



**COLORADO SCHOOL OF MINES**  
EARTH • ENERGY • ENVIRONMENT

# CSCI 370 Final Report

TerraCity

Nick Cregan  
Shaun Kannady  
Jonah Fallon  
Xavier Adams

Client: Josh Rands

Revised June 13, 2024

CSCI 370 Summer 2024

Prof. Kelly

Revision	Date	Comments
New	05/{13-17}/2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XI. Team Profile
Rev – 2	05/{20-24}/2024	Completed Sections: VI. System Architecture
Rev – 3	05/{27-31}/2024	Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations
Rev - 4	06/{3-7}/2024	Updated Sections: VI. System Architecture VII. Software Test and Quality a. created more tests b. added test results  Completed Sections: IX. Project Completion Status X. Future Work XI. Lessons Learned
Rev - 5	06/{10-14}/2024	Updated All Sections

Table 1: Revision History

## Table of Contents

I. Introduction.....	3
II. Functional Requirements.....	3
III. Non-Functional Requirements.....	3
IV. Risks.....	4
V. Definition of Done.....	4
VI. System Architecture.....	4
VII. Software Test and Quality.....	7
VIII. Project Ethical Considerations.....	16
IX. Project Completion Status.....	17
X. Future Work.....	17
XI. Lessons Learned.....	19
XII. Acknowledgments.....	19
XIII. Team Profile.....	19
Appendix A – Key Terms.....	20

## I. Introduction

The following document details a software application designed to analyze the amount of parking available within a designated geographic area. It outlines the product's purpose, functionalities, architecture, and quality.

The software's core functionality is to calculate the percentage of land dedicated to parking within a defined geographical area. This area is represented by either some arbitrary amount of coordinates that represents a polygon or one pair of coordinates that represents the center of a circle and a radius (in meters) to go along with the circle. The application analyzes the designated area and determines the percentage of land used for on and off street parking.

This project is proposed by Josh Rands, founder and CEO of TerraCity. The application is designed to benefit a variety of stakeholders. Public transportation planners need insight on the amount of parking in an area in order to make informed decisions regarding public transportation implementation and development plans. Additionally, this product may aid the average consumer in making transportation decisions based on available parking near their desired destination.

Data sources are identified during the development process. Following model training, ongoing maintenance is not anticipated. Any modifications following the finished product that requires additional development or maintenance is out of this project's scope.

## II. Functional Requirements

- A. Input: Take in a JSON in GeoJSON ([geojson.org](http://geojson.org)) format.
- B. Process the input into a bounding box that captures all of the area within the circle/polygon
  - a. Should reject input if something is off (see section VII, part I for more detail)
- C. Generate some image using altered input coordinates (includes images being sharpened and dividing an image into multiple subimages to ensure that the images are zoomed in enough for the model to be able to extract features from)
- D. Run the image(s) through the semantic segmentation model that specializes in image classification and run the image(s) through the OSM database
  - a. Model is trained on thousands of images
- E. Output: A percentage of land use dedicated to parking lots. Use a DICE score to measure/test the model. The output is two separate images, including the OSM database output and the model output.

## III. Non-Functional Requirements

The only non-functional requirement we face is that our model must run offline in a relatively efficient manner. Our model must not experience significant slowdown on an average computer system while offline. Code quality, scalability, and readability are also important factors that have been considered. These factors allow our client to integrate our product quickly and make needed changes in the future as necessary.

## IV. Risks

None of us have any familiarity with computer vision models. The risk also applies to all libraries that may come up during development, like OSMnx or satellite image APIs, for example. This could be a major risk if we do not handle this correctly, but there are plenty of resources for the team to utilize in order to get a solid understanding of all of these technologies, so the appropriate amount of research will mitigate this risk.

The machine learning model may utilize a large portion of our processing power. The idea of sending thousands of images through a model while keeping enough pixels for the model to extract features from may require hardware better than what the team has access to. This has a significantly large impact on our group, so we need to inform the client that our model is limited in its capability to identify parking lots due to limited processing power.

Data sourced from the internet can often become outdated, or even stripped of privileges required to operate our product. This risk is somewhat likely, but would have a moderate-major impact on our product. In the effort to deliver a scalable and maintainable product, we need to ensure that our data gathering is not only replaceable if need be, but also our product is modular to ensure compatibility with data sources with other possible methods.

## V. Definition of Done

- A. List of minimal useful feature set
  - a. The program can take in a JSON in GeoJSON format and get image(s) of the specified area.
  - b. The program can identify parking lots “as accurately as possible”.
- B. Describe any tests that client will run before accepting software
  - a. We apply a comprehensive test set manually engineered by our team. This, along with the results, is supplied to the client along with the final product. The client has no plans to run any tests.
  - b. A DICE score will be used to evaluate the accuracy of the model.
- C. Specify how/when product will be delivered
  - a. We have decided that a simple Git repository will be used in order to send the deliverables. This will occur after presentations on 6/14.

