Cluster 2 : Mostly shrublands



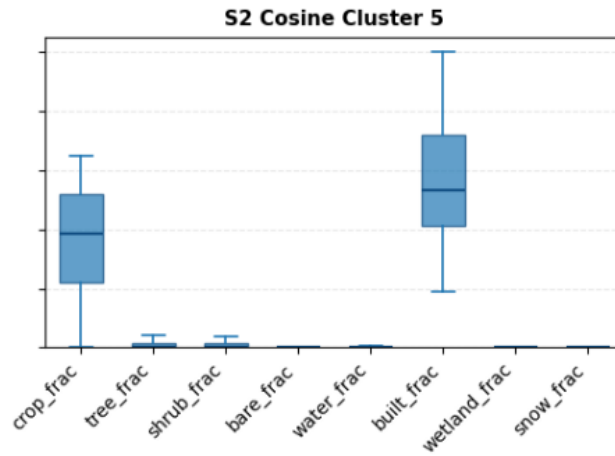
3.3.4 Cluster 3 : Himalayan areas



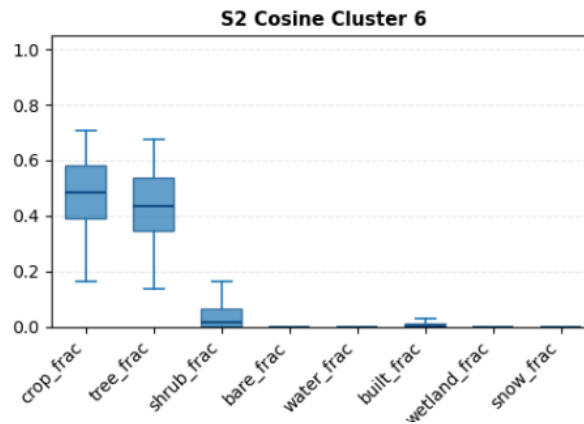
3.3.5 Cluster 4 : Mostly wetlands and riverine areas



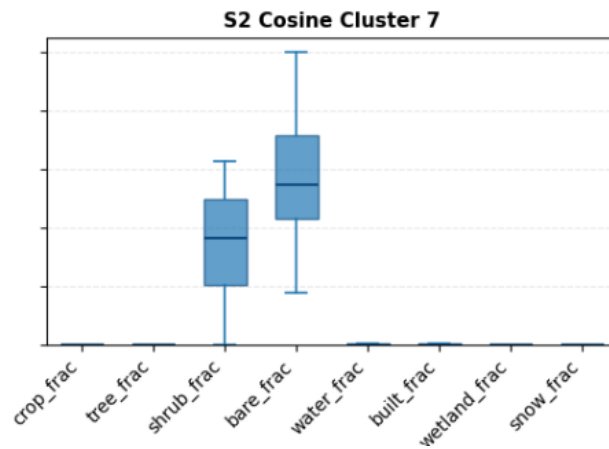
3.3.6 Cluster 5 : Urban and adjoining agriculture areas



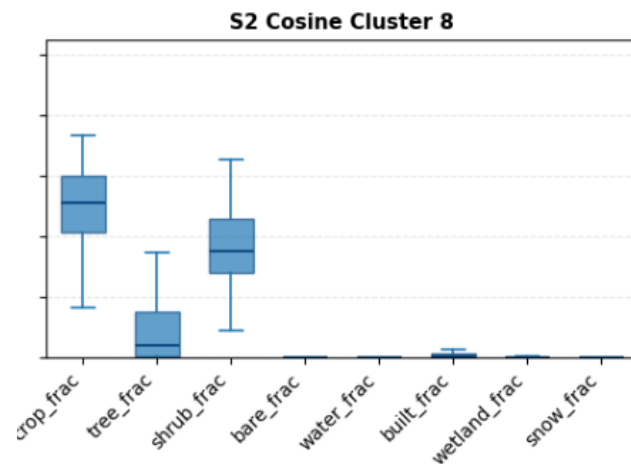
3.3.7 Cluster 6 : Crops and trees



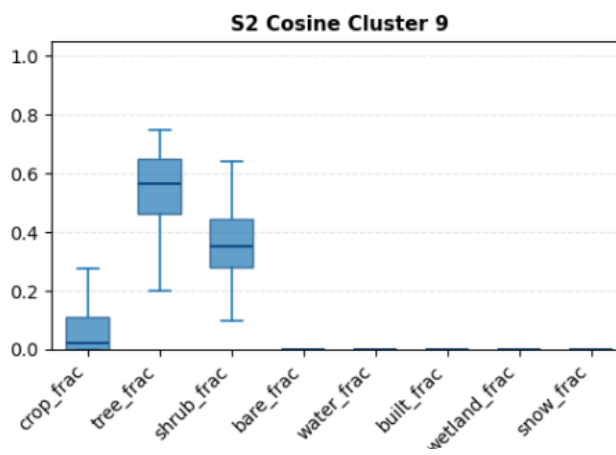
3.3.8 Cluster 7 : Bare and shrub areas



3.3.9 Cluster 8 : Crops and shrubs



3.3.10 Cluster 9 : Bare and shrub areas



Chapter 4

Phenology Clusters

4.1 Introduction

The spatial clustering approach described in Chapter 3 partitions India into ecological zones using static, annually aggregated land-cover fractions. While effective for capturing broad land-cover composition, this approach does not account for *when* vegetation greens up, peaks, and senesces over the course of a year - information that is often more diagnostic of ecosystem type and ecological condition than land cover alone. Two locations classified identically as “tree cover” may have very different growth rhythms: one may be an evergreen forest with stable year-round greenness, while the other may be a deciduous or monsoon-driven system with a sharp single peak.

This chapter introduces a complementary clustering framework built on **vegetation phenology**, which is the seasonal trajectory of the Normalized Difference Vegetation Index (NDVI) at each location. Rather than clustering on a single snapshot of land cover, we fit a compact harmonic (Fourier-series) model to each pixel’s multi-year NDVI time series and cluster on the resulting harmonic coefficients. This allows locations to be grouped by the *shape* of their seasonal vegetation cycle - timing and amplitude of greening, single versus double growing seasons, and the relative strength of semi-annual cycles - independent of their absolute land-cover label.

The phenology clusters provide an additional axis of ecological similarity: two reference and candidate sites that match on rainfall, elevation, and terrain can be further validated or distinguished by whether they share the same phenological cluster, strengthening confidence in candidate restoration or reference sites identified through the rule-based search.

