Mapping Locations in Texts

Supervisor - Prof Christopher Jones

Ce Guo | Aug 2021

Abstract

Location information is one of the essential information in natural language text, extracting it from the text and mapping it to a visual geographic information system has become a concern in natural language processing. However, due to the complexity of natural language, not all location information appears in detailed descriptions, and many locations are described using relative spatial relationships. The goal of this project is to address these difficulties.

By using effective named entity recognition methods to extract location information in natural language texts and marking them on a map in a visual geographic information system. We develop a web application to form a solution to the research problem eventually.

Content

Mapping Locations in Texts 1
Abstract 2
Content 3
1. Introduction 4
2. Background 5
2.1 Project Challenge 5
2.2 Solution Status 5
2.3 Introduction to Named Entity Recognition 6
2.4 Evaluative Method 7
2.5 Introduction to Geographic Information System 8
3. Methodology 9
3.1 Software Tool Selection 9
3.2 Data Processing 12
3.3 NER 12
3.4 Geocoding 14
3.5 Geographic Information Visualization 15
3.6 Web Application 15
4. Implementation 17
4.1 Data Processing 17
4.2 NER 22
4.3 GIS 32
4.4 Web Application 43
5. Results And Evaluation 48
5.1 NER 48
5.2 Data Layer 50
5.3 Web Application 52
6. Discussion 53
6.1 Reflection 53
6.2 Future Work 55
7. Conclusion 57
Acknowledgment 58
Reference 59

1. Introduction

With the development of the information age, a significant quantity of natural language texts are accumulated every day, such as people's daily chat conversations, text messages on websites, and processes in certain office systems.

We focus on natural language texts connected to geographic location data in this project, which are captured and kept in a variety of databases and files. However, extracting and parsing them properly is difficult. We generally cannot locate its exact position in a phrase of geographic information description due to the peculiarity of natural language in geographic location, and we can only capture it through relative spatial connections. "The British Museum" for example, can be described as "1-mile north of Waterloo". In certain cases, a new moniker was used to substitute a locality. "New York" for example, is referred to as "Big Apple".

When users record precise location information, a lot of textual effort might become tedious. However, as compared to word, images have a more intuitive visual impact on humans, and an image human-computer interface system can even offer individuals joy. This textual task will become vivid and efficient if the position in the text information can be translated into coordinates and shown on the map, and the user may interact with them.

As a result, the chance for this endeavour arose. The aim is to create a web application that identifies the location of text in geographic description information using an appropriate named entity recognition approach and then parses the actual coordinates of the place using effective geocoding. Finally, these locations are shown on the map, where users may interact with them by clicking and interacting with them. Writers can better organise a huge amount of geographical place description material this way.

2. Background

This section will provide some background information about the research study. The goal is to acquaint everyone with the project's principles and underpinnings, particularly those connected to NER and geocoding, because these two technologies are unfamiliar in the broader software sector. Other components, like as GIS and web application development, are well-known ideas. The author will give a quick overview of their background.

2.1 Project Challenge

The specific objects studied in this project are New Zealand locality data, which are sentences describing geographic locations. Let us use one of the sample data as an example to demonstrate:

"2.4 kilometers south-west Waitaki-Steward Road, 46 meters south of high terrace on east side of road."

The goal of the project is to figure out how to locate the precise place mentioned in this sentence. This actual position is the intersection of two relative positions, and its name does not appear in this description. That is why we say the natural language is complex, and we must investigate and discover solutions in the project.

2.2 Solution Status

(Melo and Martins, 2017) conducted research on text-based document geocoding and summarized some of the existing four solutions for predicting latitude and longitude geospatial coordinates based on text content.

- The early document geocoding system used heuristic methods to query the place names mentioned in the text.
- Probabilistic language modelling method uses the words that appear in the geo-referenced training document to generate a model and then predicts the words that have not been seen before in the test document.
- Combinations of different models and heuristic models, including clustering process, feature selection methods and language models constructed from different sources.
- The latest method is based on the discriminative classification model.

The current mature solution systems include:

- STEWARD ("Spatio-Textual Extraction on the Web Aiding Retrieval of Documents"), which uses machine learning-based NER to find location labels in texts, is also a solution closer to the research of this project (Lieberman *et al.*, 2007).
- Frankenplace uses a new indexing method called geoboost to boost the terms associated with the cells on the discrete global grid, and finally presented in the form of a heat map in the visual map (Adams, McKenzie and Gahegan, 2015).
- Web-a-Where, which retrieves the name of the place mentioned in the text of the

webpage and determines the geographic location pointed to by each name and is dedicated to solving the problem of geographic ambiguity (Amitay, Sivan and Soffer, 2004).

2.3 Introduction to Named Entity Recognition

As the project's fundamental technology, the author would like to start with some background information on Named Entity Recognition (NER). The job of information extraction and retrieval requires the use of NER. Its goal is to detect and categorize the text elements that represent named entities (Mohit, 2014).

With the development of computer technology, the in-depth research of natural language understanding and text mining, and the rise of digital humanities research, text semantic knowledge has become more critical. Because named entities are necessary semantic knowledge in texts, identifying them has become a significant research problem (Mansouri, Affendey and Mamat, 2008).

From rule techniques to statistical machine learning, NER solutions are continually improved. This challenge is far from being addressed due to the arbitrary nature, complexity, and unpredictability of named entities. NER remains a significant and difficult research issue. Here are three main approaches to NER implementation.

2.3.1 Dictionary-based NER

The dictionary-based NER technique involves matching text phrases to terms in a dictionary. People can mix certain heuristics, such as creating words that exist in the object, to avoid strict matching. The drawback of stringent matching is that it can only detect words from the dictionary and will miss any text phrases that aren't in the dictionary. However, rigorous matching based on dictionary resources has the advantage of improving recognition accuracy (Eftimov, Koroušić Seljak and Korošec, 2017).

2.3.2 Rule-based NER

Artificial rulesets are used to label text in rule-based NER. To create rule sets, recognition algorithms employ grammar, syntax, spelling features, or dictionaries, all of which rely on manual coding rules and manually produced corpora. The recognition results of rule-based NER are excellent within the constraints of the ruleset, but it does not allow for autonomous learning. That is to say, it lacks portability and robustness (Mansouri, Affendey and Mamat, 2008). Furthermore, because such techniques are generally domain and language specific, they may not be effectively adaptable to other domains and languages, data changes result in significant maintenance costs for the ruleset.

2.3.3 Machine Learning-based NER

The goal of machine learning-based NER is to convert the recognition problem into a classification problem that can be solved using statistical classification models. In this method, the system builds a machine learning model to find the relationship between text and patterns (Mansouri, Affendey and Mamat, 2008), to identify nouns and classify them into

specific categories such as person, place, time, and others, using statistical models and machine learning algorithms.

2.4 Evaluative Method

People demand a systematic evaluation for NER recognition outcomes or model correctness. As assessment criterion, we must utilize the following data values. There are four indications in the confusion matrix: a, b, c, and d (Visa *et al.*, 2011).

- a is the number of correct negative predictions.
- b is the number of incorrect positive predictions.
- c is the number of incorrect negative predictions.
- d is the number of correct positive predictions.

	Predicted negative	**Predicted positive**
Actual negative	a	b
Actual positive	c	d

Following the steady growth of NER recognition evaluation, four frequently used metrics emerged: accuracy, recall, precision, and F1-score (Wang and Li, 2019).

$$Accuracy = \frac{a+d}{a+b+c+d}$$

$$Precision = \frac{d}{b+d}$$

$$Recall = \frac{d}{c+d}$$

The accuracy, recall, and precision values were initially used to evaluate the accuracy of the NER model, but in the case of unbalanced positive and negative samples, these indicators have significant flaws because the difference in the sample itself determines the massive difference in these indicators.

For example, there are one positive sample and 99 negative samples in a total of 100 samples, and we will identify all 100 samples as negative samples, then in the case of paying attention to positive samples:

$a = 99$，$b = 0$，$c = 1$，$d = 0$

- Accuracy = 99 / 100 = 99%
- Recall = 0 / 1 = 0%
- Precision = 0 / 0 = 0%

The result shows the massive difference, which makes it impossible to evaluate the model correctly. Therefore, we have introduced the F1-score for evaluation, and the formula is:

$$F1 = \frac{2PR}{P + R}$$

F1-score weights both Precision and Recall and comprehensively considers the model precision and recall calculation results. The value is towards the index with a smaller value. The larger the F1-score, the higher the quality of the model (Wang and Li, 2019).

2.5 Introduction to Geographic Information System

2.5.1 Geocoding

Geocoding refers to the process of converting addresses into geographic coordinates, and the opposite is reverse geocoding. (Goldberg, Wilson and Knoblock, 2007) said that there are two commonly used geocoding methods: geocoding services and database queries.

2.5.2 Geographic Information Visualization

(Elwood, 2011) has a theory that geographic information visualization is a map visualization based on geographic data, including two essential parts: map visualization and geographic data visualization.

With the development of the Internet, web maps allow users to search and browse spatial information and become extremely popular due to its convenience and low-cost features.

Having a web map is not enough because the browser can only display static pictures by default, so it is not easy to display an interactive map in the browser. If we want to achieve human-computer interaction with the map, such as dragging, zooming, marking and panning, we need the help of a suitable map library.

3. Methodology

This section introduces the methodology of project development, including software tool selection strategies, project development ideas and processes, and problems and solutions encountered in the project.

3.1 Software Tool Selection

3.1.1 NER

NER is associated with most of the resources and functionality in this project, so we needed to choose the framework associated with it carefully. This project is ultimately implemented as a web application whose most important feature is lightweight. The framework for NER should also follow the speed feature and not go against it.

One problem is how to balance speed and accuracy. Because a framework's accuracy and speed are at odds with each other, we need to evaluate choosing frameworks with a high level of accuracy while maintaining speed.

('Natural Language Processing NER – Which model to use?', 2020) examines the four currently dominant NER frameworks, NLTK, SpaCy, Stanza and Polyglot, and demonstrates that SpaCy is the fastest of them all.

	NLTK	**SpaCy**	**Stanza**	**Polylot**
Time(sec)	14	3.15	184	7.6
CPU	1 core 100%	1 core 100%	2 core 100%	1 core 100%
Memory	340MB	1.1GB	1.6GB	150MB

Performance comparison

(Shelar *et al.*, 2020) further explores SpaCy, Apache OpenNLP and TensorFlow and concludes that SpaCy has the highest accuracy and the best results according to the evaluation criteria presented in section 2.4.

	TensorFlow and Keras	**SpaCy**	**OpenNLP**
Training accuracy	0.9955	1.0000	1.0000
Training loss	0.0219	0.00000001427	OpenNLP NER model does not show loss
F1-score	0.9700	1.0000	1.0000
Prediction probability	0.9800	1.0000	0.9969

Accuracy comparison

	TensorFlow and Keras	**SpaCy**	**OpenNLP**
Time for prediction	0.2 μs	0. 2 μs	2 to 3 ms
Ease of training	Conversion of text data to	Directly works	Directly works

	numerical data is required for training of the model	on text data	on text data
Size of model for 30 data rows	2,807 bytes	41,51,896 bytes	1,870 bytes
Size of model for 15000 data rows	2,808 bytes	40,10,784 bytes	10,239 bytes

Comprehensive comparison

3.1.2 Geographic Information Visualization

The selection in Geographic Information Visualization development focuses on two main parts: the selection of maps and the selection of map manipulation libraries.

For the choice of maps, we compare the two most common maps, Google Maps and OpenStreetMap (OSM). (Pant, no date) has a study showing that Google Maps has powerful features such as route planning, satellite imagery and street view mode. However, it also has obvious limitations. It is not free, it is not open source, and it has API access restrictions. In contrast, OSM can solve these problems. Although it is not as powerful as Google Maps, it has the basic functionality to meet the project's needs. It is also open-source, the map data is updated quickly, and the community is active and rich in solutions, which fits the requirements of this project perfectly.

On the other hand, for the choice of map manipulation library, we are concerned with its support for source JavaScript and the effectiveness of the core functionality. In the same way, we chose between two commonly used open-source map libraries, Leaflet and OpenLayers. (Ledur *et al.*, 2015) research proved that Leaflet is lighter, has a more advanced API and is easier to use than OpenLayers. Users can create applications more efficiently. However, on the contrary, it also reflects the disadvantages of Leaflet, which are also the advantages of OpenLayers. Because Leaflet is overly encapsulated and cannot call the underlying API, it becomes inflexible for complex functionality and often requires the use of third-party plug-ins to do so. OpenLayers, on the other hand, has more powerful underlying functionality and community resources to support it. Users can use its API to perform complex functions. So for this project, where custom development is required for map operations, OpenLayers was a better fit for our selection.

3.1.3 Geocode

After the NER module has recognised the name of the location in the text, we need to use a suitable geocoding system to query the actual coordinates of the location, and with the location coordinates, we can label the location in the map. This project proposes to use SpaCy as a framework for NER in section 3.1.1 and OSM as a map for geographic information visualisation in section 3.1.2. The geocoding part of the functionality needs to be used in conjunction with the front and back end, so it should be developed in Python and formatted to conform to the OSM standard format. (*Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation*, no date a) investigated a well-established Python-based geocoding system, GeoPy, in which we can use a variety of geolocation services such as Google Maps, Bing Maps or Nominatim. in the geopy.geocoders class package. For

example, we will use the Nominatim class, which provides the OSM map service, and when we instantiate its geocoder, we can get the location information of the string through its location resolution method. It has a clear workflow, is free to use and comes with detailed documentation. The features it has are ideally suited to the requirements of this project, so we have chosen GeoPy as the geocoding library.

3.1.4 Web Application

The trend in software engineering from C/S frameworks to B/S frameworks is becoming more and more common, so this project caters for the mainstream development approach, and the final presentation will be a web application. Therefore, a good web framework is essential for the development of web applications. There are many benefits to using a web framework, such as simplified debugging, reduced code length and improved database proficiency. The choice of the Python-based SpaCy as the core NER framework was presented in 3.1.1, and the choice of the Python-based GeoPy as the core geocoding parsing tool was presented in 3.1.2. Therefore, we should also use a Python-based web application framework to accompany the development of the core functionality. (Ghimire, 2020) compared the two most commonly used frameworks for Python-web, Flask and Django.

(Holovaty and Kaplan-Moss, 2009) looked at the advantages of Django, which has a complete framework and a strong community, and Django, which has many features that Flask lacks, such as an easy-to-use interface, support for ORMs to handle databases, and a variety of caching mechanisms. However, the downside of Django is that as an overall platform, it relies heavily on the Django ORM and has much code that becomes very bloated for small projects. Conversely, (Grinberg, 2018) demonstrates that Flask's greatest strength is its lightweight nature, allowing applications to be developed with minimal code. Of course, this also shows that Flask is not a panacea. For example, it usually needs the help of WTForms for forms, SqlAlchemy for ORM operations, and cannot even handle asynchronous requests. However, as the core functionality of this project is NER and GIS, these shortcomings of Flask do not affect the needs of this project but rather meet the requirements of the project for lightness and speed. Therefore, we chose Flask as the web application framework for this project.

In web application development, the three layers of architecture are the presentation layer (View), the logic layer (Controller) and the data layer (Model), which is known as the MVC design pattern. The presentation and logic layers can be developed and implemented in a web framework. The data layer needs to be implemented with a suitable storage and interaction model, and its function is mainly responsible for accessing the database system and various files.

This project is a web application, and JSON is the industry standard response format, so the files are stored in JSON format. SQL is a relational database that requires an exact data format to be specified and uses a solid declarative language for queries, has transaction processing to maintain data consistency and supports complex queries. NoSQL is a non-relational database that can hold data in an unspecified format and offers excellent performance and scalability. The data in this project includes source data, training data, test data, and display data. The format (fields) of this data is defined and does not frequently

change at random. The project itself does not involve high concurrent requests or distributed storage, so the flexibility of NoSQL cannot be used in the project. MySQL is the most widely used SQL database. It is open source, has excellent performance and service stability, and has strong community support. Therefore, we chose MySQL as the database for this project.

3.2 Data Processing

Section 2.1 mentioned that the primary research background of this project is New Zealand locality data. We need New Zealand's official gazetteer and place name description data.

3.2.1 Data Source

The source of the data is the official website of the New Zealand Gazetteer (*NZGB Gazetteer | linz.govt.nz*, no date), which contains all geographical locations and features within the jurisdiction of the New Zealand Geographical Council Ngā Pou Taunaha o Aotearoa (NZGB) (*New Zealand Gazetteer of place names*, no date).

3.2.2 Data Storage

The data storage solution of the project is oriented to all the data used in the project, including source data, training data and result data. We use a double backup mechanism, and each piece of data will be stored in both the file and database in the project to ensure the security and traceability of data storage.

3.3 NER

Following the introduction of NER in section 2.3, this section continues to discuss the research methods of NER. Entities are words in the text that correspond to specific types of data. They can be numbers, times, names, locations, and geopolitical entities (GPE). The entity is the content with specific tags marked in the text by the NLP researcher.

The function of the NER part of the project is to develop and train a model that can find location entities in the text. For example, the following sentence:

"Walker Estate, 2.4 km north of Plimmerton, on main north highway."

The ideal situation is to recognize *"Walker Estate"* and *"Plimmerton"* as location (LOC), *"2.4 km"* as distance (DIS), and *"north of"* as direction (DIR). This preliminary has the conditions to locate the location described in this sentence accurately.

In this project, we will use both the rule-based model and the machine learning model to identify NER and compare them. In the following part, we will explore the theoretical recognition method of NER step by step.

3.3.1 Create Rule-based Model

As mentioned in section 3.1.1, the project uses SpaCy as the framework for NER, so it must follow its pipeline to configure the model. SpaCy provides several matching methods for rule-based NER, including token matcher, phrase matcher, dependency matcher, and entity ruler. An entity ruler is one of them, which is usually not generated during model initialization

and needs to be manually added to the pipeline. It can find matching entities in the text according to the pattern tags we set and add them to the entity set of the text (*Rule-based matching · spaCy Usage Documentation*, no date a). Because it can be used for both rule-based models and can help generate the training set required by machine learning models, it is a component we need to use in our projects.

