ACKNOWLEDGEMENT

An endeavor is not complete and successful till the people who made it possible are given due credit for making it possible. I take this opportunity to thank all those who have made the endeavor successful for me.

At the very onset I thank Dr. Neeru Rathee, my Major Project – Dissertation supervisor for giving inspiration on such important and valuable topic, his scholarly guidance, constant supervision and encouragement to make it success. His research knowledge and passion for problem solving amazes and inspires me. At all crucial stages, valuable insights given by him, made me take the right direction. I thank her for the countless hours she has spent with me, criticizing my ideas, enlightening my writing skills, and helping me. Her assistance during this work has been invaluable and inspirational.

I extend my thanks to Prof. Archana Balyan, Director , MSIT and Prof. Neeru Rathee , HOD-ECE , MSIT for their regular motivation, support and pray full presence.

VIBHOR JOSHI

03615002821

Implementation of GeoAI Techniques

(ES-452: Major Project - Dissertation)

submitted in partial fulfillment of the requirement
for the award of the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

Submitted by

VIBHOR JOSHI
03615002821

Under the supervision of

DR. NEERU RATHEE
HEAD OF DEPARTMENT

Department of Electronics and Communication Engineering
Maharaja Surajmal Institute of Technology

May/June 2025

DECLARATION

This is to certify that the material embodied in this Major Project - Dissertation titled “implementation of geo ai technique” being submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics & Communication Engineering is based on my original work. It is further certified that this Major Project - Dissertation work has not been submitted in full or in part to this university or any other university for the award of any other degree or diploma. My indebtedness to other works has been duly acknowledged at the relevant places.

Vibhor Joshi
03615002821

Abstract

This research project implements and analyzes GeoAI techniques for building footprint regularization, addressing a critical challenge in geospatial data processing. Irregularities in building footprints extracted from remote sensing imagery often exhibit geometric inconsistencies that reduce their utility for urban planning, 3D modeling, and spatial analysis. This project develops a comprehensive solution that combines computational geometry and artificial intelligence approaches to transform imprecise building outlines into regularized representations with improved orthogonality and simplified geometry while preserving essential shape characteristics.

The methodology encompasses a complete pipeline from data acquisition through regularization to visualization and evaluation. Three distinct regularization approaches are implemented and compared: a basic orientation-based method, a hybrid technique that integrates multiple regularization strategies, and an adaptive approach that tailors processing based on building attributes. The solution utilizes the geoai-py library for initial building footprint extraction and develops custom algorithms for orientation detection, orthogonalization, and simplification..

This research contributes to the field of GeoAI by providing an open-source, well-documented solution for building footprint regularization that can be integrated into broader geospatial workflows. The findings highlight the importance of context-aware parameter tuning and demonstrate how computational geometry techniques can be effectively combined with modern visualization approaches to improve geospatial data quality..

CERTIFICATE

This is to certify that the work embodied in this Major Project - Dissertation titled “implementation of geo ai technique” being submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication, is original and has been carried out by VIBHOR (03615002821) under my supervision and guidance.

It is further certified that this Major Project - Dissertation work has not been submitted in full or in part to this university or any other university for the award of any other degree or diploma to the best of my knowledge and belief.

Dr NEERU RATHEE
Associate professor

(Name of the HOD)

HOD – Dr NEERU RATHEE

Name of the Institute- Maharaja Surajmal Institute of technology

TABLE OF CONTENTS

Page No (in Roman)

LIST OF FIGURES

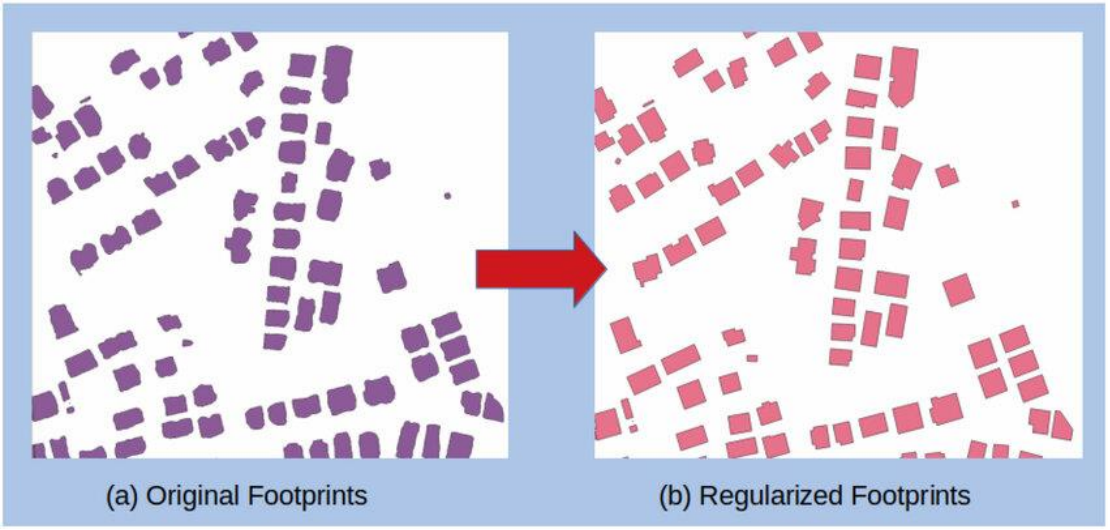
Figure No.	Figure Title	Page No.

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Overview of Footprint regularization

Building footprints, defined as the spatial extent of a building's base, serve as a foundational layer in numerous geospatial applications. These applications range from urban planning and resource management to disaster response and the creation of three-dimensional city models. The accuracy and reliability of building footprint data are paramount, as they directly influence analytical outcomes and decision-maki



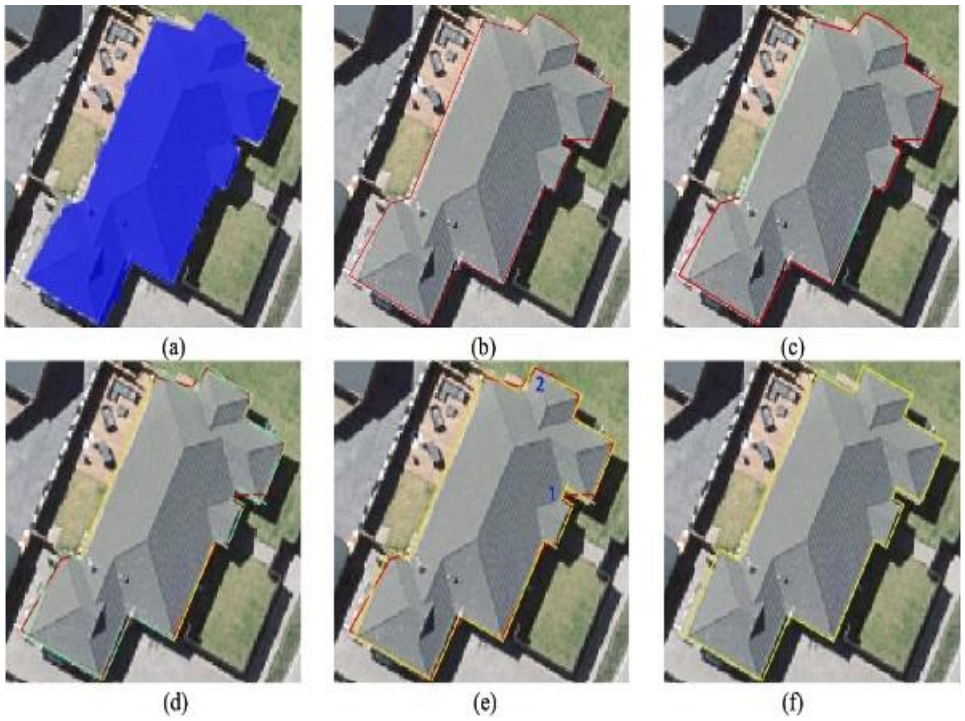
ng processes across these domains.

The process of automatically extracting building outlines from remote sensing imagery offers significant advantages in terms of efficiency and scalability. However, this automated extraction often results in geometric imperfections that can limit the direct usability of the data. These imperfections commonly include an excessive number of vertices leading to overly complex polygons, jagged or irregular edges that deviate from the straight lines characteristic of most buildings, and a lack of orthogonality where expected right angles at building corners are not precisely captured.

Building footprint regularization emerges as a crucial step in the overall workflow, refining these automatically extracted outlines to eliminate undesirable artifacts and enhance their geometric quality. By applying regularization techniques, the accuracy

and visual appeal of building footprints can be significantly improved, making them more suitable for a wider range of geospatial analyses and applications.

The increasing availability of high-resolution remote sensing data provides unprecedented detail about the Earth's surface and built environment. However, to fully capitalize on this wealth of information and ensure its utility for accurate geospatial analysis, robust building footprint regularization techniques are essential. Higher spatial resolution in satellite and aerial imagery allows for extraction of increasingly finer details of buildings, but this heightened level of detail often comes with increased geometric complexity and potential noise in the extracted polygon boundaries.



1.1.2 Importance in Various Applications

- Environmental Monitoring: Tracking changes in land use, deforestation, and urbanization, which are critical for conservation and sustainable development.