4.2 Methodology

The phenology clustering pipeline consists of five stages: (1) constructing a harmonized, gap-filled Landsat NDVI time series for India, (2) fitting a harmonic regression model to each pixel to compress the time series into five coefficients, (3) sampling these coefficients at a large number of locations and clustering them separately for tree and shrub land-cover classes, (4)

reconstructing and visually validating the phenology curves implied by each cluster centroid, and (5) assigning every pixel in the country to its nearest cluster centroid for each year, enabling year-on-year stability analysis. Each stage is described below.

4.2.1 Constructing a Harmonized Landsat NDVI Time Series

Vegetation phenology is best captured using a dense, multi-year time series of a greenness index. We use the Normalized Difference Vegetation Index (NDVI), computed from the red and near-infrared bands. To obtain sufficient temporal density and multi-decadal coverage, NDVI observations were pooled from three Landsat missions: Landsat 7 (ETM+), Landsat 8 (OLI), and Landsat 9 (OLI-2).

Because these three sensors have systematically different spectral response functions, raw reflectance values are not directly comparable across sensors. We applied the **Chastain et al. harmonization model**, which provides linear regression coefficients (slope and intercept) for converting reflectance between sensor pairs (MSI $\leftrightarrow$ OLI, MSI $\leftrightarrow$ ETM+, OLI $\leftrightarrow$ ETM+). Landsat 8 and Landsat 9 (OLI) reflectance values were harmonized to the Landsat 7 (ETM+) reference frame before computing NDVI, ensuring that the resulting greenness index is consistent across the full sensor record.

```

1  def harmonizationChastain(img, fromSensor, toSensor):
2      comboKey = fromSensor.upper() + '_' + toSensor.upper()
3      coeffList = chastainCoeffDict[comboKey]
4      slopes, intercepts, direction = coeffList
5      out = ee.Algorithms.If(
6          ee.Number(direction).eq(0),
7          dir0Regression(img, slopes, intercepts),
8          dir1Regression(img, slopes, intercepts)
9      )
10     return ee.Image(out).copyProperties(img, ['system:time_start',
11         ])
12
13 def Harmonize_L7_L8_L9(L7, L8, L9):
14     harmonized_L8 = L8.map(lambda img: harmonizationChastain(img,
15         'OLI', 'ETM'))
16     harmonized_L9 = L9.map(lambda img: harmonizationChastain(img,
17         'OLI', 'ETM'))
18     return ee.ImageCollection(L7.merge(harmonized_L8).merge(
19         harmonized_L9))

```

Listing 4.1: Harmonizing Landsat 8/9 (OLI) reflectance to the Landsat 7 (ETM+) reference frame using the Chastain regression coefficients.

Cloud and cloud-shadow pixels were masked for each sensor using its respective QA band before harmonization (`maskL7cloud`, `maskL7L8cloud`).

16-day Compositing and Gap-filling

After harmonization, NDVI was computed for every cloud-free observation. To produce a regular, analysis-ready time series, observations were composited into **16-day intervals** using a centred 16-day window around each step date. Because cloud cover and sensor revisit gaps can leave some 16-day windows empty at a given pixel, we additionally widened the search window to include the same calendar window from the year before and the year after, taking the median across all available observations within ± 8 days of the target date across the three years. This 3-year rolling window substantially reduces the number of pixels with no valid observation in a given compositing period, at the cost of mild temporal smoothing.

```

1 def Get_LS_16Day_NDVI_TimeSeries(startDate, endDate,
   harmonized_ic):
2     start, end = ee.Date(startDate), ee.Date(endDate)
3     n_steps = end.difference(start, 'day').divide(16).floor()
4
5     def make_step(i):
6         center_date = start.advance(ee.Number(i).multiply(16), '
           day')
7         years = ee.List([-1, 0, 1])
8
9         def year_image(y):
10             d = center_date.advance(y, 'year')
11             ic = harmonized_ic.select('NDVI') \
12                 .filterDate(d.advance(-8, 'day'), d.advance
13                             (8, 'day'))
14             return ee.Image(ee.Algorithms.If(
15                 ic.size().gt(0), ic.median().float(),
16                 ee.Image.constant(0).float().rename('NDVI')).
17                 updateMask(ee.Image(0))
18             ))
19
20         ndvi = ee.ImageCollection(years.map(year_image)).median()
21             .float()
22         return ndvi.set('system:time_start', center_date.millis()
23             )
24
25     return ee.ImageCollection(
26         ee.List.sequence(0, n_steps.subtract(1)).map(make_step)
27     )

```

Listing 4.2: Constructing a 16-day NDVI composite time series with a 3-year temporal window to reduce data gaps.

4.2.2 Harmonic Regression on the NDVI Time Series

Rather than clustering directly on the raw, high-dimensional NDVI time series (one value per 16-day step, per pixel, per year), we compress each pixel’s seasonal behaviour into a small set of interpretable coefficients using **harmonic regression**. This is a standard approach in remote sensing phenology analysis: the NDVI time series at each pixel is modelled as a sum of a constant term plus annual and semi-annual sinusoidal harmonics.

For a pixel observed at time t (expressed in fractional years since a reference date), the model fits:

$$\text{NDVI}(t) = \beta_0 + \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \beta_3 \sin(4\pi t) + \beta_4 \cos(4\pi t) + \varepsilon(t) \quad (4.1)$$

where the $\sin(2\pi t)$ and $\cos(2\pi t)$ terms together describe the **annual** (single growing-season) component of the curve, the $\sin(4\pi t)$ and $\cos(4\pi t)$ terms describe the **semi-annual** (bi-modal, e.g. two-monsoon) component, and β_0 is the mean NDVI level. The five coefficients $(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4)$ labelled `constant`, `sin_ann`, `cos_ann`, `sin_sem`, `cos_sem` together form a compact, five-dimensional descriptor of a pixel’s entire phenological cycle. A separate linear term in t is included during fitting to absorb any long-term trend, but is discarded before clustering since it does not describe seasonal shape.

```

1 HARMONIC_BANDS = ['constant', 'sin_ann', 'cos_ann', 'sin_sem', '
   cos_sem']
2
3 def add_harmonic_bands(img):
4     t = img.date().difference(ee.Date(REFERENCE_DATE), 'year')
5     tImg = ee.Image(t).float().multiply(2 * math.pi).rename('t')
6     return img.addBands([
7         ee.Image(1).double().rename('constant'),
8         tImg,
9         tImg.sin().rename('sin_ann'),
10        tImg.cos().rename('cos_ann'),
11        tImg.multiply(2).sin().rename('sin_sem'),
12        tImg.multiply(2).cos().rename('cos_sem'),
13    ])
14
15 harmonic_ts = ndvi_ts_3yr.map(add_harmonic_bands)
16 independents = ['constant', 't', 'sin_ann', 'cos_ann', 'sin_sem',
   'cos_sem']
17
18 regression = harmonic_ts.select(independents + ['NDVI']) \
19     .reduce(ee.Reducer.linearRegression(len(
20         independents), 1))
21
22 harmonic_coeffs = (
23     regression.select('coefficients')
24     .arrayProject([0])

