
GLAM — Geocoding via LINZ Address Matching



Liam Morris

Supervisors:

Professor Cameron Walker

Assoc. Professor Michael
O'Sullivan

Department of Engineering Science
The University of Auckland

A THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF OPERATIONS RESEARCH AND ANALYTICS IN ENGINEERING SCIENCE,
THE UNIVERSITY OF AUCKLAND, 2025.

THIS THESIS IS FOR EXAMINATION PURPOSES ONLY AND IS CONFIDENTIAL TO THE
EXAMINATION PROCESS.

Abstract

As New Zealand's data landscape evolves, driven by population growth, technological advancements, and increased global competition, the need for efficient geospatial analytics has grown significantly. Geocoding, the process of converting named locations into geographical coordinates, plays a central role in decision-making for various industries such as emergency services, retail, and urban planning. While existing commercial and open-source geocoding solutions are available, they often suffer from high costs, slow processing speeds, security concerns due to cloud processing, and limitations in handling large volumes of historical address data.

This thesis addresses these drawbacks by developing an innovative, open-source geocoding solution tailored to New Zealand addresses. A comprehensive review of international geocoding methodologies was conducted. Methods from this survey were then selectively adapted to suit the needs of this project, and a novel method based on compositional vectors was proposed. A real dataset of addresses was sourced from New Zealand Post for evaluation, and the methods implemented for this project demonstrated performance competitive with leading commercial solutions.

The outcome of this research is the creation of a Python package that provides efficient, locally executable geocoding with a focus on speed, accuracy, and accessibility. This package, accompanied by a user guide and [demo website](#), aims to offer a practical and cost-effective solution for geocoding in New Zealand, enabling better decision-making across industries.

Acknowledgements

I would like to express my gratitude to my supervisors, Professor Cameron Walker and Associate Professor Michael O'Sullivan, not only for making this research possible, but also for their guidance in shaping the direction and success of this work.

I am also grateful to Greg Schaaf and Matt Riordan from New Zealand Post for generously providing the address data used for evaluation, without which this project would not have been possible. Their contribution was fundamental to the progress of this research.

Finally, I would like to extend my sincere thanks to all those who have supported me in one way or another during this journey. Special thanks to Niklas Pechan, Martin Ooi, Alistair Evans, Matthew Brennan, Dean Franklet, Lisa de Kluijver, and Madison Robb for their friendship and encouragement along the way. I appreciate the personal and professional support from each of you.

Liam Morris

February, 2025

Table of Contents

List of Figures	ix
List of Tables	xi
1 Introduction	1
2 Statement of Research Intent	3
2.1 Validation and Verification	4
2.2 Limitations of Scope	4
2.3 Significance of Research	4
3 Literature Survey	5
3.1 Methodology	5
3.2 Search Results and Discussion	6
3.3 Findings	7
3.4 Discussion	29
4 Methodology	31
4.1 Method Selection	31

4.2	Software	34
4.3	Data Acquisition	34
4.4	Preprocessing	36
4.5	Parsing	37
4.6	Matching	48
5	Experimentation	67
5.1	RNN Parser	67
5.2	Address Embedding Model	70
5.3	Neighbourhood Searching	83
5.4	Hybrid Methods	86
6	Results and Discussion	89
6.1	Evaluation	89
6.2	Discussion	110
7	Software Contributions	115
7.1	Demo Website	115
7.2	User Guide	116
8	Conclusions	121
	Appendix	123
A	Lookups	124
B	Literature Search Results	126

C	Geocoding Services Code	148
D	Address Modifying Prompt	151

Bibliography

List of Figures

3.1	Literature Review Search Results	6
3.2	Geocoding Process	7
3.3	Example of Tokenising an Address (␣ Indicates a Space)	10
3.4	Example Address Parsing	11
3.5	Hidden Markov Model	13
3.6	Example of Inverted Index for Addresses	17
3.7	Example Effects of Contrastive Loss Function in 2D	23
3.8	Example of Triplet Loss Calculation in 2D	24
3.9	Recurrent Neural Network	25
3.10	Bidirectional Recurrent Neural Network	26
3.11	Attention Lines Indicating Attention the Word ‘himself’ Gives to Other Words in the Sequence	27
4.1	Structure of the RNN Parser Model	43
4.2	RNN Model Outputs	44
4.3	Component-Wise Fuzzy Searching Process	51
4.4	Architecture of the Transformer-Based Address Embedding Model	60

4.5	2D Space Partitioned by KD Tree After Two Iterations	64
5.1	Training Loss for Grid Searched Models	69
5.2	Softmax Probabilities/Predictions for Example Input Address	70
5.3	Training Loss (Triplet) for Address Embedding Models	72
5.4	Training Loss (Contrastive) for Address Embedding Models	72
5.5	Training Loss for Address Embedding Models Using Easy Triplets	74
5.6	Speed Performance Comparison of Nearest Neighbour Search Methods	84
5.7	Hybrid Method Match Rates for Different k Values	88
7.1	Screenshot of Demo Website	116

List of Tables

3.1	Techniques Used in Different Geocoding Implementations.	8
3.2	Example Address Representations	12
4.1	Techniques Implemented	32
4.2	Extract of LINZ NZSA	35
4.3	Address Component Labels for RNN Parser	39
4.4	Example Addresses From Synthesisation Process	40
4.5	Postprocessing for “third floor, 5th desk, 70 Symonds Street, 101” . . .	46
4.6	House Number Field Splitting Requirements	46
4.7	Mapping Table for libpostal Address Labels	48
4.8	Matching Methods and Associated Neighbourhood Searches	49
4.9	Example Training Triplets Generated for Anchor Address “70 Symonds Street, Grafton 1010”	58
4.10	Example Triplets Converted to Training Pairs for Contrastive Loss . . .	58
5.1	RNN Parser Hyperparameter Grid Search	68
5.2	Sample of Grid Searched Model Performances	68
5.3	Sample of Models Made for Grid Search	68

5.4	Address Embedding Model Hyperparameter Values	71
5.5	Address Embedding Models Tested	71
5.6	Randomly Sampled Addresses for Initial Testing purposes	75
5.7	Results from Classifier Experimentation	86
6.1	Evaluation Dataset Example	91
6.2	Evaluation Categories	92
6.3	Alternative Geocoders	93
6.4	Geocoding API Rate Limitations	94
6.5	Pricing Comparison of Alternative Solutions	95
6.6	Geocoding API Rates for 7,500 Evaluation Addresses	96
6.7	Geocoding API Accuracy Estimations	97
6.8	Detailed Results for Google API	98
6.9	Detailed Results for AWS API	98
6.10	Parsing Results for Nicely Formatted Addresses	99
6.11	Error Classifications for Realistically and Aggressively Modified Addresses	100
6.12	Parser Error Estimates for Realistically and Aggressively Modified Ad- dresses	100
6.13	Matchers Evaluated	102
6.14	Matching Algorithm Speeds	103
6.15	Matching Algorithm Accuracies for Nicely Formatted, Realistically Modified and Aggressively Modified Addresses	104
6.16	Addresses Mismatched by TF-IDF Matcher	105

6.17 Comparison of Accuracy Scores for Implemented Methods and Alternative Solutions	108
6.18 Accuracy and Robustness Gradings for Each Geocoder	109
6.19 Complete Comparison of Implemented and Alternative Geocoders . . .	109
8.1 Literature Survey Results	127

Acronyms

CNN Convolutional Neural Network

CRF Conditional Random Fields

HMM Hidden Markov Models

LINZ Land Information New Zealand

LLM Large Language Model

LSTM Long Short Term Memory

MLP Multi-Layer Perceptron

NLP Natural Language Processing

NZ Aotearoa New Zealand

NZSA New Zealand Street Address

OOV Out of Vocabulary

OSM OpenStreetMap

PNF Postcode Network File

RNN Recurrent Neural Network

SVM Support Vector Machine

TF-IDF Term Frequency-Inverse Document Frequency

Chapter 1

Introduction

As the data landscape in New Zealand matures, driven by factors such as a growing population, technological advancements, and heightened global competition, enterprises are increasingly turning to data and optimisation strategies to enhance their operations and maintain competitiveness. This process of optimisation frequently relies on geospatial analytics, with geocoding playing a pivotal role.

Geocoding, the process of converting named locations into geographical coordinates, enables the geospatial analytics used in industries such as emergency services, retail, and urban planning. While both commercial and open-source geocoding solutions are available, these often prove to be prohibitively expensive, sluggish, and unsuitable for bulk geocoding of historical address data.

Drawing upon my experience as a Data Science Consultant in New Zealand, this thesis explores the methodologies employed by geocoding algorithms on an international scale. It further discusses the application and implementation of these algorithms for New Zealand addresses, proposing a novel solution for address matching. An evaluation of the implemented methods is conducted, comparing their performance against state-of-the-art commercially available solutions. Finally, this thesis describes how the implemented methods are packaged and made publicly available for general use.

Statement of Research Intent

Given the constraints posed by existing geocoding solutions, projects focusing on location or route-based logistics/optimisation often encounter challenges in their initial phases. The upfront costs associated with geocoding, coupled with the difficulty of assessing potential benefits without coordinate data, can impede the progress of such projects. This research aims to address these obstacles by developing an alternative solution for matching addresses to the Land Information New Zealand (LINZ) NZ Street Address (NZSA) dataset [24]. As such, the proposed solution will prioritise the following criteria:

1. **Speed:** The solution should facilitate a swift turnaround of analytics.
2. **Accessibility:** It should be freely available and not require extensive software installations or specialist hardware.
3. **Local execution:** The solution should be executable locally to meet potential security requirements surrounding personally identifiable information.
4. **Accuracy:** It should maintain sufficient precision for informed decision-making while balancing speed and portability.
5. **Ready to use:** It should not require model training or configuration before geocoding.