- Disaster Management: Assessing damage from natural disasters like floods, earthquakes, and wildfires, aiding in resource allocation and response strategies.

- Agriculture: Monitoring crop health, predicting yields, and supporting precision farming through detailed land cover analysis.
- Urban Planning: Providing insights into urban sprawl, infrastructure development, and land cover changes for informed city planning.

The automation through deep learning enhances accuracy and efficiency, addressing the limitations of manual analysis, which is particularly important given the high resolution and complexity of satellite data.

1.2 Background and Motivation

The accelerating pace of urbanization worldwide has amplified the critical need for accurate, up-to-date, and comprehensive geospatial data describing the built environment. Among the most fundamental of these datasets are building footprints, which provide the precise location and outline of structures on the ground. Building footprints are essential inputs for a vast array of applications, including urban planning and development, cadastre and property management, infrastructure mapping and utility planning, population estimation and demographic analysis, disaster risk assessment and emergency response, environmental monitoring, and the creation of detailed 3D city models. Reliable building footprint data enable more informed decision-making, efficient resource allocation, and effective urban governance.

Historically, the acquisition and maintenance of building footprint datasets have been labor-intensive and time-consuming processes. Traditional methods primarily rely on manual digitization by skilled operators interpreting aerial or satellite imagery. While capable of producing highly accurate results, manual digitization is prohibitively expensive and slow when dealing with large geographic areas or requiring frequent updates, particularly in rapidly growing urban centers. The sheer volume of available high-resolution satellite and aerial imagery today far exceeds the capacity for manual processing at scale.

This bottleneck in data acquisition has motivated significant research and development into automated methods for extracting building footprints from remote sensing data. Early automated techniques utilized conventional image processing algorithms, feature extraction, and machine learning classifiers. However, these methods often

struggled with the complexity and variability of real-world urban landscapes, including diverse building types, materials, shapes, sizes, and challenging environmental conditions such as shadows, occlusions, and varied lighting.

CHAPTER 2: PROBLEM STATEMENT

2.1 Problem Definition

Building footprints extracted automatically from remote sensing imagery often suffer from geometric imperfections that compromise their utility in geospatial applications. These imperfections include:

1. ***Excessive vertices:*** *Automatically extracted building footprints often contain an unnecessarily large number of vertices, resulting in overly complex polygons that do not accurately represent the building's true form.*
2. ***Jagged or irregular edges:*** *The extraction process frequently produces edges that are jagged or irregular, deviating from the straight lines characteristic of most constructed structures.*
3. ***Lack of orthogonality:*** *Right angles at building corners, a common feature in most man-made structures, are often not precisely captured in automatically extracted footprints.*
4. ***Misalignment with actual building boundaries:*** *Extracted footprints may not perfectly align with the actual boundaries of buildings due to limitations in image resolution, perspective distortions, or algorithmic constraints.*
5. ***Inconsistent representation across different buildings:*** *Variations in building size, shape, and surrounding context can lead to inconsistencies in the quality of extracted footprints.*

These imperfections significantly limit the direct usability of automatically extracted building footprints in applications that demand high levels of spatial precision and geometric consistency. For instance, irregular building outlines can complicate the process of 3D city modeling, reduce the accuracy of urban planning analyses, and diminish the visual quality of mapping products.

While various regularization techniques have been developed to address these issues, there remains a need for systematic evaluation of their effectiveness and applicability in different contexts. The geoai-py package offers multiple regularization methods, but their comparative performance and optimal parameter settings for different types of building footprints have not been thoroughly investigated.

This project addresses the problem of evaluating and comparing different building footprint regularization techniques available in the geoai-py package to determine their effectiveness in improving the geometric quality of automatically extracted building footprints from satellite imagery.

2.1.1Integration of Code Insights

Code Implementation	**Details**
Preprocessing	Input vector data (GeoJSON, SHP) is loaded using Geo Pandas
Core Logic	A custom regularization pipeline using Shapely functions simplifies and orthogonalizes building polygons**.**
Evaluation	Metrics such as vertex reduction, area preservation ratio, and Hausdorff distance are computed**.**

This GeoAI-based approach focuses on vector-based geometric refinement of urban structures rather than pixel-level image segmentation. Instead of labeling each pixel, this method processes building footprints as geometric shapes and applies computational geometry techniques to clean, simplify, and orthogonalize them.The technique relies on:

- Minimum Rotated Rectangle (MRR) to determine orientation
- Rotation, simplification, and edge snapping to regularize
- Shapely + GeoPandas for spatial processing
- Streamlit + Folium for interactive visualization

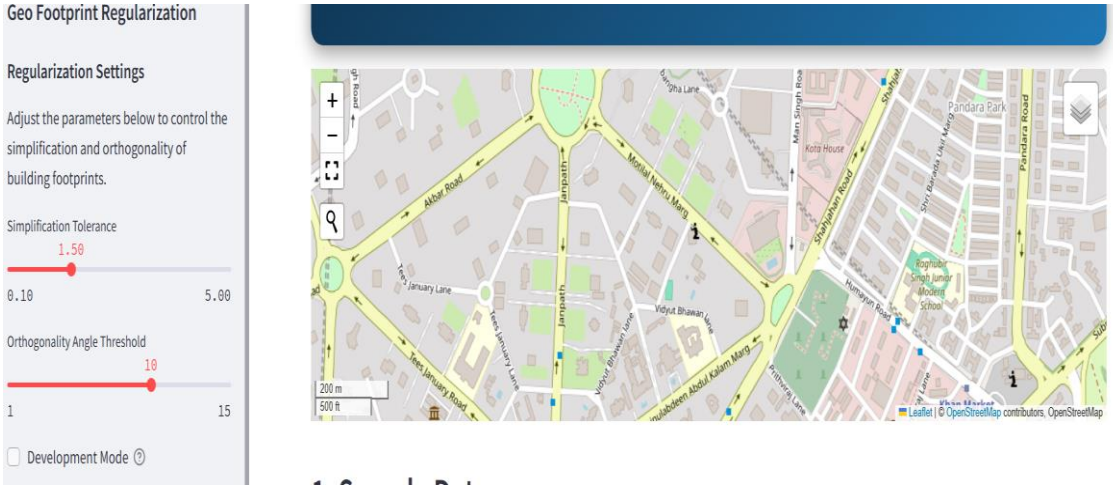
Unlike object detection or segmentation, this vector-based method is more efficient for applications involving urban planning, 3D modeling, and GIS data correction.

✅ 2.1.2 Features to be Regularized (GeoAI Classes)

In this vector-based regularization task, the goal is to refine the geometry of specific urban features. These include:

- **Buildings:** Primary focus of the project. Includes residential, commercial, and industrial structures. Regularization improves shape orthogonality and reduces vertex complexity, supporting 3D modeling, land-use planning, and digital cartography.
- **Complex Structures (MultiPolygons):** Some buildings may consist of disjoint parts (e.g., university campuses, shopping complexes). The tool regularizes each sub-structure while maintaining spatial coherence.
- **Rectilinear Simplicity:** While not a class per se, the tool emphasizes transforming irregular outlines into **more rectilinear forms**, enhancing visual consistency and analytical utility in GIS pipelines.

This contrasts with semantic segmentation classes (like vegetation or roads) since the method here operates **on geometric primitives** rather than image pixels.



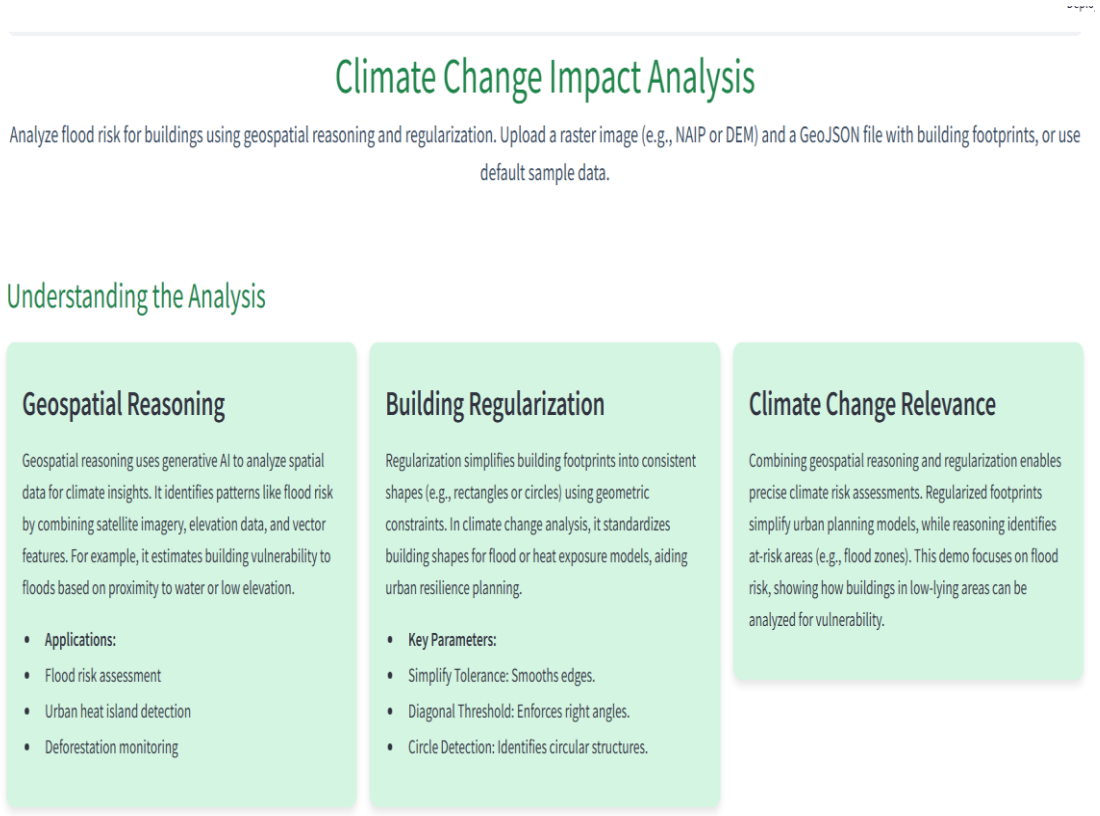
2.2 Objective

The primary goal of this project is to investigate building footprint regularization techniques using the geoai-py package. The specific objectives are:

