# **Project: Detection and Monitoring of Land Use Changes (Deforestation) in Córdoba, Argentina using Satellite Images**

### **Steps of the Document:**

1. Project Summary
2. Detailed Work Plan
3. Additional Considerations
4. Expected Outcomes
5. Resources and References
6. Team Division and Roles
7. Workflows and Synchronization
8. Bottlenecks and Proactive Solutions

## **Project Summary**

This project aims to develop a robust change detection platform using pre-trained models and ensemble learning to analyze satellite images at two time points (t1 and t2), eliminating the need for manual data tagging. The tool will identify and map land-use changes—such as deforestation or urban expansion—and provide an accessible interface, exportable reports, and a well-documented API. By serving authorities, NGOs, researchers, and the public, it enables efficient monitoring, informed decision-making, and timely action.

## **Detailed Work Plan**

### **Data Collection**

#### ***Identification of Data Sources***

* **Select free satellite image providers**, such as Sentinel-2 and Landsat 8 (using Google Earth Engine)
* **Verify the availability** of images for dates t1 and t2 (filter, map, reduce).

#### ***Image Download***

* **Download images** for times t1 and t2.
* **Ensure images have minimal cloud coverage** for clear analysis.

### **Data Preprocessing**

#### ***Image Correction***

* **Apply atmospheric corrections** using algorithms like DOS (Dark Object Subtraction) or the Sen2Cor module for Sentinel-2.
* **Perform geometric corrections** to precisely align the images spatially.

#### ***Image Registration***

* **Use image registration techniques** to perfectly align the t1 and t2 images at the pixel level.

#### ***Band Selection and Filtering***

* **Select relevant spectral bands** (e.g., red, green, blue, near-infrared bands for vegetation analysis).
* **Apply kernels (filters)** like Gaussian Blur to reduce noise and enhance edge detection.

### **Exploratory Data Analysis**

#### ***Calculation of Spectral Indices***

* **Calculate NDVI**, EVI, NDBI, and other relevant indices to highlight vegetation, built-up areas, and water bodies.
* **Generate thematic maps** based on these indices.

#### ***Preliminary Mapping***

* **Create initial maps** to visualize potential change areas.

### **Change Detection Modeling**

#### ***Implementation of Change Detection Techniques***

* **Apply pixel-based methods** like Image Differencing and Image Ratioing.
* **Employ object-based methods** using clustering segmentation (e.g., K-means, Mean Shift).

#### ***Development of Advanced Algorithms***

* **Research pre-trained models** such as:
  + [Change Detection ML Models](https://github.com/satellite-image-deep-learning/techniques?tab=readme-ov-file#change-detection)
  + [Awesome-Geospatial GitHub Repository](https://github.com/sacridini/Awesome-Geospatial?tab=readme-ov-file)

#### ***Application of Post-Processing Methods***

* **Use morphological filtering techniques** to refine results.
* **Apply edge detection algorithms** to improve spatial accuracy.

### **Model Validation and Evaluation**

#### ***Model Validation***

* **Compare model results** with reference data or high-resolution images.
* **Calculate accuracy metrics**, such as Kappa Index, Overall Accuracy, User's and Producer's Accuracy.

#### ***Temporal and Spatial Analysis***

* **Identify areas with significant changes** and analyze spatial patterns.
* **Evaluate the magnitude and direction** of land-use changes.

#### ***Report Generation***

* **Compile findings** into detailed reports.
* **Prepare visualizations and thematic maps** to communicate results.

### **API Development**

#### ***API Architecture Design***

* **Define the structure, endpoints, and necessary functionalities**.
* **Consider authentication protocols** and request handling.

#### **Implementation of Queue System with Redis**

##### **Integration of Redis as a Queue System**

* **Configure Redis** to manage a queue of incoming requests.
* **Implement logic** to enqueue image processing requests.

##### **Limiting Concurrency**

* **Set limits** on the number of images processed simultaneously.
* **Use workers** to process queued tasks in a controlled manner.

#### **Request Handling**

* **Design mechanisms** to inform users about the status of their request (queued, processing, completed).
* **Implement timeouts and retries** in case of failures.

#### **API Development**

* **Use FastAPI** to build the API.
* **Implement endpoints to**:
  + Upload images and necessary data.
  + Submit processing requests to be queued in Redis.
  + Retrieve results and request statuses.

#### **Optimization and Security**

* **Ensure efficient request handling** without overloading the backend.
* **Implement authentication and authorization** if necessary.

#### **Documentation**

* **Create clear and detailed documentation** for API users.
* **Include examples** of requests and responses.