```

```

24         .arrayFlatten([independents])
25         .select(HARMONIC_BANDS)    # drop the trend term 't'
26         .clip(india)
27     )

```

Listing 4.3: Adding harmonic basis bands and fitting per-pixel harmonic regression via `ee.Reducer.linearRegression`.

The regression is fit independently at **every pixel** across India at 30m resolution using `ee.Reducer.linearRegression`, which performs an ordinary least squares fit of the NDVI time series against the six basis functions (constant, linear trend, and the four sinusoidal terms). The output is a single multi-band image where each band holds one fitted coefficient, giving a five-dimensional “phenology fingerprint” image for the country. Because the full-country regression at 30 m resolution exceeds Earth Engine’s per-request memory limits, the computation was tiled into a 3×3 grid of regional tiles for export, then re-mosaicked back into a single seamless image for downstream sampling and clustering. The image can be accessed using:

```

1     tile_images = [
2         ee.Image(f'projects/ee-ictdiitd/assets/apoorva/
3             harmonic_coeffs_ndvi_2020_2023_tile_{i}_{j}')
4         for i in range(3) for j in range(3)
5     ]

```

For the primary analysis, the harmonic model was fit over a **3-year window** (July 2020 to June 2023).

4.2.3 Sampling and Clustering by Land-Cover Class

Phenology is expected to differ systematically between broad vegetation types - for instance, the seasonal growth curve of a forest is shaped differently from that of a shrubland, even where both experience the same climate. To avoid conflating these distinct phenological regimes into a single undifferentiated cluster set, clustering was performed **separately for each LULC class** of interest, using the IndiaSAT v3 land-cover classification (Chapter 2) to restrict the sample to pixels of a given class. The two classes analysed are **tree cover** and **shrub/scrub**

Sampling Strategy

Rather than clustering on every pixel in the country (computationally prohibitive at 30m resolution) a stratified random sample of **20,000 S2 Level-13 grid cells** was drawn from the pan-India grid described in Section 3.2 of Chapter 3. Within each sampled grid, up to 100 pixels matching the target LULC class were randomly sampled from the harmonic coefficient image, masked to the relevant class. Sampling was executed in 100 batches of 200 grids each to stay within Earth Engine’s export size limits, with each batch exported as a separate CSV to Google Drive and later recombined locally.

```

1 def sample_harmonic_for_class(grid, lulc_codes, cls_name):
2     mask = lulc.remap(ee.List(lulc_codes),
3                       ee.List.repeat(1, len(lulc_codes)), 0)
4     samples = harmonic_and_lulc.updateMask(mask).sample(
5         region=grid.geometry(), numPixels=100, scale=30,
6         geometries=True, seed=42
7     )
8     return samples.map(lambda f: f.set({
9         'grid_id': grid.get('s2_id'), 'lulc_class': cls_name
10    }))
11
12 # 100 batches x 200 grids = 20,000 grids sampled per class
13 BATCH_SIZE, num_batches = 200, 100
14 for cls_name, lulc_codes in {'tree': [6], 'shrub': [12]}.items():
15     for batch_index in range(num_batches):
16         batch_grids = sampled_grids.filter(
17             ee.Filter.inList('s2_id', batch_ids))
18         batch_samples = batch_grids.map(
19             lambda g: sample_harmonic_for_class(g, lulc_codes,
20             cls_name)
21         ).flatten()
22         ee.batch.Export.table.toDrive(
23             collection=batch_samples,
24             description=f'harmonic_{cls_name}_batch_{batch_index}
25             :03d}',
26             fileFormat='CSV'
27         ).start()

```

Listing 4.4: Sampling harmonic coefficients within a target LULC class, batched across 20,000 grids.

Clustering on Harmonic Coefficients

The recombined per-class sample (the five harmonic coefficients for each sampled pixel) was standardised using `StandardScaler` (zero mean, unit variance per coefficient) and clustered using **k-means** (`sklearn.cluster.KMeans`). Standardisation is important here because the harmonic coefficients have different natural scales - the constant term (mean NDVI level) and the seasonal amplitude terms do not vary over the same range - and without rescaling, k-means would be dominated by whichever coefficient happens to have the largest numeric spread.

```

1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import StandardScaler
3
4 def fit_kmeans(X, k, cls_name):
5     scaler = StandardScaler()
6     X_scaled = scaler.fit_transform(X)

```



```

7 km      = KMeans(n_clusters=k, random_state=42, n_init=10)
8 labels  = km.fit_predict(X_scaled)
9 print(f"{cls_name}: k={k}, inertia={km.inertia_:.2f}")
10 return km, scaler, labels
11
12 km_tree, scaler_tree, labels_tree = fit_kmeans(X_tree, k=8,
13         cls_name='tree')
14 km_shrub, scaler_shrub, labels_shrub = fit_kmeans(X_shrub, k=5,
15         cls_name='shrub')

```

Listing 4.5: Standardising and clustering harmonic coefficients independently per LULC class.

The final cluster counts were set to $k = 8$ for **tree cover** and $k = 5$ for **shrub/scrub cover**. These values were chosen empirically by inspecting the reconstructed phenology curves (Section 4.2.4) for several candidate values of k and selecting the smallest k at which clusters remained visually and ecologically distinguishable, without producing near-duplicate clusters.