To use the entity ruler, we need to add an entity pattern object to it. Entity patterns are dictionaries with two keys: "label", specifying the label to assign to the entity if the pattern is matched, and "pattern", the match pattern. The entity ruler accepts two types of patterns (*Rule-based matching · spaCy Usage Documentation*, no date a):

- Phrase pattern for exact string matching (string).

```
1. {"label": "ORG", "pattern": "apple"}
```

- Token patterns (lists) that have a dictionary describing a token.

```
1. {"label": "GPE", "pattern": [{"LOWER": "SAN"}, {"LOWER": "Francisco"}]}
```

Because this project is about identifying location names, section 3.2.1 mentions that the resource we obtained is a dictionary of the New Zealand Gazetteer. So we choose the first type of entity pattern, the phrase pattern. Finally, we generated the rule-based NER model by adding the created entity ruler as a pipeline to the language model.

3.3.2 Create Training Data

Creating a training set is the basis for generating a machine learning-based NER model. This section explains the format of the spacy training set and the process of using entity ruler and rule-based NER model to work together to create it.

The required form of the spacy training set is as follows:

```
1. TRAIN_DATA = [(text, {"entities": [(start_char, end_char, label)]})]
```

It is unrealistic to manually label every sentence (especially in extensive training data), so it is an excellent way to use the previous entity ruler and rule-based NER. We can obtain the start_char, end_char and label of the entity in the text based on the entity set they recognize, and then create a training set that meets our required format through simple code processing. It is important to note that the training set obtained at this time may not be completely accurate. The purpose of this operation is to enable users to reduce the workload of manually labelling entities. If we want to use this training set to train a machine learning model, we must manually check it to keep it correct.

3.3.3 Create Machine Learning-based Model

In this project, we use supervised learning in machine learning, which is the process by which a system learns from a set of inputs with known labels. We treat the annotated and labelled text in section 3.3.2 as known data and use them for training the machine learning model. We divide the input data into two categories: training data and validation data. While there is no fixed ratio between the two, a good rule of thumb is to use 80% of all annotated data for training and the remaining 20% for validation. The project provides two formats for storing training and validation data. One is a JSON file, which is suitable for training with

code. The other is a binary file, suitable for training with configuration files.

The unit of the training process is a single data iteration (epoch) in which the SpaCy framework uses the training data to tune the statistical model by a pre-defined algorithm. The SpaCy framework checks the model's accuracy by comparing the input labels with the predicted correct labels and adjusting them accordingly. When all the training data is viewed and predicted, an epoch is completed. The model is then tested for accuracy against the validation data. Because the validation data is not part of the training process, it can be used to verify the model's accuracy. The training data is then randomised and passed into the model for the next epoch. There is no standard value for the number of epochs in the same proportion as the previously split annotated data, but a good rule of thumb is to set it at ten epochs.

The training process is complete when the model repeats the process for the specified number of iterations. The SpaCy framework then tunes the model's parameters and finally provides us with two versions of the model: the most recent version and the best version. We can store the models for later use using the model saving API provided by SpaCy.

3.3.4 Using Models to Identify Entities in Text

After obtaining the NER models by the above methods, the SpaCy framework allows us to use these models to perform named entity recognition on text. SpaCy provides an API for loading models and generating a NLP instance, which can then be used to parse the text to obtain a collection of entities in the text. The location entities we are interested in are also in this collection, and we can identify them by their label attributes and obtain their names for use in subsequent steps of the project.

Due to the diversity of natural language, there can be many ways to describe the location, and they are often not detailed location descriptions. Such as *"4 miles north of Auckland Road"*, *"5 miles from the end of Auckland Road"*, and *"At the intersection of Auckland Road and New Port Road"*. These uncertain relative positions increase the difficulty of text recognition because it is impossible to rely on manual rule sets to match all the descriptions of relative positions.

After looking at the data to be tested, we found that most location descriptions had two forms, so we also set up two mechanisms for determining the location. One can directly locate the location name, such as *"Auckland Road"*. We will recognize this location as “Auckland Road”. The other is identifying the *"direction of location"* format, such as *"4 miles north of Auckland Road"*. We will recognize this place as “north of Auckland Road”. As we cannot rely on manual rules to cover all cases, we can only identify the two cases above. Although this is not a perfect solution and will reduce the accuracy of our system, it is already an excellent solution to the technical difficulty of identifying relative positions.

3.4 Geocoding

3.4.1 Coordinate Geocoding

In section 3.3.4, we have obtained the location names, which means that the NER part of the work is almost complete, and all that remains to be done is to translate the location names

into the corresponding coordinates to be rendered on the map. Here we need to use the GeoPy tool mentioned in section 3.1.3 and use the API it provides for converting strings into coordinates.

3.4.2 Coordinate Validation and Coordinate System

Once we have found the coordinates corresponding to the location, we need to validate them, as the coordinates parsed by the geocoding service may deviate from the actual coordinates of the location. The coordinate format provided by Gazetteer is the New Zealand Map Grid (NZMG) (*New Zealand Map Grid (NZMG)*, no date) , which is not based on geometric projections, but instead uses complex polynomial expansions. Although this has the advantage of minimal scale distortion, it is a uniquely New Zealand projection. Therefore people may be challenging to use or program it into computer software. Therefore, we will need to convert its coordinates and use the online conversion tool available on the official website to convert NZMG coordinates to standard WGS84 coordinates via a web request (*New Zealand Coordinate Conversions*, no date).

3.5 Geographic Information Visualization

3.5.1 Map Elements and Interactions

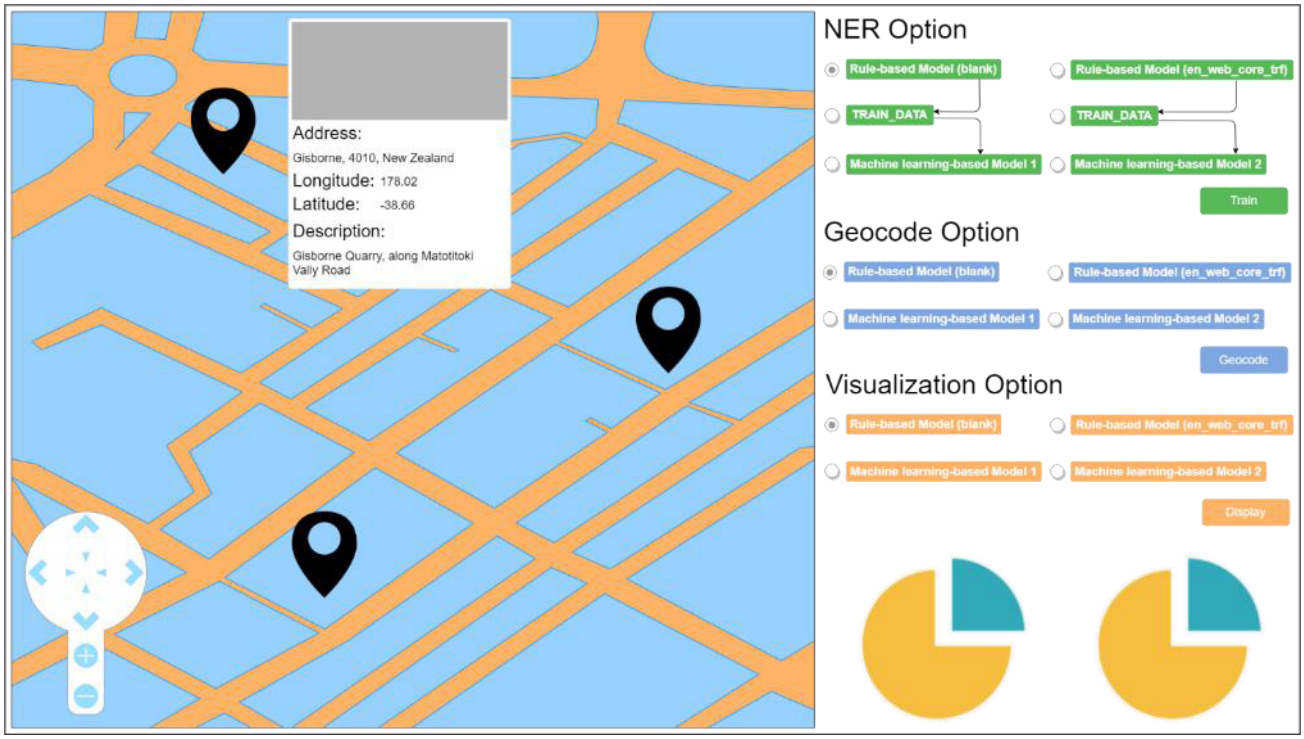
In section 3.1.2, we describe the OpenLayers map manipulation library that provides APIs for manipulating the OSM. We can use them to pan and zoom the map, add (delete) elements to the map and implement click (hover) interaction events for map elements (*OpenLayers v6.6.1 API - Index*, no date).

3.5.2 Data Visualization

The data visualisation for this project consists of a map element data visualisation and a statistical chart data visualisation. In the map element data visualisation, we also use the OpenLayers map manipulation library API to display locations as markers on the map and display information about the location as pop-ups and text in the appropriate places next to the markers. This project uses ECharts, an open-source JavaScript visualisation library that provides a wide range of visual charts, beautiful effects, and deep interactive exploration to design statistical charts (*Apache ECharts*, no date). We will choose an appropriate form of chart to display the data of interest in the project.

3.6 Web Application

3.6.1 User Interface



UI

3.6.2 Functionality

The web application is divided into four functional sections: the map section, the NER section, the geocoding section and the visualisation section.

The map section contains the map display and map interaction functions. Users can pan and zoom the map, click on the map elements and view the location information of the markers.

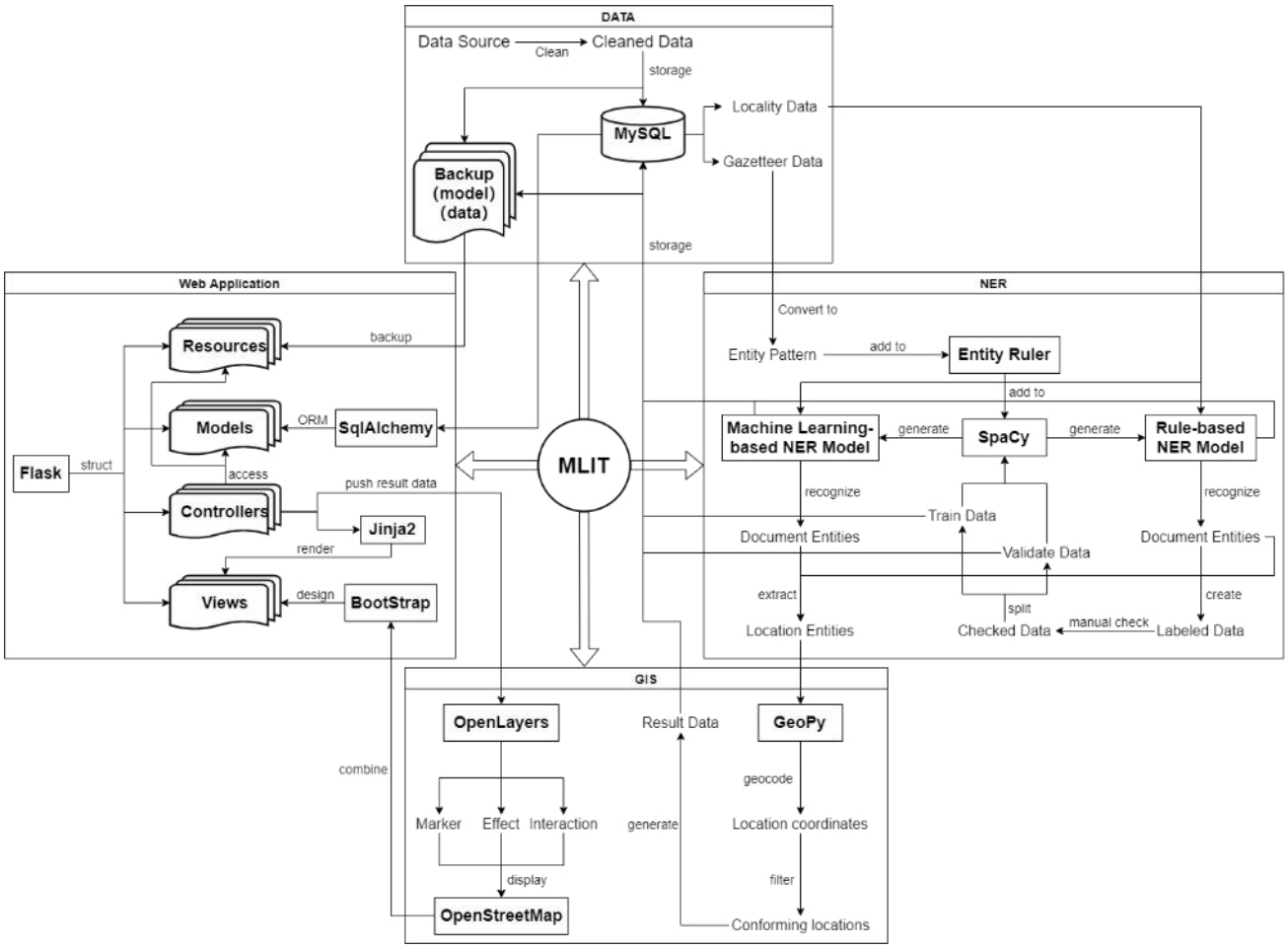
The NER section contains training functions for training models and data. Users can train two rule-based models, two training data based on them and two machine learning models based on the two training data.

The geocoding section contains functions for geocoding locality data and generating the resulting data. Users can use four different models to obtain the result data after geocoding the identified location data.

The visualisation section contains map data visualisation and statistical chart visualisation. The user can choose to display the four different results of the geocoded data on the map and see the statistical analysis of the results.

4. Implementation

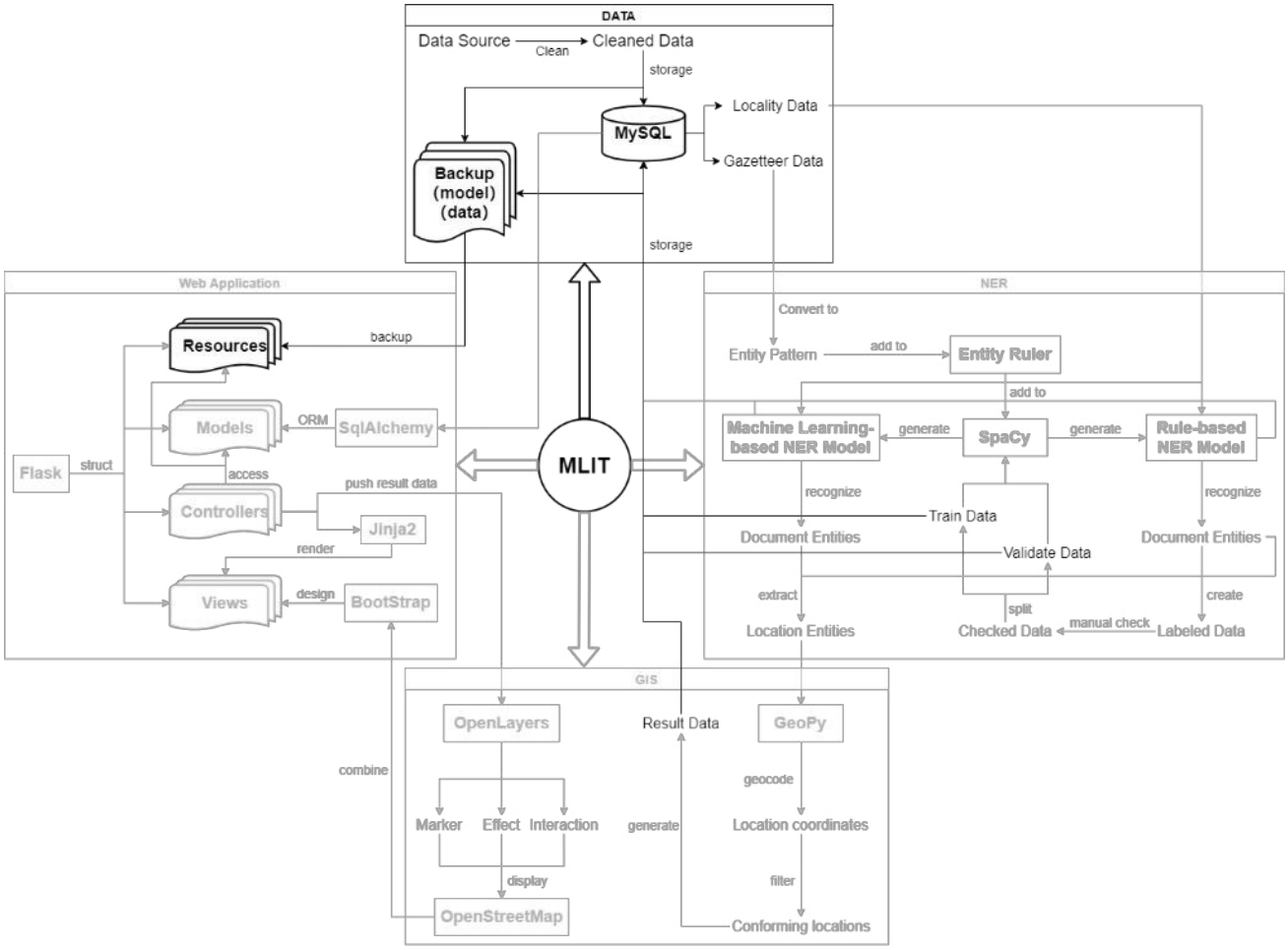
This section describes how each part of the project is implemented. We will explain how each part of the project is connected from the macro-architecture to the micro-code, how the software development and framework make sense, and how to move logically towards the final goal.



Overall project structure

4.1 Data Processing

In section 3.2.1, we have explained how to obtain a valid data source. So in the data processing section, we will focus on describing the implementation of the data cleaning and data storage processes in the project.



Data processing

4.1.1 Clean Data

The Locality data and the Gazetteer data we obtained from the official website are shown below.