2.1 Validation and Verification

The success of this project can be measured against the suggested criteria. Speed and accuracy comparisons will be made against commercial solutions and free/open-source alternatives. These comparisons will be made with a sample dataset of NZ addresses.

2.2 Limitations of Scope

Some potential areas to be explored in subsequent projects could be:

- Extension to named locations and non-residential addresses
- Implementation and comparison to more existing methodologies
- Generalisation to other countries

2.3 Significance of Research

This research will provide a free and open-source solution for bulk geocoding NZ addresses.

Chapter 3

Literature Survey

This survey explores geocoding techniques used in the literature, with a focus on methods designed to match lookup/query addresses to a reference address database and assess their performance and limitations under different conditions.

3.1 Methodology

A document search was conducted in July 2023 using a combination of Google Scholar and Scopus.

Google Scholar was used first with a generic search for “address matching techniques” limited to 20 results. This provided a generic starting point as Google Scholar automatically searches for synonyms and similar words. Keywords were then identified from relevant papers in this initial search, and Scopus was used to conduct a more systematic search. Results were then combined and manually filtered based on title and abstract to produce a refined list before being sought for retrieval. Details on the manual filtering are given in §3.2.

The query used in Scopus was:

```
TITLE-ABS-KEY (
  geocod*
  AND address
  AND ( matching OR parsing OR tokenization OR tokenisation OR
        linkage OR resolution )
  AND ( local OR speed OR compar* OR cloud )
)
AND ( LIMIT-TO ( LANGUAGE , "English" ) )
```

3.2 Search Results and Discussion

Figure 3.1 presents a summarised flowchart of inclusion/exclusion for this survey. The full breakdown is included in Appendix B.

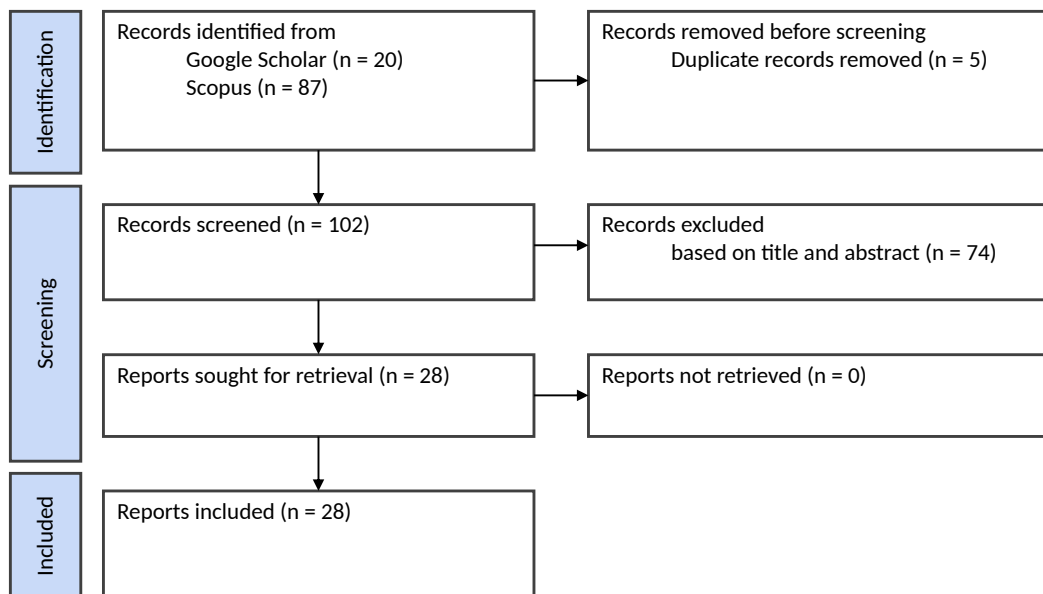


Figure 3.1: Literature Review Search Results

Results that were filtered out typically discussed the application of geocoding algorithms to various problems rather than the development of the geocoding algorithms themselves. This is hard to avoid as the applications of geocoding algorithms are often

used to solve a similar problem as the algorithms themselves, e.g., matching patient records in health data may make use of the patients' addresses.

The initial search using Google Scholar provided a good base for the search. These results provided a good range of keywords used to construct the systematic search with Scopus. This approach was helpful as crafting a high-quality search with Scopus limited the number of papers for inclusion without restricting publication date or missing any key techniques. The result was a tractable list of papers that cover a wide variety of methodologies and demonstrate the evolution of techniques over time.

3.3 Findings

The literature discussed many approaches to address matching, with varied degrees of complexity. The geocoding process typically followed a variant of this three-step process:

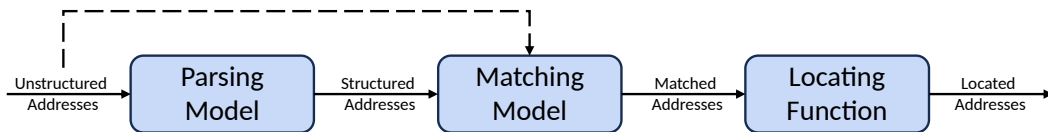


Figure 3.2: Geocoding Process

The implementation of this process varies based on the supporting reference databases and the chosen models. In particular, for deep learning models, the parsing and matching models may be combined, in which case the unstructured addresses are fed directly into the matching model (dotted arrow). A locating function is also sometimes required if the address reference database does not include location information and is instead accompanied by an addressing system.

Within the parsing/matching/locating models, multiple techniques may be used together to produce the best result. This review breaks down address-matching algorithms into their composite techniques. The review also focuses on the parsing and matching process, as the NZSA includes coordinate data, removing the need for a locating function.

A summary of the techniques used in the literature is provided in Table 3.1. Note that some of these techniques are applicable to both parsing and matching. These

techniques are included under the section that they are most commonly used. Papers that propose multiple methods or ensemble methods are included once under each methodology used.

Method	Used By
Parsing	
Rules-Based	[44, 5, 26]
Hidden Markov Models	[11]
CRFs*	[11, 59, 3]
Matching	
Blocking	[11, 3, 13]
Fuzzy Matching	[29, 43, 59, 44, 58, 26]
Indexing	[16]
Vectorising	[29, 43, 60, 26, 10]
Pretrained Embedding	[11, 27, 30]
Custom Embedding	[60] (RNN/CNN)** [43, 30, 32, 57, 13] (Transformers)
Classification Models	[11, 30, 59, 3, 57, 27, 26]
Dataset Generation	[43, 30, 32]

* Conditional Random Fields (CRF)

** Recurrent Neural Network (RNN), Convolutional Neural Network (CNN)

Table 3.1: Techniques Used in Different Geocoding Implementations.

3.3.1 Preprocessing

Before training and/or inference, most geocoding algorithms applied some preprocessing to reduce the heavy lifting required by the subsequent models. This section discusses the most common preprocessing steps in the literature.

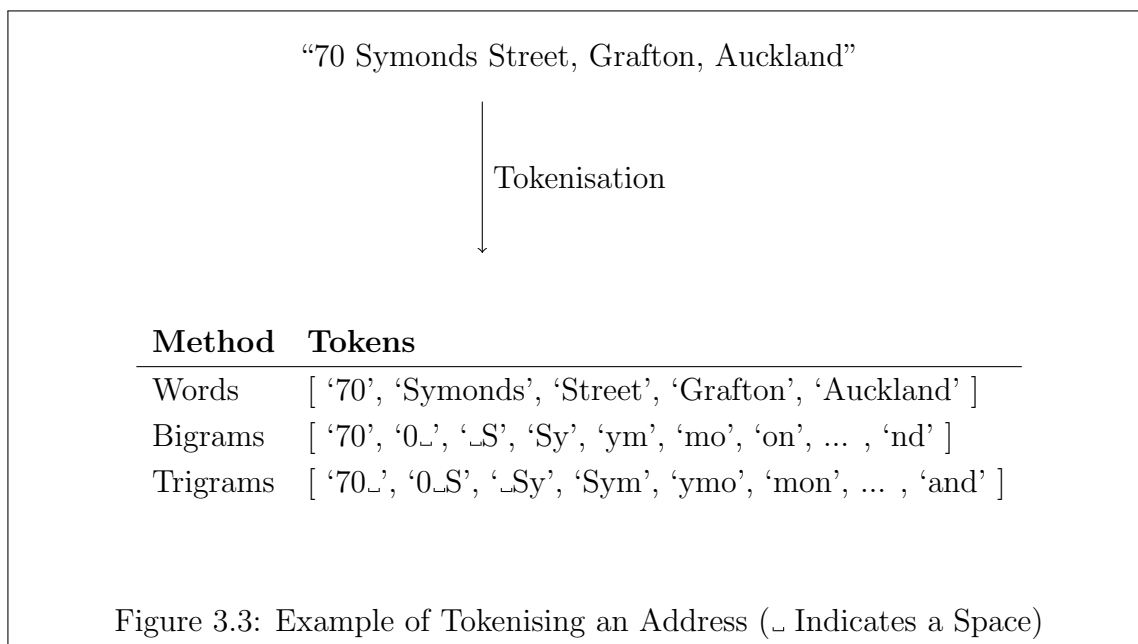
Cleaning

Cleaning methods focus on removing irrelevant information from input addresses. The exact cleaning rules depend on the application and context of the model, but [31] suggests the following:

- **Deduplication:** This is relevant for both training and inference. Duplication in training data should be avoided to protect against overfitting. Duplicated data during inference leads to redundant processing time and slower inference.
- **Lowercasing/uppercasing:** Using a common letter casing reduces the vocabulary sizes required for modelling, and there is usually no non-linguistic meaning to the letter casing.
- **Removing special symbols:** For many address formats around the world, special characters such as `*^%&` are not used but may be included in accompanying information such as building names.
- **Trimming whitespace:** Removing leading, trailing, and otherwise excessive whitespace improves the standardisation of the address data before being inputted into the model.
- **Specific adjustments:** Some datasets may benefit from specific cleaning depending on their context, e.g., removing (or adding) ‘New Zealand’ when all addresses are known to be within NZ.