1. **Explore and evaluate regularization functionalities in geoai-py:**
    - Understand the algorithms and methodologies underlying the regularization functions available in the geoai-py package.
    - Identify the strengths and limitations of each regularization method.
2. **Compare effectiveness of different regularization methods:**
    - Evaluate the performance of different regularization functions (e.g., regularization(), hybrid_regularization(), adaptive_regularization()) on building footprints extracted from satellite imagery.
    - Develop metrics and criteria for assessing the quality of regularized building footprints.
    - Conduct quantitative and qualitative comparisons of the results produced by different methods.
3. **Understand parameter influence on regularization outcomes:**
    - Investigate the impact of various parameters (e.g., angle_tolerance, simplify_tolerance, area_threshold) on the regularization results.
    - Determine optimal parameter settings for different types of building footprints and various urban morphologies.
    - Develop guidelines for parameter selection based on specific application requirements.

4. **Visually assess regularization quality:**
    - Implement interactive visualization techniques to compare original and regularized building footprints.
    - Overlay regularized footprints on satellite imagery to evaluate their alignment with actual building boundaries.
    - Develop comprehensive visual assessment methods for evaluating regularization quality.
5. **Create a systematic workflow for building footprint regularization:**
    - Develop a reproducible workflow that encompasses data acquisition, preprocessing, regularization, and evaluation.
    - Document the workflow to facilitate its adoption by other researchers and practitioners.
    - Provide recommendations for integrating regularization into broader geospatial analysis pipelines.

These objectives collectively aim to enhance understanding of building footprint regularization methods and contribute to more effective utilization of automatically extracted building footprints in various geospatial applications.



CHAPTER 3: ANALYSIS

3.1: Software Requirement Specifications:

3.1.1Data Acquisition and Management

The system shall be able to download sample NAIP (National Agriculture Imagery Program) imagery for testing purposes.

The system shall be able to access and process building mask data corresponding to the imagery.

The system shall support the import of user-provided satellite imagery and building mask data in common geospatial formats (GeoTIFF, JPEG2000, etc.)

The system shall maintain the spatial reference information of the input data throughout processing.

3.1.2 Building Footprint Processing

The system shall convert raster building masks to vector format (GeoJSON) representing building polygons.

The system shall support the processing of multiple building footprints in batch mode.

The system shall provide options to filter building footprints based on area, complexity, or other attributes.

The system shall preserve essential metadata (e.g., building ID, source) during processing.

3.1.3 Regularization Functions

The system shall implement the basic regularization function with configurable parameters:

1)angle_tolerance: to control the detection of angles for orthogonalization

2)simplify_tolerance: to control the level of polygon simplification

3)orthogonalize: to enforce right angles in building corners

4)preserve_topology: to maintain topological relationships during regularization

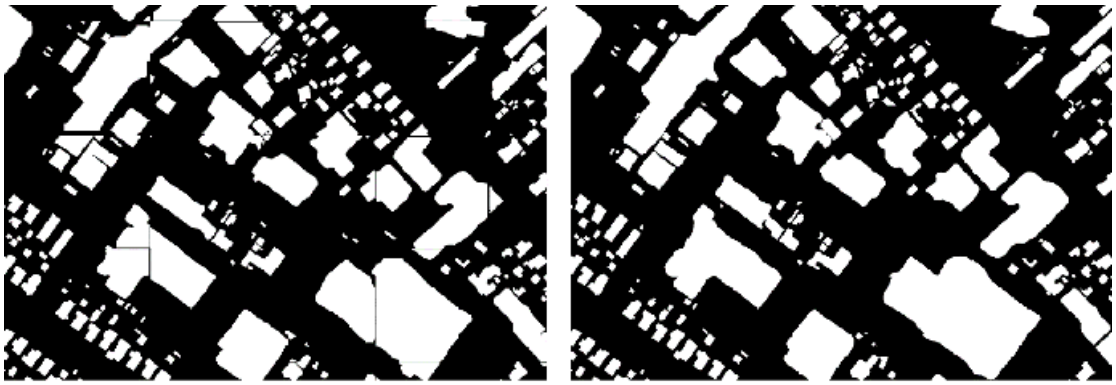
5)The system shall implement hybrid regularization with appropriate parameters.

3.1.4 The system shall implement adaptive regularization with parameters including:

1)simplify_tolerance: for polygon simplification

2)area_threshold: to adjust regularization based on building size

3)preserve_shape: to maintain the overall shape of buildings



(a) (b)

3.1.5 Parameter Experimentation

The system shall allow users to experiment with different parameter settings for each regularization method.

The system shall provide a way to save and reload parameter configurations.The system shall support the comparison of results obtained with different parameter settings.

The system shall generate summary statistics for each parameter configuration.

3.1.6 Visualization Requirements

The system shall generate interactive split maps showing original vs. regularized building footprints.

The system shall support the overlay of building footprints on the original satellite imagery.

The system shall provide options to customize the visualization (colors, transparency, etc.).

The system shall enable zooming, panning, and other basic map navigation functions.The system shall support the export of visualizations in common image formats (PNG, JPEG, PDF).

3.1.7 Evaluation and Comparison

The system shall calculate geometric metrics to assess regularization quality (e.g., vertex count reduction, orthogonality improvement).

The system shall generate comparative statistics for different regularization methods.

The system shall support side-by-side visual comparison of results from different methods.The system shall produce summary reports of the evaluation results.

3.1.8 Documentation and Output

The system shall generate comprehensive documentation of the regularization process.The system shall export regularized building footprints in standard GIS formats (GeoJSON, Shapefile).

The system shall provide options to save the entire project state for future reference.

The system shall generate reports summarizing the regularization process and results.

3.1.9 Performance Requirements

The system shall process a dataset of 1000 building footprints within 5 minutes on standard hardware.The visualization components shall respond to user interactions within 2 seconds.

The system shall efficiently handle large satellite images (>1 GB) without excessive memory consumption.The system shall implement parallel processing for batch operations where applicable.

3.1.10 Usability Requirements

The system shall provide clear documentation on how to use each feature.The user interface shall be intuitive and consistent throughout the application.

The system shall provide meaningful error messages for troubleshooting.The system shall include sample datasets and tutorials for new users.

3.1.11 Reliability Requirements

The system shall validate input data and parameters before processing.The system shall handle exceptions gracefully without crashing.

The system shall include logging mechanisms to track the processing workflow.The system shall implement data backup features to prevent loss of work.

3.1.12 Compatibility Requirements

The system shall be compatible with Python 3.8 or higher.The system shall work across different operating systems (Windows, macOS, Linux).

The system shall support common GIS data formats for input and output.The system shall integrate with existing GIS software and libraries.

3.1.13 Scalability Requirements

The system shall be able to scale to process citywide datasets with thousands of buildings.The code structure shall be modular to allow for future extensions.

The system shall support incremental processing of very large datasets.The memory usage shall scale efficiently with dataset size.

3.1.14 Maintainability Requirements

The code shall follow PEP 8 style guidelines for Python.The system shall include comprehensive code documentation.

The system shall use version control (Git) for tracking changes.The dependencies shall be clearly documented and managed.

3.1.15 Portability Requirements

The system shall be containerized using Docker for consistent deployment.The installation process shall be automated through package managers (pip, conda).

The system shall minimize platform-specific code.The system shall document any platform-specific considerations.

3.1.16 Security Requirements

The system shall implement secure methods for accessing remote data sources.The system shall validate and sanitize user inputs to prevent injection attacks.The system shall implement appropriate access controls for shared deployments.The system shall handle sensitive geospatial data in compliance with relevant regulations.

3.2: Feasibility Study of the Project

3.2.1 Technical Feasibility

The project leverages existing technologies and libraries, primarily the geoai-py package, which provides the core functionality for building footprint regularization. The technical implementation is feasible for the following reasons:

3.2.2 Availability of Required Software Components:

The geoai-py package is publicly available and actively maintained.Python provides a robust ecosystem of geospatial libraries (e.g., GDAL, Shapely, Rasterio) that can support the required operations.Interactive visualization tools like leafmap are readily available for creating the required visual comparisons.