East	North	Locality
2338613	5579915	RIDGE RD 20YRDS SW OF SHEEPYARDS 1.5MILES NW OF CORRIEDALE YARDS 7YRDS BEHIND CUTTING ON N SIDE OF RD IN SHELTER BELT
2177600	5548500	2m SOUTH OF LOWER FENCELINE OF DSIR TRIAL C (2ND FROM TOP) 9m FROM SW CORNER.
2740430	6015832	
2522335	5985171	500M NE ALONG PUGH'S RD FROM INTERSECTION WITH RANZAU RD 50M FROM PUGH'S RD ON N SIDE.
2692680	6049260	1.6km N OF OTAKI BOROUGH. N END TAYLORS ROAD, W OF RAILWAY LINE, N OF DAIRY SHED ALONG RACE ON TERRACE S OF PEAT-FILLED VALLEY
2467170	5729405	SE CORNER OF FIELD. A11 ON LINCOLN COLLEGE MIXED CROPPING
2755800	6280700	Kawakawa Bay, NW shore of Lake Taupo, east road of Otutira Catchment, 600 m north of lake
2690400	6041400	6km SSW OF OTAKI RAILWAY STATION. IN PADDOCK BEHIND HOMESTEADS, 52m DUE SOUTH OF LARGE MACROCARPA TREE BESIDE FARM GATE, IN LINE WITH PINE TRE
2768200	6351700	3km ENE of Fitzgerald Glade on SH 5
2805657	6375949	Near main A drain, 80 m. east of Kaituna River.
2407771	5870040	GRANVILLE STATE FOREST, 14.5KM SE OF TOTARA FLAT, ON DISSECTED TCE REMNANT BETWEEN MIDDLE AND LEFT BRANCHES OF WAIPUNA CK, IN FOREST AT END OF
2713550	6378930	2.75km north east of Hamilton City Post Office, 10m north of race, 10m east of fence on west side, paddock located 300m east of main bend by b
2417446	5876248	1KM E OF BROWN'S SAWMILL, 10M UPSLOPE N OF RD. 18KM E OF HUKARERE
2515062	5832541	Mt Palm (photo D25)
2490806	5823786	MAF TRIAL SITE ON ROSEWOOD
2804405	6380192	Bells Rd, end of farm race at high inland dune, side of sandpit
2308030	5472384	1/3 MILE S OF MAIN BRIGHTON-GREEN ISLAND RD, ON N SIDE OF TRACK TO WALDRONVILLE CAR RACING CIRCUIT
2970058	6311112	GISBOURNE-RUATORIA HIGHWAY: 2 KM NORTH OF MAUNGATUNA SCHOOL BETWEEN HIKUWAI RIVER AND HIGHWAY 35: 500M SOUTH ANAURA BAY T.O
2798595	6384929	20m NE of Papamoa Beach rd, midway between 2 farm houses
2530317	5905315	FOREDUNE IMMED S TARAMAKAU RIV MOUTH SW CNR OF PADDOCK S OF LAST HOUSE ALONG KUMARA BEACH RD,10M IN FROM RD, 3M FROM S FENCE
2467600	5729900	DSIR Farm, adjacent to Lincoln Campus
2821024	6295400	Compartment 69 Kaiangaroa Forest - about 20 metres from FRI lysimeter same pit as VUW tephra studies.
2783304	6281011	6KM NW OF TAUPO, ARATIAVIA RD, 1.5KM N OF INTERSECTION WITH VIEW RD, CUTTING E SIDE OF THE RD.
2466979	5729905	NW CORNER OF FIELD A10 LINCOLN COLLEGE MIXED CROPPING FARM
2266400	5617400	300M FROM SMOST PT OF PROPERTY 50CM W OF E BOUNDARY FENCE, TARA HILLS HCRS
2713991	6284618	4km East of Waimiha, SE corner of intersection Ongarue Stream Road and Waione Road.
2192359	5446633	SEPTIC TANK PIT PADDOCK S3 1M FROM E BOUNDARY FENCE 5M W OF JTN OF ACCESS RACE AND MAIN N RACE
2740563	6015532	
2903000	6300670	Mangatu Road, 2 km SE Wairere Road
2348407	5574236	100YRDS W OF GATE ON TRACK UP BROCKMAN'S HILL. PIT 5YRDS S OF TRACK

Original Locality Data

name	status	crd_latitude	crd_longitude	info_description
15 Mile Creek	Official Approved	-40.79825	172.514222	Feature shown on: NZMS260 M25 Edition 1 1984; NZTopo50-BN23 GR 590 835
17 Mile Creek	Official Approved	-40.807972	172.4705	A stream the flows off the Wakamarama Range, south-west of Mt Stevens. Feature shown on: NZTopo5
1st Basin	Unofficial Recorded	-41.838556	172.8105	Feature shown on: NZMS260 N29 Edition 1 1984 Limited Revision 1996
1st Staircase	Unofficial Recorded	-38.450361	174.642722	Feature shown on: NZMS260 R17
2nd Basin	Unofficial Recorded	-41.850472	172.79575	Feature shown on: NZMS260 N29 Edition 1 1984 Limited Revision 1996
2nd Staircase	Unofficial Recorded	-38.416187	174.638151	Feature shown on: NZMS260 R17
3rd Basin	Unofficial Recorded	-41.86	172.783472	Feature shown on: NZMS260 N29 Edition 1 1984 Limited Revision 1996
45th Parallel marker	Unofficial Recorded	-44.99919	171.05264	Feature shown on: NZMS260 J41 Edition 1 1984
4th Basin	Unofficial Recorded	-41.875417	172.770417	Feature shown on: NZMS260 N29 Edition 1 1984 Limited Revision 1996
5 Mile Creek	Official Approved	-41.746722	172.908778	Feature shown on: NZMS260 N29 Edition 1 1984 Limited Revision 1996; NZTopo50-BR25 924783
A1 Creek	Unofficial Recorded	-42.215204	171.893184	Feature shown on: NZMS260 L31 Edition 2 1996
Aan River	Unofficial Recorded	-46.202139	166.981417	Feature shown on: NZMS260 B46 B47
Aaron Creek	Official Approved	-41.040645	172.825871	Feature shown on: NZMS260 N26 Edition 1 1986; NZTopo50-BP24 854538-849591
A. A. Thomas Hills	Unofficial Replaced	-77.776132	163.868994	A line of prominent summit ridges/hills at the north end of Blue Glacier, that extend from Bettle Peak eas
Abaft Gully	Unofficial Recorded	-44.034639	168.979639	Feature shown on: NZMS260 F38 Edition 1 1995
Abandon Brook	Unofficial Recorded	-44.067891	169.001528	Feature shown on: NZMS260 F38 Edition 1 1995
Abbassia Camp	Unofficial Recorded	-39.440944	175.67425	Feature shown on: NZMS260 T20 Edition 1994

Original Gazetteer Data

For the Locality data, we focus on three columns (East, North and Locality), where the Locality column stores the description of the target location we need to identify and find, while the East and North columns represent the target location's NZMG coordinates.

For the Gazetteer data, we are looking at the following columns (name, status, crd_latitude, crd_longitude and info_description), where the name column is the name of the location, the status column represents the status of the location, the crd_latitude and crd_longitude columns record the WGS84 coordinates of the location and the info_description column stores the description of the location.

After acquiring the data source, data cleansing is necessary, which involves cleaning redundant data and correcting invalid data.

Redundant data is not needed or of interest to the project, and we need to remove it to reduce the data storage load on the system. We do this by wrapping the data tables into views that need to be used at the data layer. In Locality data, we remove the location descriptions whose contents are empty strings, as they are meaningless.

```
1. SELECT North, East, Locality FROM locality_data WHERE Locality != ''
```

SQL query for filtering locality data

North	East	Locality
6648900	2597900	Pakaraka, North Auckland. South side main north highway , opposite Pakaraka School.
6653600	2583600	Okaihau, North Auckland. 200 yd west of junction of main north highway and Kerikeri road, 2.8 km east of Okaihau P.O.
6629600	2579500	Mataraua, south of Kaikohe, North auckland. East side of Kaikohe - Donnelly's Crossing road, 0.8 km north-east Mataraua School outside property
6625800	2576600	Waimatenui, North auckland. 5.4 km north-north-east Waimatenui Hall on Kaikohe-waimatenui road on property of H. Lyon.
6618500	2545500	Tutamoe plateau, North Auckland. Roadside, East side of Kaikohe-Doneelly's Crossing road and 2.4 km north-east of Tutamoe School.
6580000	2596800	Arapohue, North auckland. East side of Dargaville-Arapohue road, .25 mile north of junction with Simpkins Road on property of A. Hamlyn.
6666700	2541700	Omahuta Forest (Compartment 8). Lefthand side of track to Kopi.
5767300	2406300	On bench 200 ft above and south of main highway, 1.1 miles west of Porters Pass.
6517500	2661300	Waiwera Hill, North Auckland. 1.5 miles north-west of Waiwera P.O. on main highway outside property of W.J. Johnstone.
6534600	2658400	Warkworth, North Auckland. 1 km along Goatleys Road from main highway outside property of Macey.
6543200	2649400	Wellsford, North Auckland. 1.5 miles south-east of Wellsford on south side of Rusty Brook Road outside proprty of Mr F. Coop.
6589500	2650600	Side road off road to One Tree Point wharf, .75 miles from main highway to Whangarei.
6615000	2619800	Ruatangata West, on roadside, 0.7 km along Worsnops Road from Ruatangata, east side of road.
6613200	2624800	Ngararatunua Cone, Kamo, Whangarei Co.
6157040	2750835	Mangaweka Hill, 6 miles north of Mangaweka and 8 moles south of Taihape. On "island" formed by straightening road.

Cleaned Locality Data

In the case of the Gazetteer data, the operation is a little more complicated, as we query the data table for data containing the following statuses with the following SQL:

```
1. SELECT DISTINCT status FROM gazetteer_data
```

SQL query for filtering status types in gazetteer data

status
Official Assigned
Unofficial Replaced
Unofficial Discontinued
Official Altered
Unofficial Recorded
Official Approved
Official Validated
Official Adopted
Official Official By Other Legislation
Official Valid
Unofficial Collected
Unofficial Original Māori Name

Status of Gazetteer Data

We choose the data that is officially relevant because they are valid and accurate. In reviewing the data, we find that any descriptions with 'Feature shown on:' only record image information and do not have a specific description of the location, so these are redundant. In the same way as Locality data, we will also exclude descriptions with empty strings.

info_description
Fishing pool on the Tongaririo River.Feature shown on: NZMS260 T19 Edition 2 1994
Fishing pool on the Tongaririo River.Feature shown on: NZMS260 T19 Edition 2 1994
Fishing pool on the Tongaririo River.Feature shown on: NZMS260 T19 Edition 2 1994
Fishing pool on the Tongaririo River.Feature shown on: NZMS260 T19 Edition 2 1994
Fishing pool on the Tongaririo River.Feature shown on: NZMS260 T19 Edition 2 1994

Redundant data in Gazetteer Data

```
1. SELECT name, status, crd_latitude, crd_longitude, info_description FROM gazetteer_data WHER
   E `status` LIKE 'Official%' AND info_description NOT LIKE '%Feature show
   on:%' AND info_description != ''
```

SQL query for filtering gazetteer data

name	status	crd_latitude	crd_longitude	info_description
Abel Glacier	Official Assigned	-43.329576	170.6381	Leading from Garden of Eden Ice Plateau and draining into Perth River about half a mile above its confluence with Adve
Abel Janszoon Glacier	Official Assigned	-43.554196	170.152539	Glacier on north side of Mount Tasman
Abel Lake	Official Assigned	-43.33927	170.638249	Lake on west side of Main Divide, feeding into the Perth River
Abel Tasman National Park	Official Assigned	-40.91	172.966111	Between Totaranui and Marahau
Abel Tasman Point	Official Assigned	-40.800035	172.921583	On the headland between Ligar Bay and Wainui Inlet. Northeast of Tarakohe. Block III Totaranui Survey District.
Abel Tasman Roadstead	Official Assigned	-40.777632	174.019604	Strait off east of Rangitoto ki te Tonga / D'Urville Island.
Abseil Peak	Official Assigned	-44.060919	169.677636	Peak 2435, north west of Mt Huxley in the Huxley Range.
Acacia Bay North	Official Assigned	-38.708627	176.029164	Bay, Lake Taupo, between Waikereru Point and Te Kopua Point.
Acacia Bay South	Official Assigned	-38.712122	176.027139	Bay, Lake Taupo, between Waikereru Point and Otuparae Point.
Access Peak	Official Assigned	-44.736195	167.9158	South of the confluence of Claddau River West Branch and Claddau River South Branch, appoximately 4.8 km west of H
Acheron Flat	Official Assigned	-43.411765	171.605482	Flat on the Left Bank of the Rakaia River South-east of the Acheron River, 7km South-east of Lake Coleridge locality. Bl
Acheron Lakes	Official Assigned	-45.165083	168.0949	For the two small lakes in the headwaters of the Upukerora River (midway between the lakes)
Achilles Peak	Official Assigned	-43.567983	170.727685	On Two Thumb Range, south of Mount Alma and divided by Alma Col
Aciphylla Creek	Official Assigned	-43.278656	170.704831	Creek providing access to Adams Range
Ada Flat	Official Assigned	-42.329991	172.601618	A river flat on Waiau River between the Ada Homestead and Little Lake.
Adam Creek	Official Assigned	-44.207259	168.223006	A southern tributary of Gorge River just east of "Jacobs Creek"
Adam Creek	Official Assigned	-43.720775	169.209166	Stream crossing Paringa-Haast Road. Flows into Tasman Sea 35 chains south-west of Harvey Creek. Block X Arnott SD.
Adams Burn	Official Assigned	-45.256278	167.346424	Flows easterly into Lake Te Au about halfway along western shore. Esk Survey District.

Cleaned Gazetteer Data

On the other hand, invalid data is data that is flawed and faulty. For example, 'DATA' and 'data' are two different words to the computer in subsequent natural language processing. We need to standardise the data format. Location descriptions in locality data that are entirely in uppercase letters need to be standardised to lowercase representation. In terms of orientation descriptions, we also found various representations of 'south', such as "S" and "SOUTH", which need to be standardised into the exact representation. These need to be harmonised into a single representation. We use Python on the back end to process these strings.

```
TODO Replace other expressions to standardized expression, for example: replace “S” to “south”
cleaned_text = cleaned_text.lower()
```

Uniform description

4.1.2 Data Storage

The project's data storage system uses a "double backup" model, where all critical data is archived in the file system and the database, a common means of data storage in software project development. This practice effectively prevents data loss or corruption and allows the choice of two data loading methods during development depending on the state of the network.

We implement both storage methods in the back-end, and the data to be stored contains data sources, training data, and recognition results data. In the file mode, we use the API provided by Python to wrap into a custom method for storing the data.

```
FUNCTION save_data(file, data):
    TODO Save data to the file
    with open(file, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=4)
```

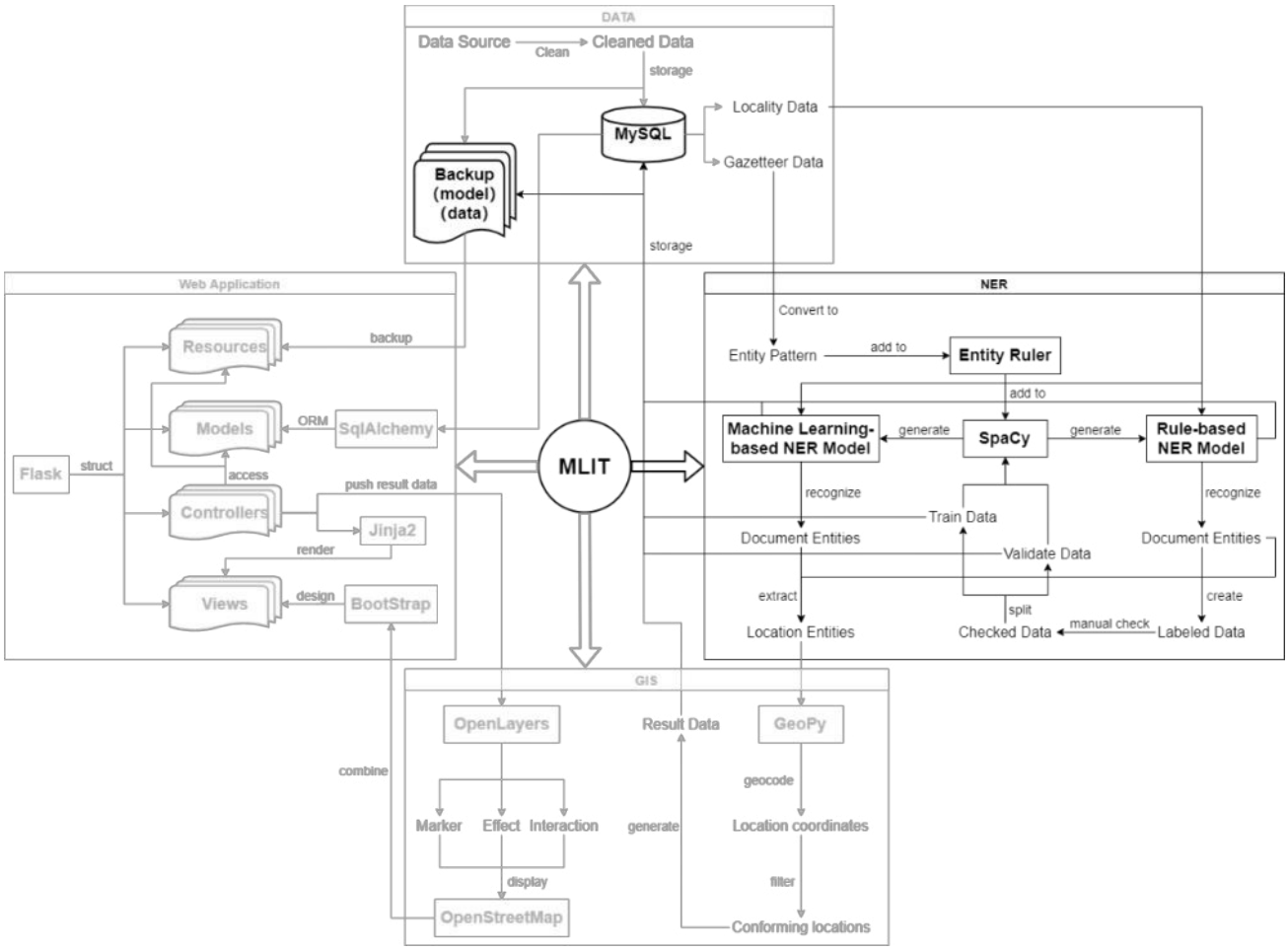
Save data to the file

In the database mode we use the API provided by SqlAlchemy to store the data.

```
FUNCTION add_train_data(train_data):
    TODO Save data to the database
    db.session.add_all(train_data)
    db.session.commit()
```

Save data to the database

4.2 NER



NER

4.2.1 Create Entity Ruler

The first step in creating an entity ruler is to generate an entity pattern. This project requires the recording of three label types of entity pattern, “LOC” (non-GPE locations), “GPE” (counts, cities, states) and “DIR” (directions) (*Annotation Specifications · spaCy API Documentation*, no date).