Tokenisation

Tokenisation is the process of segmenting a single string into a list of smaller, representative strings. These tokens represent a unit of text that a natural language processing (NLP) model can learn a meaning for. The tokens could be words from the original string, subwords like ‘un’ and ‘likely’, or character-based n-grams, where the list is a rolling window of n-length strings [43].



Intuitively, using words and subwords as tokens provides the most direct link between language and model (statistical, deep learning, or otherwise), as any representations learned by models can be mapped back to whole words in the source language. The main disadvantage of this for address matching algorithms is it requires a very large vocabulary. It is also prone to out-of-vocabulary (OOV) errors when the model encounters a new word that was not present in training data. The use of subwords improves this problem for many NLP tasks.

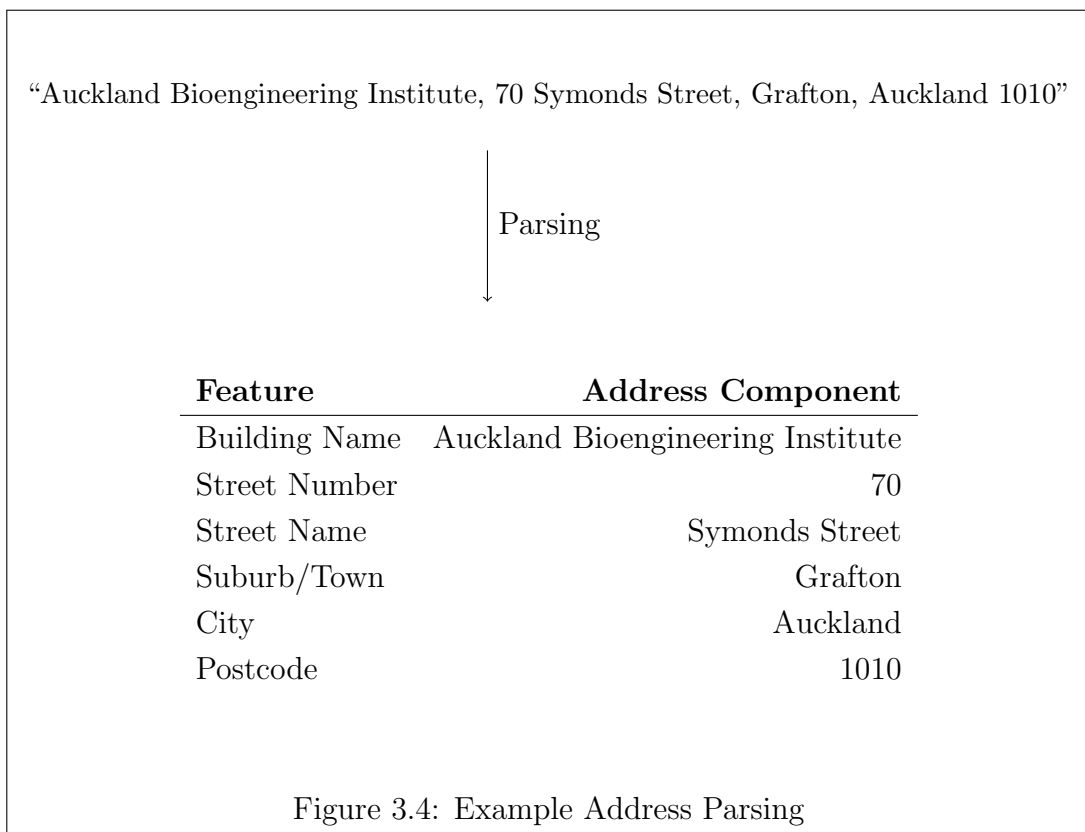
For example, a model may not recognise the word ‘electroencephalography’, but it can be tokenised into ‘electro’, ‘encephalo’, and ‘graphy’. This allows deep learning models to understand a whole word from its parts. However, addresses are extra prone to OOV errors as they include many proper nouns and typos. The use of subwords does not provide many advantages here. For this reason, the use of tokens such as bigrams and trigrams is common in the literature [29, 26, 10, 11, 27, 60, 43, 30, 32, 57, 13].

Using n-grams reduces the vocabulary size and the likelihood of OOV errors, improving efficiency and generalisability. The relationship between n-gram size and vocabulary is exponential, $V = k^n$, where V is the vocabulary, k is the cardinality of the set of allowed characters after cleaning, and n is the n-gram size. This increase in vocabulary allows NLP models to learn more about the semantic meaning of text and sequential

characters. However, it also increases the storage size required for the model, as parameters need to be stored for every token in the vocabulary.

3.3.2 Parsing

As addresses are often stored as unstructured text, parsing this data is a natural first step taken on the road to address matching. Parsing — also referred to as segmentation — is the process of converting an unstructured address to a structured/labelled address, i.e., identifying labels for each part of the unstructured address.



Complications arise when converting from unstructured to structured data as the same address can be represented in many different ways [26], for example:

Address	Format Variation
70 Symonds Street, Grafton, Auckland 1010	Fully qualified address
ABI, 70 Symonds Street	Includes a building name
70 Symonds Street	Street number and name only
70 Symonds Street, Auckland Central	Incorrect (neighbouring) suburb
70 Symonds St, AKL	Uses abbreviations for street and city
70 Simmonds Street, Auckland	Includes typographical error

Table 3.2: Example Address Representations

These are just a handful of problematic examples that are commonly found in unstructured address data. Depending on the parsing model used, there can also be difficulty identifying components for new/unseen streets and suburbs. This is due to some parsing models remembering known locations rather than parsing the addresses based on their structure.

The literature discussed several approaches to address matching that are designed to combat issues with address representations such as these. The next sections summarise these methods and discuss when each is most effective.

Rule-Based

The most basic way to parse an address is by identifying patterns in the address format/dataset. This may involve searching for known suffixes for states/provinces, streets, etc. Construction of these rules can be time-consuming and requires thorough inspection of the dataset. Once the rules have been defined, regular expressions (regex) are an effective way to implement them. Regex works by defining patterns that will quickly match specific components of an address.

Rule-based parsing is used successfully in [26] with semi-structured data, but it lacks the flexibility required for truly unstructured address data. For example, the regex pattern:

```
\$(?P<street_number>\d*)\s*(?P<street_name>[a-zA-Z\s]~)
```

will successfully parse all addresses in the format <street number> <space> <street name>, but if there are any other address formats in the data, this method will fail. It is therefore not useful in circumstances with unstructured address data.

Hidden Markov Models

Hidden Markov Models (HMM) are a statistical method of estimating a sequence of hidden states, given a sequence of observed events based on predefined transition and emission probabilities.

HMMs use the following definitions:

- X is the sequence of hidden states
- Y is the sequence of observations
- a_{ij} is the probability of transitioning from state i to state j
- π_i is the probability of beginning the sequence in state i
- $b_j(k)$ is the probability of emitting observation k in state j

A set of states and observations can then be visualised like so:

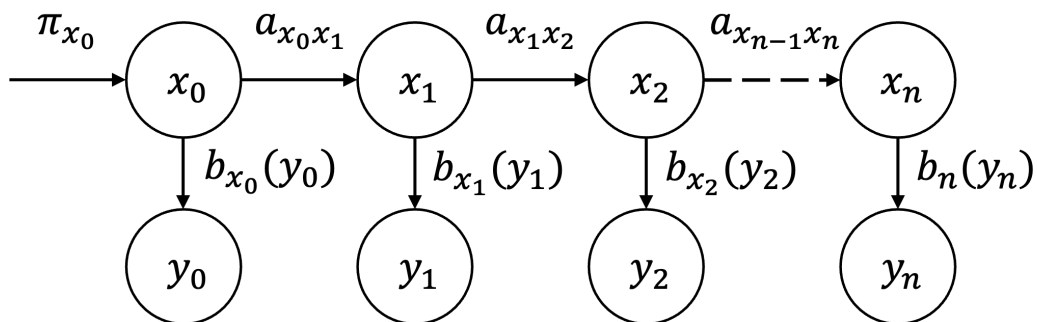


Figure 3.5: Hidden Markov Model

In the context of address parsing, HMMs can be used after tokenising an unstructured address string, treating the address feature labels as hidden states and the tokens from the string as the observed sequence. The transition and emission probabilities can be learned from a dataset using maximum likelihood estimation. During inference, the

Viterbi algorithm can be used to compute the most likely labels given an observed address [11].

HMMs are memoryless, which means that the transitional probabilities only depend on the current state, disregarding what has come previously. This represents a short-coming of HMMs for the purposes of modelling an address string, as addresses are often better interpreted as a whole [11]. This is intuitive when considering numbers in an address string. For example, numbers towards the end of the string will have the same transition probabilities as numbers near the beginning of the string, even though digits at this position are much more likely to represent a postcode compared to a street number and vice versa.