## VI. System Architecture

### Overall Architecture

The product utilizes four main streams to generate an output for the user: a user interface, an API interface, a computer vision model, and OSMnx.

Figure 1 demonstrates the high level system architecture that the user will experience when using the product. They will provide a valid input to our API interface (a polygon or a circle formatted as a GeoJSON file). For example, the terminal input `python .\main.py .\example_jsons\example-eval.json` will process the polygon or circle described by the input json `.\example_jsons\example-eval.json` into a bounding box and then run the rest of the program.

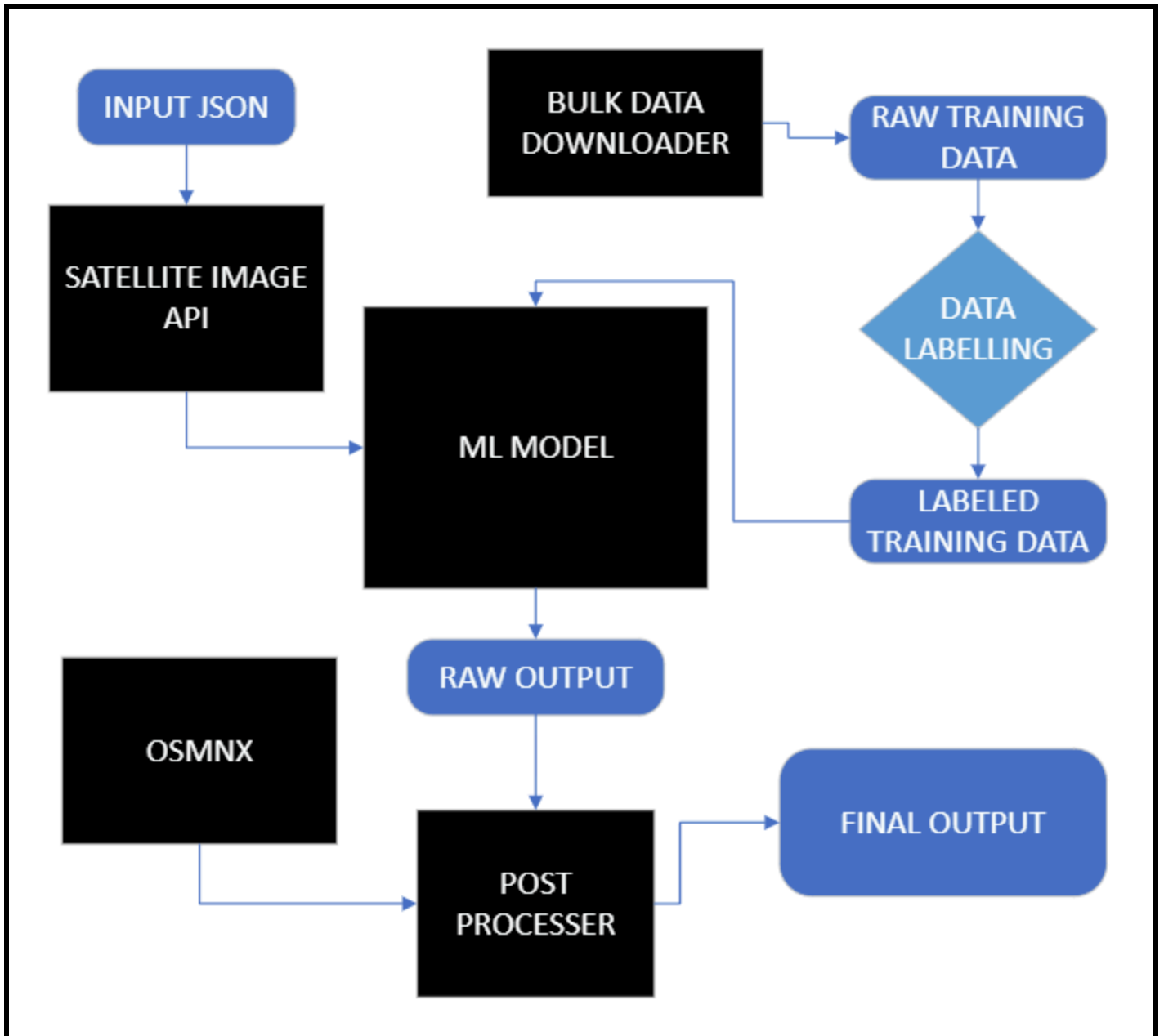


Figure 1: Overall System Architecture

Figures 2 and 3 show an example of what an input JSON may look like.

```
{
  "topic": "parking",
  "message": {
    "messageType": "evaluate",
    "analysisRegion": {
      "type": "Circle",
      "coordinates": [
        -106.00936222076416,
        39.60103199343892
      ],
      "radiusM": 4000
    },
    "modelPath": "./path/to/trained/model",
    "outputPath": "./output/here"
  }
}
```