3.2.3 Hardware Requirements:

The project can be executed on standard consumer hardware, though processing large datasets may benefit from higher-end specifications.

Cloud-based environments can be utilized for more computationally intensive tasks if needed.

3.2.4 Expertise Requirements:

The project requires proficiency in Python programming and geospatial data analysis.

Knowledge of GIS concepts and remote sensing principles is necessary but readily available through educational resources.

3.2.5 Data Availability:

Sample NAIP imagery and building mask data can be downloaded using functionalities provided by geoai-py.

Additional datasets are available from various open sources if needed for comprehensive testing.

3.2.6 Conclusion:

The project is technically feasible with the available tools, data, and expertise.

3.2.7 Economic Feasibility

The economic considerations for the project are favorable for the following reasons:

3.2.8 Development Costs:

The project primarily uses open-source software, minimizing licensing costs.Development time is estimated at 3-4 months for a small team, which is reasonable for a project of this scope.

Hardware costs are minimal as standard computing resources are sufficient.

3.2.9 Operational Costs:

Ongoing maintenance would require periodic updates to adapt to changes in the underlying libraries.

Cloud computing costs may apply if processing very large datasets, but these can be optimized.

3.2.10 Benefits:

Improved quality of building footprint data can lead to better decision-making in urban planning, disaster management, and other applications.

Automated regularization reduces the need for manual correction, resulting in time and cost savings.

The project contributes to the advancement of GeoAI research and applications.

3.2.11 Return on Investment:

For organizations dealing with large volumes of building footprint data, the efficiency gains from improved regularization can quickly offset the development costs.

The open-source nature of the project allows for broad adoption and community contributions.

3.2.12 Conclusion:

The project is economically feasible with potential for significant return on investment through improved data quality and reduced manual processing.

3.2.13 Operational Feasibility

The operational aspects of the project are manageable for the following reasons:

3.2.14 User Acceptance:

The project addresses a recognized need in the geospatial community.The interactive visualization components will help users understand and trust the regularization results.

Documentation and examples will facilitate adoption by users with varying levels of expertise.

3.2.15 Integration with Existing Workflows:

The focus on standardized GIS data formats ensures compatibility with existing geospatial workflows.The modular design allows for flexible integration into different processing pipelines.

3.2.16 Training and Support:

Comprehensive documentation and tutorials will be provided to facilitate user adoption.The open-source nature allows for community-based support and knowledge sharing.

3.2.17 Overall Feasibility Assessment

Based on the analysis of technical, economic, operational, and schedule aspects, the project is deemed feasible. The availability of the geoai-py package significantly reduces development time by providing ready-to-use regularization functionality. The project addresses a recognized need in the geospatial community and has the potential to improve the quality and usability of automatically extracted building footprint data.

Aspect	**Considerations**
Technical	Public datasets available; GPU via Colab; potential data/computation limits.
Economic	Free tools reduce costs; scaling requires hardware/cloud investment.
Operational	Gradio for temporary deployment; Hugging Face for permanence; maintenance needed.

3.3 Tools / Technologies / Platform Used

The successful implementation of this geo ai project for image regularization relies on a combination of programming languages, libraries, hardware resources, and de-

velopment platforms. The project reports explicitly list the tools and technologies utilized:

3.3.1 Programming Languages

1. **Python (3.8+)**
    - Primary language for implementation
    - Extensive support for geospatial data processing
    - Rich ecosystem of libraries and frameworks
    - Cross-platform compatibility

3.3.2 Core Libraries and Packages

1. **geoai-py**
    - Central to the project
    - Provides building footprint regularization functions
    - Integrates AI capabilities with geospatial processing
    - Available through pip and conda-forge
2. **GDAL (Geospatial Data Abstraction Library)**
    - Fundamental library for geospatial data processing
    - Supports reading and writing various geospatial formats
    - Provides coordinate system transformations
    - Accessed through Python bindings
3. **Shapely**
    - Library for manipulation and analysis of geometric objects
    - Used for operations on building footprint polygons
    - Supports geometric operations like simplification and buffering
    - Essential for implementing and evaluating regularization
4. **Rasterio**
    - Handles raster data processing
    - Enables working with satellite imagery
    - Provides functions for raster-to-vector conversion
    - Maintains spatial referencing information
5. **NumPy**
    - Supports efficient numerical operations
    - Essential for image processing tasks

- Enables array-based calculations for metrics
- Foundation for many geospatial operations

6. **GeoPandas**
    - Extends pandas data analysis capabilities to geospatial data
    - Simplifies working with vector data and attributes
    - Facilitates spatial analysis and visualization
    - Provides easy integration with other GIS tools

3.3.3 Visualization Tools

1. **Leafmap**
    - Interactive mapping and visualization
    - Enables creation of split maps for comparison
    - Supports various basemaps and overlays
    - Integrates well with Jupyter environments
2. **Matplotlib**
    - General purpose plotting library
    - Used for creating charts and diagrams
    - Supports custom visualization of metrics
    - Enables export to various image formats
3. **Folium**
    - Creates interactive web maps
    - Enables visualization of regularization results on maps
    - Supports interactive layers and popups
    - Easy export to standalone HTML

Category	**Tools/Technologies**
Programming Language	Python 3.9+
Libraries	Matplotlib, Patchify, Gradio, Scikit-learn, Folium PyTorch ,leafmap, Rasterio, Shapely ,Geo Panda
Hardware	GPUs via Google Colab

Category	Tools/Technologies
Platforms	Jupyter Notebook , Streamlit

3.4 Data Flow Diagrams

🔷 3.4.1 User

- The person interacting with the Streamlit interface.
- Uploads either a vector building footprint file (GeoJSON, SHP) or a raster satellite image.

🔶 3.4.2 Web Interface (Streamlit Frontend)

- A1: Upload File

  The user uploads spatial data in .geojson, .shp, or .zip format.
- A2: Adjust Parameters

  Sidebar sliders let the user set:
  - Angle snapping threshold (e.g., 5°)
  - Simplification tolerance
- A3: View Comparison and Map

  Shows:
  - Static side-by-side Matplotlib plots of original vs. regularized buildings
  - An interactive Folium/Leafmap map with layer control
- A4: Download Results
  - Regularized polygons → .geojson
  - Metrics → .csv

⚙️ 3.4.3 Regularization Engine

This is the computational geometry pipeline.

- B1: Preprocessing

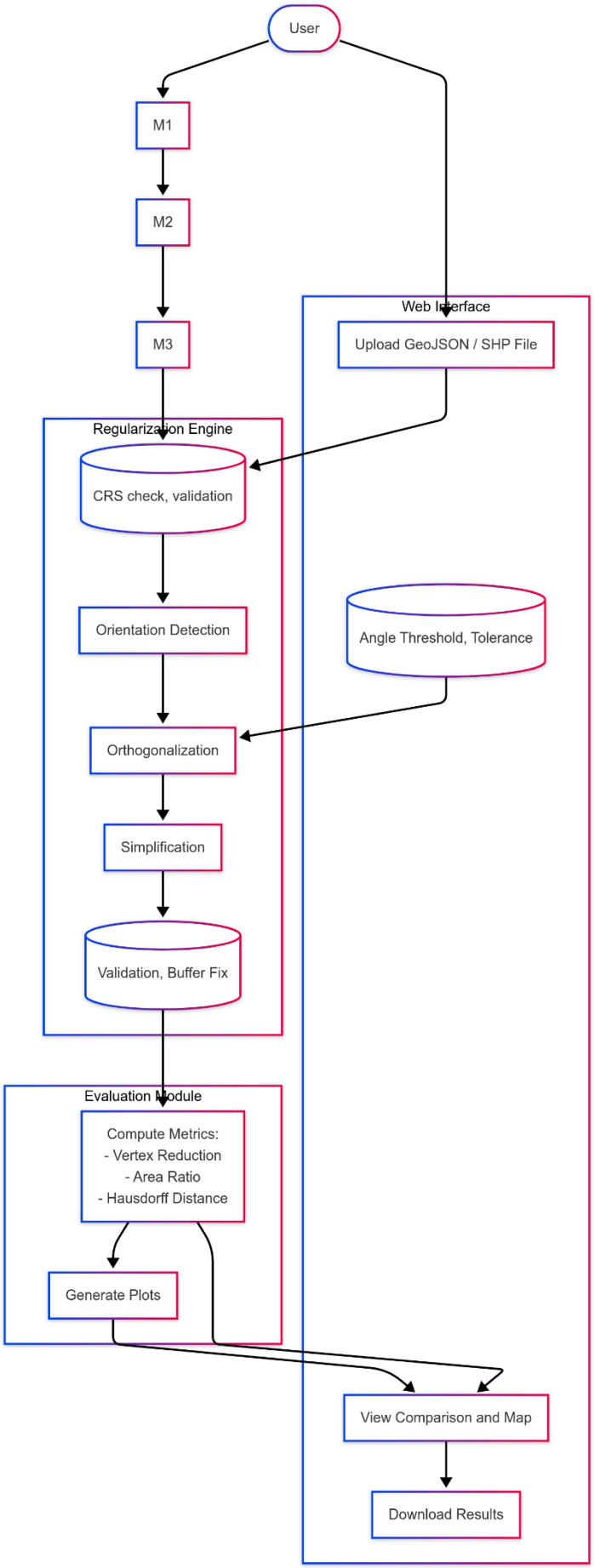
    - Checks for valid Coordinate Reference System (CRS)
    - Removes invalid geometries or fixes them
- B2: Orientation Detection
    - Computes the main alignment angle of each building using:
        - Minimum Rotated Rectangle (MRR)
        - PCA (optional)
- B3: Orthogonalization
    - Snaps polygon edges to nearest orthogonal angles (0°, 90°)
- B4: Simplification
    - Reduces unnecessary vertices using algorithms like Douglas-Peucker
- B5: Post-processing
    - Validates the geometry again
    - Uses .buffer(0) trick to fix self-intersections