### **Front-End Development**

#### **UI/UX Design**

* **Create wireframes and prototypes** of the user interface.
* **Focus on usability and accessibility** for non-technical users.

#### **Integration with API and Queue System**

* **Connect the front-end application** with the back-end API.
* **Implement functionalities to**:
  + Display the processing status of requests.
  + Notify users when their request has been processed.
* **Ensure efficient communication** and error handling.

#### **Implementation of Interactive Features**

* **Incorporate interactive maps** using libraries like Leaflet or Mapbox.
* **Allow users to visualize different layers** and compare between t1 and t2.

### **Deployment and Testing**

#### **Hosting Configuration**

* **Select a cloud platform** (AWS, Azure, Heroku) for deployment.
* **Configure necessary servers and databases**.

#### **Redis Implementation in Production**

* **Set up Redis instances** in the production environment.
* **Ensure scalability and availability** of the queue service.

#### **Application Deployment**

* **Deploy back-end and front-end services** in the production environment.
* **Configure SSL certificates and domains** if necessary.

#### **Testing and Quality Assurance**

* **Conduct exhaustive testing** of functionality, performance, and security.
* **Test the queue system under high loads** to ensure robustness.
* **Fix bugs and optimize code** as necessary.

#### **Launch Preparation**

* **Plan a launch strategy**.
* **Prepare communication materials and guides** for users.

## **Additional Considerations**

### **Request Management and Scalability**

#### **Redis Implementation for Load Handling**

* **Use Redis as a messaging system** to enqueue and manage processing tasks.
* **Avoid model saturation** by limiting the number of concurrent processes.

#### **Resource Optimization**

* **Reduce the need for large backend instances** by optimizing resource usage.
* **Scale horizontally** by adding more workers if necessary.

### **Application of Different pre-trained Models and Algorithms**

* **Explore and compare multiple change detection models** mentioned in the provided list, such as:
  + **Siamese Networks** for similarity-based change detection.
  + **Transformer-based models** like ChangeFormer.
  + **Unsupervised methods** like PCA and K-means Clustering.
  + **Lightweight deep learning models** like TinyCD.

### **Application of Advanced Kernels and Filters**

* **Use Gaussian filters** to smooth images and reduce noise.
* **Apply edge detectors** like Canny or Sobel to enhance change detection.

### **Handling Cloud and Shadow Data**

* **Implement methods to detect and mask clouds and shadows** that may affect model accuracy.
* **Use cloud detection algorithms** like Function of Mask (Fmask).

## **Expected Outcomes**

### **Functional Change Detection Model**

* **Achieve** at least a 0.75 Kappa Index and 85% Overall Accuracy in detecting land-use changes between t1 and t2.
* **Successfully validate results** against reference datasets or higher-resolution imagery to ensure reliability.

### **Accessible Platform**

* **Provide a user interface** enabling non-technical users to navigate through different time points, visualize indices (e.g., NDVI), and compare layers without requiring advanced technical knowledge.
* **Offer a clearly documented, versioned API** that external stakeholders can easily integrate with their systems, receiving standardized data formats (e.g., JSON, GeoJSON).

### **Efficient Queue System**

* **Maintain continuous image processing** on a dedicated cloud instance (e.g., EC2) without overloading system resources.
* **Enforce** a maximum number of concurrent tasks to keep CPU and memory usage within healthy ranges (e.g., CPU <80%, memory <70%).
* **Support horizontal scaling** by adding more workers as needed, ensuring 24/7 operation without service interruptions or crashes.

### **Reports and Documentation**

* **Generate comprehensive, reproducible reports** that include methodologies, monitored areas, validation metrics, and resulting visualizations.
* **Produce standardized, easily shareable reports** (PDF, CSV, GeoJSON) suitable for regular distribution (e.g., monthly) to authorities, NGOs, and other stakeholders.

## **Resources and References**

### **Datasets and Tools**

* **Satellite Images**: Sentinel-2, Landsat 8 (Google Earth Engine)
* **Processing Tools**: QGIS, GeoPandas, GDAL, Python (NumPy, Pandas, Scikit-learn, TensorFlow, PyTorch).
* **Visualization Libraries**: Matplotlib, Seaborn, Plotly, Leaflet.
* **Backend Tools**: FastAPI, Redis, Celery (for asynchronous task management).

### **Models and Algorithms**

* **Siamese Networks**: For similarity-based change detection.
* **U-Net and Variants**: For semantic segmentation.
* **Transformers**: Models like ChangeFormer for multi-scale detail capture.
* **Unsupervised Models**: PCA, K-means, DBSCAN.

