International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 05 | May 2026 www.irjet.net p-ISSN: 2395-0072

TERRASPOTTER: PRECISION AFFORESTATION TOOL

Om Sunil Borekar[1], Prof. S. M. Shelke[2], Vishwaja Vinayak Kakulate[3], Prasad Dattraya Dhotre[4], Pradnya Harishchandra Gajre[5]

[1345]*Department of Computer Engineering, STES's Sinhgad Academy of Engineering, Pune, Maharashtra, India*
[2]*Professor, Department of Computer Engineering, STES's Sinhgad Academy of Engineering, Pune.*

Abstract - *Cities are growing fast and so is the need for smarter afforestation, but most tree-planting efforts today fail due to poor planning, bad data, and random site selection. TerraSpotter: Precision Afforestation Tool solves this by providing a data-driven platform that identifies and manages optimal tree-planting locations scientifically. The system is built using React, Spring Boot, PostgreSQL, and a Python Flask ML microservice, integrating OpenStreetMap and Leaflet.js for interactive mapping while fetching real environmental data via Open-Meteo APIs. A machine learning model then recommends suitable tree species based on temperature, rainfall, soil moisture, and local climate conditions. Cloudinary handles image storage while Brevo manages email notifications. What makes TerraSpotter stand out is its community validation system where real users verify and rate locations, keeping the data trustworthy and up to date. Together, geospatial analysis, machine learning, and crowdsourcing make afforestation planning smarter, more transparent, and genuinely community-driven.*

Key Words: Afforestation, Geospatial Analysis, Machine Learning, Tree Species Recommendation, Crowdsourcing, OpenStreetMap, Environmental Data, Precision Planting

1.INTRODUCTION

Afforestation is becoming increasingly important as urbanization continues to reduce green cover [1],[4]. Although many people are interested in planting trees, most efforts fail due to poor site selection, lack of proper environmental data, and limited coordination [2],[6]. Although many people are interested in planting trees, most efforts fail due to poor site selection, lack of proper environmental data, and limited coordination. There is currently no single platform that combines all these aspects in a simple and effective way.

TerraSpotter: Precision Afforestation Tool is designed to address this gap. It is a web-based platform that helps users identify suitable plantation sites, validate data through community participation, and manage plantation activities efficiently. By integrating geospatial mapping, environmental data, and machine learning, the system enables more accurate and practical afforestation planning.

1.1 Motivation

Rapid urbanization is consuming green cover faster than plantation drives can keep up. Most of these efforts fail because site selection lacks scientific basis, and reliable environmental data is rarely accessible to those who need it. There is no platform today that brings data, mapping, and community participation together in one place Terra Spotter is built to be exactly that.

1.2 Problem Definition

Plantation sites are often chosen without any validation, ignoring critical factors like soil type, rainfall, and climate conditions. Tree species are selected by intuition rather than science. Relevant data remains scattered across disconnected tools with no unified system for mapping, analysis, or progress tracking. There is also no way for communities to verify site information, give feedback, or monitor results lleaving most afforestation efforts unaccountable and ineffective [3],[5],[13].

1.3 Proposed System

To address the stated problems, TerraSpotter is engineered as a multi-tier web application that integrates all the above features via a suite of highly interconnected components. First off, the application interface, implemented based on React - a JavaScript-based User Interface (UI) library, delivers a highly convenient and engaging map-based interface, driven by Leaflet.js and OpenStreetMap (OSM) by [4]. The server-side, which includes user authentication, business logic implementation and inter-component communications is based on Spring Boot technology stack. Database is provided by PostgreSQL that hosts user profiles, land data records and recommendation results. Also, TerraSpotter incorporates a custom microservice built using Python Flask which hosts ML recommendation algorithm.
Environmental data is obtained real-time through integration with Open-Meteo APIs [5] and used as input for ML-based recommendation engine as [1],[7],[9] that takes into account such parameters as temperature, rainfall, soil moisture, and climatic conditions[8],[10],[12] when determining the most suitable type of trees for afforestation at certain locations. Cloudinary is responsible for storing images uploaded by users on their plantation sites while Brevo sends automated email

notifications. To ensure platform's integrity over the long run, we've developed an automatic community validation mechanism where each planting site can be rated and verified by other users.