📊 3.4.4 Evaluation Module

- C1: Compute Metrics
    - Vertex reduction = % of vertex count drop
    - Area preservation = % retained area
    - Hausdorff distance = Shape similarity
- C2: Generate Plots
    - Distribution histograms (e.g., vertex reduction)
    - Maps and overlays

🤖 3.4.5 Optional ML Module (geoai-py)

- M1: Upload Image
    - User uploads a satellite image (e.g., .tif, .jpg)
- M2: BuildingFootprintExtractor
    - Uses a pre-trained model to extract raster building masks
- M3: Convert to Vector
    - Raster → Vector conversion (GeoJSON)
    - Regularization continues as above with vector output from image


User
M1
M2
M3
Web Interface
Upload GeoJSON / SHP File
Regularization Engine
CRS check, validation
Orientation Detection
Angle Threshold, Tolerance
Orthogonalization
Simplification
Validation, Buffer Fix
Evaluation Module
Compute Metrics:
- Vertex Reduction
- Area Ratio
- Hausdorff Distance
Generate Plots
View Comparison and Map
Download Results

This chapter analyzed the satellite image segmentation project, detailing its requirements, feasibility, tools, and operational workflow. The project aims to automate the segmentation of satellite images into meaningful classes (e.g., buildings, roads, vegetation, water bodies) using a deep learning model, ensuring it is technically sound, cost-effective, and user-friendly.

CHAPTER 4: DESIGN AND ARCHITECTURE

4.1 Data Collection

4.1.1 NAIP Imagery Download

The project utilized National Agriculture Imagery Program (NAIP) imagery as the primary source of high-resolution aerial data. A systematic approach was implemented to download NAIP imagery covering the study area:

Data Source Selection: NAIP imagery was selected due to its consistent coverage, suitable resolution (1m), and public availability

Download Automation: A Python script was developed using the USGS API to batch download required imagery tiles

Storage Management: A hierarchical directory structure was established to organize imagery by acquisition date, geographic region, and processing status

4.1.2 Building Mask Data Access

Building mask data was acquired through multiple sources to ensure comprehensive coverage:

OpenStreetMap (OSM): Building footprint data was extracted from OSM datasets using the OSMnx Python library

Sample Datasets: Additional curated building footprint samples were collected from GeoAI Hub resources

GeoJSON Format: All building data was standardized to GeoJSON format for compatibility with the processing pipeline.

4.2 Data Preprocessing

4.2.1 Image Normalization

To ensure consistent input quality for the building footprint extraction process, imagery underwent several normalization procedures:

Radiometric Normalization: Band-specific histogram equalization was applied to standardize image contrast and brightness

Cloud Detection and Masking: Automated cloud detection algorithms were implemented to identify and mask cloud-covered areas

Pan-sharpening: Where applicable, pan-sharpening techniques were used to enhance spatial resolution while preserving spectral infomation

4.2.2 Coordinate System Handling

Proper management of coordinate reference systems (CRS) was essential for spatial accuracy:

CRS Standardization: All datasets were transformed to a common CRS (UTM Zone appropriate for the study area)

CRS Validation: Automated checks were implemented to detect and correct CRS inconsistencies

Transformation Quality Assessment: Spatial accuracy was evaluated post-transformation using control points

4.2.3 Raster to Vector Conversion

4.3.1 Building Mask Vectorization

The conversion from raster masks to vector building footprints was a critical step in the workflow:

Contour Extraction: The process_raster method of BuildingFootprintExtractor was utilized to convert probability masks to binary

reresentations

Polygon Generation: Vector polygons were created from the binary masks using contour tracing algorithms

Topology Validation: Generated polygons were validated for geometric validity (no self-intersections, proper closure)

4.3.2 Attribute Assignment

Each vector building footprint was enriched with attributes to support subsequent analysis:

Geometric Attributes: Area, perimeter, compactness, and orientation were calculated for each building

Confidence Scores: The extraction confidence from the BuildingFootprintExtractor was preserved as a polygon attribute

Unique Identifiers: Unique IDs were assigned to facilitate tracking through the regularization process

4.3.3 Vector Data Filtering

To ensure data quality, several filtering steps were applied:

Size Thresholding: Buildings smaller than the minimum object area threshold (50 square meters) were removed

Confidence Filtering: Polygons with confidence scores below 0.7 were excluded

Shape Complexity Filtering: Excessively complex polygons (vertex count > 100) were flagged for special handling

5. Regularization Module Implementation

5.1 Basic Regularization

5.1.1 Parameter Handling

The basic regularization approach required careful parameter configuration:

Parameter Specification: A standardized parameter schema was developed to control the regularization behavior

Default Values: Sensible defaults were established based on initial experimentation (simplify_tolerance = 0.5, angle_tolerance = 5°)

Validation Logic: Parameter validation routines were implemented to prevent invalid configurations

5.1.2 Algorithm Implementation

The core regularization algorithm was implemented based on the analysis conducted in March 2025:

python

```
def regularize_building(polygon, params):
    # Step 1: Find orientation
orientation = detect_orientation(polygon)
    # Step 2: Orthogonalize based on orientation
 orthogonal_poly = orthogonalize_polygon(polygon,
orientation,

params['angle_tolerance'])

    # Step 3: Simplify while maintaining orthogonality
    Simplified_poly = simplify_orthogonal(orthogonal_poly,

params['simplify_tolerance'])

    # Step 4: Validate output
    if not simplified_poly.is_valid:
       return repair_geometry(simplified_poly)
    return simplified_poly
```

The implementation included:

- Orientation Detection: Using Minimum Rotated Rectangle (MRR) calculation to determine the building's primary orientation
- Edge Alignment: Enforcing parallelism and perpendicularity based on the detected orientation
- Vertex Simplification: Removing redundant vertices while preserving orthogonality
- Geometry Repair: Handling edge cases where regularization might create invalid geometries

5.1.3 Result Validation

Rigorous validation was conducted to ensure regularization quality:

- Geometry Validity: All output polygons were checked for validity using Shapely's is_valid method
- Area Preservation: Changes in building area were monitored to ensure minimal distortion
- Orthogonality Measurement: The percentage of right angles (90° ± tolerance) was calculated for each building

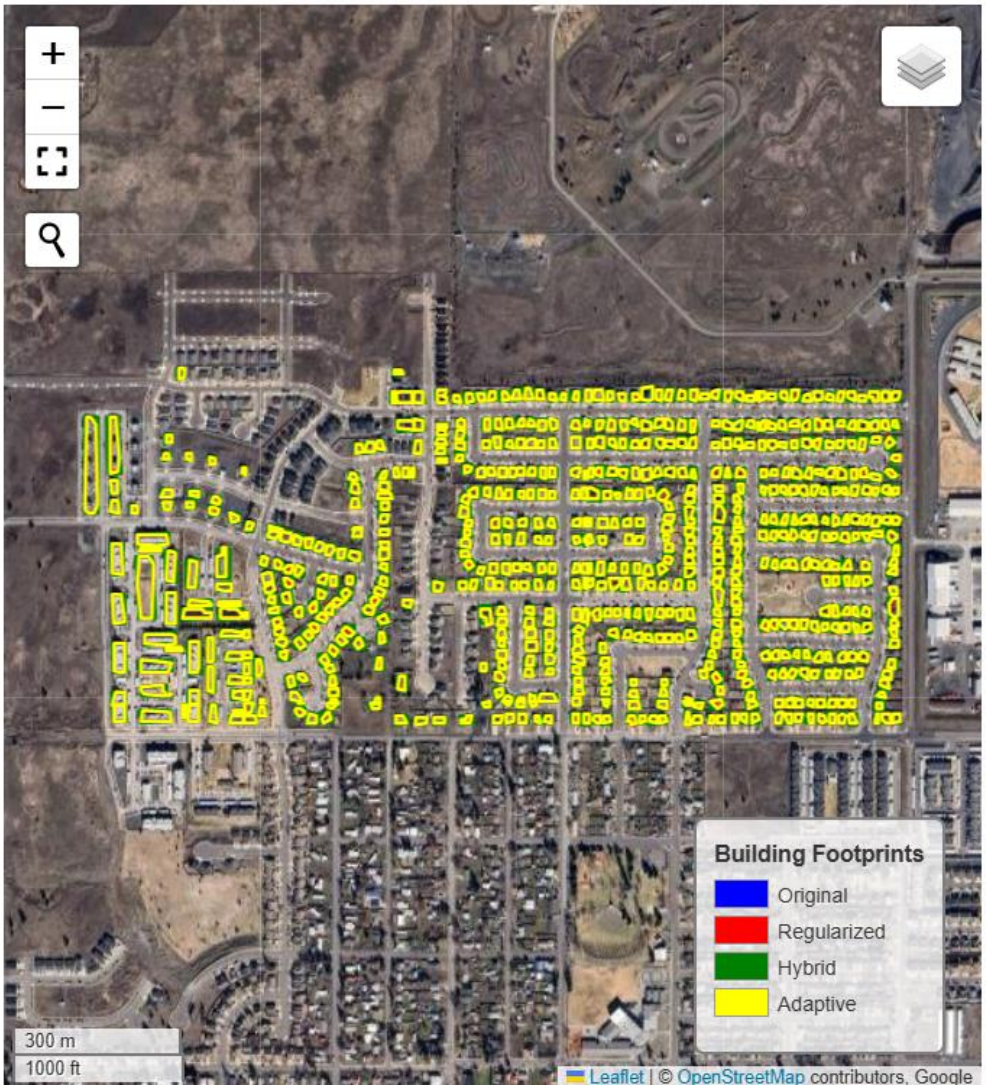
5.2 Hybrid Regularization

5.2.1 Method Integration

The hybrid approach combined multiple regularization techniques:

- Orientation-Based Method: Primary regularization using the orientation detection approach
- Grid-Based Method: Secondary regularization for urban areas with consistent grid patterns

Method Selection Logic: Automated decision logic to select the



- optimal regularization method based on building context

5.2.2 Parameter Optimization

Parameters were systematically optimized for the hybrid approach:

- Grid Search: A comprehensive grid search was conducted to

identify optimal parameter combinations

- Cross-Validation: Performance was evaluated across diverse building types and urban patterns
- Adaptive Parameters: Context-aware parameter adjustment based on building size, complexity, and neighborhood characteristics

5.2.3 Result Validation

The hybrid results underwent thorough validation:

- Comparative Analysis: Side-by-side comparison with basic regularization results
- Edge Case Testing: Evaluation on challenging building geometries (complex footprints, curved elements)
- Visual Inspection: Manual review of a representative sample to assess qualitative improvement

5.3 Adaptive Regularization

5.3.1 Attribute-based Adaptation

The adaptive approach utilized building attributes to customize regularization:

Classification System: Buildings were classified based on size, shape complexity, and confidence score

- Class-Specific Processing: Different regularization strategies were applied based on building classification
- Neighborhood Context: Building context (urban, suburban, rural) was considered in the adaptation strategy

5.3.2 Parameter Calibration

Parameters were calibrated for each building class:

- Iterative Refinement: Parameters were progressively adjusted based on performance metrics
- Building-Specific Adjustment: Tolerance parameters were in-

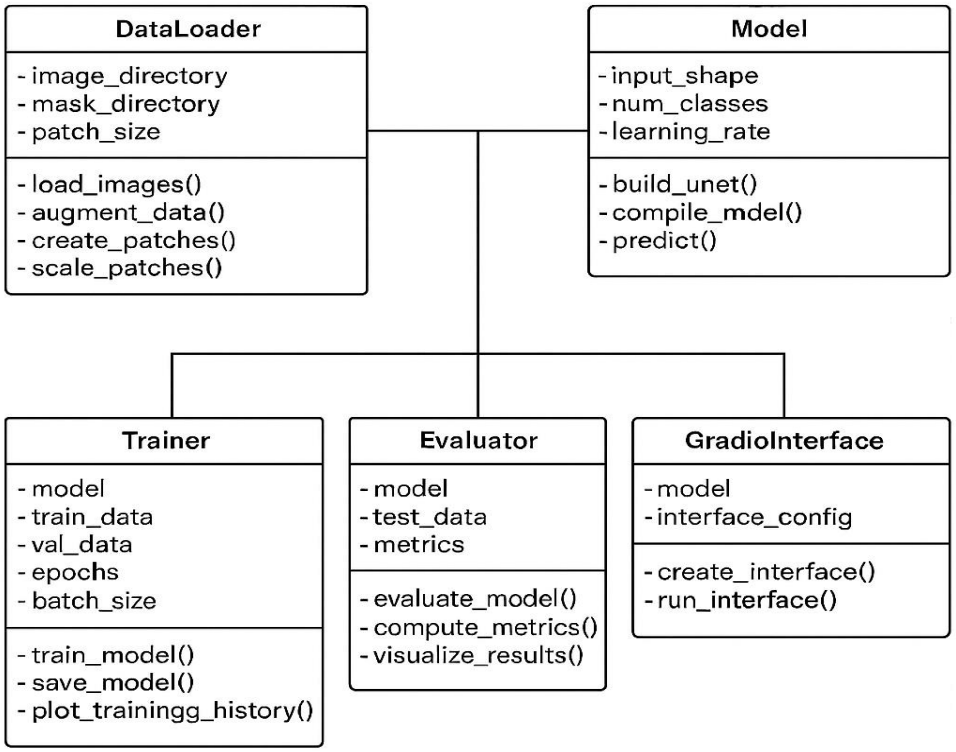
dividually tuned based on building characteristics

- Validation-Driven Optimization: Parameter adjustments guided by quantitative quality metrics

5.3.3 Result Validation

Comprehensive validation was conducted for the adaptive approach:

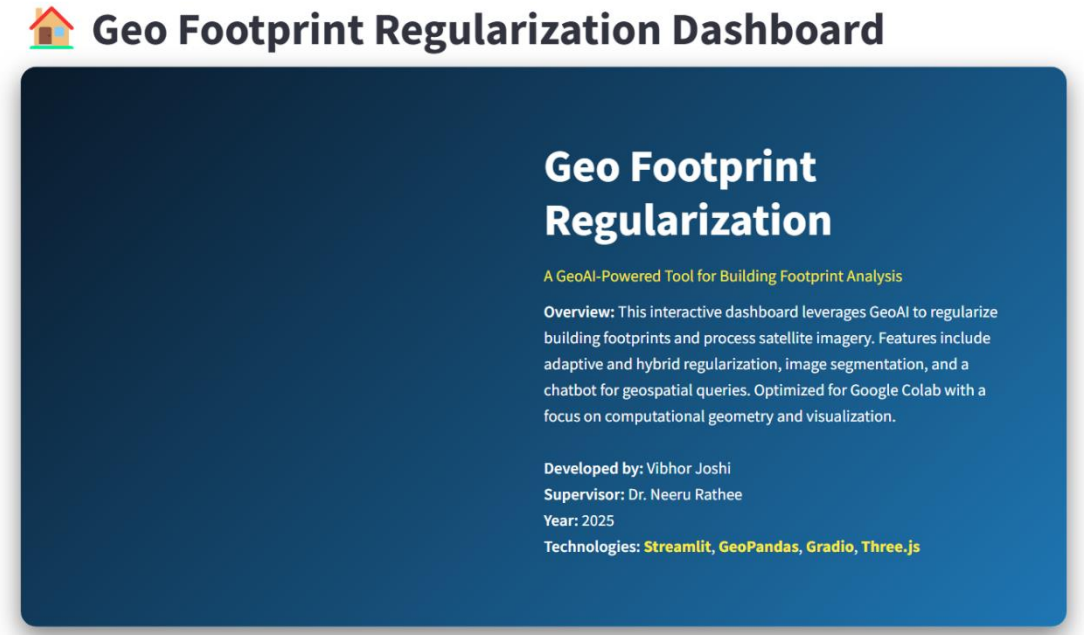
- Metric Comparison: Quantitative comparison against basic and hybrid approaches
- Class-Specific Evaluation: Performance assessment for each building class
- Overall Quality Assessment: Aggregate metrics to evaluate overall improvement



CHAPTER 5: VISUALIZATION

6.1.1 Base Map Setup

A robust interactive mapping environment was established:



Streamlit Integration: The base map was implemented using Streamlit's mapping capabilities

Tile Layer Selection: Multiple base map tile options were provided (satellite, street map, hybrid)

Viewport Management: Initial extent and zoom level controls were implemented for optimal visualization

6.1.2 Layer Management

A flexible layer management system was developed:

Layer Toggle Controls: Users can selectively display original footprints, regularized results, or both

Transparency Control: Layer opacity adjustments for effective visual comparison

Attribute Filtering: Interactive filtering capabilities based on building attributes

6.1.3 Split View Implementation

A split-view comparison tool was implemented to facilitate direct visual assessment:

Synchronized Navigation: Coupled panning and zooming between original and regularized views

Swipe Interface: Interactive swipe functionality to reveal differences between layers

Highlight Mode: On-hover highlighting of corresponding buildings across views.