Why cluster on coefficients rather than on the raw time series? An alternative approach (explored during early development but not used in the final pipeline) was to cluster the raw NDVI time series directly using `tslearn.TimeSeriesKMeans`, combined with a hierarchical divisive-splitting procedure (repeatedly sub-clustering any cluster whose internal distortion exceeded a threshold). This time-series clustering approach was found to be dominated by the *mean greenness level* of the curve: pixels were grouped primarily by how green they were on average, rather than by the *shape* of their seasonal cycle. Two pixels with the same single-peak monsoon-driven seasonal pattern but different average greenness (e.g. due to canopy density) would be split into different clusters, while a single-peak and a bimodal curve with similar mean greenness could be merged into the same cluster. Clustering on the five harmonic coefficients instead treats the amplitude and phase of the annual and semi-annual cycles as separate, equally-weighted features, which better preserves distinctions in the *timing and number of growing seasons*.

4.2.4 Curve Reconstruction and Cluster Validation

To interpret and validate each cluster, the fitted harmonic coefficients for a cluster centroid were used to reconstruct a synthetic, smooth phenology curve over a single annual cycle:

$$\widehat{\text{NDVI}}(t) = \beta_0 + \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \beta_3 \sin(4\pi t) + \beta_4 \cos(4\pi t), \quad t \in [0, 1) \quad (4.2)$$

```

1 def reconstruct_curve(coeffs, n_points=46):
2     """coeffs: [constant, sin_ann, cos_ann, sin_sem, cos_sem]"""
3     t = np.linspace(0, 1, n_points, endpoint=False)
4     omega = 2 * math.pi

```

```

5     return (coeffs[0]
6           + coeffs[1] * np.sin(omega * t) + coeffs[2] * np.cos(
              omega * t)
7           + coeffs[3] * np.sin(2*omega*t) + coeffs[4] * np.cos
              (2*omega*t))
    
```

Listing 4.6: Reconstructing a smooth annual phenology curve from a coefficient vector.

Centroid phenology curves were generated per land-cover class to validate the cluster solution, which looked as follows:

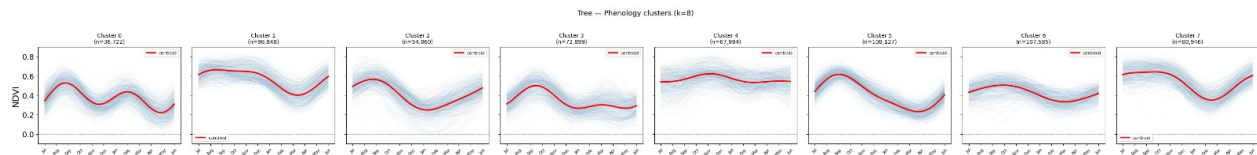


Figure 4.1: Tree clusters

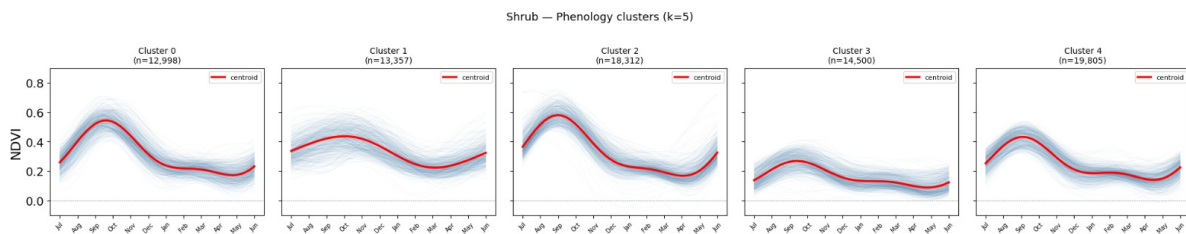


Figure 4.2: Shrub clusters

For each cluster, the reconstructed centroid curve is plotted in red over a 12-month axis (July–June, matching the harmonic window), overlaid with a faint background of individual pixel curves reconstructed from a random subsample of up to 500 pixels assigned to that cluster. This visualises both the typical seasonal shape of the cluster and the spread of curves around it.

4.3 Year-on-Year Cluster Assignment and Stability

A natural question once a phenology cluster vocabulary has been established for one reference period (2020–2023) is whether the same vocabulary remains meaningful, and whether individual locations remain in the same cluster, across other years. This is relevant both as a validation of cluster stability and as a way of detecting locations whose phenology has changed over time due to land degradation, restoration, etc.

4.3.1 Per-window Harmonic Coefficient Images

The same harmonic regression procedure described in Section 4.2.2 was repeated for rolling 3-year windows ending in each year of interest (e.g. a window ending in 2003 covers July

2001 to June 2004). Each window’s harmonic coefficient image was masked to the combined tree and shrub LULC classes and exported as a separate GEE asset, since computing the regression on demand for every year would be prohibitively slow for repeated queries.

```

1 def build_harmonic_coeffs_for_window(end_year, window_size=3):
2     start_year = end_year - window_size + 1
3     start = ee.Date.fromYMD(start_year, 7, 1)
4     end = ee.Date.fromYMD(end_year + 1, 6, 30)
5     # ... build harmonic NDVI time series and fit regression as
        in Listing 4.3 ...
6     return coeffs # 5-band image: constant, sin_ann, cos_ann,
        sin_sem, cos_sem
7
8 combined_codes = [6, 12] # tree + shrub, masked together for
    storage efficiency
9 for end_year in WINDOW_YEARS:
10     coeffs_masked = build_harmonic_coeffs_for_window_masked(
11         end_year, combined_codes, window_size=3)
12     ee.batch.Export.image.toAsset(
13         image=coeffs_masked.toFloat(),
14         assetId=f'{ASSET_DIR}/harmonic_coeffs_window_{end_year}',
15         region=india, scale=30, maxPixels=1e13,
16         pyramidingPolicy={'default': 'mean'})
17     .start()

```

Listing 4.7: Building and exporting masked harmonic coefficient images for each 3-year window.

Masking to only the tree and shrub classes before export substantially reduces GEE asset storage requirements compared to exporting unmasked coefficients for the full country at 30 m resolution for every year.