Figure 2: Input JSON (circle case)

```
{
  "topic": "parking",
  "message": {
    "messageType": "evaluate",
    "analysisRegion": {
      "type": "Polygon",
      "coordinates": [
        [
          -105.22361,
          39.74544
        ],
        [
          -105.21381,
          39.75081
        ],
        [
          -105.22258,
          39.75677
        ],
        [
          -105.22361,
          39.74544
        ]
      ]
    },
    "modelPath": "./path/to/trained/model",
    "outputPath": "./output/here"
  }
}
```

Figure 3: Input JSON (polygon case)

After the polygon/circle is processed into a bounding box, where a satellite image is then obtained based on the bounding box, the corresponding satellite image(s) will then be sharpened and fed to the computer vision model. NOTE: satellite images may need to be split up into multiple subimages in order to ensure that images are zoomed in so that the model and OSMnx will be able to process them effectively. The model will generate an output of designated parking in that area as an image with binary pixel values (white : parking, black : no





box and return an image of binary black and white pixels (white : parking, black : no parking) as another metric for TerraCity or clients of TerraCity to use to understand the full picture of parking availability in an area.

## Post-Processing

The direct results from the model are noisy and often have protrusions. These factors cause our parking predictions to be overstated, and disrupt any visuals made from the outputs. Through some post processing, these images can be cleaned of noise, and rounded to mitigate the protrusions aforementioned. Figure 5 demonstrates this post processing on an example output from the machine learning model.

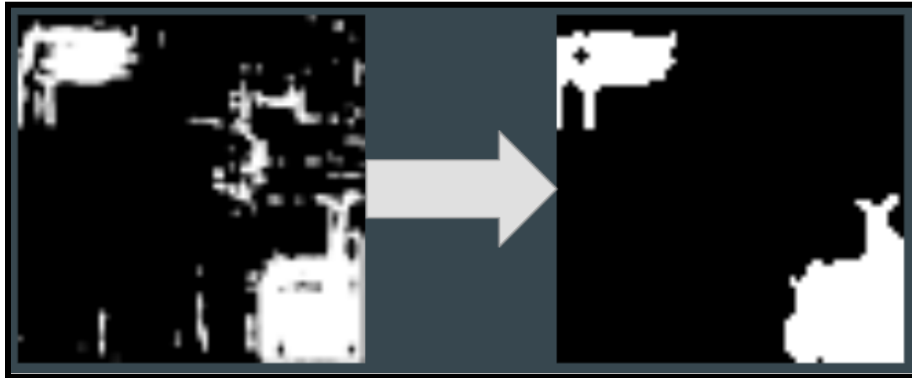


Figure 5: Post Processing Model Output Mask

Additionally, to make the outputs easier to visualize, the post processing can include geometric polygon prediction, where an output mask from our model can be regularized to shapes more commonly seen in parking lots. This allows for easy recognition of identified parking lots to be seen in larger input areas.

Figure 6 demonstrates a mask fully post processed and overlaid on the input image. The three stages of post processing are shown from left to right in the figure, ending with geometric shape regularization (in this case a rectangle).



Figure 6: Geometric Shape Regularization

## VII. Software Test and Quality

**\*\*NOTE: All tests are run in Python 3.12.4 on VS Code. \*\***

### Part I: JSON Input Tests

Test Name	Category	Setup	Action	Expected Result	Actual Result
Invalid JSON file	JSON Parser	Input invalid JSON file	Run python .\main.py .\main.py	Error: Please provide a JSON file (Usage: python3 main.py filename.json)	Error: Please provide a JSON file (Usage: python3 main.py filename.json)
No Input	JSON Parser	Input doesn't include file	python .\main.py	ValueError: Only provide one argument (Usage: python3 main.py filename.json)	ValueError: Only provide one argument (Usage: python3 main.py filename.json)
Multiple Inputs	JSON Parser	Input includes multiple files	python .\main.py .\example_jsons\example-eval-circle.json .\example_jsons\example-eval-circle-too-big.json	ValueError: Only provide one argument (Usage: python3 main.py filename.json)	ValueError: Only provide one argument (Usage: python3 main.py filename.json)
Invalid JSON topic	JSON Parser	Make a JSON file where the topic is not equal to "parking"	python .\main.py .\example_jsons\example-eval-invalid-topic.json	Error: Invalid topic. Expected 'parking', got {topic}.	Error: Invalid topic. Expected 'parking', got {topic}.
No message field	JSON Parser	Make a JSON file with no message	python .\main.py .\example_jsons\example-eval-no-message.json	Error: Invalid message type. Expected 'train' or 'evaluate', got "".	Error: Invalid message type. Expected 'train' or 'evaluate', got "".
Invalid message type	JSON Parser	Make a JSON file with message type not of 'evaluate' or 'train'	python .\main.py .\example_jsons\example-eval-invalid-message-type.json	Error: Invalid message type. Expected 'train' or 'evaluate', got '{message_type}'.	Error: Invalid message type. Expected 'train' or 'evaluate', got '{message_type}'.
No Analysis Region Field	JSON Parser	MAke a JSON file with no analysis region	python .\main.py .\example_jsons\example-eval-no-analysis-region.json	Error: Invalid analysis region type. Expected 'Circle' or 'Polygon', got "".	Error: Invalid analysis region type. Expected 'Circle' or 'Polygon', got "".
Invalid Analysis Region Type	JSON Parser	Make a JSON file with an invalid analysis region type.	python .\main.py .\example_jsons\example-eval-invalid-analysis-region-type.json	Error: Invalid analysis region type. Expected 'Circle' or 'Polygon', got	Error: Invalid analysis region type. Expected 'Circle' or 'Polygon', got