Models trained on the OntoNotes 5 corpus support the following entity types:

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

named-entities annotation

After reading the Gazetteer data from the database, we used the data in the "name" column as "LOC" labelled entities. We used the data in the "land_district" column as "GPE" labelled entities. Then defined the eight directions as custom 'DIR' labelled entities.

```
TODO Create a list record pattern
FOR each location of locations:
    TODO Create a pattern instance with the label "LOC"
    patterns.append(pattern)
END FOR
FOR each GPE of GPEs:
    TODO Create a pattern instance with the label "GPE"
    patterns.append(pattern)
END FOR
FOR each direction of directions:
```

```
    TODO Create a pattern instance with the label "DIR"
    patterns.append(pattern)
END FOR
```

Create entity pattern

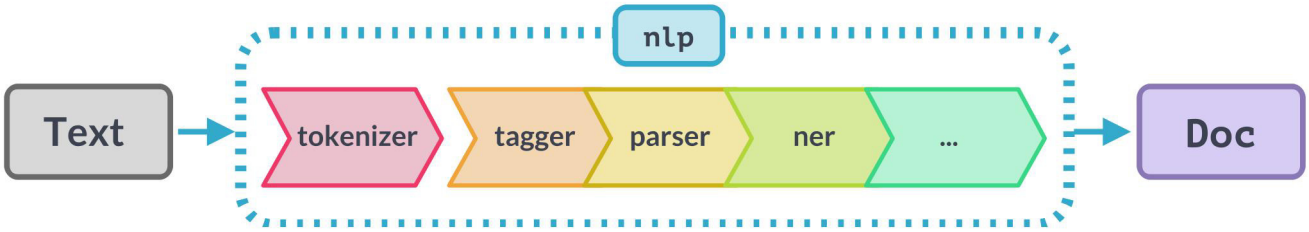
Once the entity pattern has been generated, we need to add it to the entity ruler via the API provided by SpaCy.

```
entity_ruler.add_patterns(patterns)
```

Add entity pattern to entity ruler

4.2.2 Create Rule-based NER Model

Section 3.3.1 introduced SpaCy's pipeline mechanism, which typically consists of a tagger, a lemmatizer, a parser, and an entity recognizer. SpaCy transforms text into Doc objects, which are cascaded through the pipeline. Each pipeline component returns the processed Doc object, which is passed on to the next component (*Language Processing Pipelines · spaCy Usage Documentation*, no date a).



SpaCy pipeline

We create two rule-based models, one created purely using the entity ruler and the other using SpaCy's model combined with the entity ruler. The difference is that the first uses an empty language model, which only has a pipeline of the entity ruler we added (*Rule-based matching · spaCy Usage Documentation*, no date a) . The second uses an existing language model combined with the entity ruler pipeline, which has multiple pipelines allowing for more annotations and labels, so we can use more tools to process the text, which could potentially improve the accuracy of NER (*Rule-based matching · spaCy Usage Documentation*, no date b). However, this is only a guess, and the complexity of natural language is such that it is impossible to determine which model works better, and we will provide a detailed comparison in section 5.

The creation is also simple. Just add the entity ruler we have already made in section 4.2.1 to the NLP model. The first NLP model is an empty, simple English language model created using the code below.

```
nlp = English()
entity_ruler = nlp.add_pipe('entity_ruler', config=config)
```

Add entity ruler to English model

The second model is slightly different, and we need to choose a suitable SpaCy language

model for the combination. SpaCy provides four trained English language models, and their accuracy for NER is shown below (*English · spaCy Models Documentation*, no date).

	en_core_web_sm	en_core_web_md	en_core_web_lg	en_core_web_trf
ENTS_P	0.84	0.85	0.85	0.90
ENTS_R	0.83	0.85	0.85	0.90
ENTS_F	0.84	0.85	0.85	0.90

SpaCy English model comparison

We see that "en_core_web_trf" is the model that works best for NER, so we choose it to generate the second model. In the "add_pipe()" method, we can also set the value of the before parameter to determine the position of the entity ruler in the pipeline, e.g. before="ner" means we place the entity ruler before the ner component.

```
1. nlp = spacy.load('en_core_web_trf')
2. entity_ruler = nlp.add_pipe('entity_ruler', before='ner', config=config)
```

Add entity ruler to "en_core_web_trf" model

Once the two models were created, we used the API provided by SpaCy to store them separately for future use.

```
1. nlp.to_disk(model_path)
```

Save NER model to local

We can verify the models' pipelines from the generated files and list their functions below. We can see that the second model has more pipelines and functionality than the first model (*Language Processing Pipelines · spaCy Usage Documentation*, no date b).

Name	Component	Description
transformer	Transformer	Assign the tokens and outputs of a transformer model.
tagger	Tagger	Assign part-of-speech-tags.
parser	DependencyParser	Assign dependency labels.
entity_ruler	EntityRuler	Assign named entities based on pattern rules and dictionaries.
ner	EntityRecognizer	Assign named entities.
attribute_ruler	AttributeRuler	Assign token attribute mappings and rule-based exceptions.
lemmatizer	Lemmatizer	Assign base forms to words.

Component list

```
1. "pipeline":[
2.     "entity_ruler"
3. ]
```

The pipeline of the first NER model

```
1. "pipeline":[
```

```
    "transformer",
    "tagger",
    "parser",
    "entity_ruler",
    "ner",
    "attribute_ruler",
    "lemmatizer"
]
```

The pipeline of the second NER model

4.2.3 Create Training Data

In section 3.3.2, we introduced the idea of relying on the rule-based NER model and the entity ruler to create training data, and it became clear that we needed to generate training data in the following format.

```
TRAIN_DATA = [(text, {"entities": [(start_char, end_char, label)]})]
```

Format of TRAIN_DATA

Once the Doc object has been pipelined through the NER model, we can access the list of entities in the text via the "doc.ents" property. For each entity in the list, we can obtain its start and end position in the text via its "start_char" and "end_char" properties and its label information via its "label_" property. So we can easily manipulate the list of entities to produce the data format we want by using the following code.

```
TODO Obtain the Doc instance processed by NER model and create a list to record entity
FOR each entity of entities in Doc:
    entities.append((ent.start_char, ent.end_char, ent.label_))
END FOR
IF entity list has entity:
    train_data = [text, {"entities": entities}]
END IF
```

Create train data generated by each text

In section 4.2.2, we mentioned that the project generates two types of rule-based NER models with different functions, and each of them has a different list of entities that can be obtained. For example, we process the above code for the following description.

"Leading from Garden of Eden Ice Plateau and draining into Perth River about half a mile above its confluence with Adverse Creek. Tyndall Survey District."

```
[
```

```
    "Leading from Garden of Eden Ice Plateau and draining into Perth River about half a mile above its confluence with Adverse Creek. Tyndall Survey District.",
    {
        "entities": [
            [13, 39, "LOC"],
            [58, 69, "LOC"],
            [114, 127, "LOC"]
        ]
    }
]
```

Train data for the first NER model

```
[
    "Leading from Garden of Eden Ice Plateau and draining into Perth River about half a mile above its confluence with Adverse Creek. Tyndall Survey District.",
    {
        "entities": [
            [13, 39, "LOC"],
            [58, 69, "LOC"],
            [70, 87, "QUANTITY"],
            [114, 127, "LOC"],
            [130, 137, "GPE"]
        ]
    }
]
```

Train data for the second NER model

We can see that both models identify "Garden of Eden Ice Plateau" (locations from 13 to 39, labelled "LOC"), "Perth River" (locations from 58 to 69, labelled "LOC") and "Adverse Creek" (locations from 114 to 127, labelled "LOC"). However, the second model additionally identifies "about half a mile" (locations from 70 to 87, labelled "QUANTITY") and "Tyndall" (locations from 130 to 137, labelled "GPE"). This phenomenon is because they have different pipelines. The first model only includes our defined entity ruler pipeline, which contains only our defined entity patterns labelled "LOC", "GPE", and "DIR". It cannot recognise entities outside of these entity patterns. The second model contains the entity ruler pipeline and the other pipelines in the "en_core_web_trf" model to recognise more entities.

So far, it looks like we have completed the construction of our training data, but there is one more step to check our annotations for accuracy. The safest thing to do is to check the annotation of each sentence manually. However, as mentioned in section 3.3.2, due to the large amount of data we have (around 8000+ descriptions), it is not practical to check them all manually, and SpaCy provides a component called PhraseMatcher. When we need to match large lists of terms, we can use PhraseMatcher and create Doc objects instead of markup schemas, which is more efficient overall (*Rule-based matching · spaCy Usage Documentation*, no date c). Therefore, we have chosen to use it to help us do a basic check and then do a manual spot check, which is implemented as follows.

```
TODO Obtain the Doc instance processed by NER model and create a set record entity position
FOR each entity of entities in Doc:
    set.update(range(entity.start, entity.end))
END FOR
TODO Create a matcher and add location dictionary to matcher
TODO Get the matches after matcher retrieves the text
FOR each match of matches, obtain its start and end position:
    IF neither the start nor the end position is in the set record position:
        TODO Create a Span instance as the omitted location and add it to Doc's entities
    END IF
END FOR
```

Use PhraseMatcher check labelled train data

We use the model to identify the entities in the current text and record their locations, then use the "make_doc()" method to add all the location names in the location dictionary to the matcher. The matcher processes the current text again and checks if location entities in the text exist in the dictionary but are not correctly identified in the entity list. If any are found, we add them to the entity list. This completes the work of checking the entity annotations using the PhraseMatcher, and we add all the text to the final training data after checking them one by one. Finally, we manually sampled 200 of them to ensure their accuracy and saved them.

```
TODO Create a list record train data
FOR each text of texts:
    TRAIN_DATA.append(train_data)
END FOR
TODO Save the TRAIN_DATA to local
```

Generate and save TRAIN_DATA

4.2.4 Create Machine Learning-based NER Model

In section 3.3.3, we described a method for training a machine learning-based NER model. Moreover, in this section, we describe how to implement it. First, we need two already labelled input sets, the training and validation sets. We split the “TRAIN_DATA” obtained in section 4.2.3 into an 8:2 ratio as required in the method. 80% of them for training records and 20% for validation records.

```
length = int(len(TRAIN_DATA) * 0.8)
train_data = TRAIN_DATA[:length]
valid_data = TRAIN_DATA[length:]
```

The training process then requires a training set and a validation set in binary format, as required by SpaCy. We design a transformed binary file (.spacy) method based on SpaCy's API.

```
FUNCTION generate_spacy_bin(data_set, output_path):
    TODO Create an empty English language model and a binary Doc instance
    FOR each data of the data set:
        TODO Use the model to process the data’s text and create an entity list
        FOR each entity of the data’s annotation:
            TODO Create Span instance by entity’s start and end position and label
            IF span exists:
                TODO Add the span to the entity list
            END IF
        END FOR
        TODO Update the Doc’s entities with the entity list
        TODO Add the Doc to the binary Doc
    END FOR
    TODO Save the binary Doc to local
```

We then apply this method to convert the training and validation sets into binary format and store them.

```
NLPHelper.generate_spacy_bin(train_data, train_spacy_path)
NLPHelper.generate_spacy_bin(valid_data, valid_spacy_path)
```

Finally, SpaCy requires us to have a configuration file to assist in training, which can be generated by overriding the base configuration provided by SpaCy (*Training Pipelines & Models · spaCy Usage Documentation*, no date). This can be achieved by changing the base configuration file. We assign the binary path we have just obtained to its "train" and "dev" attributes.

```
[paths]
```

```
2. train = train_spacy_path
3. dev = valid_spacy_path
```

Then we execute the following command to generate a custom configuration file.

```
1. python -m spacy init fill-config base_config.cfg custom_config.cfg
```

This leaves us with the final step in the machine learning-based NER model training, and we continue to call the commands provided by SpaCy to perform the training process. The parameters we need to specify are the name of the custom configuration file (custom_config.cfg), the location where we want to save it (model_path), the training data path (train_spacy_path) and the validation data path (valid_spacy_path).

```
1. python -m spacy train custom_config.cfg --output model_path --paths.train train_spacy_path
   --paths.dev valid_spacy_path
```

The entire process of training a machine learning-based NER model finished. Section 4.2.3 explained that since we have two sets of training and validation data, this project will also generate two machine learning-based NER models. Like the rule-based NER models, we will also compare them and describe the results in Section 5.

```
============================= Training pipeline =============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     49.96    0.00    0.00    0.00    0.00
  0     200        220.14   4235.13   73.12   74.33   71.96    0.73
  0     400        258.62   2525.23   74.11   76.10   72.22    0.74
  0     600        487.55   2785.10   78.39   82.37   74.77    0.78
  0     800        849.03   3315.41   76.64   76.66   76.61    0.77
  1    1000        539.51   3181.93   79.03   81.38   76.82    0.79
  1    1200        693.34   3580.39   78.22   78.91   77.53    0.78
  2    1400        878.03   4171.04   80.04   81.30   78.82    0.80
  2    1600       1270.94   4325.03   81.48   81.75   81.21    0.81
  3    1800       1542.47   4607.68   80.94   81.36   80.51    0.81
  4    2000       1787.08   4845.59   81.47   82.69   80.28    0.81
  6    2200       3007.46   5259.12   81.62   82.88   80.39    0.82
  7    2400       2803.61   5359.57   77.89   78.27   77.51    0.78
  9    2600       3183.01   5040.87   81.58   82.58   80.61    0.82
 10    2800       3035.13   4456.77   80.98   81.12   80.85    0.81
 12    3000       3043.26   3995.21   80.13   80.60   79.67    0.80
 13    3200       2823.52   3538.46   81.19   83.31   79.18    0.81
 15    3400       2708.71   3251.66   80.53   81.29   79.79    0.81
 16    3600       2952.04   3157.43   80.74   81.55   79.94    0.81
 18    3800       2706.53   2780.14   79.53   80.19   78.88    0.80
```

The first machine learning-based NER model

```
============================= Training pipeline =============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     51.74    0.00    0.00    0.00    0.00
  0     200        215.51   3889.40   73.65   75.60   71.80    0.74
  0     400        178.63   2094.76   77.80   79.04   76.60    0.78
  0     600        406.03   2311.91   77.79   78.01   77.56    0.78
  0     800        273.13   2553.18   78.80   79.43   78.18    0.79
  1    1000        419.68   2669.03   80.76   80.89   80.64    0.81
  1    1200        708.95   2874.23   81.66   83.28   80.10    0.82
  2    1400        516.83   3151.05   81.56   81.24   81.88    0.82
  2    1600        734.24   3488.35   82.14   81.63   82.65    0.82
  3    1800        997.20   3675.25   81.35   81.41   81.28    0.81
  4    2000       1130.47   3855.76   83.18   83.44   82.91    0.83
  6    2200       1319.68   4084.06   83.20   82.82   83.58    0.83
  7    2400       1636.17   4439.47   81.96   81.76   82.16    0.82
  9    2600       1667.35   4118.43   82.99   82.70   83.28    0.83
 10    2800       1631.65   3879.30   81.82   81.22   82.42    0.82
 12    3000       1567.50   3469.46   82.32   82.18   82.47    0.82
 13    3200       2486.77   3510.38   82.93   82.92   82.94    0.83
 15    3400       1557.99   3129.20   82.19   82.14   82.25    0.82
 16    3600       1613.07   3074.71   83.11   82.91   83.32    0.83
 18    3800       1606.17   2883.96   82.86   82.24   83.48    0.83
```

The second machine learning-based NER model

4.2.5 Using Models to Identify Entities in Text

The concrete implementation of model recognition is simple, and we have used it several times in the previous sections. All that is needed is to load the model we want to use and process the text to get the entities in the text.

```
1. nlp = spacy.load(model_path)
2. doc = nlp(text)
3. entities = doc.ents
```

In section 3.3.4, we introduced two rules for identifying locations in the text. One is to locate the location name directly, and the other is to identify the "direction of location". The first is more accurate when the text describes a specific location, such as "Auckland Road". Because the sentence itself describes Auckland Road, it is an exact location rather than a relative one. However, the second approach is more accurate when the text describes a relative location, such as "4 miles north of Auckland Road", because the location described is a specific direction from the identified location. After looking at the pattern of locality data, we found that the earlier the location name appears in the sentence, the more it matches the

location described in the sentence.

Therefore we examine the list of entities we have obtained. First, we need to determine whether the location entity describes a specific location or a relative location by finding out if the location entity has a format that matches the "direction of location". If we find a description around the current location entity in this format, we determine that it describes a relative location. Conversely, we determine that it describes a specific location (or a relative location that does not satisfy the "direction of location" format, but we ignore this rare case).

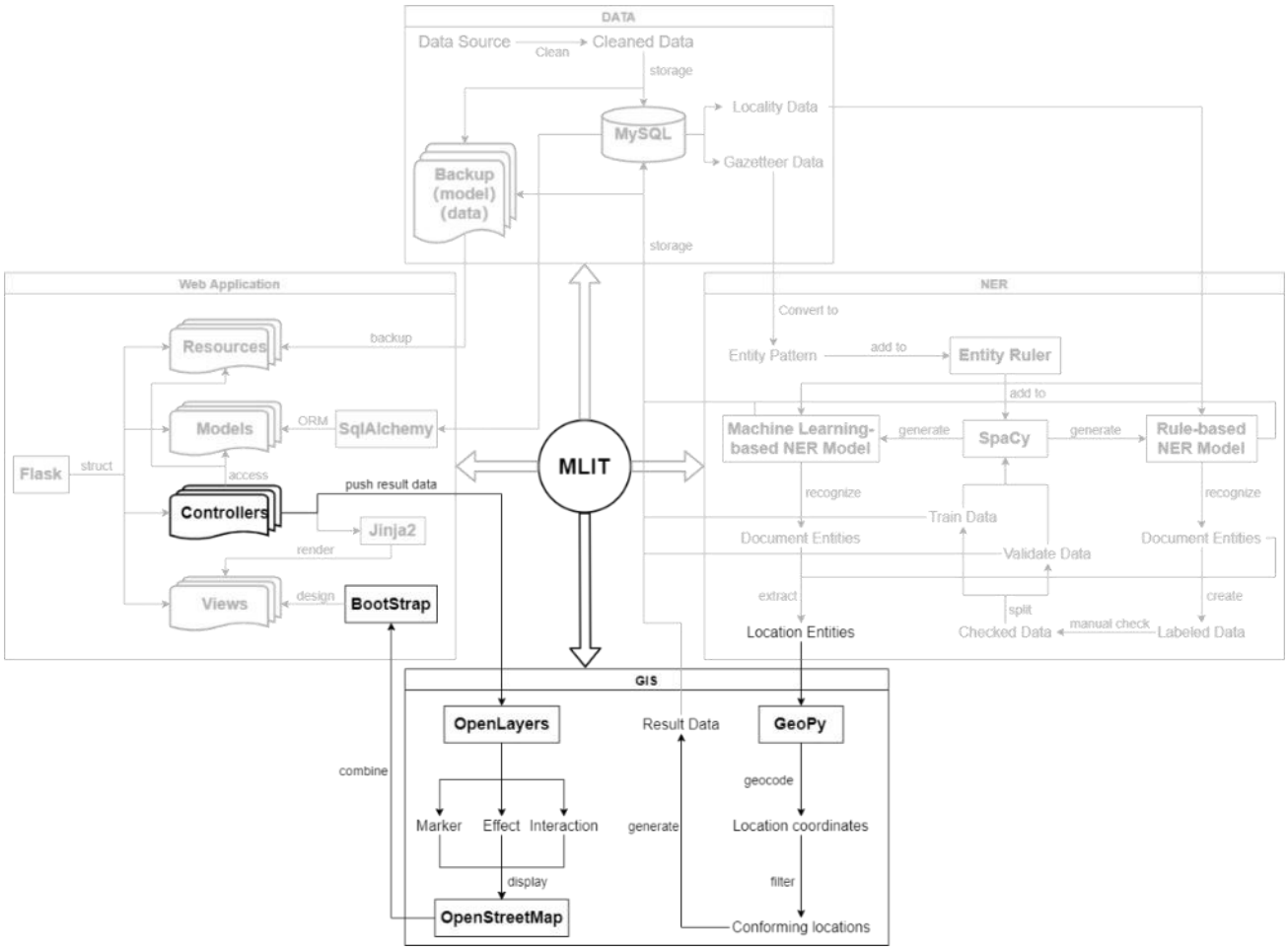
```
TODO Create two lists to record relative and specific location entities
FOR each entity of entities:
    IF The current entity is a location:
        IF The current entity is the first entity of entities:
            TODO The entity is not a relative one because it cannot match the format
            TODO Add the entity to the specific location list
        ELSE:
            IF The previous entity matches the two conditions:
                TODO Add the entity to the relative location list
            ELSE:
                TODO Add the entity to the specific location list
            END IF
        END IF
    END IF
END FOR
```

In the above method, we will check if the text satisfies the following two conditions when finding a location entity in the text.