2. SYSTEM ARCHITECTURE

2.1 Component Interaction

The frontend uses React and interacts with the backend using RESTful API calls. Upon submission of a plantation site by the user, the backend stores the land information, acquires the environmental information using the site coordinates through Open-Meteo, and provides input values such as temperature, precipitation, and soil types to the Flask ML microservices, which return tree species and their probabilities [1]. The process is illustrated in Fig. 1.

2.2 Technology Stack

The frontend utilizes React, Vite, along with Tailwind CSS and React-leaflet for maps. The backend utilizes Spring Boot framework with Java programming language using layered architecture design. Database used in the application is PostgreSQL, while JPA acts as ORM layer for database interaction.
The ML-based component is developed using Python and Flask framework. Other external tools used in the application include Open-Meteo for environment, Cloudinary for images, and Brevo for emails.

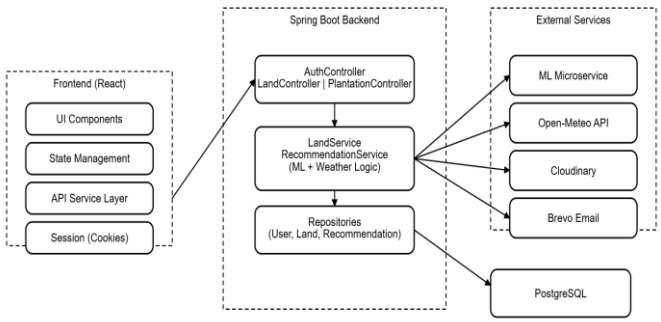

Fig -1: SSystem Architecture

3. IMPLEMENTATION DETAILS

3.1 Frontend Implementation

3.1.1 React Structure

The frontend is built using React and Vite, giving users a fast and smooth experience across all pages. The application follows a component-based structure where each part of the UI is broken into reusable modules. App.jsx manages routing and user session, while main.jsx initializes the application. Key components include Navbar, Login, Browse, SiteDetail, and PlantationForm, each handling a specific part of the user journey.

3.1.2 Key Pages

The Login page handles user authentication and starts a session on successful sign-in. The Browse and Map page displays all available plantation sites on an interactive map. The Plantation Form lets users submit land details and upload site photos. The Site Detail page brings everything together showing land information, tree recommendations, uploaded images, and community reviews in one place.

3.1.3 Map Integration

Maps are powered by Leaflet.js, which renders interactive and zoomable map views directly in the browser. OpenStreetMap (OSM) [6],[11] provides the base map tiles and geospatial data. Users can explore plantation locations, click on markers, and interact with site data visually without leaving the page

3.2 Backend Implementation

3.2.1 Architecture Layers

Backend implementation uses Spring Boot and has a clear three-tier architecture. The Controller tier accepts all HTTP requests and forwards them to their corresponding services, such as AuthController for authentication requests and LandController for land-related requests. The Service tier has all the business rules that perform land calculations, generate recommendations, and validate data. The Repository tier acts close to the database and reads from and writes to PostgreSQL using Spring Data JPA without writing any queries.

3.2.2 Request Flow

When the frontend sends a request, the controller receives and validates it first. It then passes the request to the service layer, which processes the logic and calls external APIs if needed. The repository handles database operations and returns the result back up the chain. The final response is sent to the frontend in JSON format. This flow is shown in Fig. 3.

3.2.3 Business Logic

The backend manages land creation, image uploads, and tree recommendation generation end to end. When a new land record is saved, it automatically fetches environmental data from Open-Meteo, builds the ML input parameters, and calls the Flask microservice to get recommendations. Responses from external APIs are mapped into structured database entities and persisted. The backend also handles input validation, error responses, and transaction management to keep data consistent throughout.