4.3.2 Nearest-Centroid Cluster Assignment per Pixel

For each yearly window, every pixel’s harmonic coefficient vector is assigned to the **nearest cluster centroid** (in Euclidean coefficient space) from the reference-period clustering solution (Section 4.2.3). This is implemented in Earth Engine by computing, for each candidate cluster centroid, a distance image equal to the Euclidean distance between the pixel’s coefficient vector and that centroid, then taking a per-pixel arg-min over all candidate centroids via `qualityMosaic` on the negated distance.

```

1 def make_cluster_image_from_centroids(coeffs_img, centroids_np,
    cls_name):
2     """centroids_np: shape (k, 5), in ORIGINAL (unscaled)
        coefficient space."""
3     distance_images = []
4     for i, centroid in enumerate(centroids_np):
5         centroid_img = ee.Image.constant(centroid.tolist()) \

```

```

6         .rename(ee.List(HARMONIC_BANDS))
7     dist = (coeffs_img.subtract(centroid_img)
8             .pow(2).reduce(ee.Reducer.sum()).sqrt()
9             .rename('distance'))
10    cluster_id_img = ee.Image.constant(i).rename('cluster_id')
11    distance_images.append(dist.addBands(cluster_id_img))
12
13    dist_ic = ee.ImageCollection(distance_images)
14    return (dist_ic
15            .map(lambda img: img.addBands(
16                img.select('distance').multiply(-1).rename('
17                    neg_distance'))
18            .qualityMosaic('neg_distance')    # picks min-distance
19            .select('cluster_id')
20            .toInt())

```

Listing 4.8: Nearest-centroid cluster assignment via per-centroid distance images and `qualityMosaic`.

Note that cluster centroids are stored and applied in the **original (unscaled) coefficient space** - the inverse of the `StandardScaler` transform used during fitting - so that nearest-centroid distances computed in Earth Engine correspond correctly to distances in the original harmonic coefficient units.

4.3.3 Per-grid Majority Vote and Export

Pixel-level cluster assignments are aggregated to the S2 Level-13 grid level by taking the **modal (most frequent) cluster ID** across all pixels within each grid cell, using `reduceRegions` with `ee.Reducer.mode()`. This produces a single representative cluster label per grid per year per LULC class, which is exported as a CSV containing only the grid ID and assigned cluster ID.

```

1  grid_clusters = cluster_img.reduceRegions(
2      collection=s2_grids, reducer=ee.Reducer.mode(),
3      scale=30, tileSize=4
4  )
5  grid_clusters = grid_clusters.map(lambda f: ee.Feature(None).set
6      ({
7          'grid_id': f.get('s2_id'), 'cluster_id': f.get('mode')
8      })))
9  ee.batch.Export.table.toDrive(
10     collection=grid_clusters,
11     description=f'stability_{cls_name}_window_{end_year}',
12     fileFormat='CSV', selectors=['grid_id', 'cluster_id']

```

```
13 | ).start()
```

Listing 4.9: Aggregating pixel-level cluster assignments to grid-level labels via majority vote, and exporting per-year, per-class results.

The resulting per-year, per-class CSVs (one row per grid, giving its assigned cluster for that year) are loaded into a merged pandas DataFrame and used to analyse **transition patterns**: how often, and in which direction, a grid’s phenology-cluster label changes from one year’s window to the next. Grids that remain in the same cluster across consecutive years provide confidence that the phenology fingerprint at that location is stable; grids that transition between clusters are candidates for closer investigation, since a phenological shift may indicate a change in vegetation condition, land-use, or degradation/recovery status that is not yet visible in static land-cover datasets.

4.4 Summary

The phenology clustering pipeline complements the static spatial clusters of Chapter 3 by characterising *when* vegetation greens and senesces, not just *what* land-cover class is present. By compressing each pixel’s multi-year NDVI time series into five interpretable harmonic coefficients and clustering separately within tree and shrub land-cover classes, the method produces a phenology vocabulary ($k = 8$ tree clusters, $k = 5$ shrub clusters) that can be applied consistently across years to track cluster stability and detect phenological change. Within the broader restoration site identification workflow, phenology cluster membership offers an additional, time-series-derived criterion for validating whether a candidate site identified by the Environmental Similarity App (Chapter 2) genuinely shares the seasonal vegetation dynamics of its reference site, rather than only matching on static environmental variables.

Chapter 5

Case Study: Auroville Botanical Gardens

5.1 Introduction

While the preceding chapters describe the design and methodology of the Environmental Similarity App and the spatial and phenology clustering pipelines, this chapter presents a real-world application of the Main App in collaboration with the **Auroville Botanical Gardens (AVBG)**, a restoration and conservation organisation working in the Pandalgudi area of Tamil Nadu. This case study documents the full workflow as it was actually used in practice: the reference site, the parameter choices made, the iterative feedback received from the AVBG team, and the next steps now underway. It is included here both as a worked example of the Main App’s intended use case and as a record of the practical lessons learned from a live collaboration.

5.1.1 Background

AVBG had been documenting the historical flora of the region using the *Flora of the Presidency of Madras*, a colonial-era botanical survey, as part of an effort to reconstruct what the native vegetation of the Pandalgudi area looked like before large-scale land-use change.

AVBG’s underlying question was a natural fit for the Main App’s **Use Case 2: Finding reference sites**: they had a restoration site at Pandalgudi and wanted to identify other locations in the surrounding landscape that already exhibited similar ecological conditions - candidate intact reference sites that could be visited for field study to understand what a healthy end-state of restoration at Pandalgudi might look like.

5.2 Building the Rule Set for Pandalgudi

Working with the team, we used the Main App’s nine-step workflow to characterise the ecological fingerprint of the Pandalgudi restoration site and search for similar areas within

its ecoregion.

5.2.1 Reference Site and Parameter Selection

The Pandalgudi restoration boundary was used as the reference site (Step 2). Following Step 3, the environmental layers were configured based on the values automatically extracted at the reference location, refined iteratively using Step 5 validation:

- **Annual precipitation:** 500–900 mm/year
- **Elevation:** 60–120 m
- **Terrain:** Broad Flat Areas, Broad open slopes
- **LULC:** Trees

For the temporal layers (Step 4), the team selected a **base year of 1985** and a **restoration start year of 2018**, with change detection configured to retain pixels that were classified as **Trees at both the base year and the restoration start year** i.e. areas that have remained continuously forested since 1985. This choice reflects **Use Case 2** (finding reference sites): rather than looking for areas that degraded and might now be restorable, the goal was to find areas that have *stayed* intact, since these represent plausible analogues for what a fully restored Pandalgudi might eventually resemble. Land degradation, fire, and natural forest masks were left unset for this initial pass, and the inference year (Step 6) was set to 2024.