- The previous entity is one word away from the target entity to determine if the phrase satisfies the "word of location" form.
- The previous entity is marked as "DIR" to determine if the phrase satisfies the form "direction of location".

If both of these conditions are met, we consider the location is relative because it satisfies the "direction of location" format. Then we record them in the relative location list. Conversely, we consider the location is not relative and record it in the list of specific locations. Finally, we return the relative and specific location lists and use them for subsequent GIS processing.

4.3 GIS



GIS

4.3.1 Geocoding

Section 4.2.5 obtained the list of relative and specific locations extracted from the text, which means that the project has progressed from NER to GIS.

The next step was to select the target location from the list of relative and specific locations. We first need to determine if the current text describes a relative or specific location based on the state of both the relative and specific location lists.

```
IF The relative location list exists item:
    TODO The text describes a relative position
ELSE:
    TODO The text describes a specific position
```

Determine what the current text describes

We then find the target location in the selected list. The target location needs to satisfy two conditions.

- It can be successfully geocoded by GeoPy.
- It belongs to New Zealand.

We set a flag bit and work with the information provided by the geocoding to find and record the target location.

We chose Nominatim's API to query feature data that matches the OSM map (*Overview - Nominatim Documentation*, no date), so we needed to instantiate a Nominatim type geolocator and override its parameters as Nominatim's default user_agent violates Nominatim's usage policy and can lead to HTTP errors (*Nominatim Usage Policy*, no date).

We have thousands of location descriptions to test, and each location's coordinate query takes time to process the service response. Therefore we need to use RateLimiter to generate the geocode instance when handling bulk data, which can gracefully handle error responses and add latency when needed (*Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation*, no date b).

```
1. geolocator = Nominatim(user_agent="mlit")
2. geocode = RateLimiter(geolocator.geocode, min_delay_seconds=1)
3. locations = geocode(location_name, exactly_one=False)
```

Geocoding location

Since there is a possibility of renaming places, for example, London in both the UK and the US, therefore we need to set the exact_one parameter to False when using geocode to resolve location coordinates so that it returns a list of locations with all the location information for the query location name.

Each location has a raw attribute in the list of locations to get all the information about it. For example, if we look up the location "Garden of Eden Ice Plateau", we can get the following information.

```
1. {'place_id': 257997152, 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://os
   m.org/copyright', 'osm_type': 'relation', 'osm_id': 9690373, 'boundingbox': ['-43.3345227',
    '-43.3042201', '170.6092522', '170.7241012'], 'lat': '-43.31936345', 'lon': '170.71358226
   059598', 'display_name': 'Garden of Eden Ice Plateau, Westland District, West Coast, New Z
   ealand / Aotearoa', 'class': 'natural', 'type': 'glacier', 'importance': 0.7}
```

Geocoded information example

We can observe and extract some important attributes and information, including "place_id", "importance", "boundingbox", "lat", "lon" and "display_name". GeoPy provides a mechanism to sort all geographic information in descending order by the "importance" attribute value, the more suitable locations come first. The "lat" and "lon" attribute values help us locate the location's coordinates. We can also determine the identification accuracy by comparing the range of the "boundingbox" with the coordinates of the location being measured. The "display_name" attribute value describes the region in which the location is located and can therefore help us filter for locations belonging to New Zealand. Using London as an example, we can get the following list of locations with the attributes we are interested in.

```
1. {'place_id': 326508358} {'boundingbox': ['51.2867601', '51.6918741', '-0.5103751', '0.33401
   55']} {'lat': '51.5073219'} {'lon': '-0.1276474'} {'display_name': 'London, Greater London,
    England, United Kingdom'} {'importance': 0.9407827616237295}
```

```
2. {'place_id': 256872490} {'boundingbox': ['51.5068696', '51.5233122', '-0.1138211', '-0.0727
   493']} {'lat': '51.5156177'} {'lon': '-0.0919983'} {'display_name': 'City of London, Great
   er London, England, United Kingdom'} {'importance': 0.6865111547516773}
3. {'place_id': 310657642} {'boundingbox': ['42.8236747', '43.1436747', '-81.4096068', '-81.08
   96068']} {'lat': '42.9836747'} {'lon': '-81.2496068'} {'display_name': 'London, Southweste
   rn Ontario, Ontario, N6A 1G4, Canada'} {'importance': 0.6759177286656098}
4. {'place_id': 257760707} {'boundingbox': ['42.8245667', '43.0730461', '-81.3906556', '-81.10
   70784']} {'lat': '42.9537654'} {'lon': '-81.2291529'} {'display_name': 'London, Southweste
   rn Ontario, Ontario, Canada'} {'importance': 0.6759177286656098}
5. {'place_id': 256890671} {'boundingbox': ['37.079759', '37.15226', '-84.126262', '-84.035957
   ']} {'lat': '37.1289771'} {'lon': '-84.0832646'} {'display_name': 'London, Laurel County,
   Kentucky, 40741, United States'} {'importance': 0.5512922449371916}
```

Information geocoded by London

Below we illustrate the specific implementation of this project for filtering, extracting and validating target locations. When we find the first location in the list of locations located in New Zealand, it is the most appropriate location for the query location that meets the geographical range criteria, and we select it as a result and add it to the list of results.

```
1. IF The location entity successfully geocoded:
2.     FOR each location of geocoded locations:
3.         IF location belongs to New Zealand:
4.             TODO location is the target location, add it to result list
5.         END IF
6.     END FOR
7. END IF
```

Find the target location

We will verify the target location when we get it. Section 4.1.1 explains that for each description, we can get the East and North of it, and section 3.4.2 explains that we will use the coordinate conversion system provided on the website to convert it to WGS84 latitude and longitude. We compare this latitude and longitude to the range of the boundingbox to determine if the location we find matches the area described in the text.

```
1. IF The target location’s coordinate is in the boundingbox area:
2.     TODO The target location is correct
```

Verify the target location

4.3.2 Geographic Information Visualization

We have obtained a list of parsed geographic information about the location through geocoding. The next step is to visualise the geographic information and display it on a web

map. 3.5.1 describes how we will use the API provided by OpenLayers to build the OSM, add map effects and implement map interactions. They correspond to the three core development parts: initialising the map and its components, adding the resultant data visualisation layers to the map, developing the map's pan and zoom functionality, and click interaction pop-ups.

The project provides a method for initialising the map, which determines the initial position of the map display via the centre and initZoom parameters, the zoom range of the map via the maxZoom and minZoom parameters and finally initialises a map with an OSM layer.

```
FUNCTION initMap(center, initZoom, maxZoom, minZoom) {
    TODO Initialising map by properties
    RETURN map
}
```

Initialising the map

Once the map has been created, we obtain the geocoded data via an HTTP request and use it to generate coordinate point instances to be displayed on the map. This is achieved by first fetching the result_locations completed in the previous section via a get request, then converting the latitude and longitude into coordinates via the OpenLayers proj.fromLonLat() method and assigning them to the geometry property of the Feature entity. Each Feature entity is then assigned an icon style via the setStyle() method. Finally, all the features are placed on a vector layer and this layer is added to the map using the addLayer() method. This completes our task of adding the result data to the map in the form of icons.

```
FUNCTION initFeature(dataType) {
    TODO Use get method request for result locations
    FOR each location of result locations:
        TODO Use location’s coordinate to generate point’s feature
        TODO Use image to generate point’s style
        pointFeatures.push(pointFeature);
    END FOR
    TODO Create a vector layer and use pointFeatures as its resource
    map.addLayer(vectorLayer);
}
```

Initialising the feature

With the above steps, we have built up the basic functionality of the geo-information visualisation, which can display the location of our parsed completion on the map. However, as an information visualisation system, it lacks the necessary descriptive information and dynamic interaction effects. So we continue to add these elements to the coordinate points.

The first thing is about adding descriptive information. We can display some helpful information obtained by geocode, such as latitude, longitude and address, together with the description of the parsed location in a panel next to the icon. It is achieved by adding custom properties and assigning their values when generating the Feature entity.

```
let iconFeature = new ol.Feature({
    TODO Add custom properties
});
```

Add custom properties to feature entities

The next task is to display them in the panel, which we do in the following way. First, add an overlay layer to the map via the “addOverlay()” method and display the location information panel on this layer.

```
FUNCTION initOverlay() {
    TODO Create an overlay
    map.addOverlay(overlay);
    RETURN overlay;
}
```

Initialising the overlay

We then add the panel to the overlay layer. We extract the information stored in the feature by using the "getProperties()" method and move the overlay to the position of the point so that the panel can be displayed next to it by using the "setPosition()" method. We then design a panel in HTML and place the information we need to display in it. Finally, the "popover('show')" method is called to display the panel.

```
TODO Use point.getProperties() to obtain point’s properties
overlay.setPosition(coordinate);
TODO Create HTML element as an information panel
$(popup).popover('show');
```

Create information panel and display next to the feature

This completes the first half of the process of adding description information. Once we have the description panel, we need an interactive function to trigger its display and a dynamic effect for this process. The approach in this project is to use mouse clicks on the icon points to trigger the panel's display and add a flight animation when switching between clicks on different icon points. The following section describes the development of dynamic interaction effects.

We initialise an "onOverlayClick()" method to monitor the map click event. When we click on the map, we use the "getEventPixel()" method to get the pixel position of the click. This pixel location is then queried via the "forEachFeatureAtPixel()" method to see if this pixel location was clicked on a feature and retrieved the feature entity.

```
FUNCTION onOverlayClick(overlay) {
    TODO Add event to map onClick ({
        TODO Use “getEventPixel()” to obtain the clicked pixel point
        TODO Use “forEachFeatureAtPixel()” to obtain the point on the pixel point
    });
}
```

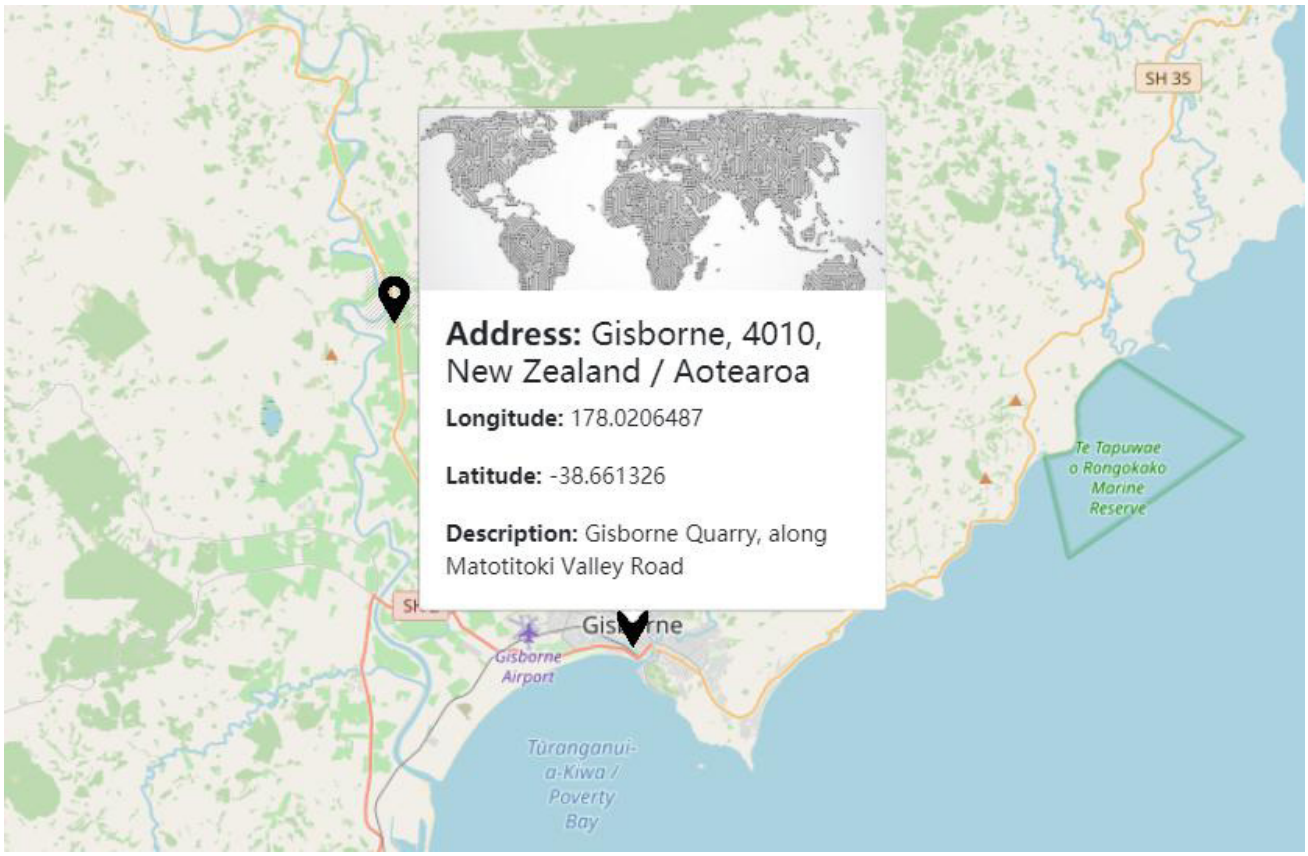
Create event listener when click on overlay

We then design an animation effect for the flight after the click. The core idea is to control the smooth deflation and movement of the map by calling back the animate() method at fixed intervals.

```
FUNCTION flyTo(location, done) {
    TODO Create method to repeat animation callback();
    TODO Create method to excuate animation animate(
        callback();
    );
}
```

Create animation effect

Finally, we combine the above describing the information panel and the dynamic interaction development to implement the geographic information visualisation part of the work.



Final result

4.3.3 Data Visualization

The use of Echarts for data visualisation of the results in this project is described in section 3.5.2. We designed the data presentation for each model result with six dimensions

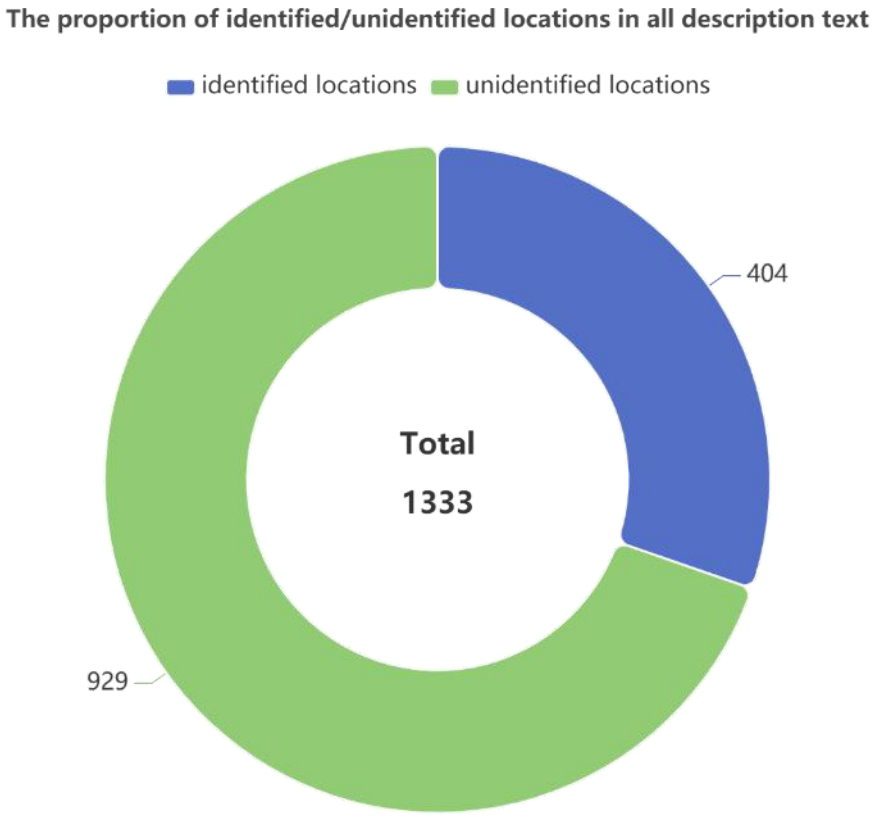
- the proportion of identified/unidentified locations in all description text
- the proportion of relative/specific locations in identified locations
- the proportion of successfully/unsuccessfully geocoded locations in identified locations
- the proportion of relative/specific locations in successfully geocoded locations
- the proportion of correct/incorrect coordinates in successfully geocoded locations
- the proportion of relative/specific locations in correct coordinates

We have already stored this data in the implementation logic in section 4.3.1, so in this section we describe the process for implementing the visualisation. The main way to create and display a visual chart is as follows. We initialise the chart using the “init()” method and set the properties in its options using the “setOption()” method..