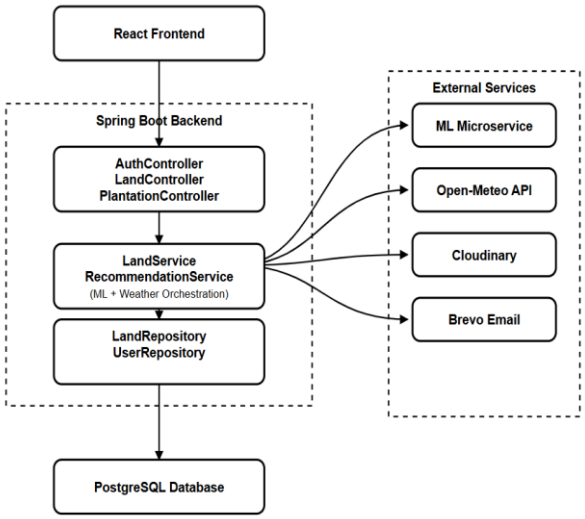

Fig -2: Backend Flow Diagram

3.2 Database Design

3.3.1 Overview

PostgreSQL serves as the backbone of the system's data layer, handling all structured and relational application data. The schema follows normalized design principles, with primary and foreign key constraints keeping everything consistent. Spring Data JPA sits on top as the ORM layer, mapping Java entity classes directly to database tables and taking care of CRUD operations through clean repository interfaces.

3.3.2 Key Tables

There are eleven tables in the schema, which include all functional areas of the system. Users have credentials, user profiles, and user roles, which have unique constraints for emails and check constraints for roles being either USER, ADMIN, or MODERATOR. Land details is a table containing geographical data related to plantation land, such as latitude and longitude, polygon shape in JSONB format, land area in square meters, type of soil, availability of water, and land owner information. Land recommendations record the recommendation results from the machine learning algorithm, including plant name, suitability score as double precision, and recommendation reason in text format. Land reviews store ratings between one and five using check constraints, feasibility analysis, and review comments for each user.

3.3.3 Relationships and Constraints

Relational integrity is enforced end-to-end through foreign key constraints. Users have a 1:N of relationship with land details via created_by and plantation_user_id keys. Each land record cascades into recommendations, reviews, verifications, and plantation records. The land verifications table enforces a unique constraint on (land_id, user_id) so each user gets exactly one vote per land entry. Cascade delete and set-null behaviors are explicitly defined per constraint to handle parent record removal cleanly. JSONB columns cover polygon coordinate arrays and ML metadata where flexible, schema-free storage makes more sense than rigid columns.

3.3.4 Indexing and Query Optimization

Indexes sit on all foreign key columns land_id, user_id, created_by, completion_id to keep joins and filtered queries fast. Suitability score supports composite queries for ranked recommendation retrieval per land entry. Proximity-based land filtering uses stored centroid lat/lng values with Haversine distance calculations handled at the service layer. bigserial primary keys provide auto-incremented IDs with minimal locking under concurrent inserts.

Chart -1: Name of the chart

Table Name	Description	Key Fields
Users	Stores user account and authentication data	user_id, name, email, password
Land Details	Stores plantation site information	land_id, location, area, soil_type
Land Images	Stores image URLs for land	image_id, land_id, image_url
Recommendations	Stores ML-based tree recommendations	rec_id, land_id, plant_name, score
Reviews	Stores user feedback and ratings	review_id, land_id, rating, comment
Plantation Records	Tracks plantation progress	plantation_id, land_id, status, date

3.4 API INTEGRATION

3.4.1 Open-Meteo API

Open-Meteo is a free weather API used to collect real-time environmental data for any given location. When a user submits a plantation site, the backend takes the land coordinates and fires two HTTP GET requests to Open-Meteo one to the forecast endpoint for temperature and soil moisture, and one to the archive endpoint for historical rainfall data. The backend then processes these values to compute mean temperature, annual rainfall, and soil category, which are used as input parameters for the ML model.