### **Recommended Readings**

* **Articles and Publications**: Review recent publications on change detection in remote sensing.
* **GitHub Repositories**: Explore implementations of mentioned models to better understand their applications.

**Note**: The integration of Redis as a queue system is crucial for efficiently handling processing requests and avoiding overloading the model and backend. By enqueuing requests, we ensure that each task is processed in order and limit the number of concurrent processes, allowing the use of more modest resources without sacrificing performance.

**Suggested Technical Implementation**:

* **Use Celery with Redis**: Consider using Celery as an asynchronous task system that works alongside Redis to handle queues. This facilitates task definition and worker management for processing them.
* **User Notifications**: Implement mechanisms for users to check the status of their requests and receive notifications when processing is complete.
* **Scalability**: The queue-based architecture allows horizontal scaling by adding more workers if demand increases.

## **Team Division and Roles**

### **1. Pre-Trained Models and Experimentation Team**

**Objective**: Evaluate and compare different pre-trained models using the project's specific data.

**Tasks**:

* **Subgroup 1**: Experimentation with traditional models (e.g., Siamese Networks, PCA, K-means).
* **Subgroup 2**: Experimentation with deep learning-based models (e.g., U-Net, ChangeFormer).
* **Subgroup 3**: Application of ensemble learning techniques to combine results and optimize accuracy.
* **Subgroup 4**: Post-processing of results (e.g., refinement with morphological filtering, edge detection).

**Key Roles**:

* **Remote sensing and machine learning specialists**.
* **Programmers familiar with deep learning frameworks** like TensorFlow or PyTorch.
* **Analysts to validate model results**.

**Team Size**: 6-8 people.

### **2. Processing and Optimization Team (Redis and Backend)**

**Objective**: Implement a robust system to manage processing requests using Redis as a queue system and ensuring an efficient backend.

**Tasks**:

* **Configure Redis** to handle queued requests.
* **Implement workers** for asynchronous image processing.
* **Optimize the API (FastAPI)** to limit concurrency and ensure scalability.
* **Document the system flow and integrations**.

**Key Roles**:

* **Backend developers experienced in FastAPI, Redis, and Celery**.
* **Specialists in distributed systems optimization**.

**Team Size**: 4-5 people.

### **3. Data Preprocessing and Analysis Team**

**Objective**: Prepare input data and perform exploratory analysis to ensure models function correctly.

**Tasks**:

* **Download and verify satellite images** for t1 and t2.
* **Preprocess images** (atmospheric corrections, image registration, band selection).
* **Calculate relevant spectral indices** (NDVI, NDBI, etc.).
* **Generate preliminary thematic maps** to identify areas of interest.

**Key Roles**:

* **Remote sensing and image processing specialists** (familiar with tools like QGIS, GDAL).
* **Programmers experienced in data analysis** (Python, NumPy, Pandas).

**Team Size**: 4-5 people.

### **4. Front-End and Visualization Team**

**Objective**: Create an intuitive and accessible interface for users to visualize results and understand detected changes.

**Tasks**:

* **Design the front-end** (wireframes, prototypes).
* **Implement interactive maps** using Leaflet or Mapbox.
* **Connect the front-end with the API** to display results dynamically.
* **Provide options to download reports and visualizations**.

**Key Roles**:

* **UI/UX designers**.
* **Front-end developers** with experience in React or similar frameworks.
* **API integrators**.

**Team Size**: 4-5 people.

## **Workflows and Synchronization**

To ensure teams work cohesively:

* **Establish Team Leaders**: Each team should have a leader responsible for coordinating tasks, resolving blockers, and reporting progress to the rest of the team.
* **Weekly Meetings**: Synchronization meetings to share progress, resolve dependencies, and adjust priorities.
* **Collaboration Platforms**: Use tools like Trello or Jira for task management and Slack or Discord for communication.
* **General Schedule**: Follow an iterative plan where each team delivers partial results that integrate at the end.

### **Example of Coordinated Workflows**

**Weeks 1-2**:

* **Data Preprocessing Team** works on data preparation.
* **Models Team** starts basic tests with a reduced dataset.
* **Redis Team** sets up the queue system for initial tests.
* **Front-End Team** designs wireframes and basic prototypes.

**Weeks 3-4**:

* **Models Team** finalizes tests and delivers optimized results.
* **Redis Team** integrates their system with the API.
* **Data Preprocessing Team** delivers data ready for modeling and validation.
* **Front-End Team** begins API integration.