				'{analysis_region_type}'.	'{analysis_region_type}'.
Circle: two pairs of coordinates	JSON Parser	Make a JSON file with the circle center being two pairs of coordinates.	python .\main.py .\example_jsons\example-eval-circle-two-coordinates.json	Error: Invalid center coordinate. Please try again.	Error: Invalid center coordinate. Please try again.
Circle: >2 pairs of coordinates	JSON Parser	Make a JSON file with polygon coordinates as the center.	python .\main.py .\example_jsons\example-eval-circle-polygon-coordinates.json	Error: Invalid center coordinate. Please try again.	Error: Invalid center coordinate. Please try again.
Circle: radius small	JSON Parser	Make a JSON file with a small radius.	python .\main.py .\example_jsons\example-eval-circle-too-small.json	Error: Area of the given coordinates from the JSON is too small ({area} sq. km). The current lower bound is 0.05 sq. km. Revise and try again.	Error: Area of the given coordinates from the JSON is too small ({area} sq. km). The current lower bound is 0.05 sq. km. Revise and try again.
Circle: radius large	JSON Parser	Make a JSON file with a large radius.	python .\main.py .\example_jsons\example-eval-circle-too-big.json	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km. Revise and try again.	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km. Revise and try again.
Circle: invalid longitude	JSON Parser	Make a JSON file with an invalid longitude.	python .\main.py .\example_jsons\example-eval-circle-invalid-latitude.json	Error: Asking for an invalid longitude. Please try again.	Error: Asking for an invalid longitude. Please try again.
Circle: invalid latitude	JSON Parser	Make a JSON with an invalid latitude.	python .\main.py .\example_jsons\example-eval-circle-invalid-longitude.json	Error: Asking for an invalid latitude. Please try again.	Error: Asking for an invalid latitude. Please try again.
Circle Edge Case: 180W/180E crossed	JSON Parser	Make a JSON file with a low/high longitude that	python .\main.py .\example_jsons\example-eval-circle-180WE-crossed.json	Error: Distance value is too large. Please try again.	Error: Distance value is too large. Please try again.

		crosses the 180W/E line.			
Circle Edge Case: North Pole	JSON Parser	Make a JSON file with a radius above 85.06N.	python .\main.py .\example_jsons\example-eval-circle-north-pole.json	Error: Coordinate value {coordinate_value} is out of range (max latitude is 85.06). Please try again.	Error: Coordinate value {coordinate_value} is out of range (max latitude is 85.06). Please try again.
Circle Edge Case: South Pole	JSON Parser	Make a JSON file with radius below 85.06S.	python .\main.py .\example_jsons\example-eval-circle-south-pole.json	Error: Coordinate value {coordinate_value} is out of range (min latitude is -85.06). Please try again.	Error: Coordinate value {coordinate_value} is out of range (min latitude is -85.06). Please try again.
Training Case	JSON Parser	Make a JSON file with message type 'train'.	python .\user_interface\split.py .\example_jsons\example-train.json	TODO: Implement training functionality	TODO: Implement training functionality
Polygon: <3 Coordinate Pairs	JSON Parser	Make a JSON file with less than three coordinate pairs making up the polygon.	python .\main.py .\example_jsons\example-eval-polygon-lt3-coordinate-pairs.json	Error: Got {vertices} vertices, which cannot make a polygon. Fix the JSON and try again.	Error: Got {vertices} vertices, which cannot make a polygon. Fix the JSON and try again.
Polygon: area small	JSON Parser	Make a JSON file with a polygon representing a small area.	python .\main.py .\example_jsons\example-eval-polygon-too-small.json	Error: Area of the given coordinates from the JSON is too small ({area} sq. km). The current lower bound is 0.05 sq. km. Revise and try again.	Error: Area of the given coordinates from the JSON is too small ({area} sq. km). The current lower bound is 0.05 sq. km. Revise and try again.
Polygon: area large	JSON Parser	Make a JSON File with a polygon representing a large area.	python .\main.py .\example_jsons\example-eval-polygon-too-big.json	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km. Revise and	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km.