3.4.2 Machine Learning API

The Python Flask microservice exposes a predict endpoint that accepts four query parameters temp, rainfall, soil, and climate. The backend calls this endpoint after resolving

environmental values from Open-Meteo. The ML service loads a pre-trained decision tree model (tree model.pkl) using joblib, runs predict proba on the input, and returns a ranked JSON list of suitable tree species [7],[9],[14] with confidence scores and planting reasons. The backend maps these results to Land Recommendation entities and saves them to PostgreSQL.

3.4.3 Cloudinary

Cloudinary handles all image storage in the system. When a user uploads site photos, the backend receives the files and sends them to Cloudinary using its Java SDK. Cloudinary returns a secure URL for each uploaded image, which the backend stores in the land images or plantation completion images tables. The frontend then fetches and renders these URLs directly, keeping image serving fast and scalable without loading the backend server.

3.4.4 Brevo Email Service

Brevo is used for all transactional emails in the system. During signup, the backend generates a 4-digit One Time Password (OTP), stores it temporarily, and sends it to the user via a POST request to the Brevo SMTP API with an HTML email body. A welcome email is also sent after successful account creation. Email failures are handled gracefully they are logged but do not block the main signup or authentication flow.

3.5 Authentication

3.5.1 Signup and Login

When a user registers, the backend validates the input fields and checks if the email already exists in the database. A 4-digit OTP is generated, stored temporarily, and sent to the user via Brevo email for verification. Once the OTP is confirmed, the password is hashed using BCrypt and the user record is saved to the users table in PostgreSQL. During login, the backend loads the user by email, compares the submitted password against the stored hash, and returns an error if they do not match.

3.5.2 Session and JWT Handling

On successful login, the backend generates a JSON Web Token (JWT) containing the user ID, assigned role, and expiry timestamp. This token is returned to the frontend and attached as an Authorization header in all subsequent API requests. A JWT validation filter intercepts every incoming request, verifies the token signature and expiry, and populates the Spring Security context with the authenticated user's identity and roles. A refresh token mechanism is also supported to maintain session continuity without requiring repeated logins.

3.5.3 Protected Routes

Routes that require authentication are secured through Spring Security's filter chain. Every request to a protected endpoint passes through the JWT filter before reaching the controller. If the token is missing or invalid, the backend returns a 401 Unauthorized response. If the user is authenticated but lacks the required role or ownership, a 403 Forbidden response is returned. Service layer methods perform additional checks for example, verifying that the requesting user owns the land record before allowing edits or deletions. This flow is shown in Fig. 3.

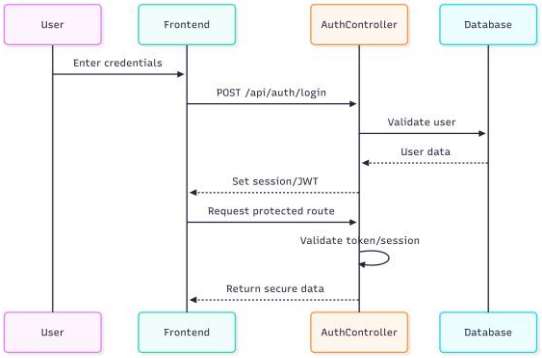

Fig -3 : Authentication

3.6 Core Feature Implementation

3.6.1 Land CRUD Operations

Users are able to perform CRUD operations on plantations sites using the REST API provided by the Land Controller component. The information received by the system upon submission includes the location of the centroid of the plantation site, its polygon shape, area (in square meters), type of soil, accessibility to water, and ownership data. Once the information is validated by the Land Service component, the entity is mapped to an instance of the Land class and saved using the Land Repository component management workflows.