**Weeks 5-6**:

* **Teams adjust models**, optimize Redis, and conduct joint API and visualization tests.
* **All teams work on integration tests** and final adjustments.

## **Bottlenecks and Proactive Solutions**

### **1. Data Download and Preprocessing**

**Potential Bottleneck**:

* Without preprocessed and aligned satellite images, the Models and Experimentation Teams cannot begin their tasks.

**Prerequisites Needed**:

* **Availability Verification**: Ensure images for t1 and t2 are downloaded in time.
* **Basic Preprocessing**:
  + Atmospheric and geometric corrections applied.
  + Image registration completed (precise spatial alignment).
  + Relevant band selection ready.

**Proactive Solution**:

* **Assign top priority to the Data Preprocessing Team during the first two weeks**.
* **Use placeholder preprocessed datasets** while completing the full image processing pipeline.

### **2. Redis Queue System Configuration**

**Potential Bottleneck**:

* Without Redis configured, the backend cannot handle multiple processing requests, delaying integrated tests between the Backend and Models Teams.

**Prerequisites Needed**:

* **Initial Redis configuration** in a local or development environment.
* **Basic connection between Redis and the backend API** to enqueue requests.
* **A simple script simulating workers processing queued tasks**.

**Proactive Solution**:

* **Dedicate early time (Week 2)** to implement a minimum viable product (MVP) of Redis integrated with the API.
* **Test Redis with simple tasks** before connecting it with real models.

### **3. Data Inputs for the Models Team**

**Potential Bottleneck**:

* Pre-trained models cannot be tested without preprocessed and aligned datasets.

**Prerequisites Needed**:

* **Subset of preprocessed images (t1 and t2)** ready for initial tests.
* **Clear specification of necessary spectral bands** for each model.
* **An initial pipeline delivering images in the expected format** for the models.

**Proactive Solution**:

* **Generate a small preprocessed test dataset** to use while completing the full pipeline.
* **Document dataset characteristics required by each model**.

### **4. Backend and Models Integration**

**Potential Bottleneck**:

* If models are unavailable or API endpoints are not ready, the Backend and Models Teams will block each other.

**Prerequisites Needed**:

* **Basic functional API endpoint** that accepts input images and returns a dummy response.
* **Initial version of a pre-trained model connected to the backend** (even with limited functionality).
* **Local experimentation pipeline with models** ready before attempting API integration.

**Proactive Solution**:

* **Work in parallel on a simple model** (like PCA or Image Differencing) while advanced models are prepared.
* **Prioritize having basic functional API endpoints from Week 3**.

### **5. Validation and Metrics**

**Potential Bottleneck**:

* Without a clear flow to validate model results (metrics, reference data), it will be difficult to iterate and improve results.

**Prerequisites Needed**:

* **Generation of thematic maps and initial metrics** (like NDVI, NDBI) during exploratory analysis.
* **High-resolution datasets or local references** for comparison.
* **Basic visualization tools** to interpret initial results.

**Proactive Solution**:

* **Create automated scripts to calculate metrics and generate maps from Week 2**.
* **Document validation criteria from the start**.

### **6. Front-End and Visualization**

**Potential Bottleneck**:

* Without available API and processed results, the Front-End Team will lack real data to display.

**Prerequisites Needed**:

* **Test data (dummy model results)** ready to integrate with the front-end.
* **Basic API structure functional** to retrieve results.
* **Mockups or initial prototypes of thematic maps** to iterate the design.

**Proactive Solution**:

* **Provide the Front-End Team with simulated data** to work in parallel.
* **Establish clear communication between teams** to update API endpoints quickly.

# **Optional Advanced Approaches**

## **1. Integration of Vision-Language Models for Enhanced Remote Sensing**

### **1.1 Overview**

In addition to the traditional or deep learning methods described earlier, Vision-Language Models (VLMs) offer extended capabilities in identifying, describing, and explaining changes in remote sensing (RS) imagery. These models combine visual information (satellite images) with textual or linguistic input/output, providing functionalities like captioning, question answering, and descriptive change detection.