Conditional Random Fields

Conditional Random Fields (CRF) are a relaxation of HMMs that allow for more complicated sequence modelling.

There are several variations of CRFs, but all of them work by removing restrictions on the transition matrix $A = (a_{ij})$. This allows for states to be dependent on backward and/or forward-looking states, as well as other observations [10].

These relaxations make fitting and inference of CRFs more complex than HMMs, but also provide extra flexibility, which is useful in the context of addresses [10]. An example of this is the removal of independence between observations, as it is expected that real-world addresses will include interaction and dependencies, e.g. postcodes are related to city names [11].

libpostal

libpostal (stylised in lowercase) is an open-source library built in C for parsing street addresses around the world [38]. It is referenced and used frequently in the literature [11, 10, 59], and uses CRFs to parse addresses in many different languages with 99.45% accuracy. libpostal is trained on OpenStreetMap (OSM) data, which provides a fairly consistent structure for all of New Zealand. There are no officially provided binaries for libpostal, which can make installations tricky (especially on Windows). libpostal also requires downloading large dependencies before it can be used, which limits portability.

Normalising

Normalising is a common postprocessing step after parsing an address. This corrects the formatting of the address to improve matchability to the address reference database [43]. The implementation of normalisation therefore depends on the matching algorithm and address reference database being used. However, a common step is lengthening abbreviations for streets, e.g. *st* → *street* and states, e.g. *NY* → *New York*.

3.3.3 Matching

Matching techniques are the core of address-matching algorithms and are a specific case of the general entity resolution problem. The literature discusses a wide variety of these algorithms and their applications to address matching. The composite parts of address-matching techniques are extracted and discussed in the following sections.

Fuzzy Matching

Fuzzy matching is a simple and effective way to perform address matching. The term ‘fuzzy matching’ refers to a group of algorithms and distances for determining either a similarity or dissimilarity score via a direct comparison of two strings. These are generally based on an edit distance or an intersection of characters/tokens between the two words [29].

For two input strings, *S1* and *S2*, some of the most common fuzzy matching methods are:

- **Hamming distance** [4] measures the minimum number of substitutions required to transform *S1* into *S2*.
- **Levenshtein distance** [28] measures the edit distance as the minimum number of operations required to transform *S1* to *S2*. Edit operations include insertions, deletions, and substitutions.
- **Jaro Winkler (JW)** [56] builds upon Jaro distance [23], which is based on the number of matching characters and transpositions between *S1* and *S2*. JW

adds an inflation factor for having matching prefixes between $S1$ and $S2$. A JW distance of 0 indicates a perfect match, and 1 indicates no matching characters.

- **Cosine similarity** measures similarity between two non-zero vectors by using the cosine of the angle between them [50]. First, $S1$ and $S2$ are encoded as vectors where each entry is a count of occurrences for each letter in them. The words are similar if their encoding vectors are parallel ($\cos(0) = 1$) and dissimilar if they are perpendicular ($\cos(90) = 0$).

There are a plethora of different fuzzy matching distances/algorithms beyond those listed here. Some of these incorporate additional logic such as string alignment, weightings, and other techniques to improve their performance [50]. As a result, each of these algorithms has its own pros and cons. For example, due to character encoding, cosine similarity does not consider the order of characters within the strings. This means that anagrams will have a perfect match score despite potentially being different words.

Despite the wide variety of fuzzy matching algorithms, there are some common drawbacks:

- **Speed:** Fuzzy matching algorithms are computationally expensive. Levenshtein has a time complexity of $\mathcal{O}(n_1n_2)$, where n_1 and n_2 are the lengths of the two strings being compared), which quickly becomes intractable when working with either long strings or large datasets.
- **Colloquialisms:** Fuzzy matching algorithms lack the ability to identify synonyms or abbreviations. In the case of addresses, an abbreviation of ‘High Street’ \rightarrow ‘High St.’ can significantly impact a similarity score and prevent a successful matching.

Indexing

Indexing is used to optimise retrieval of relevant records from a database given a search query. Indexing can also be applied to many forms of data by making small changes to the index structure and ordering. The most applicable method for matching addresses is called an inverted index [9]. Inverted indexing, at a high level, is extracting tokens from records in the database and storing these as keys in the index. Figure 3.6 demonstrates this concept using words as tokens on a small address database.

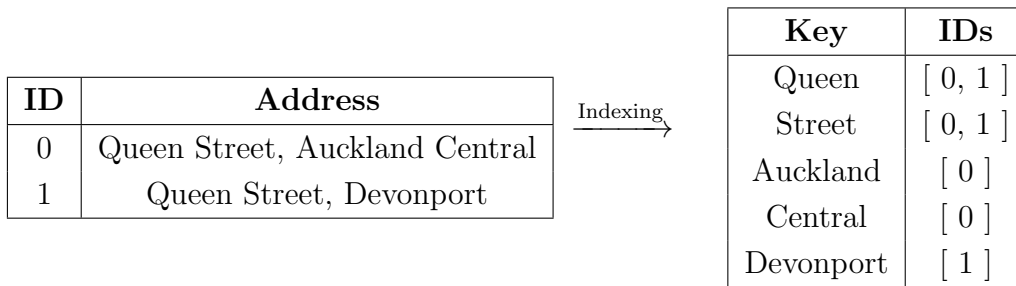


Figure 3.6: Example of Inverted Index for Addresses

A query address can then be tokenised, and IDs can be looked up using the inverted index. The most repeated ID can be considered the best match, or the top n IDs can be passed through to a more thorough match classifier. [9] implements a reverse index using structured address data and creates an index for each address component. This ensures that recommended addresses contain tokens from all address fields rather than many matches in one field.

Reverse indexing also allows the addition of synonyms, abbreviations, place names, etc., to addresses with no potential penalties. For example, ‘University of Auckland’ could be included in the tokens for the address ‘70 Symonds Street’; then, if the query includes any of these tokens, the address will be returned as a potential match. This comes at the cost of a larger index. The query process can also be extended to allow fuzzy matching, where all keys within a maximum edit distance of the query are returned.

This method of indexing was effectively used in the literature, but there were several drawbacks:

- Adept at returning potential matches, but further processing is often required to find a single best match.
- Requires expensive upfront creation of the index.
- Requires storage of a large index table.
- Index querying can be slow, but parallelisation and specialised data structures can mitigate this.

Blocking

Blocking is a technique used for increasing computational tractability when attempting to match a structured or semi-structured address [11]. Blocking uses an address feature, known as a blocking key, to perform filtering using an exact match before address pairs are fed through a matching function. This reduces the complexity from $\mathcal{O}(nm)$ to $\mathcal{O}(nm/b)$, where n is the number of addresses attempting to be matched, m is the size of the address reference database, and b is the number of blocks (unique values in the blocking key column). This allows matching algorithms to scale better when working with large datasets [10].

Blocking is an effective technique, but it relies on having a feature that partitions datasets well. This means it must be available for most addresses, have a relatively uniform distribution of values, and ideally have a high number of unique values. Blocking also relies on the key matching exactly to the reference dataset, which prevents an address from being matched correctly if someone has entered the wrong zip code, for example [10].

Some implementations of blocking remedy this by performing a fuzzy match with a maximum edit distance rather than using an exact match in an attempt to capture typos and neighbouring areas [11]. This system relies on neighbouring zip codes having a similar value, which is not always the case depending on the country.

Vectorisation

Vectorisation is used frequently in the literature in one of two ways:

- Representation of an *address* as a vector
- Representation of an *address pair* as a vector

The most common ways these appeared in the literature were Term Frequency-Inverse Document Frequency (TF-IDF) and similarity metrics.

TF-IDF

TF-IDF [51] is a popular method for vectorising text documents. In the case of address matching, each address is considered a document. The process is as follows:

1. Build vocabulary: Tokenisation must be performed on the entire database. These tokens will be represented by an index in the vectorised representation of an address.
2. Compute TFs: Count how many times each token appears within each address. This is the term frequency.
3. Compute IDFs: The IDF is calculated as the natural logarithm of the ratio of total documents to documents containing that token.
4. Compute TF-IDF score: The TF-IDF score is the product of the TF and IDF score for each term in an address.

For example, given the address database:

$$D = \begin{array}{|c|c|} \hline \mathbf{ID} & \mathbf{Address} \\ \hline 0 & \text{Queen Street, Auckland Central} \\ 1 & \text{Queen Street, Devonport} \\ \hline \end{array}$$

Using words as tokens, the vocabulary becomes the set of all words in the database:

{Queen, Street, Auckland, Central, Devonport}

The TF-IDF representations of the addresses are then:

$$\text{Queen Street, Auckland Central} \implies \begin{bmatrix} 1 \times \ln(2/2) \\ 1 \times \ln(2/2) \\ 1 \times \ln(2/1) \\ 1 \times \ln(2/1) \\ 0 \times \ln(2/1) \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ 0.7 \\ 0.7 \\ 0 \end{bmatrix}$$

$$\text{Queen Street, Devonport} \implies \begin{bmatrix} 1 \times \ln(2/2) \\ 1 \times \ln(2/2) \\ 0 \times \ln(2/1) \\ 0 \times \ln(2/1) \\ 1 \times \ln(2/1) \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.7 \end{bmatrix}$$

An address query can then be converted to its TF-IDF representation, and a vector distance calculation such as Euclidean distance or cosine similarity can then be used to find the nearest address in the database [1].