3.6.2 Recommendation System

Once a land record is saved, the backend automatically triggers the recommendation pipeline through fetch And Save Recommendations (). It calls Open-Meteo to fetch mean temperature, annual rainfall, and soil moisture for the site coordinates, then derives a climate zone using simple threshold logic. These four parameters are sent as query inputs to the Flask ML microservice at the /predict endpoint. The microservice runs the trained decision tree model and returns a ranked JSON list of suitable tree species with confidence scores and planting reasons. The backend maps each result to a Land Recommendation entity and saves it to PostgreSQL. If the ML service is unavailable, a fallback list of common species is used to maintain user experience. This pipeline is shown in Fig.

3.6.3 Image Upload

Users can upload site photos during land submission and plantation completion. The backend receives image files as multipart form data, passes them to Cloudinary Service, which uploads each file using the Cloudinary Java SDK and

returns a secure URL. These URLs are stored in the land images or plantation completion images tables and served directly to the frontend for rendering without any additional processing on the server.

3.6.4 Reviews and Community Verification

Any registered user can submit a rating between 1 and 5 along with written feedback for a plantation site through the reviews endpoint. The backend checks for an existing review from the same user and performs an upsert to prevent duplicate entries. Average ratings are computed on the frontend from the returned review list. For verification, users can cast an APPROVE or REJECT vote on pending land entries. A unique constraint on the land id and user id combination in the land verifications table ensures each user can vote only once per site. Admin users can then use these votes to make final approval decisions.

3.6.5 Plantation Tracking

Users can formally record when they plan to start a plantation by submitting details like planned date, team size, method, and number of trees to plant. This creates a plantation starts record linked to the land and user. On completion, users submit a plantation completions record with actual trees planted, remaining capacity, and proof photos uploaded to Cloudinary. This two-stage tracking gives the system a clear view of plantation lifecycle from planning through to completion and allows other users to see real progress on any site.

3.7 System working / flow

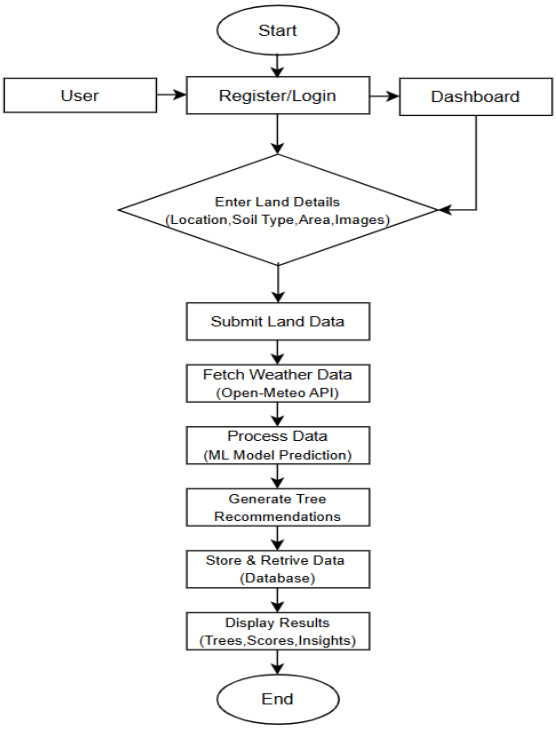

Fig -4: System Flow

3.8 Results and output

The developed system provides a user-friendly interface that allows users to explore different features such as Browse, Submit, History, About, and Contact through a clean and responsive web design. The platform displays available plantation sites with relevant details, images, and filters, enabling users to easily navigate and understand land information. The profile dashboard further shows user activity, statistics, and plantation progress, making the system interactive and informative. These outputs demonstrate that the frontend successfully presents complex data in a simple and visually appealing manner.