				try again.	Revise and try again.
Polygon: >=1 invalid latitude	JSON Parser	Make a JSON file with a polygon with at least one invalid latitude value. NOTE: we choose +/- 85.06 to ensure area calculations are accurate later on.	python .\main.py .\example_jsons\example-eval-polygon-invalid-latitude.json	Error: Coordinate value {coordinate_value} is out of range (max min latitude is +/- 85.06). Please try again.	Error: Coordinate value {coordinate_value} is out of range (max min latitude is +/- 85.06). Please try again.
Polygon: >=1 invalid longitude	JSON Parser	Make a JSON file with a polygon with at least one invalid longitude value.	python .\main.py .\example_jsons\example-eval-polygon-invalid-longitude.json	Error: Coordinate value {coordinate_value} is out of range (min max longitude is +/- 180). Please try again.	Error: Coordinate value {coordinate_value} is out of range (min max longitude is +/- 180). Please try again.
Polygon Edge Case: Crossing Antimeridian (180W/E)	JSON Parser	Make a JSON file with a polygon with an area crossing the 180W/E line.	python .\main.py .\example_jsons\example-eval-polygon-180WE-crossed.json	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km. Revise and try again.	Error: Area of the given coordinates from the JSON is too big ({area} sq. km). The current upper bound is 250 sq. km. Revise and try again. NOTE: this case tends to have unpredictable output. Avoid this case at all costs during product use.
Polygon Edge Case: North Pole	JSON Parser	Make a JSON file with a polygon with at least one latitude value in the north	python .\main.py .\example_jsons\example-eval-polygon-north-pole.json	Error: Coordinate value {coordinate_value} is out of range (max latitude is 85.06). Please try	Error: Coordinate value {coordinate_value} is out of range (max latitude is 85.06). Please try

		pole.		again.	again.
Polygon Edge Case: South Pole	JSON Parser	Make a JSON file with a polygon with at least one latitude value in the south pole.	python .\main.py .\example_jsons\example-eval-polygon-south-pole.json	Error: Coordinate value {coordinate_value} is out of range (min latitude is -85.06). Please try again.	Error: Coordinate value {coordinate_value} is out of range (min latitude is -85.06). Please try again.

Table 2: Product Testing (JSON Parser)

### Part II: Satellite Image API Tests

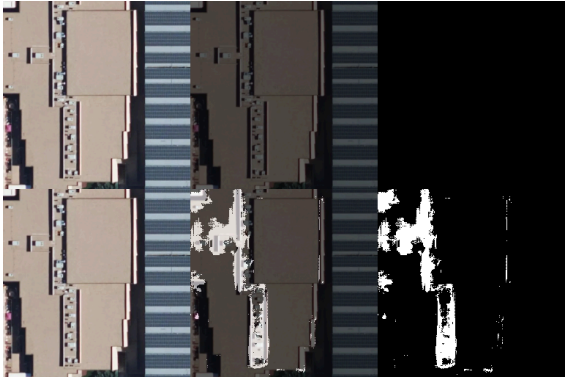

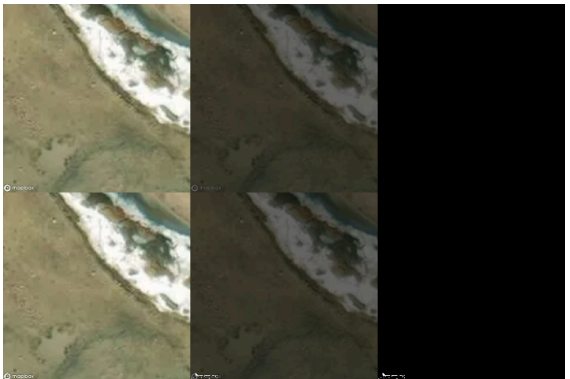
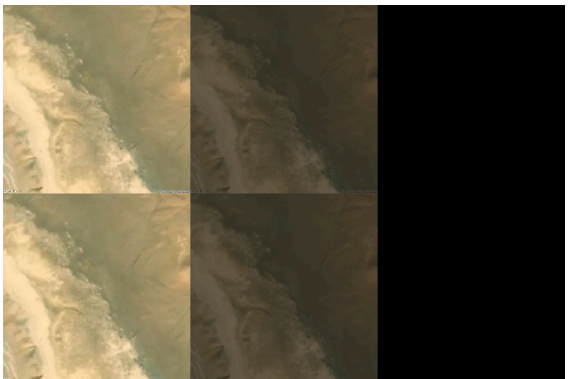
Test Name	Category	Setup	Action	Expected Result	Actual Result
Single Invalid Coordinate	Satellite Image API - Single Image Query	Hardcode invalid coordinate	Run Standalone API request	Error #: "Coordinate lat#, lon# outside of bounds"	Error #: "Coordinate lat#, lon# outside of bounds"
Multiple Invalid Coordinates	Satellite Image API - Single Image Query	Hardcode invalid coordinates	Run Standalone API request	Error #: "Coordinates lat#, lon# etc.. outside of bounds"	Error #: "Coordinates lat#, lon# etc.. outside of bounds"
Logfile Correctness	Satellite Image API - Bulk Data Downloader	Set up a standard image size, 2x2 grid. manually create logfile	Run Bulk Data Downloader and compare to correct output	Accurate Lat/Lon logging	Accurate Lat/Lon logging
Grid Precision	Satellite Image API - Bulk Data Downloader	Set grid size: 3, 4, 5	Run downloader, compare image output count to 9, 16, 25	GridSize^2 images generated	GridSize^2 images generated