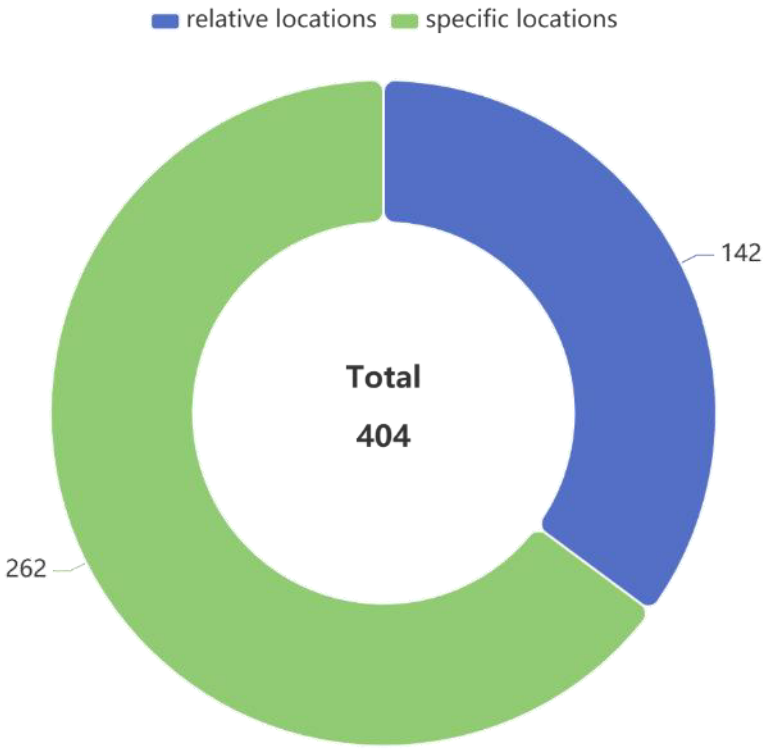
```
1. let myChart = echarts.init(chartDom);
2. let option = {...};
3. myChart.setOption(option);
```

As an example, we show the final visualisation of the results of the first Rule-based NER

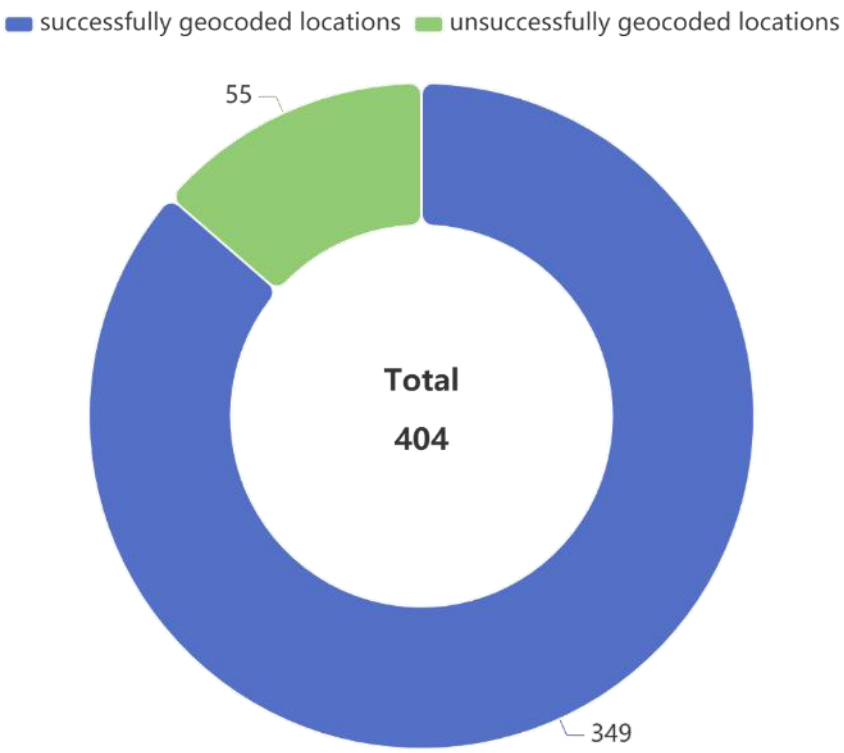
model.


The proportion of identified/unidentified locations in all description text
identified locations
unidentified locations
404
Total
1333
929

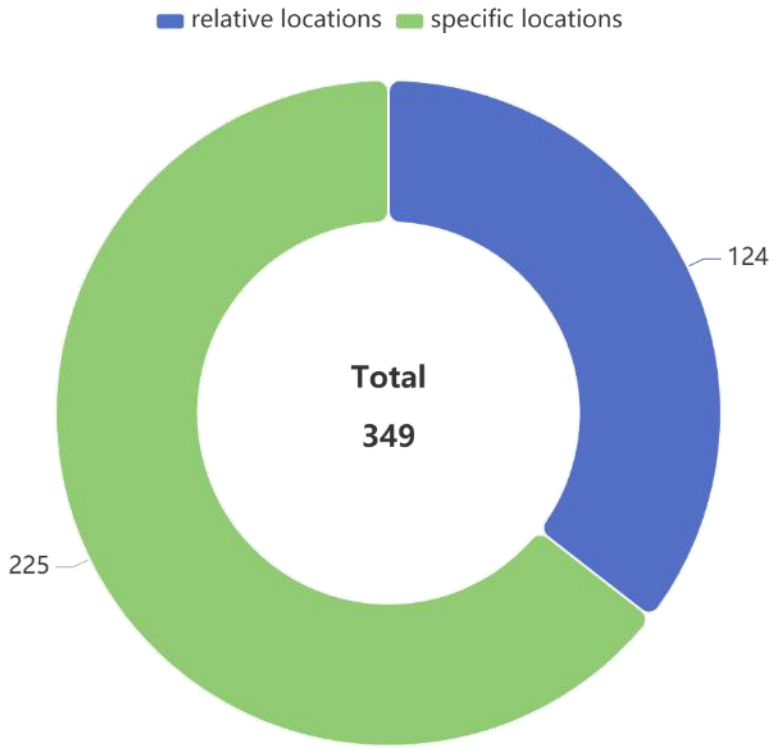
The proportion of relative/specific locations in identified locations



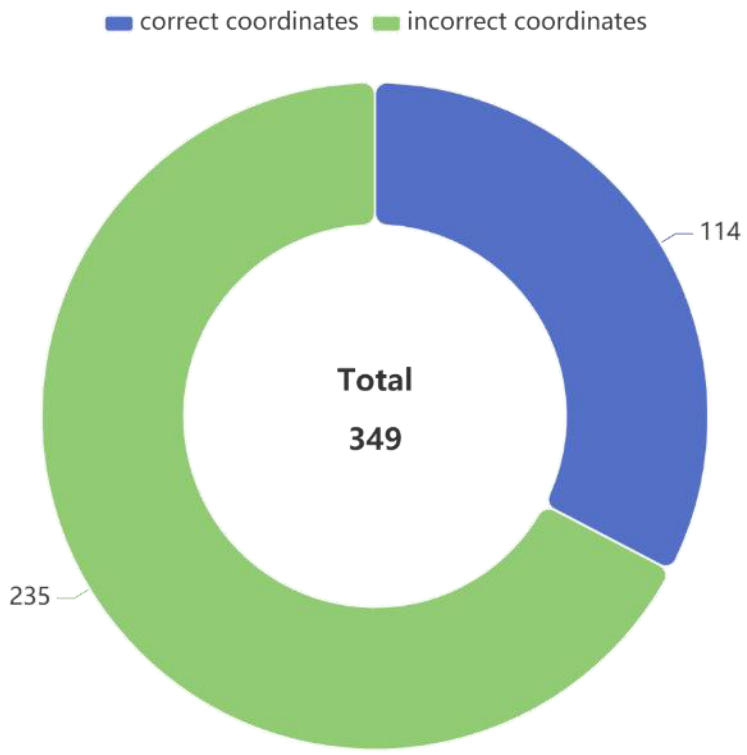
The proportion of successfully/unsuccessfully geocoded locations in identified locations

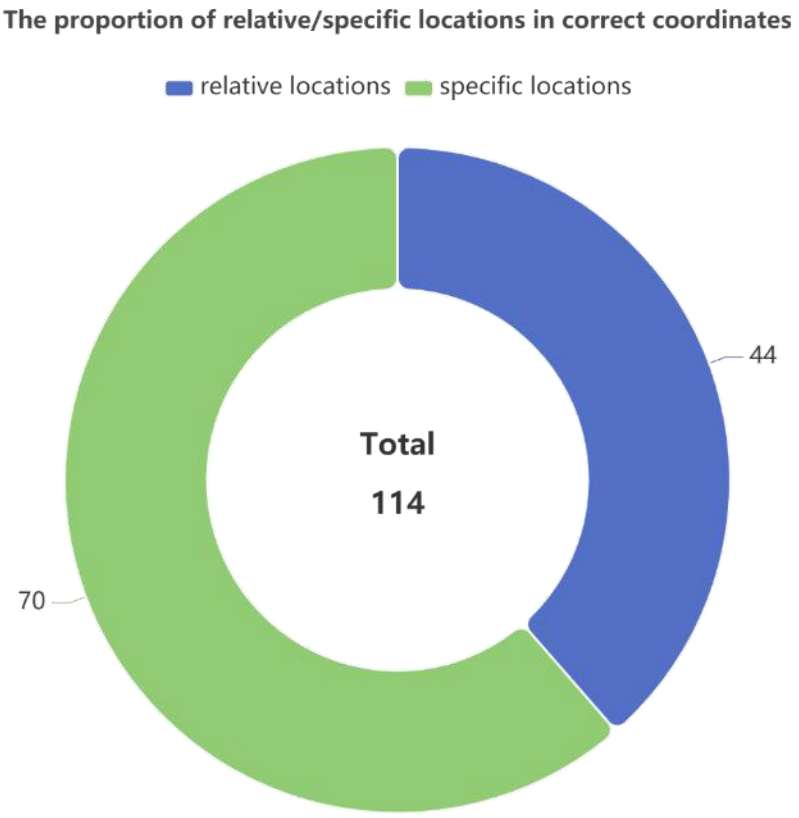


The proportion of relative/specific locations in successfully geocoded locations

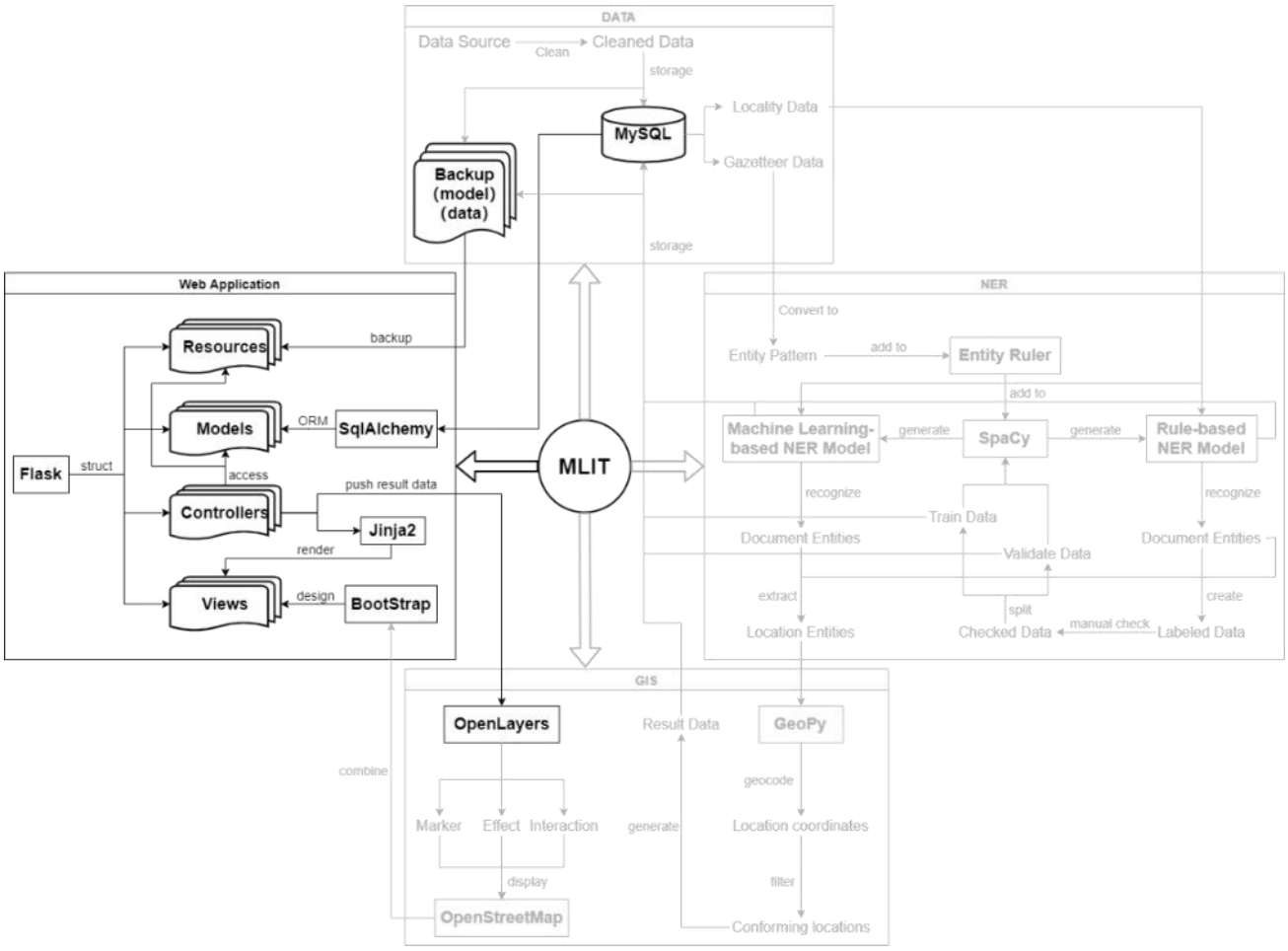


The proportion of correct/incorrect locations in successfully geocoded locations





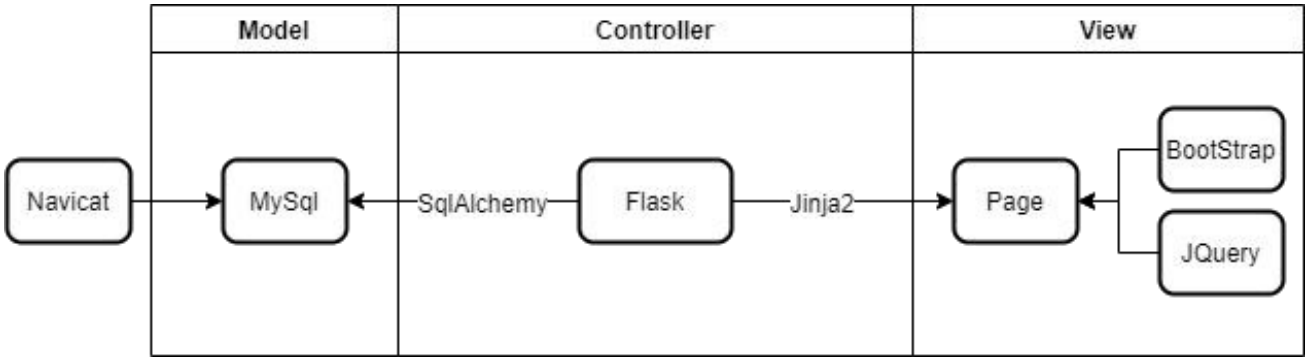
4.4 Web Application



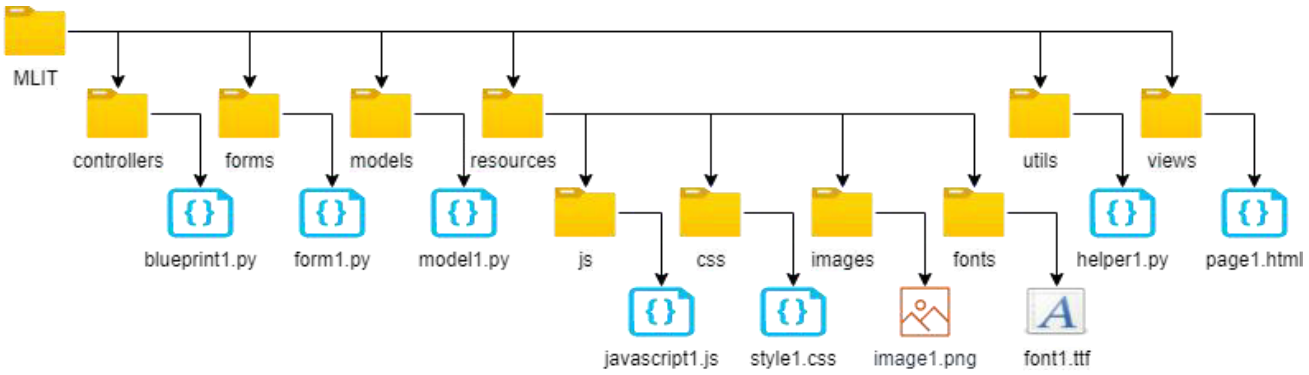
Web Application

4.4.1 Structure Design

The project is described in section 3.1.4 as using the MVC design pattern to develop web applications that can guarantee the flexibility and maintainability of the software. We have designed the technical structure as follows.



Technical Flow



Resource Flow

4.4.2 Web API

The primary process of a web application is that the user sends an HTTP request to the server using the client, the server accepts the request, calls the data layer resources for logical processing, then returns the response content to the client, and finally, the client logically processes the response content and renders it to the user. This section shows the functionality included in the project in the form of the Web API.

Web API	Parameter	Return	Description
'/'		Index page	Get index page content.
'/train/model/<model_type>/'	model_type	Trained model	Get the trained model of the selected type.
'/train/data/<data_type>/'	data_type	Trained data	Get the trained data of the selected type.
'/geocode/<model_type>/'	model_type	Result data	Identify and geocode the location in the text using the selected model. Get the location result data.
'/display/<data_type>/'	data_type	Result data	Displaying location and statistics charts from the result data.