5.2.2 Exported Rule Set

The full rule set, as exported using the JSON export feature, is reproduced below for reference and reuse:

```

1 {
2   "rainfall": {
3     "min": 500,
4     "max": 900
5   },
6   "elevation": {
7     "min": 60,
8     "max": 120
9   },
10  "terrain": [
11    "Broad Flat Areas",
12    "Broad open slopes"
13  ],
14  "spatial": null,
15  "land_degradation": null,
16  "fire": null,
17  "lulc": [
18    "Trees"

```

```

19 ],
20   "change_detection": {
21     "from": [
22       "trees"
23     ],
24     "to": [
25       "trees"
26     ]
27   },
28   "naturalForests": null,
29   "metadata": {
30     "project_name": "Flora of the Presidency of Madras -
31       Pandalgudi",
32     "description": "Finding reference sites similar to pandalgudi
33       polygons",
34     "contact": "Anand Kr.p - anand82546@gmail.com",
35     "use_case": "Finding reference sites"
36   },
37   "years": {
38     "train": {
39       "base": 1985,
40       "restoration": 2018
41     },
42     "infer": {
43       "base": 1985,
44       "current": 2024
45     }
46   }
47 }

```

Listing 5.1: Exported rule set JSON for Pandalgudi

This JSON can be pasted into both the App’s *Load rules from JSON* text box to instantly reproduce the Pandalgudi rule set. It is also the rule set loaded by default for the “Pandalgudi, Tamil Nadu” entry in the Accompanying App’s predefined site list.

5.2.3 Generated Report

Running *Show Report* on this rule set produces a summary table of all active parameters, their values, and the total area of the matched pixels. The generated report for the Pandalgudi rule set is shown below.

Layer	Values
Project Name	Flora of the Presidency of Madras - Pandalgudi
Description	Finding reference sites similar to pandalgudi polygons
Contact	Anand Kr.p - anand82546@gmail.com
Use Case	Finding reference sites
Rainfall	500 - 900 mm
Elevation	60 - 120 m
Terrain	Broad Flat Areas, Broad open slopes
Lulc	Trees
Change Detection	From (1985): trees To (2018): trees

Figure 5.1: App-generated parameter report for Pandalgudi

5.3 Results: Identifying Candidate Reference Sites

5.3.1 Within the Pandalgudi Ecoregion

Running the rule set above against the ecoregion containing Pandalgudi returned a set of candidate pixels matching all specified conditions. Visually, the output picked out a number of forested patches that have remained under continuous tree cover since 1985 and share the reference site's rainfall, elevation, and terrain profile - areas of flat or gently sloping land within the same precipitation band.

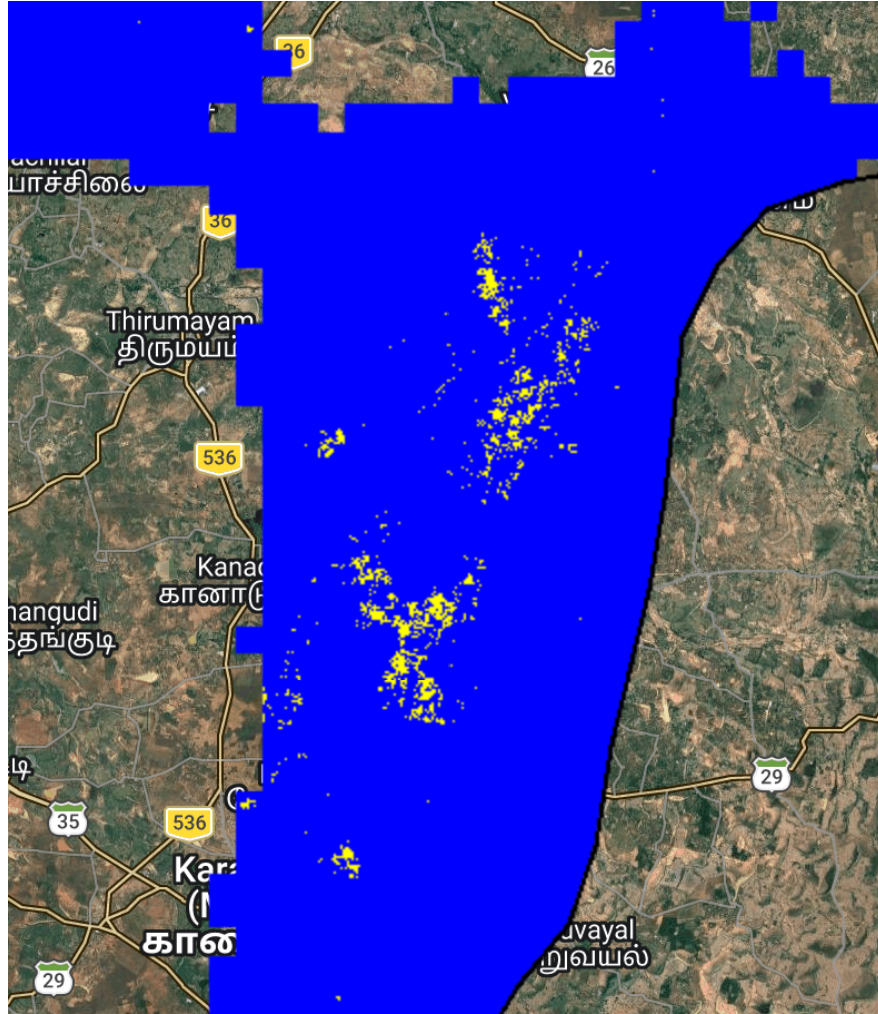


Figure 5.2: Some of the final computed pixels (yellow) for Pandalgudi

5.3.2 Extending to Surrounding Ecoregions

Because the AVBG team was interested in a broader search radius than a single ecoregion, the same rule set was also re-run against the ecoregions immediately surrounding the one containing Pandalgudi, by re-selecting the ecoregion in Step 1 while keeping the Step 3–7 rules unchanged. This produced additional candidate areas beyond the original ecoregion boundary, expanding the pool of sites available for field investigation.

5.4 Current Status: Ground Verification

AVBG was satisfied with the candidate reference sites identified through this process and is now proceeding with **ground verification** of the highlighted areas - visiting the candidate sites in the field to confirm whether they do, in fact, represent suitable ecological reference points for the Pandalgudi restoration project. This verification work is ongoing at the time of writing and is, by design, the most important and most difficult step in the overall workflow:

the app’s output is explicitly intended as a tool to narrow the search space and generate field-testable hypotheses, not as a substitute for on-the-ground ecological judgement.