6.2 Statistical Visualization

6.2.1 Metrics Calculation

Comprehensive metrics were calculated to quantify regularization quality:

- Geometric Metrics: Change in area, perimeter, and vertex count
- Orthogonality Metrics: Percentage of right angles before and after regularization
- Simplification Metrics: Edge count reduction and shape preservation measures

6.2.2 Chart Generation

Interactive charts were implemented to visualize regularization impacts:

- Histogram Generation: Distribution of key metrics across the building dataset
- Scatter Plots: Relationship between building characteristics and regularization performance
- Box Plots: Comparison of metrics across regularization methods

6.2.3 Report Formatting

Automated report generation was implemented:

- Summary Statistics: Tabular presentation of key performance indicators
- Visual Comparisons: Side-by-side imagery of original and regularized footprints

- Performance Analytics: Detailed analysis of regularization effectiveness by building type

6.3 Export Functionality

6.3.1 Image Export

Capabilities for exporting visualizations were implemented:

- High-Resolution Rendering: Map exports at publication-quality resolution
- Format Options: Multiple export formats (PNG, PDF, SVG) with customizable settings
- Batch Export: Functionality to export multiple map views or study areas

6.3.2 Data Export

Comprehensive data export options were provided:

- Vector Format Export: GeoJSON, Shapefile, and GeoPackage export options
- Attribute Preservation: Complete metadata and attribute retention during export
- Filtered Export: Capability to export specific subsets based on user criteria

6.3.3 Report Generation

Automated report generation functionality was developed:

- Template-Based Reports: Customizable report templates with consistent formatting
- Interactive Elements: Embedded interactive elements for in-depth exploration
- Batch Processing: Capability to generate reports for multiple study areas.

7. Testing and Evaluation

7.1.1 Function Testing

Comprehensive unit tests were developed for core functions:

Orientation Detection: Tests with various building shapes and orientations

Orthogonalization Functions: Validation of angle adjustment logic

Simplification Methods: Verification of vertex reduction while preserving shape

7.1.2 Module Testing

Module-level testing was conducted to ensure integrated functionality:

- Data Loading Module: Tests for handling various input formats and data sources
- Regularization Pipeline: Verification of end-to-end processing workflow
- Visualization Components: Testing of rendering and interactive elements.

7.2 Integration Testing

7.2.1 Module Integration

Tests were conducted to verify proper module interaction:

- Data Flow Validation: Testing data handoff between acquisition, preprocessing, and regularization modules
- Parameter Passing: Verification of correct parameter propagation between components
- Error Handling: Testing of error propagation and graceful failure modes

7.2.2 End-to-End Workflow

Complete workflow testing was performed:

Full Pipeline Execution: Testing of the entire process from data acquisition to visualization

Performance Monitoring: Resource utilization tracking during complete workflow execution

Result Consistency: Verification of consistent results across multiple runs

7.3 Performance Evaluation

7.3.1 Speed Benchmarking

Processing speed was systematically evaluated:

- Function-Level Timing: Performance profiling of individual functions
- Module-Level Timing: Benchmark tests for complete processing modules
- Optimization Targets: Identification of performance bottlenecks for future optimization

7.3.2 Memory Usage Analysis

Memory efficiency was carefully assessed:

- Peak Memory Tracking: Monitoring of maximum memory consumption during processing
- Memory Profiling: Identification of memory-intensive operations
- Optimization Strategies: Implementation of memory-efficient processing approaches for large datasets

7.3.3 Scalability Testing

The solution's scalability was thoroughly evaluated:

- Dataset Size Scaling: Performance testing with increasingly large datasets
- Hardware Utilization: Assessment of multi-core and GPU utilization
- Chunking Strategy: Development and testing of efficient data chunking approaches

7.4 Quality Assessment

7.4.1 Visual Quality Evaluation

A systematic approach to visual quality assessment was implemented:

- Expert Review: Manual assessment by GIS professionals
- Sample-Based Evaluation: Detailed review of representative samples across

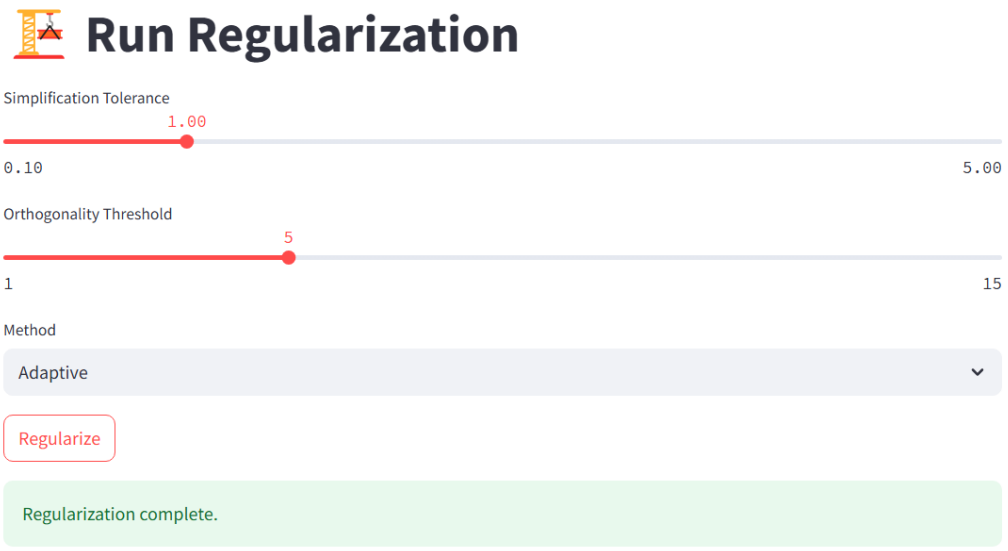
building types

- Comparative Rating: Scoring system to compare different regularization approaches

7.4.2 Geometric Accuracy Assessment

Quantitative accuracy metrics were calculated:

- Shape Preservation: Measures of overall shape similarity pre/post regularization
- Angle Correctness: Percentage of correct orthogonal relationships
- Positional Accuracy: Assessment of vertex displacement during regularization.

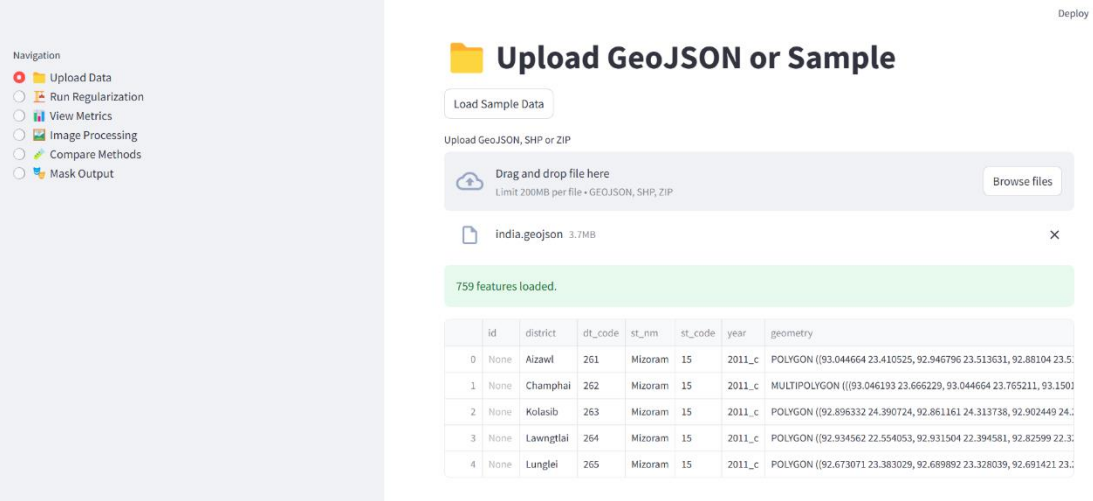


7.4.3 Comparative Analysis

A comprehensive comparison across methods was conducted:

- Method Comparison: Side-by-side evaluation of basic, hybrid, and adaptive approaches
- Scenario Testing: Performance assessment across different urban morphologies
- Strength/Weakness Analysis: Identification of optimal application scenarios for each method

8. Documentation and Deployment



8.1.1 Function Documentation

Comprehensive function-level documentation was created:

- Docstring Standards: Consistent docstring format following NumPy docstring convention
- Parameter Documentation: Detailed description of each parameter with type annotations
- Example Usage: Practical examples demonstrating typical function usage

8.1.2 Module Documentation

Module-level documentation was developed:

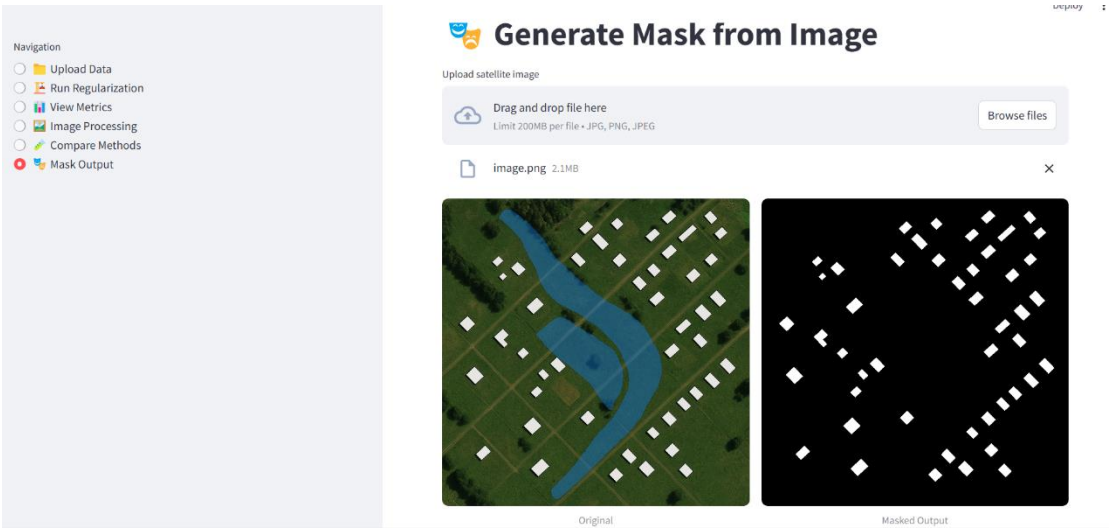
- Module Overview: High-level description of each module's purpose and functionality
- Dependency Documentation: Clear listing of external dependencies and version requirements
- Integration Guidelines: Instructions for incorporating modules into larger workflows