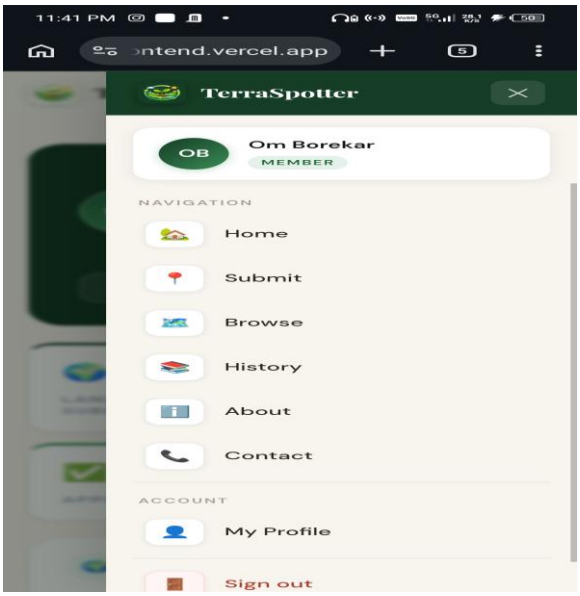

Fig -5: Navigation

Users can submit land for plantation by filling a structured form and marking the exact land boundary using an interactive map interface. The polygon drawing feature allows precise selection of land area, ensuring accurate data collection for further processing. Additional details such as land title, description, ownership, and images can also be provided. This functionality ensures that the system captures complete and reliable input data from users. The land submission and boundary selection process is illustrated in Fig -6.

Once the user submits the land details, the system processes the input and securely stores it in the database. The submitted land is then displayed in the browse section, where users can view multiple land parcels along with important information such as location, area, land type, and basic description. This helps users easily explore and compare different plantation sites. When a user selects a specific land, the system opens a detailed view showing complete information along with uploaded images and user reviews, if available.

After this, the system automatically collects environmental data such as temperature, rainfall, and soil conditions using external APIs. These parameters are then passed to the machine learning model, which analyses the data and generates suitable tree species recommendations. The output includes a list of recommended trees along with relevance or suitability scores, helping users clearly understand which trees are best suited for that particular land. In addition, the system ensures that the recommendations are stored and can be viewed again without reprocessing. This complete workflow, starting from browsing land to receiving intelligent recommendations, is illustrated in Fig -8 and Fig -9