5.5 Next Steps

Beyond completing ground verification at Pandalgudi, this collaboration has prompted discussion of two follow-on directions:

1. **Replicating the workflow for additional reference sites.** AVBG and the development team agreed to repeat this end-to-end process for further sites where AVBG already has a reference location in mind, in order to build up a stronger evidence base for the overall similarity-search methodology across multiple independent cases rather than a single example.
2. **Retrofitting the app for AVBG’s specific use case.** Initial discussions have also considered whether a new, purpose-built app, or a retrofit of the existing Main App, should explicitly incorporate perspectives specific to AVBG’s broader botanical documentation work - for example, proximity to natural forest patches, consistency of tree cover over time, or proximity to culturally or ecologically significant landmarks such as temples, which can often coincide with patches of relatively undisturbed vegetation in this region.

Appendix A

Technical Guide for the Apps

A.1 Overview

This chapter is a developer-facing guide for anyone who wishes to understand, maintain, or extend the GEE applications described in the previous chapter. It covers the organisation of the codebase, the role of each script module, and step-by-step instructions for common modifications such as adding a new environmental layer, adjusting the masking logic, or adding a new predefined restoration site to the Accompanying App.

All source code is written in **Google Earth Engine JavaScript** (client-side GEE API). The scripts are hosted on the GEE Code Editor and are also available on GitHub at:

<https://github.com/OjasviBansal/MTech-Project>

No local installation is required. All computation runs on Google Earth Engine's cloud infrastructure.

A.2 Code Organisation

The codebase is organised into two primary Google Earth Engine projects: **gee-app**, which contains the Main App implementation, and **gee-app2**, which contains the Accompanying App with predefined reference sites. Each project consists of multiple scripts responsible for specific environmental layers, filters, and visualisation components.

<code>gee-app/</code>	
<code>main</code>	(Main application)
<code>bioclim</code>	(Precipitation layer)
<code>temp</code>	(Temperature layer)
<code>elevation</code>	(Elevation layer)
<code>soil</code>	(Soil layer)
<code>terrain</code>	(Terrain layer)
<code>fire</code>	(Fire layer)

ldd	(Land degradation layer)
lulc	(Land use / land cover)
natural_forests	(Natural forests layer)
ONE_map	(ONEs layer)
change_det	(Change detection layer)
snic	(SNIC segmentation)
sizebased	(Size-based filtering)
spatial	(Spatial clustering)
ndvi_clusters	(Phenology clustering)

A similar structure exists for `gee-app2` as well.

The modular structure means that adding, removing, or modifying a single environmental layer only requires changes to its corresponding layer module and a small update to the main app entry point.

A.3 How the Rule Engine Works

Understanding the rule engine is essential before modifying or extending the application. The rule engine combines all active environmental constraints into a single binary mask that identifies pixels satisfying every selected condition.

1. Each active layer module produces a **binary GEE Image** (a mask) where pixels satisfying the layer's condition have value 1 and all others have value 0.
2. All active masks are combined using a pixel-wise **AND operation** (implemented using `.and()` in GEE). A pixel remains valid only if it satisfies every active rule.
3. The final combined mask is clipped to the region of interest and visualised.

In GEE JavaScript, this looks like:

```
1 // Start with a mask where all pixels pass
2 var combinedMask = ee.Image(1);
3
4 // Apply AND logic for each active layer
5 if (precipitationActive) {
6   combinedMask = combinedMask.and(precipitationMask);
7 }
8
9 if (elevationActive) {
10  combinedMask = combinedMask.and(elevationMask);
11 }
12
13 if (terrainActive) {
14  combinedMask = combinedMask.and(terrainMask);
15 }
16
17 // Continue for remaining active layers...
```

```
18  
19 var result = combinedMask  
20   .selfMask()  
21   .clip(roi);
```

Listing A.1: AND combination logic

The `selfMask()` call converts 0-valued pixels to transparent (masked) pixels so that only matching areas are displayed.

A.4 Adding a New Environmental Layer

This section gives a step-by-step walkthrough for adding a new environmental layer to the Main App. Each layer is implemented as a standalone JavaScript and is responsible for:

- Creating its own UI controls
- Managing visualisation on one or more maps
- Generating a binary mask image
- Exporting/importing rule settings
- Returning masks to the rule engine for AND computation

This section demonstrates how to add a new hypothetical **Aridity Index** layer. The same pattern applies to any numeric or categorical GEE dataset.

A.4.1 Step 1: Create the Layer Module

Create a new file:

`gee-app/aridity.js`

The module should expose standard functions used throughout the application:

- `getPanel()` - builds UI controls
- `reloadAndGetImage()` - regenerates binary mask
- `getLoadedImage()` - returns current mask
- `getRule()` - exports current settings
- `applyFromJSON()` - restores settings
- `setROI()` - injects ROI and map references

Example implementation:

```
1 // aridity.js  
2  
3 var roi_boundary = null;
```

```
4 var loadedImage = null;
5
6 var minBox;
7 var maxBox;
8
9 var DATASET =
10 ee.Image('projects/sat-io/open-datasets/global-ai-et0/
    aridity_index');
11
12 exports.setROI = function(roi) {
13   roi_boundary = roi;
14 };
15
16 exports.getPanel = function() {
17   var panel = ui.Panel();
18   panel.add(
19     ui.Label('Aridity Index')
20   );
21   minBox =
22     ui.Textbox({
23       placeholder: 'Minimum',
24       value: '0'
25     });
26   maxBox =
27     ui.Textbox({
28       placeholder: 'Maximum',
29       value: '10000'
30     });
31   panel.add(minBox);
32   panel.add(maxBox);
33   return panel;
34 };
35
36 exports.reloadAndGetImage =
37 function() {
38   var minVal =
39     parseFloat(
40       minBox.getValue()
41     );
42   var maxVal =
43     parseFloat(
44       maxBox.getValue()
45     );
46
47   var image =
48     DATASET.select('b1')
49       .clip(
50         roi_boundary
```

```

51         );
52
53         loadedImage =
54             image.gte(minVal)
55                 .and(
56                     image.lte(
57                         maxVal)
58                 )
59                 .selfMask();
60         return loadedImage;
61     };
62
63     exports.getLoadedImage =
64     function() {
65         return loadedImage;
66     };
67
68     exports.getRule =
69     function() {
70         return {
71             min:
72                 parseFloat(
73                     minBox.getValue()
74                 ),
75             max:
76                 parseFloat(
77                     maxBox.getValue()
78                 )
79         };
80     };
81
82     exports.applyFromJSON =
83     function(minVal,maxVal){
84         minBox.setValue(
85             minVal
86         );
87         maxBox.setValue(
88             maxVal
89         );
90         exports.reloadAndGetImage();
91     };

```

Listing A.2: Template for a new layer module

A.4.2 Step 2: Register the Layer in main.js

Import the module:


```
1 var aridity = require('users/yourname/repository:
2 layers/aridity');
```

Listing A.3: Importing the new layer

Add its panel to Step 3:

```
1 step3Panel.add(
2     aridity.getPanel()
3 );
```

Listing A.4: Adding UI panel

Inject ROI when selected:

```
1 aridity.setROI(
2     roi,
3     currentMap
4 );
```

Listing A.5: Passing ROI

A.4.3 Step 3: Include the Layer in Rule Evaluation

Locate the AND-combination logic (Steps 5/8).