Web API

4.4.3 User Interface

Train
Rule-based
Train Model
Rule-based
Train Data
Update
Rule-based
Update Data
Display
Rule-based
Display Data
Data Statistic

Train
Rule-based
Train Model
Rule-based
Train Data
Update
Rule-based
Update Data
Display
Rule-based
Display Data
Data Statistic
The proportion of identified/unidentified locations in all description text
identified locations
unidentified locations
404
Total
1333
929

Address: Plimmerton, Porirua, Porirua City, Wellington, 5026, New Zealand / Aotearoa
Longitude: 174.8652978
Latitude: -41.0819811
Description: Walker Estate, 2.4 km north of Plimmerton, on main north highway.
South Island / Te Waipounamu
Train
Rule-based
Train Model
Rule-based
Train Data
Update
Rule-based
Update Data
Display
Rule-based
Display Data
Data Statistic
The proportion of identified/unidentified locations in all description text
identified locations
unidentified locations
404
Total
1333
929

UI

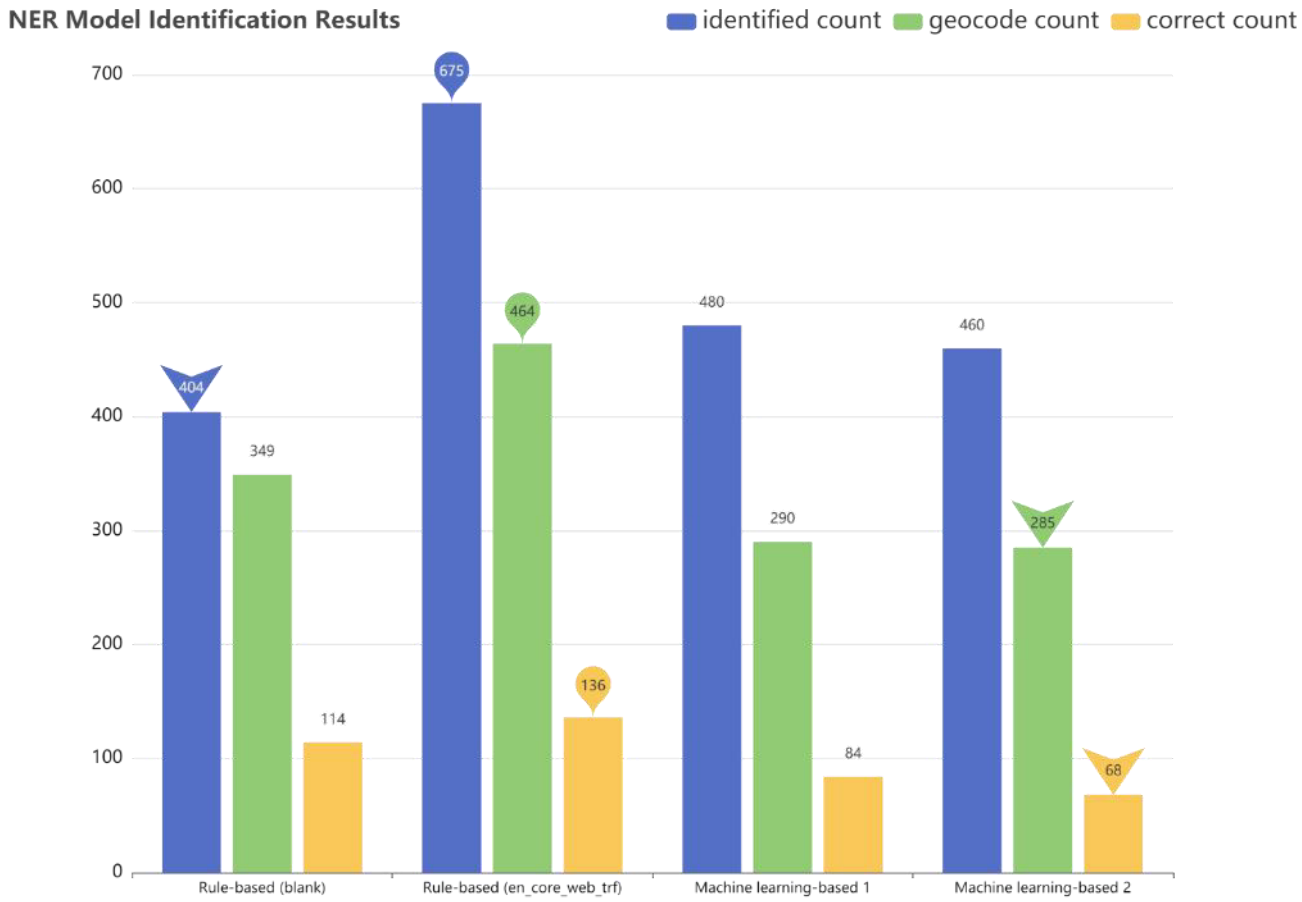
5. Results And Evaluation

5.1 NER

In this part, we compare and contrast the identification outcomes of the four NER models used in the research, as well as assess their accuracy.

5.1.1 NER Model Identification Results

Model name	Total count	Identified count	Geocode count	Correct count
Rule-based (blank)	1333	404	349	114
Rule-based (en_core_web_trf)	1333	675	464	136
Machine learning-based 1	1333	480	290	84
Machine learning-based 2	1333	460	285	68



NER Model Identification Results

These are the results of the identification statistics for the four models and the concept of the relevant metrics.

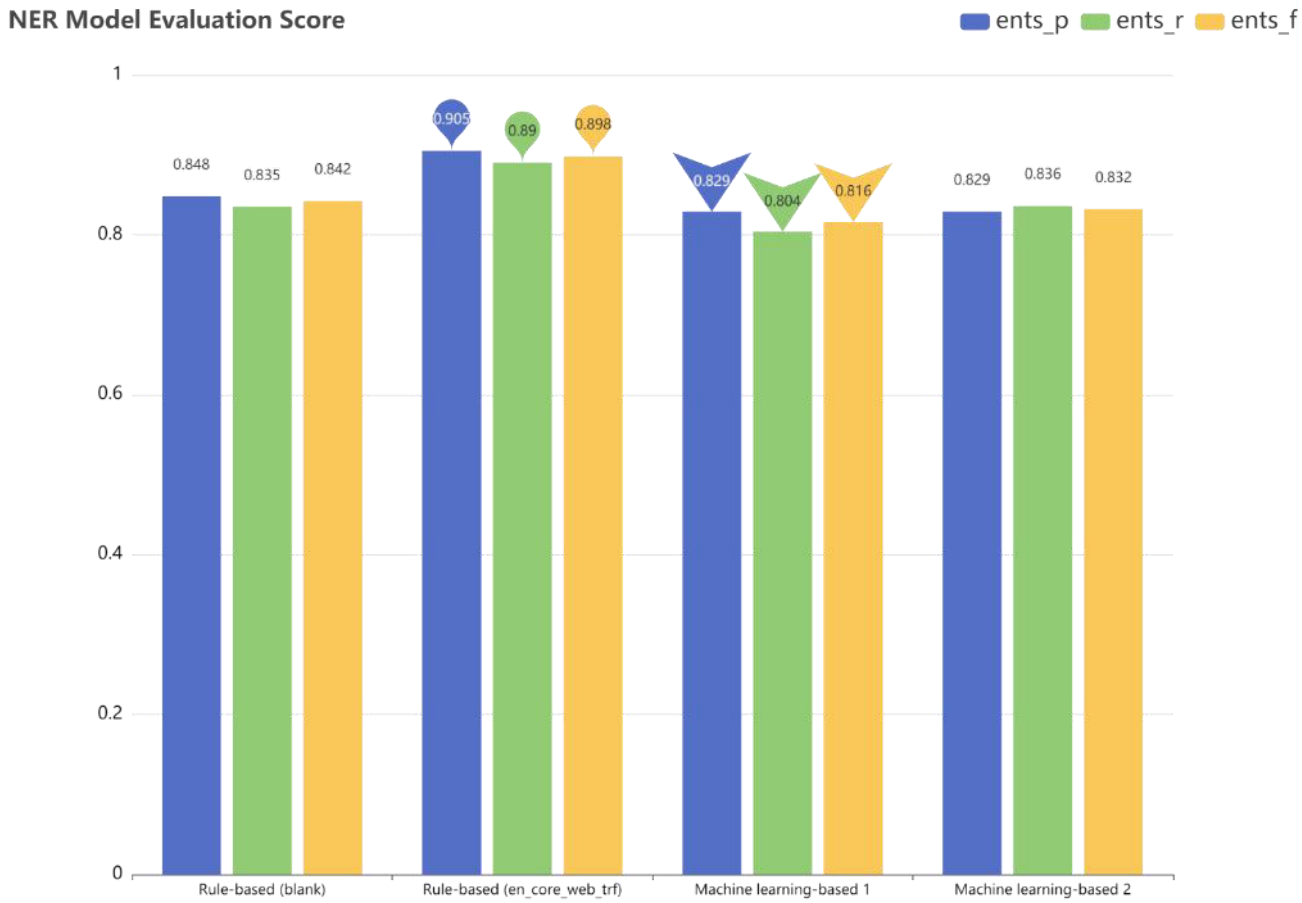
- Total count: the total number of locality data descriptions to be measured.
- Identified count: the number of locality data that can be identified.
- Geocode count: the number of descriptions that can be geocoded after being identified.

- Correct count: the number of descriptions that are located within the geocoded location area.

From the statistical results, we can see that the rule-based (en_core_web_trf) model has the most three indicators, so we can consider it the best of the four models. Although the rule-based (blank) model has the least number of recognitions, it has the highest geocoding accuracy rate. 349 of the 404 recognition results can be successfully geocoded. The indicators of the two machine learning-based models are very close. Although they have a higher number of recognitions than the rule-based (blank) model, their accuracy is not as excellent. Overall, the rule-based model outperforms the machine learning-based model in terms of recognition.

5.1.2 NER Model Evaluation

Model name	ents_p	ents_r	ents_f
Rule-based (blank)	0.848	0.835	0.842
Rule-based (en_core_web_trf)	0.905	0.89	0.898
Machine learning-based 1	0.829	0.804	0.816
Machine learning-based 2	0.829	0.836	0.832



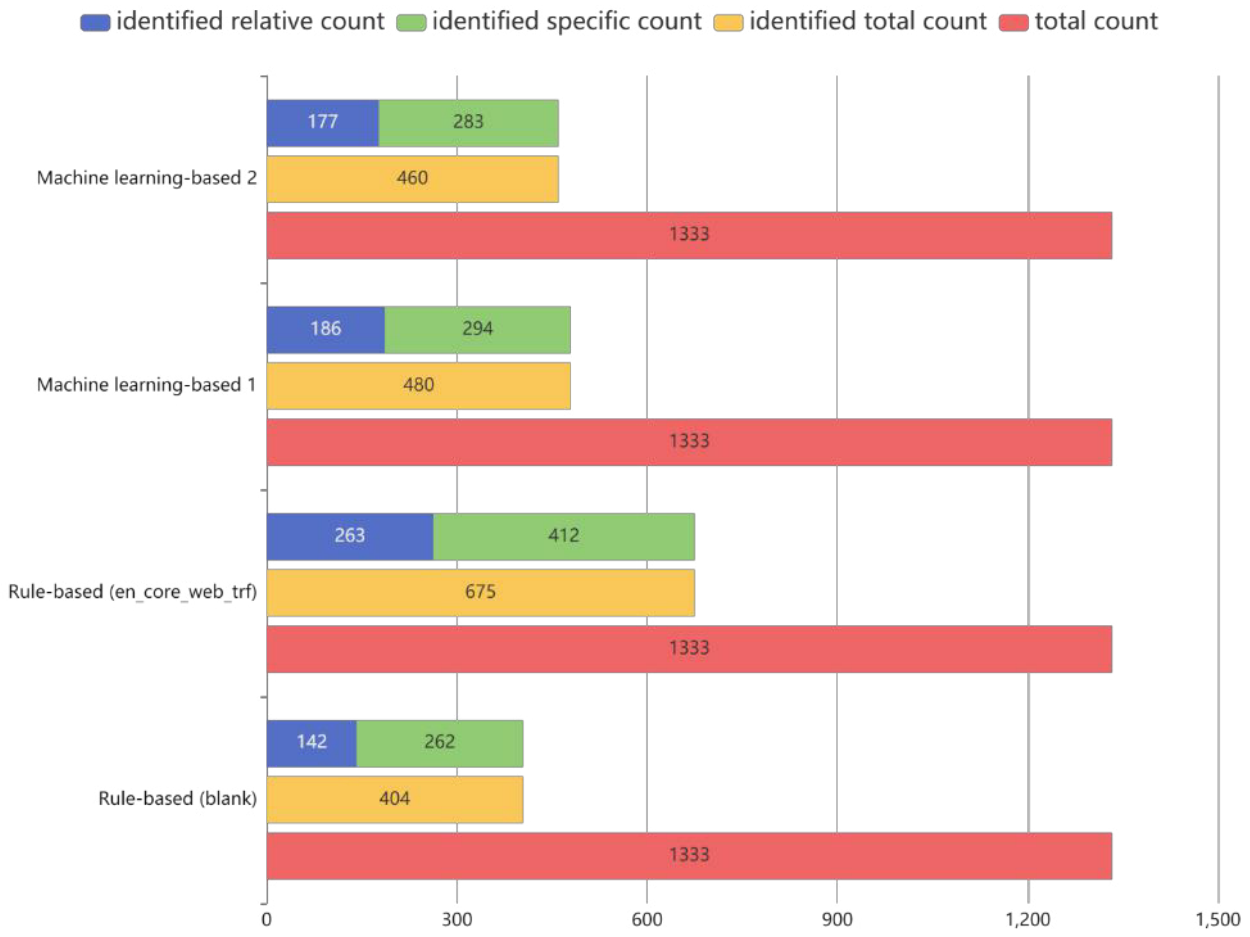
NER Model Evaluation Score

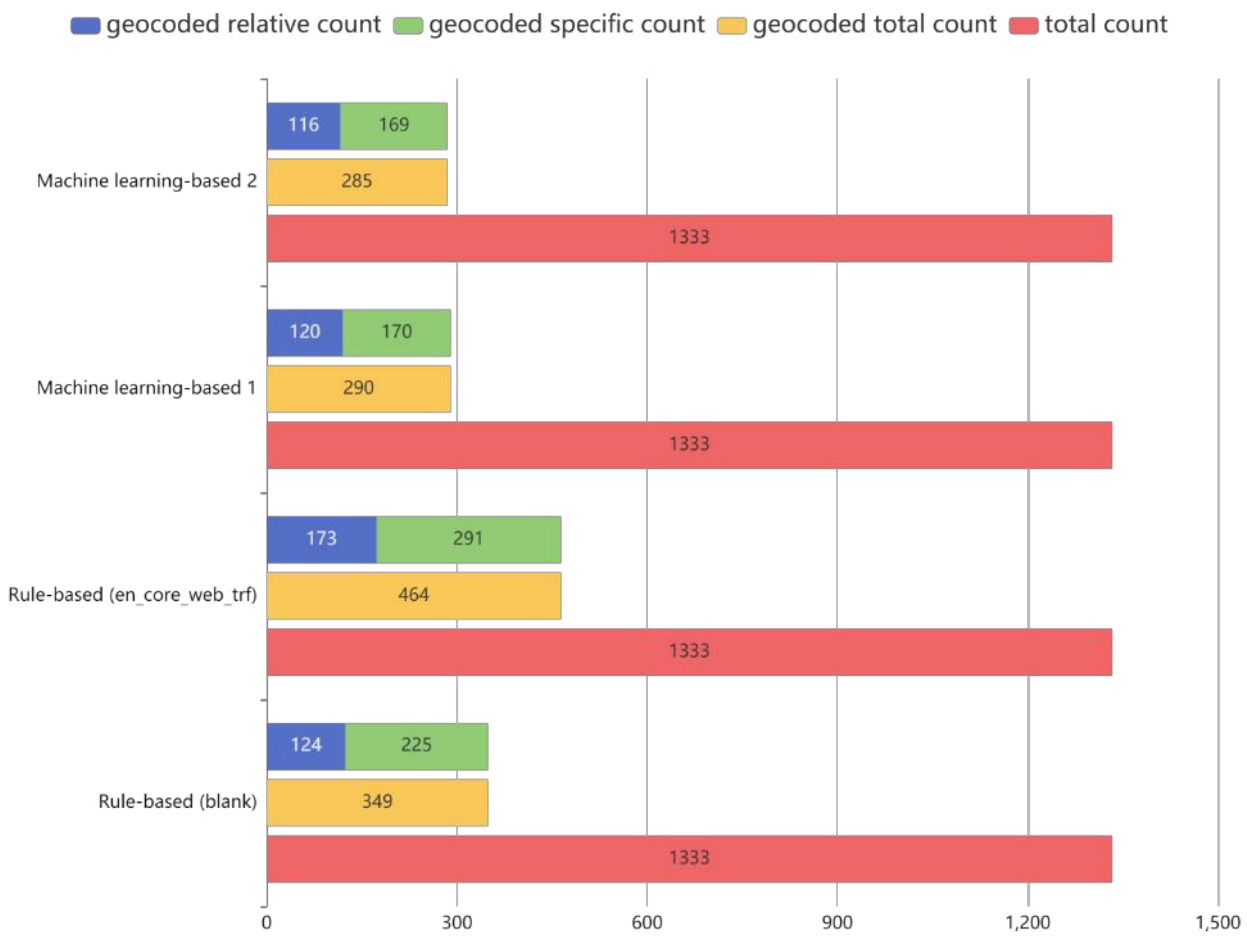
We introduce the indicators and methods of NER model evaluation in section 2.4. The above tables and graphs show the precision, recall and f1-score of the four models.