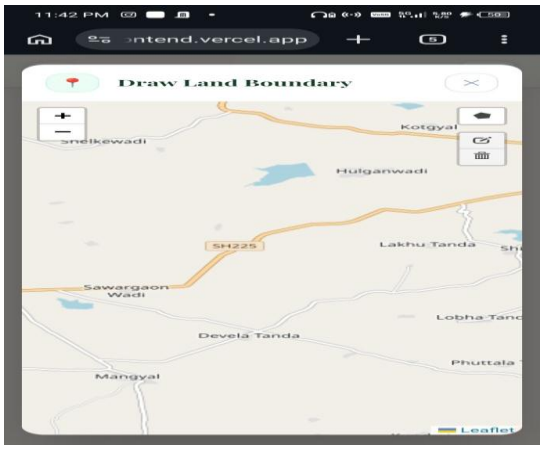

Fig -6: Draw Boundary

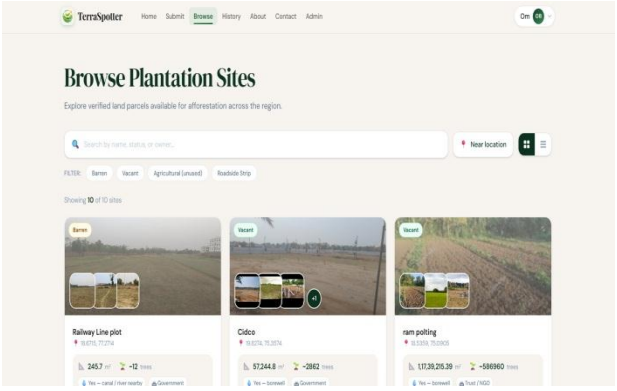

Fig-7: Browse Plantation Sites

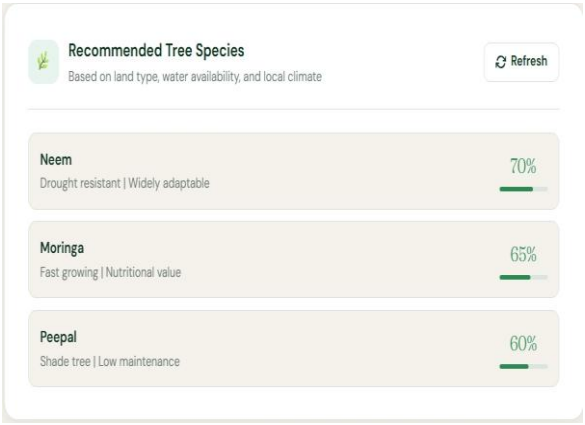

Fig- 8: Ml Recommendation

4. CONCLUSION

In this paper, TerraSpotter: Precision Afforestation Tool was successfully implemented as a web-based system for identifying and managing suitable plantation sites. The platform combines mapping, environmental data, and machine learning to provide accurate tree recommendations and support better decision-making. It also encourages community participation through validation and feedback features. Overall, the system makes afforestation planning more practical, organized, and data-driven. In the future, the system can be enhanced by adding mobile support, advanced analytics.

5. REFERENCES

[1] X. Zhou and X. Zhang, "Individual Tree Parameters Estimation for Plantation Forests Based on UAV Oblique Photography," IEEE Access, vol. 8, pp. 97654–97663, May 2020.

[2] R. Reitberger, N. Pattnaik, and Y. Lu, "Urban tree placement analysis: a GIS-based approach for identifying suitable planting locations in Munich," in Proc. Conf., Sept. 2024..

[3] S. Badamasi et al., "Geospatial Mapping of Agricultural Land Suitability in Zaria, Nigeria," FUDMA J. Sci., vol. 6, no. 4, pp. 212–221, Aug. 2022.

[4] K. Schulze et al., "Pantropical distribution of short-rotation woody plantations," Mitig. Adapt. Strateg. Glob. Change, vol. 28, no. 28, pp. 1–22, May 2023.

[5] N. Patel et al., "Agroforestry site suitability for wasteland greening," J. Geomatics, vol. 18, no. 1, pp. 50–66, Apr. 2024.

[6] C. Bakolo et al., "Optimal locations for green space initiatives via GIS and AHP," Environ. Syst. Res., vol. 13, no. 46, 2024.

[7] N. Yadav, S. Rakholia, and R. Yosef, "Decision support system for tree selection and plantation," Forests, vol. 15, no. 1219, pp. 1–17, Jul. 2024.

[8] S. Tsiaras and C. Domakinis, "Use of GIS in Selecting Suitable Tree Crop Cultivation Sites in Mountainous Less Favoured Areas: An Example from Greece," Forests, vol. 14, no. 6, 2023.

[9] S. Yousefi et al., "Identification of the most suitable afforestation sites by machine learning methods: A case study in Iran," 2021.

[10] C. Yağcı et al., "GIS-based site suitability analysis of afforestation in Konya province (Turkey) using AHP

method," Turkish J. Geographic Information Systems, vol. 3, no. 2, pp. 89–95, 2021.

[11] M. Topuz and M. Deniz, "Application of GIS and AHP for land use suitability analysis: case of Demirci district (Turkey)," Humanities & Social Sciences Communications, vol. 10, article 115, 2023.

[12] A. Gholizadeh et al., "Model application in evaluating land suitability for Oak and other forest plantations (Quercus robur etc.)," Land Use Policy, 2020.

[13] M. A. E. AbdelRahman et al., "Enhancing land suitability assessment through integration of topography, soil, and climate factors," Scientific Reports, 2025.

[14] M. Armin and A. S. M. Abdolrassoul, "A Fuzzy Multi-Criteria Assessment of Land Suitability for Afforestation with Eucalyptus grandis," J. Applied Biosciences, 2010.