Existing layers return binary masks through:

```
reloadAndGetImage()
```

Add:

```
1 var aridityMask = aridity.reloadAndGetImage();
2
3 if (aridityMask) {
4     combinedMask =
5         combinedMask
6         .and(aridityMask
7             );
8 }
```

Listing A.6: Including layer in AND logic

This ensures the new layer participates in the global AND computation.

A.4.4 Step 4: Extend JSON Export

Locate the JSON export routine:

```
1 rules.aridity =  
2 aridity.getRule();
```

Listing A.7: Export rule settings

This stores:

```
1 {  
2     "min":100,  
3     "max":800  
4 }
```

A.4.5 Step 5: Extend JSON Import

During import:

```
1 if (json.aridity){  
2     aridity.applyFromJSON(  
3         json.aridity.min,  
4         json.aridity.max  
5     );  
6 }
```

Listing A.8: Restore rule settings

A.4.6 Step 6: Test the Layer

Verify:

1. The controls appear in Step 3
2. ROI updates propagate correctly
3. The binary mask is generated by `reloadAndGetImage()`
4. Export/import restores values
5. Tightening or widening thresholds changes candidate pixels
6. AND logic correctly updates the final result

A.5 Adding a New Masking Layer

Adding a masking layer follows the same modular pattern as adding an environmental layer: create a new module, implement the standard functions (e.g., `getPanel()`, `reloadAndGetImage()`, `getRule()`, `applyFromJSON()`, `setROI()`), and register the layer in `main.js`. The key difference is that masking layers are configured in **Step 7** rather than Step 3 and are used to

exclude pixels from the final suitability result. Consequently, masking layers are applied after the environmental AND computation, typically by removing pixels from the combined mask rather than contributing an additional suitability condition.

A.6 Adding a New Predefined Site to the Accompanying App

The Accompanying App (`gee-app2`) simplifies user interaction by providing pre-configured environmental and historical rule sets for specific, real-world restoration projects across India. Rather than requiring users to manually input individual geographic boundaries, baseline years, or precise biophysical thresholds, these configurations are managed globally via a structured JavaScript array.

A.6.1 The `projectRegions` Array Structure

Predefined sites are stored inside a global list variable named `projectRegions` within the main application entry script. Each site is defined as a structured dictionary containing explicit geographic coordinates, baseline timeline arrays, and bounding environmental intervals. A standard site registration block uses the following object schema:

```

1 {
2   name: 'Unique Display Title',
3   roi: getEcoRegionROI(longitude, latitude),
4   center: [longitude, latitude],
5   years: [preDegradationYear, restorationYear, baselineYear,
6           currentYear],
7   rainfall: [minPrecipitation, maxPrecipitation],
8   elevation: [minElevation, maxElevation],
9   terrain: [listOfValidTerrainClassIDs],
10  change_detection: [[preDegradationClasses], [
11                      restorationClasses]],
12  lulc: [listOfValidLULCClassIDs],
13  project_name: 'Official Institutional Project Title',
14  description: 'Detailed abstract outlining ecological
15               background and restoration targets.',
16  contact: 'Lead Practitioner Name - email@domain.com',
17  usecase: 'Primary classification framework (e.g., Finding
18           reference sites)'
19 }
```

The configuration profile relies on the helper function `getEcoRegionROI(lon, lat)`, which programmatically intersects the site coordinates with the RESOLVE Ecoregions (2017) dataset clipped to the administrative boundary of India. This boundary forms the active region of interest (ROI) used by the rule engine.

A.6.2 Step-by-Step Instructions to Add a New Site

To add a new pre-configured restoration entry to the application, follow these procedural steps:

1. Open the main script file for Gee App 2.
2. Locate the initialization block defining the array list variable:

```
var projectRegions = [ ... ];
```
3. Insert a new structured object profile block (following the structural pattern in object schema in 3.6.1) as an independent, comma-separated entry inside the array array bracket boundaries.
4. Provide geographic values for the primary coordinates (`lon`, `lat`) corresponding to the geographic center of the targeted field implementation.
5. Calibrate bounding threshold criteria brackets (`rainfall`, `elevation`, `terrain`) based on validated empirical parameters characterizing the local ecological target baseline.
6. Save changes and execute a clean application re-compilation to verify the entry populates correctly inside the interactive project dropdown menu component.

A.7 GEE-specific Notes for Developers

- **Lazy evaluation:** GEE uses deferred (server-side) evaluation. Avoid calling `.getInfo()` inside loops, as each call is a synchronous round-trip to the GEE server and will make the app slow. Batch all `.getInfo()` calls or use `evaluate()` with a callback for non-blocking execution.
- **Scale parameter:** Always specify the `scale` parameter in `reduceRegion()` and `reduceRegions()` calls. Using a scale much finer than the native dataset resolution inflates computation time without improving accuracy.
- **Clipping early:** Clip images to the ROI before any computations to reduce the spatial footprint of server-side operations.
- **Layer loading messages:** All long-running computation is wrapped in UI loading indicators (`ui.Label` updates) so users receive feedback while results are computed.
- **Quota limits:** Heavy use of high-resolution datasets (e.g., IndiaSAT at 5 m resolution over a large ecoregion) can exceed GEE's user memory quota. If this occurs, increase the `scale` in the relevant layer module or restrict the ROI.
- **Asset management:** Custom datasets (IndiaSAT, India WRIS, ONEs) are stored as GEE assets. Their asset paths are defined at the top of each layer module and may need to be updated if assets are moved or a newer version is published.