The main strengths of TF-IDF are the numerical representation of addresses and the importance weighting of terms based on their frequency. Its drawbacks include:

- The TF-IDF representation does not consider the order of tokens in the address
- Depending on the choice of tokens, the upfront calculation to vectorise the database can be expensive.
- TF-IDF representations for addresses are particularly sparse, as an address string is relatively short compared to the vocabulary size.
- Tokens from queries that are not included in the reference database are ignored.
- Does not consider colloquialisms.

Similarity Vectors

The literature also used vectorisation to represent a pair of addresses. This was most commonly achieved by computing a series of similarity metrics for the address pair. When these similarity metrics are computed for the Cartesian product of the query and reference datasets, they can be treated as features for a binary or ternary classification model to label the address pairs as match or non-match (or potential match in the case of a ternary model). The size of these vectors varies; [10] uses Jaro-Winkler distance on different address components for a total of 5 features, whereas [26] uses 17 edit and token similarity metrics.

As an example, Jaro-Winkler and Levenshtein distances are used to create similarity vectors that are two elements long. These similarity vectors can be calculated for the query address “Beech Rd” against a simple address database, D , with only three records:

$$D = \begin{array}{|c|c|} \hline \mathbf{ID} & \mathbf{Address} \\ \hline 0 & \text{Queen Street} \\ 1 & \text{First Avenue} \\ 2 & \text{Beach Road} \\ \hline \end{array} .$$

First, Jaro-Winkler and Levenshtein distances are calculated for every reference address:

ID	Jaro-Winkler	Levenshtein
0	JW(Beech Rd, Queen Street) = 0.61	L(Beech Rd, Queen Street) = 9
1	JW(Beech Rd, First Avenue) = 0.31	L(Beech Rd, First Avenue) = 11
2	JW(Beech Rd, Beach Road) = 0.89	L(Beech Rd, Beach Road) = 4

This produces the three similarity vectors: $[0.61, 9]$, $[0.31, 11]$, and $[0.89, 4]$. These vectors are fed through the fitted classification model, which outputs labels or match likelihoods. In this case, it might look something like $[0.2, 0.1, 0.9]$, where 0.9 is the highest likelihood of a match, so the address with index 2 (Beach Road) is returned as the match.

This combination of similarity scores and a classification model has proven to be quite effective despite its relatively simple approach [10, 26]. However, it does have some significant drawbacks:

- Extremely computationally expensive - most papers report that the pairwise calculation of one similarity metric is already restrictive in terms of speed. This approach can be significantly slower depending on the length of the vectors used [10].
- Requires careful selection of similarity metrics to avoid highly correlated features
- Classification models are limited by similarity metrics weaknesses. For example, since similarity metrics can not recognise abbreviations, the classification model inherits this problem.

Embedding

Embedding is similar to vectorisation, but the representation of an address/address pair is learned via a deep-learning model. There are many pretrained models for word/token/sentence embedding used in the literature. These include word2vec [37], BERT [12], and GloVe [40]. A pretrained model can be used directly to create address

embeddings, or fine-tuning can be done to enhance the model’s fit to the desired address reference database. Alternatively, an entirely new embedding model can be created using a variety of deep learning models [43, 30, 32, 57, 13].

The trade-offs of these approaches are:

- Deep learning models excel at capturing sequential information and semantic meaning. This means the embedded vectors are likely to capture much more information than traditional vectorisation.
- Deep learning models are able to interpret colloquialisms such as street/st.
- Pretrained models offer a head start but are trained to represent a much larger corpus of text. This means they will often have more parameters than required to learn an address representation, slowing down training and inference.
- Training or fine-tuning a deep learning model requires a carefully crafted dataset. Generalisability is important, so the training dataset needs to be representative of all possible addresses. The address pairs used for training also need to balance easily discernible pairs with more difficult ones to allow the model to learn progressively.
- Deep-learned embeddings are likely to be much larger than traditional vectors, which increases storage and slows down inference.

Siamese Networks Siamese networks, also known as twin networks, input two or more vectors through identical subnetwork models, producing multiple outputs/embeddings for comparison [7]. The loss function for the siamese network employs a distance function such as cosine or Euclidian distance to optimise the embeddings in a way that clusters similar addresses together. The two most common loss functions are contrastive and triplet [17].

Contrastive loss learns from input pairs, where an input pair is labelled as 1 for matching and 0 for non-matching [19]. The definition of contrastive loss is:

$$L(\vec{x}_1, \vec{x}_2, y) = y \cdot D(\vec{o}_1, \vec{o}_2) + (1 - y) \cdot \max\{M - D(\vec{o}_1, \vec{o}_2), 0\}$$

where:

- \vec{x}_1 and \vec{x}_2 are two inputs;
- y is the binary similarity label (one indicating similar and zero dissimilar);
- \vec{o}_1 and \vec{o}_2 are the associated outputs inputting \vec{x}_1 and \vec{x}_2 to the subnetwork;
- D is the distance metric to measure vector similarity (e.g. Euclidean);
- and M is a hyperparameter controlling the minimum distance between dissimilar pairs.

The max function ensures that the loss is zero if the distance between dissimilar pairs is greater than the margin. This enforces a minimum separation between dissimilar pairs by only incurring a loss if the computed distance is less than M . This concept is visualised for two dimensions in Figure 3.7.

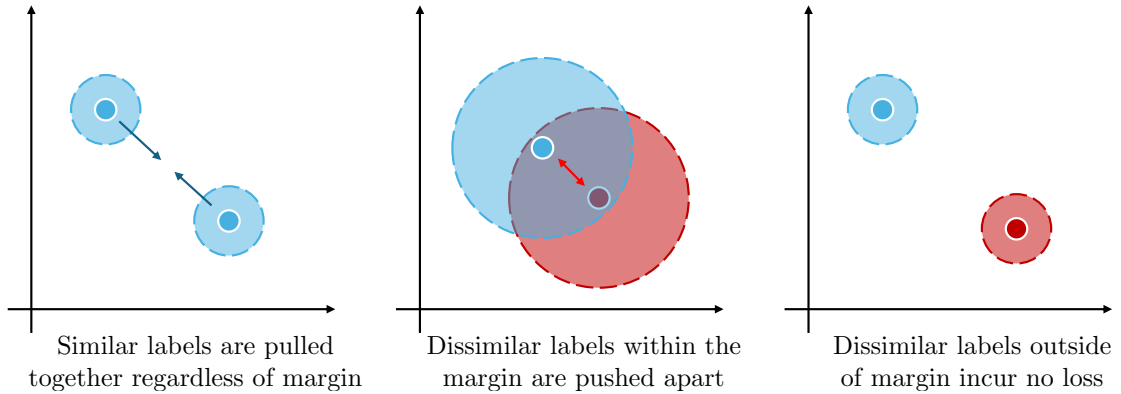


Figure 3.7: Example Effects of Contrastive Loss Function in 2D

Triplet loss learns from input triplets, where each triplet contains an anchor case, a positive (matching) example, and a negative (non-matching) example [6]. The definition for triplet loss is:

$$L(\vec{A}, \vec{P}, \vec{N}) = \max(D(f(\vec{A}), f(\vec{P})) - D(f(\vec{A}), f(\vec{N})) + \alpha, 0)$$

Where:

- $\vec{A}, \vec{P}, \vec{N}$ are the anchor, positive, and negative inputs respectively
- $f(\vec{A})$ denotes the result from inputting \vec{A} to the subnetwork

- D is the distance metric to measure vector similarity
- α is a hyperparameter controlling the margin between positive and negative pairs

By using α , no loss is incurred if the distance between the anchor and positive cases is sufficiently lower than the distance between the anchor and the negative case. Figure 3.8 shows this in two dimensions.

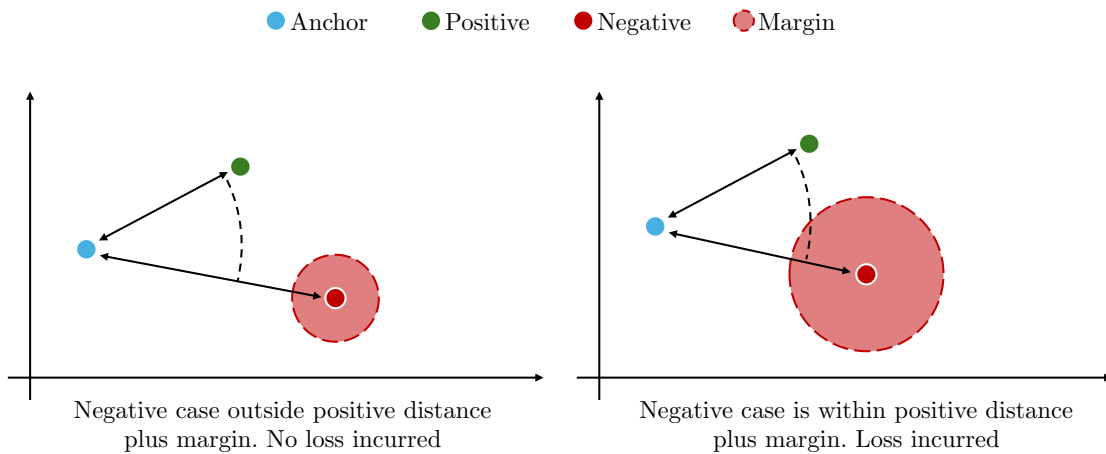


Figure 3.8: Example of Triplet Loss Calculation in 2D

As triplet loss is calculated based on three inputs, it is often more robust to noisy data and can learn to differentiate between levels of similarity. This is better when the true similarity exists on a continuum rather than being binary. The trade-off is that triplet loss requires more training data as one loss calculation requires three inputs instead of two [17].