### **1.2 Potential VLM Frameworks**

Below are some state-of-the-art frameworks and ideas that could be explored or adapted to our use case:

* **ChangeMinds: Multi-Task Framework for Remote Sensing Change Detection and Captioning**A unified multi-task framework for both change detection (CD) and change captioning (CC). Integrates spatiotemporal modeling through a specialized ChangeLSTM and utilizes cross-attention for joint optimization. It can identify changes and also provide a textual description of those changes in a single workflow.
* **GEOBench-VLM: Benchmarking Vision-Language Models for Geospatial Tasks**A benchmark that tests VLMs on geospatial-specific tasks such as object counting, fine-grained categorization, and temporal analysis. Highlights the gaps in current VLMs when dealing with geospatial complexity and large-scale earth observation data.
* **GeoChat: Grounded Large Vision-Language Model for Remote Sensing**A Large Vision-Language Model tailored for high-resolution RS imagery. It handles small-object detection, diverse scales, and region-specific dialogues. GeoChat supports multitask conversational features, allowing spatially grounded question answering and referencing—ideal for interactive platforms where users query localized changes.
* **RS5M and GeoRSCLIP**
  + **RS5M**: A large-scale dataset of 5 million RS image-text pairs.
  + **GeoRSCLIP**: A domain-specific vision-language model fine-tuned on RS tasks such as zero-shot classification, cross-modal retrieval, and semantic localization. Useful for bridging the gap between general-purpose VLMs and RS-specific needs.
* **VHM: Versatile and Honest Vision Language Model for Remote Sensing Image Analysis**A domain-focused model using a large-scale RS dataset (VersaD). Integrates CLIP-based visual encoders and Vicuna-based language models for improved multimodal instruction following and accurate analysis of RS imagery.

### **1.3 Integration Considerations**

* **User Queries and Conversational Workflow**Incorporate a chatbot-like interface (e.g., using GeoChat or a custom fine-tuned LLM) that allows users to ask domain-specific questions (“Show me areas with significant deforestation”) and get immediate visual or textual responses.
* **Multi-Task Learning**Combine change detection, captioning, and Q&A tasks under a single model or ensemble approach, leveraging pre-trained networks from the frameworks above.
* **Data Requirements**Large volumes of annotated or partially labeled text-image pairs may be required for optimal model fine-tuning. Leverage existing RS datasets (e.g., RS5M) for initial training.
* **Performance and Resource Management**Deploy large models efficiently—consider techniques like model pruning, quantization, or using GPU-backed instances to handle high-resolution imagery and complex queries in real time.

## **2. Potential Use of Vector Embeddings for Natural Language-Driven Change Detection**

### **2.1 Overview**

Vector embeddings—especially those derived from CLIP-based models—can capture the semantic content of an image in a high-dimensional vector space. By comparing embeddings from two different time points (t1 and t2), it becomes possible to detect changes without relying solely on pixel-level differencing. Moreover, if these embeddings are aligned with text embeddings, you can perform semantic searches and queries via natural language (e.g., “Find regions where forest was replaced by housing between 2020 and 2023”).

### **2.2 Workflow**

1. **Image Embedding Generation**
   * Use a CLIP-based or similar model to convert satellite images at each time point into vectors representing their semantic content.
2. **Temporal Comparison**
   * Compute the distance or similarity between embeddings at t1 and t2 to detect potential changes (higher distance often indicates more significant change).
3. **Text Alignment and Semantic Querying**
   * If the model also embeds text into the same vector space, compare the image embeddings to textual embeddings (e.g., “forest,” “urban area,” “solar panels”) to identify and label the nature of changes.
4. **Results Interpretation**
   * For deeper insights, integrate an LLM to summarize changes in a more natural form:  
     *“Between 2018 and 2021, this area shows decreased similarity to ‘forest’ and increased similarity to ‘houses,’ indicating housing development in what used to be a forest.”*

### **2.3 Benefits and Considerations**

* **Advantages**
  + Less sensitive to minor variations like slight radiometric differences or haze.
  + Enables natural language-based queries to find specific types of changes (“deforestation,” “new solar farms,” etc.).
* **Limitations**
  + Still requires good image alignment to avoid large geometric offsets.
  + Seasonal changes or shadows might cause false positives/negatives unless carefully managed.
  + Fine-grained or small-scale changes might be missed if not prominent enough to alter the overall semantic embedding.
* **Examples of Use Cases**
  + Tracking environmental changes: deforestation, wetland reduction, crop rotation.
  + Monitoring infrastructure developments: new roads, bridges, parking lots.
  + Searching for transformations in urban planning or industrial expansions.

### **2.4 Potential Extensions**

* **Multimodal LLM Summaries**
  + Apply a large language model to generate textual descriptions of changes over a broad region, essentially automating report creation.
* **Integration with Queue System**
  + Embed the vector comparison pipeline into the existing Redis-based queue system to handle large batches of images efficiently.
* **Hybrid Approach**
  + Combine embedding-based methods with traditional supervised or unsupervised techniques for enhanced reliability and interpretability.