Table 3: Product Testing (Satellite Image API)




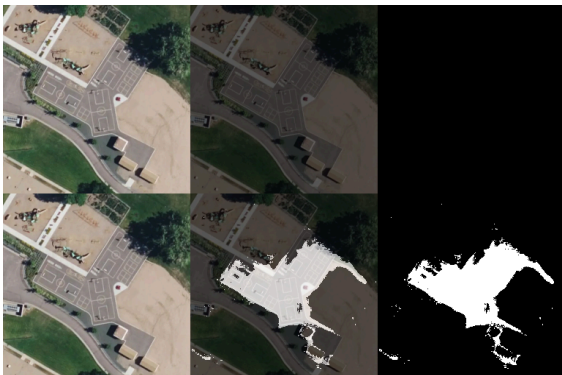
### Part III: Object Detection Tests

Test Name	Category	Setup	Action	Ground Truth (top), Model Output (bottom) NOTE: model outputs are before post processing!
-----------	----------	-------	--------	--

<p>Typical Urban Case (more zoomed out)</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Typical Urban Case II (zoomed out)</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Typical Parking Lot</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Typical Parking Lot II</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	

Zoomed In Building	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	
Body of Water	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	
Mountain Image	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	
Desert Image (N: 21.80890, S: 21.77553, W: -0.10225, E: 0.00090)	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	



<p>Forest Image (N: 0.16926, S: -0.10702 , W: -70.20029, E: -69.87885)</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Typical Residential Area (no parking lots)</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Edge Case: tennis courts</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	
<p>Edge Case: playground with basketball courts/four square</p>	<p>Object Detection</p>	<p>Train model, get model ready for prediction</p>	<p>Send corresponding satellite image through the model</p>	

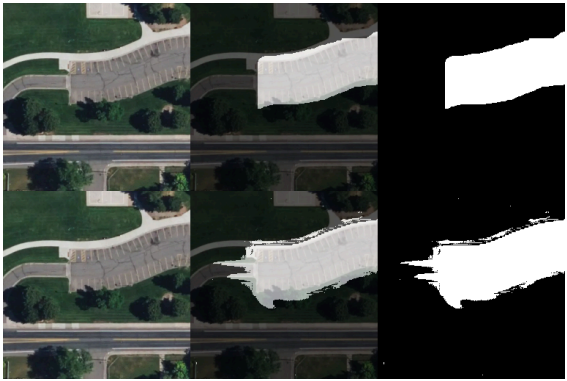
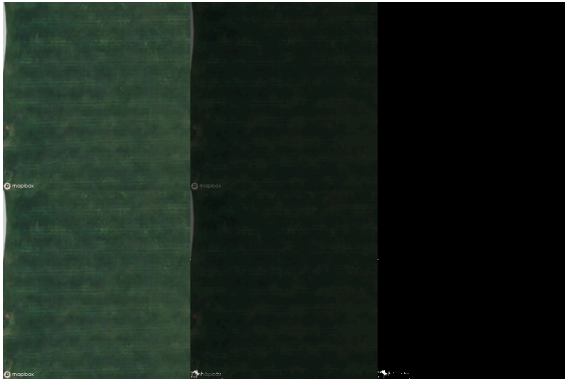
Abnormally Shaped Parking Lot	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	
Grass	Object Detection	Train model, get model ready for prediction	Send corresponding satellite image through the model	

Table 4: Product Testing (Object Detection)

## VIII. Project Ethical Considerations

In our model, we utilize satellite imagery as input data. This type of data can raise privacy concerns such as the potential exposure of addresses.

Continuing with the ethics of satellite imagery, we as a group have little idea about how accurately satellite images fed to the model reflect what the given area looks like in the present. The satellite images that are taken by either the Google Maps API or the Mapbox API are certainly not live images, so these images will not reflect any changes to the landscape done between the date when the images were taken and the analysis to be done by TerraCity.

The same idea applies for the OSM database. There are going to be many cases where the information stored in OSM's database does not match up with the information seen in the satellite imagery, so the value of updated satellite imagery cannot be overstated for this project.

Additionally, OSM's database will not contain every parking amenity on the planet, leading to loads of false negatives that the machine learning model is used to mitigate. It is up to municipalities to update the OSM database, so it may be worth informing clients of TerraCity that this is the case if they want maximum accuracy in their analyses.