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks particularly suited to interpreting sequential data [14]. The design of RNNs allows them to handle variable length inputs (like addresses), and also propagates a hidden state through the layer allowing for a ‘memory’ as shown in Figure 3.9.

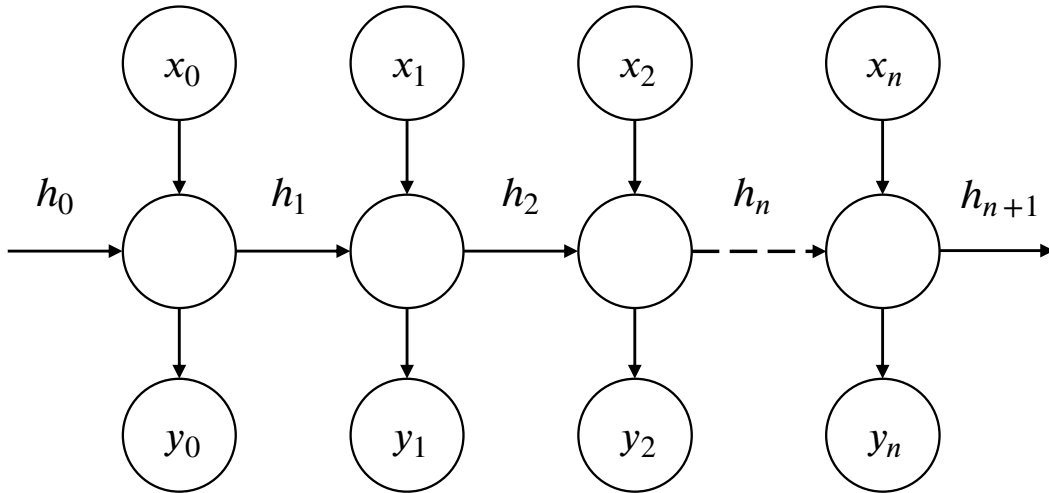


Figure 3.9: Recurrent Neural Network

In Figure 3.9:

- x_i represent inputs to the layer. These are typically produced via tokenisation from the raw input address (see §3.3.1).
- h_i represent the hidden state passed between nodes. These values allow the RNN to remember what it has passed over earlier in the sequence. Note that h_0 is slightly different, as this is a learned parameter that does not depend on the sequence input values.
- y_i represent the outputs of the layer. These values contain information the RNN has observed in the sequential data and are typically fed into more feed-forward layers within the neural network before becoming labels.

Traditional RNNs suffer from vanishing and exploding gradients, which can make training difficult. Improved architectures such as Long Short Term Memory (LSTM) [20] and Gated Recurrent Units (GRU) [8] have been designed to mitigate these issues. Both of these solutions implement gating to regulate the flow of information through a cell.

Being able to carry forward information is the key strength of RNNs. Bidirectional RNNs build on this strength by stacking RNN layers in opposing directions [46].

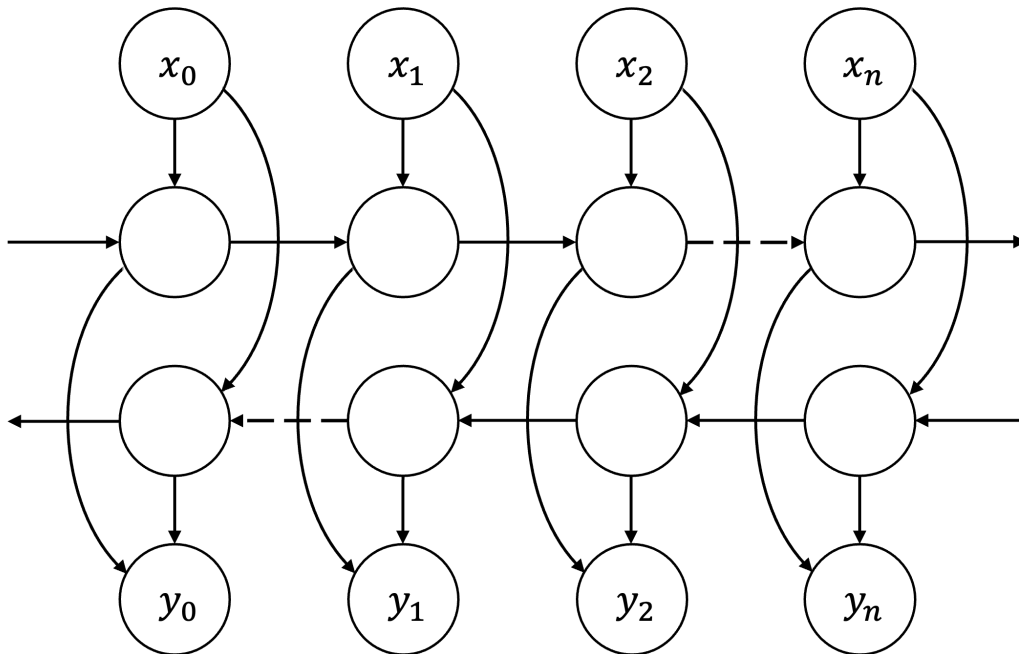


Figure 3.10: Bidirectional Recurrent Neural Network

Figure 3.10 demonstrates a bidirectional RNN. In this configuration, outputs are informed by previous and future information in the sequence. This is particularly relevant for sequences such as addresses, which are better interpreted as a whole rather than purely sequentially.

For example, when parsing “St Lukes Road”, the abbreviation ‘St’ could be improperly interpreted as an abbreviation for ‘street’ when strictly reading left to right. When using a bidirectional RNN, information from later in the address (‘Road’) is considered when choosing a label for earlier parts of the text. This helps to prevent misinterpreting fields in addresses with ambiguous text.

Transformers

Transformers [54] are a newer architecture in the world of deep learning and are currently the favoured model for many different applications, including language translation, sentiment analysis, and sequence-to-sequence tasks like address matching.

The defining trait of transformers is the attention mechanism within the encoder-decoder architecture. The use of attention during the encoding of an input effectively

reviews the entire sequence and places a weighting on certain tokens before making a prediction [54]. This differs from RNNs, where the hidden state is propagated along a sequence and continuously modified. This is what makes transformers adept at handling NLP tasks as long-range dependencies are not lost.

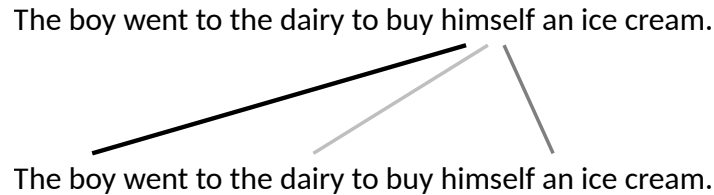


Figure 3.11: Attention Lines Indicating Attention the Word ‘himself’ Gives to Other Words in the Sequence

Figure 3.11 demonstrates how attention enables one word in the sequence to focus on other words in the sequence regardless of their location. Due to the non-sequential architecture, transformers utilise positional encoding to provide this information. This ensures transformers are still able to understand the ordering of the sequence.

Classification

In the literature, classification models were used in combination with vectorisation and embedding techniques to classify address pairs into matches, non-matches, and sometimes also potential matches.

The most common classification models used were logistic regression, Support Vector Machine (SVM), random forests, xgboost, and Multi-Layer Perceptron (MLP). Linear models like logistic regression and support vector machines tend to have the lowest performance and are outperformed by random forests, xgboost, and MLP. This is likely due to their ability to capture non-linear relationships between features in the address representations. MLP also has the ability to be integrated directly into a deep learning model, which avoids sending data between multiple different models.

Classification models tend to perform well in the literature [11, 30, 59, 3, 57, 27, 26], but they are also a computationally expensive approach. This is because they operate on some form of encoded address (embeddings or similarity vectors) and require comparing encoded query addresses to each address in the lookup database.

3.3.4 Data Generation

Cases were also common in the literature where no authentic human-entered addresses were available. In these cases, a dataset was generated/augmented in order to train a model. This process typically began with an existing, structured address dataset.

The first step in building a training dataset is determining which fields are necessary. Fields that frequently appear in real addresses will often not be present in a structured dataset, e.g., building names. These fields need to be generated to supplement the existing dataset.

Once the additional features have been generated to augment the dataset, the addresses need to be converted to unstructured addresses. In order to best prepare a model for true unstructured addresses, the following techniques were used [43]:

- Random Delimiting: The use of random delimiters to separate address fields. The length of the delimitation may also vary.
- Field dropout: Randomly choose to withhold some features.
- Field shuffling: Shuffle the order of some fields. For example, reverse the town and the city.
- Abbreviations: Replace common words with their abbreviations, e.g. street \rightarrow st
- Ordinal and cardinal conversions: replace numbers with an alternative representation, e.g., 1 \rightarrow first or 1st.
- typographical errors: Randomly insert typographical errors such as deletions, additions, substitutions and transpositions.

Correspondence between characters in the addresses and their source fields must be maintained to be used as a label for training parsing models. In the case of matching models, correspondence to the source address must also be maintained.

To create a balanced dataset for a matching model, some unstructured to structured pairs need to be non-matches. These should be created from a mix of complete address swaps and singular field swaps [43]. This produces a training dataset with a mixture

of reasons for addresses not matching and varying difficulties for the model to learn progressively.

3.4 Discussion

The literature for address matching covers a good range of techniques for both parsing and matching street addresses in many languages. While most of the literature was for addresses with more rigorous structures than New Zealand, many of these principles discussed are transferable.

Some gaps were identified in the literature and explored as part of this project. These are discussed in this section.