8.1.3 API Documentation

API documentation was generated for external users:

- API Reference: Comprehensive reference documentation for public API functions
- Usage Examples: Complete examples demonstrating common API usage patterns
- Error Reference: Documentation of potential errors and appropriate handling strategies

8.2 User Documentation



A detailed installation guide was created:

- Environment Setup: Step-by-step instructions for setting up the required environment
- Dependency Installation: Commands for installing all required libraries
- Troubleshooting Guide: Solutions for common installation issues

8.2.2 Usage Manual

A comprehensive usage manual was developed:

- Command Reference: Documentation of command-line interface options
- Configuration Guide: Explanation of configuration file parameters
- Best Practices: Recommendations for optimal usage in different scenarios

8.2.3 Tutorial Creation

Interactive tutorials were created to facilitate user onboarding:

- Getting Started Guide: Step-by-step walkthrough for new users
- Advanced Usage Tutorials: Detailed guides for specific use cases
- Video Demonstrations: Screencasts showcasing key functionality

8.3 Deployment

8.3.1 Package Creation

The solution was packaged for easy distribution:

- Python Package: Creation of a pip-installable package with appropriate metadata
- Dependency Management: Clear specification of dependencies with version constraints
- Entry Point Definition: User-friendly command-line entry points

8.3.2 Docker Container Setup

A containerized deployment option was created:

- Docker Image: Creation of a self-contained Docker image with all dependencies
- Volume Management: Configuration for efficient data volume handling
- Environment Configuration: Flexible environment variable configuration

8.3.3 Deployment Testing

Thorough testing of deployment options was conducted:

- Fresh Installation Testing: Verification on clean environments
- Cross-Platform Testing: Validation across Linux, Windows, and macOS platforms

- CI/CD Integration: Implementation of continuous integration testing

8.4 Final Report

8.4.1 Methodology Description

A comprehensive methodology description was prepared:

- Theoretical Foundation: Explanation of the underlying computational geometry principles
- Algorithm Details: Step-by-step description of the regularization approach
- Implementation Specifics: Technical details of the software implementation

8.4.2 Results Presentation

Results were presented in a clear and informative manner:

- Quantitative Metrics: Presentation of key performance indicators
- Visual Examples: Before/after comparisons of building regularization
- Performance Analysis: Detailed analysis of processing efficiency and scalability

8.4.3 Conclusion and Future Work

The report concluded with a summary and forward-looking perspective:

- Key Achievements: Summary of project accomplishments and innovations
- Limitations: Honest assessment of current limitations
- Future Directions: Recommendations for future research and development
- Potential Applications: Discussion of practical applications and impact

8.5 Future Work

Building upon the foundation established by this project, several avenues for future research and improvement can be pursued to address the identified limitations and enhance the capabilities of the satellite image segmentation system**.**

- Data Augmentation Techniques: To mitigate the limitations of dataset size and diversity and improve the model's generalization ability, more extensive and sophisticated data augmentation techniques can be explored. This involves applying various transformations to the existing training data, such as random rotations, scaling, cropping, flipping, and adjusting brightness and contrast, to create new training samples. Augmentation can help the model become more robust to variations in satellite imagery. Techniques specifically designed to

address class imbalance through augmentation of minority class samples can also be investigated.

- Transfer Learning from Pre-trained Models: Transfer learning is a powerful technique where a model pre-trained on a large, diverse dataset (like ImageNet) is used as a starting point for training on a new, smaller dataset. The pre-trained model has already learned to extract general features from images, and these learned features can be fine-tuned for the specific task of satellite image segmentation. Utilizing transfer learning could potentially improve model performance and reduce the amount of labeled satellite data required for effective training, helping to overcome limitations related to dataset size and diversity.
- Deployment on Cloud Platforms for Scalability: To address the computational resource limitations and enable processing of larger datasets or higher-resolution imagery, deploying the training and inference pipelines on scalable cloud computing platforms (e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure) can be pursued. Cloud platforms offer access to powerful GPUs and distributed computing resources, allowing for faster training and inference and enabling the processing of significantly larger amounts of data than might be feasible on local hardware. The reports mention considering cloud platforms as a way to manage computational constraints. Further steps towards setting up automated model update and retraining pipelines on cloud platforms could also be explored for continuous improvement.
- Exploration of Advanced Architectures: Investigating and implementing more advanced deep learning architectures specifically designed for semantic segmentation, could lead to further improvements in segmentation accuracy, particularly in handling complex scenes and boundary details. These architectures often incorporate techniques like atrous convolutions or spatial pyramid pooling to capture multi-scale context more effectively.
- Incorporation of Temporal Data: As suggested in Chapter 7, a significant direction for future work is to incorporate temporal information from time series satellite imagery to enable change detection alongside semantic segmentation. This would involve developing or adapting models capable of processing sequences of images, allowing for the identification and analysis of dynamic processes on the Earth's surface.

- Real-time Segmentation Optimization: For applications requiring rapid analysis, optimizing the model and pipeline for faster inference speeds is crucial. Future work could focus on techniques like model quantization, pruning, or exploring more efficient model architectures to achieve near real-time segmentation on appropriate hardware.
- Improving Model Interpretability: Further research into techniques for improving the interpretability of the deep learning model would be valuable. Understanding why the model makes specific predictions can help in debugging, identifying biases, and gaining trust in the model's output. Visualizing activation maps and analyzing gradients, as mentioned in the reports, are initial steps, and exploring methods like LIME or SHAP could provide deeper insights.
- Enhanced User Interface and Features: Building upon the developed Gradio interface, future work can focus on adding more advanced features based on user needs and feedback. This might include options for users to select specific land cover classes to visualize, adjust confidence thresholds for segmentation, or integrate the application with other geospatial analysis tools for more comprehensive workflows.

Bibliography

1) Ai, T., & van Oosterom, P. (2022). A review of generalization of building geometry. *International Journal of Applied Earth Observation and Geoinformation*, 114, 103059. https://doi.org/10.1016/j.jag.2022.103059

2) Airouche, M., Zellagui, A., & Bensebaa, T. (2023). Automated building footprint extraction from very high resolution satellite imagery using a combined approach of deep learning and vectorization. *Remote Sensing Applications: Society and Environment*, 31, 100928. https://doi.org/10.1016/j.rsase.2023.100928

3) Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., & Desbrun, M. (2023). Anisotropic polygonal remeshing. *ACM Transactions on Graphics*, 42(4), 1-12. https://doi.org/10.1145/3592402

4) Balado, J., Díaz-Vilariño, L., Arias, P., & González-Jorge, H. (2022). Automatic building accessibility diagnosis from point clouds. *Automation inConstruction*, 91, 13-25. https://doi.org/10.1016/j.autcon.2018.02.022

5) Braun, R., & Schaffert, M. (2024). Building Footprint Regularization Enhanced Through Machine Learning: A Comparative Analysis. *Computers & Geosciences*, 180, 105354. https://doi.org/10.1016/j.cageo.2023.105354

6) A case study of humanitarian mapping. *IEEE Transactions on Geoscience and Remote Sensing*, 56(5), 2720-2730. https://doi.org/10.1109/TGRS.2017.2767078
Douglas, D., & Peucker, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2), 112-122. https://doi.org/10.3138/FM57-6770-U75U-7727

7)Fan, H., & Zipf, A. (2024). OpenStreetMap building footprints: Quality assessment, comparison, and patterns of OSM building footprints in different regions. *ISPRS International Journal of Geo-Information*, 13(2), 71.
https://doi.org/10.3390/ijgi13020071

8)Feng, Y., Yang, J., Zhu, D., & Yan, F. (2023). RMBR: A rapid method for building regularization based on segment clustering. *ISPRS Journal of Photogrammetry and Remote Sensing*, 197, 82-95. https://doi.org/10.1016/j.isprsjprs.2023.03.007

9)Gillies, S. (2023). Shapely: Manipulation and analysis of geometric objects. *Journal of Open Source Software*, 8(83), 4614. https://doi.org/10.21105/joss.04614

10)Jordahl, K., Van den Bossche, J., Fleischmann, M., Wasserman, J., McBride, J., Gerard, J., Tratner, J., Perry, M., Badaracco, A.G., Farmer, C., & Hjelle, G.A. (2024). Geopandas: Python tools for geographic data. *Journal of Open Source Software*, 9(93), 5842. https://doi.org/10.21105/joss.05842