There will always be ethical considerations with machine learning because a model can be biased due to how it is trained. This bias can be shown in the results from the model while being hard for us to detect. This is difficult because we do not understand how the model is making each of its decisions. This has an impact on users as our model will be involved in making important travel decisions.

Tangentially related, the method in which data is labeled is crucial in how the eventual model performs. The definition of a parking lot is not exactly something that is agreed upon by everyone, from us, the programmers, to those at TerraCity, to the clients of TerraCity. The lack of uniformity in this definition across all parties will lead to a degree of ambiguity when interpreting results and the action(s) taken based on program output may not necessarily be appropriate.

Lastly, there are group members who have beliefs that conflict with the overall project. The belief is that infrastructure based on private vehicle infrastructure is less than ideal given how much space in our cities is dedicated to private vehicles. This is the urbanist, “15 Minute City” view on city planning. With this project, there is a use case that encourages people to use their own private vehicle to get to a place assuming that the model identifies parking there, contradicting the urbanist point of view. The fact that certain use cases of the software may go against the personal ethics of group members will be another hurdle that needs to be overcome for the success of the project.

## IX. Project Completion Status

### Implemented Features

The core functionalities of our project include:

- A. Reading a GeoJSON file containing latitude and longitude bounds in the shape of a polygon or a circle, along with appropriate error checking.
- B. Utilizing an API to access satellite imagery in the given area.
- C. The inclusion of both the results of a trained model and the results of an OSM database query.
  - a. Accurately identifies parking locations and returns a percentage of land that is dedicated to parking.
  - b. Can return a predicted image mask to compare to the original mask and image.

### Unimplemented Features

Unfortunately, due to time constraints within our project schedule, we were unable to incorporate a significant feature: the capability for our model to identify on-street parking or other edge cases of parking like underground parking or parking underneath a roof. The OSM output attempts to mitigate this, but the issue will persist nevertheless. Additionally, other functionalities that remain unaddressed include the ability to analyze regions of non-rectangular shapes. We are currently restricted by the API to rectangular regions but there could be ways to get very close to a circular shape using some math and transformations of the images we collect and the API requests we make. Also, other existing algorithms developed by TerraCity could be helpful in future improvements.

## X. Future Work

Currently, our model outputs a mask image that shows the areas it identifies as parking lots along with the percentage of land in the area that is identified as a parking lot. There may be a chance that the current method of returning a percentage is unhelpful for many potential future projects. We believe something that would estimate the number of available parking spaces, including the distinction between used and unused parking spots, would provide more options and potential for future work. This would involve needing to identify what kind of parking lot each lot is. Edge cases such as underground parking and parking garages would need to be identified in order to accurately predict the number of parking spaces.

As of now, our product is hoped to be used by TerraCityAI to implement an algorithm that can suggest to a user how to travel to an inputted destination, among other use cases. The algorithm would analyze parking availability along with public transportation options and prices to complete this task.

Feature	Implementation	Notes
Street/Roof/Underground/Other Parking Recognition	Label Data with inclusion of these types of parking.	This would be a difficult feature to train a model on but would provide more accurate parking availability in many areas. Things like on site data labeling or leverage of OSMnx would help here, but could lead to confusing the model (since these types of parking look different than standard parking lots), leading to the difficulty.
Non Rectangle Shapes as a Feature Space (including polygons with holes within them)	Add software functionality to be able to handle atypical shapes.	Most satellite image APIs do not have functionality to request anything but a box so this would take some thought.
Any Input that includes 180W/180E or the range (85.06N, 90N) or the range (90S, 85.06S) in the feature space	Add software functionality to handle continuous surfaces across these boundaries as opposed to treating them like fences.	The reason we deny the 180W/180E inputs right now is to guard against excessively large radius inputs for the circle input case. If there is a better way to limit the radius field of the circle input, that would help with the implementation. Additionally, area calculations will be inaccurate for the extreme north/south parts of the Earth and we doubt that any analysis of this type will be run up there, so we ignored these cases too.
More data!	To label data, we all used some sort of photo editing software (GIMP, MS Paint, etc.) and drew polygons over the original satellite images (where the parking lots are) and then used a fill tool to fill in the polygons representing parking lots white. From there, we used some sort of magic select/color select/invert tool and then filled the rest of the image in with black pixels, completing the labeled data in this way.	<p>The data that we have included in the model is mostly sourced from Denver, Houston, and Pasadena. We could not think of a way to successfully randomly sample from the US or world landscapes without being oversaturated with images with no parking lots, but we believe that a sample of satellite images that is more representative of the United States or the world would vastly improve the model. Also, there would be benefits to doing on site labeling in order to properly confirm whether or not a piece of land is dedicated to parking or not. The on site labeling would especially help with the case of street parking. See notes for more details on how we labeled data.</p> <p>It is also worth noting that most of our training data is set to very particular zoom levels, so the lack of variance there could also be addressed in the future.</p>
Any input with a	Optimize software to handle	We have the upper bound constraint in place in order