3.4.1 Performance analysis

The majority of papers included in this survey compare accuracies for different techniques. Very little analysis was performed on the speed and memory usage of the algorithms.

[45] uses a combination of blocking, fuzzy matching, and an XGBoost classification model and claims a matching speed of 100 million queries in 56 hours but does not mention how many pairwise comparisons this makes. [43] implements a variety of methods on a testing set with 1,000 rows, with the following results:

Technique(s)	Time (s)
Address-wise fuzzy matching	< 10
Component-wise fuzzy matching	> 100
TF-IDF	~ 40
LSTM + transformer + MLP	~ 1 (with GPU)

An important distinction to make here is queries vs. comparisons. A single query may be compared to an entire reference database, resulting in many comparisons. When applied to NZ, which has approximately 2.3 m residential addresses [24], a comparison

rate of 1,000 pairs/s equates to a matching speed of 1 every 40 minutes (assuming a brute force approach). Experimentation with different techniques was therefore required to balance matching speed and accuracy. This experimentation is described in chapters 4 and 5.

3.4.2 Neighbourhood Search

Neighbourhood searching refers to the process used to evaluate potential candidates when matching addresses. The literature discussed in this review did not cover neighbourhood searching in detail, but a neighbourhood search is required in some form to match addresses. Most of the literature used a brute force approach [44, 58, 16, 29, 10, 30, 60, 13, 11, 59, 3, 57, 27, 26, 43, 32], but blocking was also used occasionally [11, 3, 13].

Brute force approaches tend to be simple but time-consuming. The majority of the literature required a brute force approach due to the nature of the comparison techniques used. For example, when using a classification model to predict if two addresses are a match or not, the query address must be paired with each address in the matching database and fed through the classification model to produce a confidence score.

Nearest neighbour searches are a common problem in computer science, and it was expected that some more sophisticated techniques to partition the search space intelligently may improve performance. Again, these are discussed in chapters 4 and 5.

After reviewing the literature on address-matching methods, a selection of algorithms was chosen for development and testing. The selection process and implementation of these algorithms are described in Chapter 4.

Chapter 4

Methodology

This chapter describes the implementation details of selected methods for the literature survey in Chapter 3, as well as some new methods proposed as part of this research. All methods are implemented as part of a Python Package hosted in the Python Package Index (PyPI). This ensures that the final product can cater to a range of requirements.

4.1 Method Selection

The literature survey provides a good bearing on the techniques that have been successful in matching addresses internationally. However, not all of these were appropriate given the requirements for this project.

Table 4.1 includes a list of methods discovered during the literature review and a justification for their inclusion/exclusion in this project. The table also includes compositional vectors, search trees, and hybrid methods, as these relate to the proposed method and are explored in the coming sections.

Table 4.1: Techniques Implemented

Method	Included	Justification
Dataset generation	✓	As a real dataset of human-entered and validated address pairs was not available, this had to be generated.
Parsing		
CRF	✓	Conditional Random Fields were the most promising statistical method, and libpostal [38] provides a pretrained model.
RNN	✓	RNNs are suitable for shorter sequences compared to transformers and are computationally less expensive. This also provides a good comparison between statistical methods and deep learning methods.
Rules-based	✗	Rules-based parsers were not popular in recent literature due to their inflexibility.
HMM	✗	Hidden Markov Models showed poorer performance compared with CRF models which are implemented.
Matching		
Fuzzy matching	✓	Due to its reliability and simplicity, fuzzy matching provides a useful baseline for comparison.
TF-IDF	✓	As the proposed method is a form of vectorisation, TF-IDF provides the best comparison.
Similarity vectors	✗	Due to the expensive process of computing fuzzy distances for every address component, similarity vectors are not suitable for this project.
Compositional vectors	✓	The proposed method – described in detail in §4.6.4.

Continued on next page

Table 4.1 Continued from previous page

Method	Included	Justification
Custom embedding	✓	A custom embedding model is included for comparison as this is the deep learning equivalent of the proposed method.
Pretrained embedding	✗	Pretrained embeddings offer a lower effort alternative to custom embedding models, but are designed and trained for much more general NLP tasks which makes them excessive for address matching.
Indexing	✗	Indexing was not frequently used in the literature and is more appropriate for low-volume queries with large databases.
Hybrid methods	✓	It was expected that combinations of multiple matching methods into hybrid methods could blend speed and accuracy to produce a superior or balanced method.
Neighbourhood Search		
Brute force	✓	Brute force searching offers a good baseline for comparison with other methods.
Classification models	✓	Classification models were popular in the literature and offer accuracy advantages over brute force. It was suspected that the computational requirements would be too great for this project, but an experiment was conducted in §5.3 to confirm this.
Blocking	✓	Blocking is implemented to measure the trade-off between speed and accuracy compared to brute force.

Continued on next page

Table 4.1 Continued from previous page

Method	Included	Justification
Search trees	✓	Search trees were not included in the literature for address matching, but are described in §4.6.6 and are a common method to improve speed of distance calculations so they will be investigated here.

As the techniques in Table 4.1 are components, they are not all compatible with each other. The techniques need to be used together in a geocoding/address-matching algorithm to perform address matching. This chapter discusses how these techniques are combined to create complete address-matching algorithms.

4.2 Software

Python [53] is used to implement the methods described throughout this chapter and to build a distributable package. PyPI [15] is used to publicly host the package and make it available for users to install.

4.3 Data Acquisition

This section details the datasets acquired for this project.

4.3.1 New Zealand Street Addresses

Land Information New Zealand (LINZ) maintains and publicly provides the New Zealand Street Address (NZSA) dataset [24], which contains structured information of all residential addresses in NZ, including coordinate information. An extract of this dataset is shown in Table 4.2.

address_id	address_number	full_road_name	suburb_locality	town_city	gd2000_xcoord	gd2000_ycoord
1132625	70	Symonds Street	Grafton	Auckland	174.7665433333	-36.85594495

Table 4.2: Extract of LINZ NZSA

As the NZSA contains structured information for the vast majority of NZ addresses, it provides a good foundation for this project¹. The alternative to the NZSA dataset is to use OpenStreetMap (OSM). However, OSM data is based on the NZSA dataset, with every address including a reference to a NZSA address ID. This presents the possibility that OSM may contain outdated information about an address. OSM also stores the unit, address number, and address suffix as a single field rather than separate fields, which is less informative for matching. The NZSA is the preferred dataset for these reasons, though it still has its drawbacks, including:

- The NZSA does not include non-residential addresses. This means that business addresses are not geocodable via this dataset.
- The NZSA does not include postcodes. Postcodes are proprietary to New Zealand Post and are only made available through licence. This means postcodes cannot be looked up by matching to the NZSA and can not be used to improve search efficiency.
- The NZSA does not include some features that are common in real-world addresses, e.g. building names, levels, etc. These features must be present in order to train a model that will generalise well to real-world addresses.
- The NZSA does include features such as unit numbers and suffixes, although these features are replete with blanks as they are less common among NZ addresses. This could introduce bias if the dataset is used to train a machine learning model.
- The NZSA is clean and well-structured. While this is excellent for entity matching purposes, it means that a machine learning model trained on this data may be less generalisable to real-world addresses.

¹Please note that since the time of research, the NZSA has been superseded by the New Zealand Address [25] dataset. Several references are made to the NZSA dataset throughout this thesis as this was the dataset used to conduct the research. The two datasets are roughly equivalent, and the change has no impact on the research conducted.

Solutions and workarounds for these drawbacks are discussed in the upcoming sections of this chapter.

4.3.2 Postcode Network File

As postcodes are proprietary to New Zealand Post, there is no publicly available dataset containing postcodes. However, New Zealand Post does maintain a Postcode Network File (PNF), which is made available via a paid licence. This dataset includes all postcodes in NZ and their associated polygons.

As postcodes are often useful information for address matching, a copy of the PNF was obtained for this research. The algorithms designed as part of this project were made to support geocoding with or without a PNF, depending on whether the user has acquired a PNF licence. The package is not distributed with postcode data.

4.4 Preprocessing

A preprocessing module was designed to handle aspects of the address that can be reliably corrected or removed with regular expressions. This reduces the information present in the addresses that the models have to interpret and allows them to be focused on the more complicated aspects within addresses. The preprocessing module has five steps:

1. Remove NZ: As the NZSA dataset contains only New Zealand addresses, there is no need to extract country information from addresses. Note that this is only removed if it is present at the end of an address to avoid removing NZ from building names, etc.
2. Remove rural delivery routes: Rural delivery routes are used by NZ Post to improve delivery efficiency in rural areas. These are easily found with regular expressions and, being absent from the NZSA, are not helpful for geocoding. Rural delivery routes can, therefore, be removed from addresses.
3. Remove PO Boxes: PO boxes are also not present in the NZSA dataset and can not be geocoded.

4. Correct postcodes: When addresses are constructed by concatenating separate data fields, postcodes (and sometimes house numbers) can have leading zeros stripped (or sometimes become decimal numbers). The preprocessing module replaces numbers in an address like 630.0 with 0630 or 630, depending on their location in the address.
5. Clean whitespace: After all the previous steps have been applied, addresses can become messy with multiple consecutive whitespaces due to removed elements. Correcting this and removing leading/trailing whitespace is the final step in preprocessing addresses.

4.5 Parsing

Two parsers were implemented for this project. The first is a wrapper for libpostal, and the second is an RNN-based deep learning model. By providing these two methods of parsing:

- A comparison can be made between deep learning and statistical methods for address parsing.
- The RNN parser can be used when there is difficulty installing libpostal.
- libpostal can be used in projects where speed is critical, as it is expected to be faster due to being a statistical method.