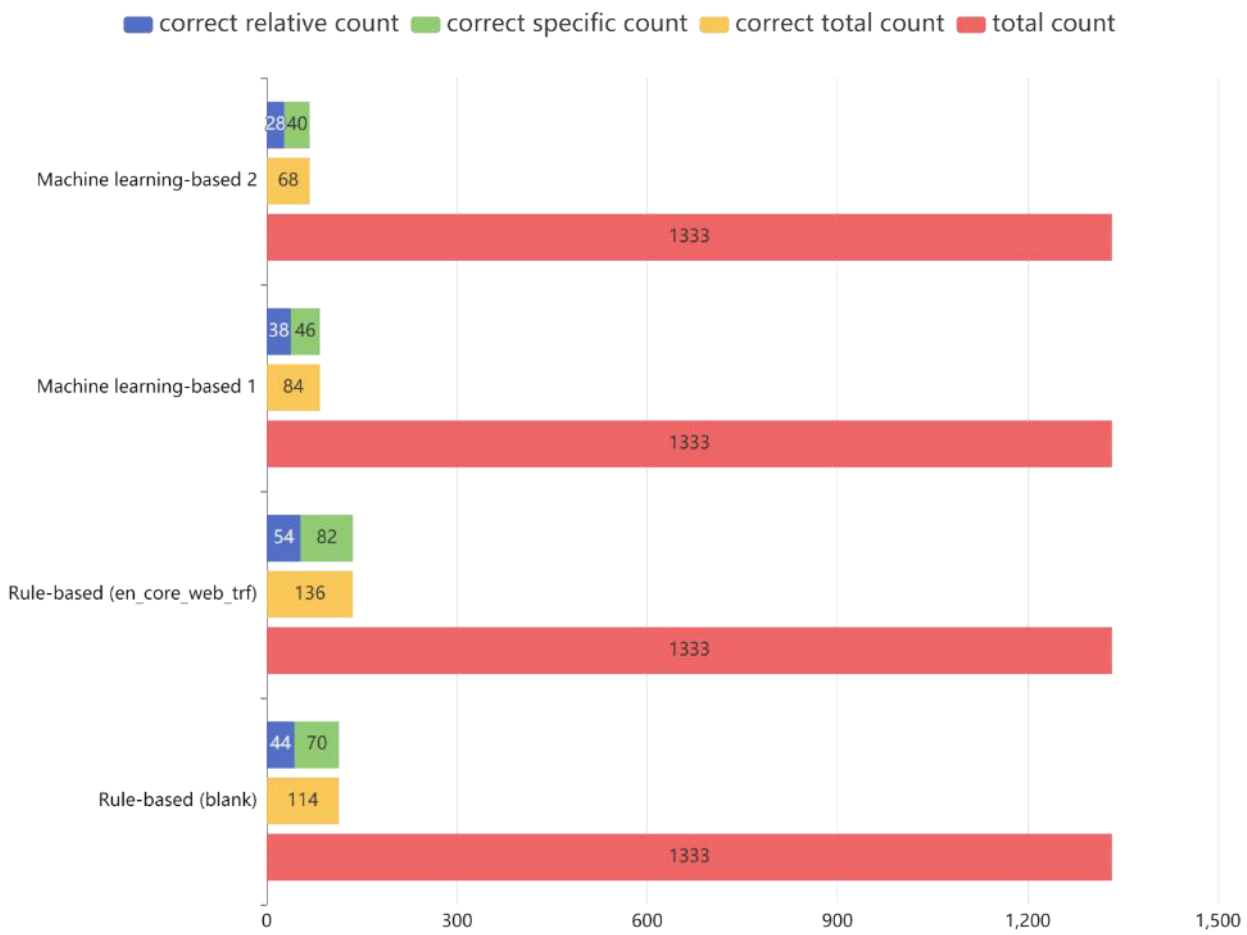
As we can see from the statistics, the rule-based (en_core_web_trf) model has the highest

number of all three metrics, followed by the rule-based (blank) model, the machine learning-based 2 model and the machine learning-based 1 model. The model metrics assessment findings are consistent with the identification results given in the preceding section, with the rule-based model outperforming the machine learning-based model in this project.

5.2 Data Layer




geocoded relative count
geocoded specific count
geocoded total count
total count
Machine learning-based 2
116
169
285
1333
Machine learning-based 1
120
170
290
1333
Rule-based (en_core_web_trf)
173
291
464
1333
Rule-based (blank)
124
225
349
1333
0
300
600
900
1,200
1,500


correct relative count
correct specific count
correct total count
total count
Machine learning-based 2
28
40
68
1333
Machine learning-based 1
38
46
84
1333
Rule-based (en_core_web_trf)
54
82
136
1333
Rule-based (blank)
44
70
114
1333
0
300
600
900
1,200
1,500

According to the graphical data above, the successful conversion rate of locality data is close to 50%, the successful conversion rate of geocoding is close to 30%, and the accurate rate of the coded location is only around 10%. This implies that the quality of the location data is poor, since there is a lot of redundant and incorrect information. The low percentage of correctly encoded locations, on the other hand, indicates a mismatch between the NZMG-based location data and the actual geographic coordinates in the locality data.

The data from another set of dimensions in the chart shows that the number of relative locations in the text descriptions is minor than specific locations. This suggests that the locality data is mainly written in a style that describes specific locations.

5.3 Web Application

The web application evaluation for this project is divided into two parts: software functionality and software quality.

The software for this project can train a rule-based and machine learning-based NER model, recognise and geocode descriptive text, and visualise the recognition results with geographic information and graphics. The research directions of the project were followed, and the requirements of the project were met.

The software quality of the project follows usability and maintainability. When the data in the interface is updated, the software uses Ajax asynchronous requests to load the page so that the interface is partially refreshed rather than redirected. It ensures the usability of the software and satisfies the user experience. The software uses different interfaces to request static elements and dynamic data for each page. The static interface returns the response of the rendered page, and the dynamic interface returns data in standard JSON format. Modular development makes the software easy to maintain and modify, and the coupling between functions is reduced, keeping the software maintainable.

6. Discussion

6.1 Reflection

6.1.1 Technical Reflection

This section will discuss the author's abilities and lessons learned throughout the endeavour. Reflection is a key component of project iteration, and the project's future progress can only be advanced by returning to the summaries at each milestone.

Early in the project, the author analysed and reflected on the gaps between personal abilities and project-specific skills, as planned learning and bridging these gaps was critical to the project's completion.

The NER portion of NLP is used since the project objective is to extract the address from the text, and SpaCy is chosen as the NER tool. As a result, the NER section focuses on understanding the fundamentals of NER, studying NER techniques, and becoming skilled in the use of SpaCy for NER.

Prior skills	Initial gap	Final gap
Data preprocessing involves the application of python in data science, such as pandas, numpy, and scipy	Low	Low
Use SpaCy to process NER tasks in a rule-based method, including token matching, phrase matching, and regular expression matching	High	Low
Use SpaCy to process NER tasks in a machine learning-based method, including processing pipelines and configuring training models	High	Low

NER Skills Reflection

Despite having a strong foundation in Python programming, the author lacks prior understanding of NER and SpaCy development expertise. The author intends to devote two weeks to studying the SpaCy official course (*Advanced NLP with spaCy · A free online course*, no date) in order to master two approaches for handling NER jobs. Then, in the following week, the software development for the project will be completed using SpaCy's official documentation (*spaCy 101: Everything you need to know · spaCy Usage Documentation*, no date). Through methodical learning of the course on the internet, the author effectively overcame the early gaps and completed the development skill set necessary for the project as the project progressed.

After utilising SpaCy to extract the location information in the text, it must be geocoded using GeoPy. As a result, the geocode section focuses on mastering the use of geocoding in GeoPy.

Prior skills	Initial gap	Final gap
Basic knowledge of python programming and programming ability	None	None
Use geopy for geocoding of text addresses, including geocoders and distance calculations	High	None

Geocode Skills Reflection

Although the author was proficient in Python for programme logic, he lacked GeoPy programming skills. The author intends to study geocoding and methodically programme development in a week using the official GeoPy documentation (*Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation*, no date a). According to GeoPy's official documentation, the author learns to geocode using Nominatim and use it to parse place names.

We need to see the geographical location in a GIS once the geographical coordinates have been acquired using GeoPy. As a result, the GIS part focuses on learning how to use OpenLayers to edit OSM and create interactive features.

Prior skills	**Initial gap**	**Final gap**
Understand the basic elements of points, polygons, polygons, layers, etc. in OSM	None	None
Proficient in drawing of points, polylines, polygons, layers, etc. and their styles in the OpenLayers map library, as well as related animation and visualization rendering techniques	None	None

GIS Skills Reflection

Because the author has four years of GIS visualisation development expertise, he is extremely knowledgeable about GIS and data visualisation. The creation of GIS-related applications in the project is expected to take two weeks. And this aspect of the project ran well the entire time.

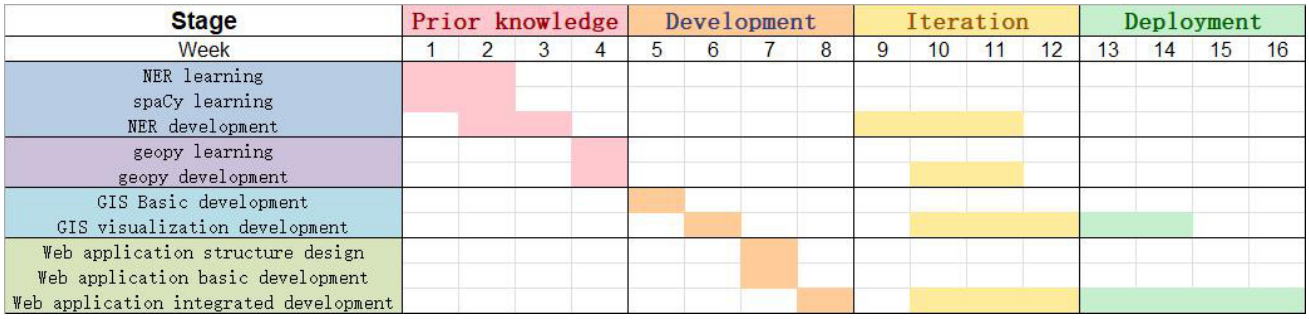
After all of the functional components have been created, they must be combined into a web application for presentation. As a result, the Web application section focuses on learning Flask web programming.

Prior skills	Initial gap	Final gap
Proficient in the basic capabilities of Web application development, including front-end technology, back-end technology, and basic database knowledge	None	None
Proficient in Web application development framework Flask and its MVC design pattern	None	None
Able to integrate GIS visualization functional components on the front-end, and integrate NLP and geocoding functional components on the back-end	None	None

Web application Skills Reflection

Because the author has four years of MVC design pattern software development expertise and two years of Flask web development experience, he is well familiar with their code standards and development principles. It is anticipated that the integrated development of the project's web application would take two weeks. The project's web application development strives to achieve the software's functionality as well as fulfil the software's quality requirements from start to end.

6.1.2 Time Reflection



Time Plan

The author utilises a Gantt chart to indicate the tasks and projected time of each phase of the project, which is expected to take four months. The project follows the design process for software research and development, the duration was correctly predicted, and the author performed the intended work on time and with a high degree of execution.

6.2 Future Work

6.2.1 Insufficient Technicality

The analysis and evaluation of the results in section 5 show that the project has much future work to remedy the current shortcomings.

Firstly, to ensure that the data is of high quality, there is a large amount of redundant and erroneous data in the NZ locality description data provided in the project that needs to be cleaned, and the irrelevant description text has a significant impact on the accuracy of the NER model. The NZMG-based location data in the locality data needed to be checked to ensure they were converted to WGS84 format and correct.

Secondly, the recognition rate of the text describing relative locations is not accurate enough for this project. Due to the diversity of relative location texts, we can use an infinite number of relative locations to describe the exact location, as there is an infinite number of relative locations for a single location. The existing methods cannot meet the ability to recognise all relative locations, so the recognition algorithm for relative locations needs to be updated in the future.

Finally, we need to optimise the machine learning-based NER model. Although the rule-based NER model outperformed the machine-learning-based NER model in this project, this project was for New Zealand locality data. For many other countries and regions, the rule-based NER model needs to constantly adjust its list of rules, which is undoubtedly a huge task and challenge, which is why the machine-learning-based NER model is necessary.

6.2.2 Inadequate Software Format

To show the software's capabilities to the user, this project is presently provided as a web application. A mature project application, on the other hand, may require numerous types of software to fulfil the demands of consumers utilising various end devices. For Windows

users, the project will also require a C/S based desktop programme, as well as an Android or iOS based application for those who prefer mobile devices.

7. Conclusion

This project describes a web-based application that uses named entities to identify location names described in natural language text in New Zealand locality data and subsequently maps them to coordinates and presents them in a GIS for data analysis.

This project satisfies the research's objectives and assesses the creation and execution of relevant techniques and instruments. We used SpaCy's NER technique to identify location entities in natural language text, GeoPy to geocode the real coordinates of the location entities, OpenLayers and OpenStreetMap to map the actual coordinates into the GIS, and Echarts to visualise the identification results. The entire process is then wrapped up in a Flask-based web application that users can interact with.

This project can be used as a fundamental solution to recognize locations described by natural language text. Furthermore, we can optimize the technical deficiencies of data quality, model recognition rate and algorithm accuracy. To expand the possibilities accessible to consumers, we can also create C/S-based and mobile applications. By resolving the above difficulties, the solution will become more exact and comprehensive.

Acknowledgment

Due to the covid-19 pandemic, the communication and guidance of the project are carried out online. I want to thank Professor Christopher Jones for leading me to overcome difficulties and ensure the smooth progress of the project.

Reference

Adams, B., McKenzie, G. and Gahegan, M. (2015) 'Frankenplace: interactive thematic mapping for ad hoc exploratory search', in *Proceedings of the 24th international conference on world wide web*, pp. 12–22.

Advanced NLP with spaCy · A free online course (no date) *Advanced NLP with spaCy*. Available at: https://course.spacy.io/en (Accessed: 26 September 2021).

Amitay, E., Sivan, N.H.R. and Soffer, A. (2004) 'Web-a-Where: Geotagging Web Content'.

Annotation Specifications · spaCy API Documentation (no date). Available at: https://spacy.io/api/annotation#named-entities (Accessed: 7 September 2021).

Apache ECharts (no date). Available at: https://echarts.apache.org/en/index.html (Accessed: 3 September 2021).

Eftimov, T., Koroušić Seljak, B. and Korošec, P. (2017) 'A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations', *PloS one*, 12(6), p. e0179488.

Elwood, S. (2011) 'Geographic information science: Visualization, visual methods, and the geoweb', *Progress in Human Geography*, 35(3), pp. 401–408.

English · spaCy Models Documentation (no date) *English*. Available at: https://spacy.io/models/en (Accessed: 8 September 2021).

Ghimire, D. (2020) 'Comparative study on Python web frameworks: Flask and Django'.

Goldberg, D.W., Wilson, J.P. and Knoblock, C.A. (2007) 'From text to geographic coordinates: the current state of geocoding', *URISA journal*, 19(1), pp. 33–46.

Grinberg, M. (2018) *Flask web development: developing web applications with python*. O'Reilly Media, Inc.

Holovaty, A. and Kaplan-Moss, J. (2009) *The definitive guide to Django: Web development done right*. Apress.

Language Processing Pipelines · spaCy Usage Documentation (no date a) *Language Processing Pipelines*. Available at: https://spacy.io/usage/processing-pipelines (Accessed: 8 September 2021).

Language Processing Pipelines · spaCy Usage Documentation (no date b) *Language Processing Pipelines*. Available at: https://spacy.io/usage/processing-pipelines#built-in (Accessed: 22 September 2021).

Ledur, C. *et al.* (2015) 'Towards a domain-specific language for geospatial data visualization maps with big data sets', in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. IEEE, pp. 1–8.

Lieberman, M.D. *et al.* (2007) 'STEWARD: Architecture of a Spatio-Textual Search Engine'.

Mansouri, A., Affendey, L.S. and Mamat, A. (2008) 'Named entity recognition approaches', *International Journal of Computer Science and Network Security*, 8(2), pp. 339–344.

Melo, F. and Martins, B. (2017) ‘Automated geocoding of textual documents: A survey of current approaches’, *Transactions in GIS*, 21(1), pp. 3–38.

Mohit, B. (2014) ‘Named entity recognition’, in *Natural language processing of semitic languages*. Springer, pp. 221–245.

‘Natural Language Processing NER – Which model to use?’ (2020) *Druva*, 5 November. Available at: https://www.druva.com/blog/natural-language-processing-ner-which-model-to-use/ (Accessed: 29 August 2021).

New Zealand Coordinate Conversions (no date). Available at: https://www.geodesy.linz.govt.nz/concord/ (Accessed: 3 September 2021).

New Zealand Gazetteer of place names (no date) *Toitū Te Whenua Land Information New Zealand*. Available at: https://www.linz.govt.nz/regulatory/place-names/find-place-name/new-zealand-gazetteer-place-names (Accessed: 28 August 2021).

New Zealand Map Grid (NZMG) (no date) *Toitū Te Whenua Land Information New Zealand*. Available at: https://www.linz.govt.nz/data/geodetic-system/datums-projections-heights/projections/new-zealand-map-grid-nzmg (Accessed: 3 September 2021).

Nominatim Usage Policy (no date). Available at: https://operations.osmfoundation.org/policies/nominatim/ (Accessed: 11 September 2021).

NZGB Gazetteer | linz.govt.nz (no date). Available at: https://gazetteer.linz.govt.nz/ (Accessed: 31 August 2021).

OpenLayers v6.6.1 API - Index (no date). Available at: https://openlayers.org/en/latest/apidoc/ (Accessed: 3 September 2021).

Overview - Nominatim Documentation (no date). Available at: https://nominatim.org/release-docs/develop/api/Overview/ (Accessed: 11 September 2021).

Pant, S. (no date) ‘Comparison of Google map features with Open Street Map’.

Rule-based matching · spaCy Usage Documentation (no date a) *Rule-based matching*. Available at: https://spacy.io/usage/rule-based-matching#entityruler (Accessed: 25 August 2021).

Rule-based matching · spaCy Usage Documentation (no date b) *Rule-based matching*. Available at: https://spacy.io/usage/rule-based-matching#models-rules (Accessed: 8 September 2021).

Rule-based matching · spaCy Usage Documentation (no date c) *Rule-based matching*. Available at: https://spacy.io/usage/rule-based-matching#phrasematcher (Accessed: 8 September 2021).

Shelar, H. *et al.* (2020) ‘Named entity recognition approaches and their comparison for custom ner model’, *Science & Technology Libraries*, 39(3), pp. 324–337.

spaCy 101: Everything you need to know · spaCy Usage Documentation (no date) *spaCy*

101: Everything you need to know. Available at: https://spacy.io/usage/spacy-101 (Accessed: 26 September 2021).

Training Pipelines & Models · spaCy Usage Documentation (no date) *Training Pipelines & Models*. Available at: https://spacy.io/usage/training#quickstart (Accessed: 9 September 2021).

Visa, S. *et al.* (2011) 'Confusion matrix-based feature selection.', *MAICS*, 710, pp. 120–127.

Wang, R. and Li, J. (2019) 'Bayes test of precision, recall, and F1 measure for comparison of two natural language processing models', in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4135–4145.

Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation (no date a). Available at: https://geopy.readthedocs.io/en/stable/ (Accessed: 31 August 2021).

Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation (no date b). Available at: https://geopy.readthedocs.io/en/stable/#geopy.extra.rate_limiter.RateLimiter (Accessed: 11 September 2021).