<p>bounding box with an area above 250 sq. km or below 0.05 sq. km.</p>	<p>bounding boxes of smaller/bigger sizes.</p>	<p>to not put excess strain on either the API, the model, or our hardware in general. With techniques like parallel processing/multithreading, we feel that this constraint could either be increased to a bigger area or eliminated entirely. Also, with better hardware, more pixels/more features could be extracted from the satellite images, which would further help in the accuracy of the model.</p> <p>On the other end, a lower bound will help avoid the case of a simple gray square or a feature space with very little features to go on. This edge case could be improved with further training, but is in place for now to ensure maximum accuracy of the result.</p> <p>NOTE: These constraints are arbitrary values and can absolutely be changed accordingly!!!</p>
---	--	---

Table 5: Future Product Features

## XI. Lessons Learned

- A. APIs provide powerful functionality for any project, yet they frequently offer functionalities that may not align perfectly with project requirements and the possessed data. Navigating this gap and transforming available data into a format compatible with the APIs capabilities can pose challenges.
- B. Addressing the various needs of clients often demands a flexible approach. Rather than relying on a singular solution, projects often utilize a blend of tactics to effectively tackle complex requirements. In our situation, we used a machine learning model along with a database of parking lots to verify and add to our results in cases that could not be handled by the model alone, such as street parking.
- C. Machine learning projects in niche areas, such as parking lot identification, often creates a need to manually label data due to the lack of data availability. This is a time-consuming process that can take weeks or months to provide an adequate amount of data for a model to learn and succeed.
- D. Heavy research in data science, an exploding field, has helped lead the way for many others to succeed in their own projects. While our specific goal has very little research done, we can apply methods and techniques from other image classification projects to help fine-tune our model. There are often similarities between each of these projects that we can use to our advantage.

## XII. Acknowledgments

- A. We would like to thank our client, TerraCityAI, for their support and trust with this project. Specifically, a big thank you to Josh Rands for helping the project go smoothly.
- B. Thank you to our advisor, Kathleen Kelly, for advising us and ensuring our project requirements were met and team dynamics/team morale were high.

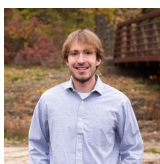
## XIII. Team Profile



Nick Cregan  
Computer Science - Business  
Senior  
Colorado Springs, CO

As a graduating Senior, Nick is excited to join the industry after the summation of this project. In his free time, he enjoys skiing, camping, and riding dirtbikes.

Advisor point of contact. Nick was responsible for most of the data gathering and contributed to key functionalities within the code base in addition to other tasks related to CSCI370 in general.



Jonah Fallon  
Computer Science - Data Science  
Junior  
Batavia, IL

Jonah is a rising Junior who is interested in the Data Science field. He runs track for Mines, competing in the 400m. He loves to spend time outside playing spikeball, volleyball, and pickleball.



Shaun Kannady  
Computer Science - Data Science  
Senior  
Aurora, CO

Shaun has grown up throughout the general Denver Metro Area for as long as he can remember. After deciding to attend Colorado School of Mines, he decided to major in Computer Science and specialize in Data Science, looking forward to what the industry has to offer. During his free time, Shaun enjoys living with his family, bowling, and playing video games.

Shaun was responsible for everything OSM/OSMnx related to the project. Additionally, he helped with the GeoJSON input processing and the overall tasks for the CSCI370 class in general.



Xavier Adams  
Computer Science - Data Science  
Senior  
Arvada, CO

Xavier is a rising senior double majoring in Computer Science + Data Science and Engineering Physics. In his free time he enjoys playing video games and hanging out with friends.

Jonah was responsible for keeping communication with the client, creating many key helper functions within the model involving preprocessing and querying, as well as contributing heavily to other CSC1370 assignments and tasks.

Xavier was responsible for much of the construction and training of the ML model used in this project, and contributed to many CSC1370 class assignments and tasks.

## Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
<i>API</i>	<i>Stands for Application Programming interface. Provides a way for various software applications to communicate.</i>
<i>Semantic Segmentation</i>	<i>Deep learning model technique that classifies each pixel to a category. ie Parking Lot or Not Parking Lot.</i>
<i>JSON</i>	<i>JavaScript Object Notation. This is a text based file format with attribute-value pairs.</i>
<i>OSM</i>	<i>OpenStreetMaps. An open license database that stores information about the world.</i>
<i>OSMnx</i>	<i>A Python library that combines OSM, networkx, and GeoPandas into one, neat library.</i>
<i>DICE Score</i>	<p><i>A measure for defining the accuracy of an object detection machine learning model. Defined by the below equation:</i></p> $\text{Dice} = \frac{2 \times \text{Area of overlap}}{\text{Total area}} = \frac{2 \times \text{Prediction} \cap \text{Ground truth}}{\text{Prediction} \cup \text{Ground truth}}$

Table 6: Appendix