4.5.1 Recurrent Neural Network

Despite the literature trending towards transformer models for NLP tasks [43, 30, 32, 57, 13], an RNN was chosen to represent deep learning methods for this project for the following reasons:

- Transformer models are typically larger than RNNs due to their attention modules. This would impact the speed criterion for this project.

- Transformers were used in the context of address matching. Address parsing is a simpler task that likely does not require the attention modules used by transformers.
- The use of transformers will be available for comparison as they are used in the address embedding model.

Labels

As the parser modules are intended to be internal modules for the package, their primary purpose should be that the extracted features are useful for matching to the NZSA dataset. A secondary priority is extracting other useful features from the address that are too difficult to identify with regular expressions during preprocessing but should not be discarded when geocoding. The labels chosen to align with these priorities are displayed in Table 4.3.

Label	Description	Example(s)
Blank	Delimiters in the address	Commas, spaces, hyphens
Building Name	The name of a location at an address	Uniservices 70 Symonds Street
Level	The floor of a building	Ground floor, Penthouse, lvl 3
Unit	Sublocations at an address	1/70 Symonds St, Apartment 5, Stall 2
First number	The main street number of the address	70 Symonds Street
First number suffix	An alternative representation of sublocation at an address	70 a Symonds St
Second number	A second number to represent a collection of dwellings in one location	70- 77 Symonds St
Street name	The name of the street	70 Symonds St
Suburb/town/city	Names of the area of the address	70 Symonds St Grafton Auckland
Postcode	Postal code	70 Symonds St 1010

Table 4.3: Address Component Labels for RNN Parser

These labels align well with the NZSA dataset, which means matching algorithms designed to follow parsing will not need to remap fields after parsing.

Dataset Generation

In order to train an RNN for address parsing, representative training data is critical. The NZSA dataset provides a good foundation, but enhancements were necessary to ensure the model would generalise well to real data. This involved generating extra features from a combination of statistical methods and lookup tables (included in

Appendix A, sourced from NZ Post[42], and Auckland Transport [52]), as well as synthesising addresses with a random process to simulate human behaviour.

Table 4.4 shows the result of processing “70 Symonds Street, Grafton, Auckland 1010” ten times.

ID	Result Address
1	mahsa- 5th floor 1/70-75 Symonds St 6523
2	5/70a Symnds Street Grafton Auckland
3	bgo house 70b Smyonds St - Grftnn Acklnd, 8939
4	ground floor cfx wacsdid 3/70f, Symonds Street
5	70-84 Symonds Streyt , Grafton Auckland
6	shed seven 70c Symonds St
7	penthoise 70 Symonds Street Grftn Acklnd 2980
8	hhfesl lvl 3 70b Symonds St., Grftn Adklnd
9	70f, Symonds Street
10	lot 13 70-72 Symonds St , Grafton Auckland 2641

Table 4.4: Example Addresses From Synthesisation Process

This demonstrates the variety of addresses generated from the process. This section explains how these features are generated and combined to form addresses such as these.

Additional Features

Three additional features are created to improve the generalisability of the model. These are building names, levels, and a combination of suburb, town, and city.

Building names are generated from the following random process:

1. Sample the number of words to make up the building name, n , from the reciprocal distribution, $f(x; a, b) = \frac{1}{x[\ln(b) - \ln(a)]}$. Where $a = 1$ and $b = 10$.
2. Let $L \sim Uniform(1, 10)$. Sample l_i from L for $i \in \{1, 2, 3, \dots, n\}$ to determine the length of word i in the building name
3. Randomly sample from the set of lowercase ASCII letters to produce the words in the building name.

4. Select a dwelling type from a lookup table.
5. Stitch the words together with spaces.

Levels are generated from the following random process:

1. Let $X_1 \sim \text{Uniform}(0, 10)$ and $X_2 \sim \text{Uniform}(0, 100)$.
2. Sample the level number, x , from X_1 70% of the time, and from X_2 30% of the time.

The final feature that is generated is `suburb_town_city`, which is generated from combining the `suburb_locality` and `town_city` features into one. This is implemented as many addresses in the NZSA use these fields in a syntactically equivalent way. This could confuse the parsing model, and their distinction is often not important for matching.

Feature Balancing

Some features are present but under-represented in the NZSA dataset. These features are regenerated in order to produce a more balanced dataset.

- Units are sampled from an exponential distribution with $\lambda = 10$ and floored to produce integers.
- Second numbers are generated by taking the first number and adding a random variable sampled from an exponential distribution with $\lambda = 10$ (so that the second number is always higher).
- Suffixes are uniformly sampled from the set of lowercase ASCII letters.

Address Synthesis

Once all the relevant fields for addresses have been generated, they need to be synthesised in a way that resembles real-world, hand-typed addresses.

The synthesis was informed by discussions found in [26], NZ Post's published addressing standards [41], and professional experience working with address data and geocoding in NZ.

The final process is the following series of 50/50 choices:

1. Include the building name?
 - (a) Add dwelling type? E.g. house, building, apartment*
 - i. Add the dwelling type before or after the building name?
2. Include the level number?
 - (a) Select level type, e.g. level, floor, lvl*
 - (b) Select numerical, ordinal, or cardinal format? E.g. level 5, fifth floor, or 5th floor
3. Include the location on the street?
 - (a) Include the unit?
 - (b) Include the first number?
 - (c) Include the suffix?
 - (d) Include the second number?
 - (e) Choose appropriate separators for the resulting selection (e.g., '/', '-', ':', ';')
4. Abbreviate the street type? E.g. Road → Rd., Street → St.*
5. Include the suburb/town/city?
 - (a) Abbreviate the suburb/town/city? This is achieved by removing vowels.
6. Include the postcode?

The components were then stitched together with randomly selected delimiters to form a complete address. Steps marked with * reference a lookup table; these are included in Appendix A.

Typos were then applied using a rate of 0.4% per character. The types of typos applied were:

- *Substitution*: A character is replaced with another that is nearby on a qwerty keyboard.
- *Transposition*: A character is switched with one of its neighbours.

- *Duplication*: A character is repeated by inserting a duplicate.
- *Deletion*: A character is removed from the address string.

The probability of typos was chosen to be 0.4% as the average length of addresses in the NZSA is 35 characters. $35 \times 0.004 \times 4$ gives an expected number of typos per address of 0.56. This expectation produced a dataset that is well balanced between no typos, a single typo, and multiple typos.

As shown in Table 4.4, this process produces a variety of training addresses with known labels. This variation also serves as a form of regularisation for the model and prevents overfitting due to the extra noise included.

RNN Model

This section describes the generic structure of the RNN-based model used. Experimentation with exact model hyperparameters is described in §5.1.

The model uses character-level encodings, a bidirectional LSTM, and feed-forward layers ending in a softmax activation to represent label probabilities. The structure of the full parsing model is shown in Figure 4.1.

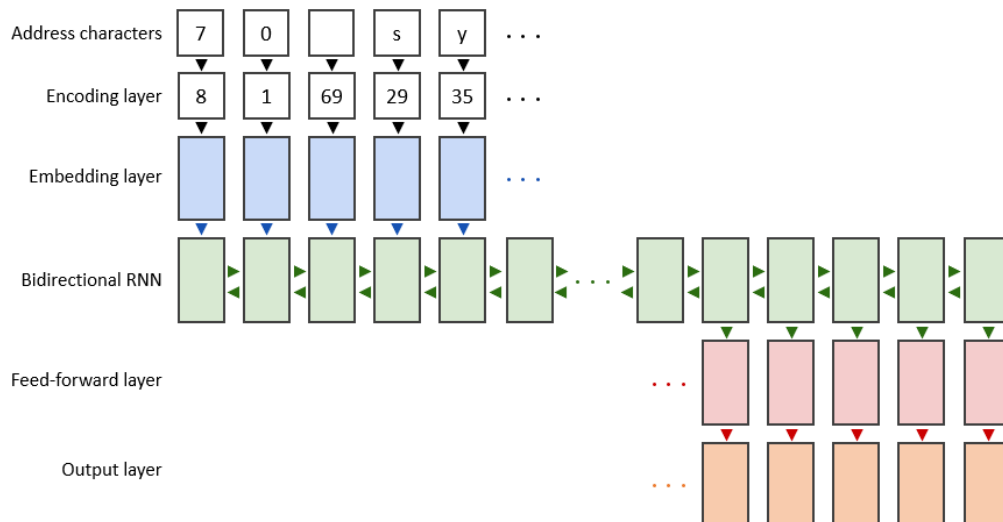


Figure 4.1: Structure of the RNN Parser Model

Encoding Layer

The encoding layer replaces tokens with their index in the vocabulary. This allows the lookups for the embedding layer to use integers rather than tokens, making computational and memory efficiency gains. Unigrams were used as tokens for encoding to keep the vocabulary size as small as possible and remove the risk of out-of-vocabulary errors when a new word is encountered at inference time.

Embedding Layer

This layer learns a vector to represent each character. This enables the model to form a ‘meaning’ for each character.

Bidirectional LSTM Layer

A bidirectional layer of LSTM cells is used to learn sequential information in the address. This means there are two layers of LSTM cells stacked in opposing directions, which allows the model to use information from both ends of a sequence to inform its outputs.

Feed-Forward Layers

The feed-forward layer uses neurons to condense the sequential information from the RNN. This is followed by another dense layer with 10 neurons (the number of labels), with softmax activation to produce a multinomial probability distribution for the output.

The outputs can then be interpreted as a confidence score of a particular character belonging to the corresponding labels. This is visualised in Figure 4.2.

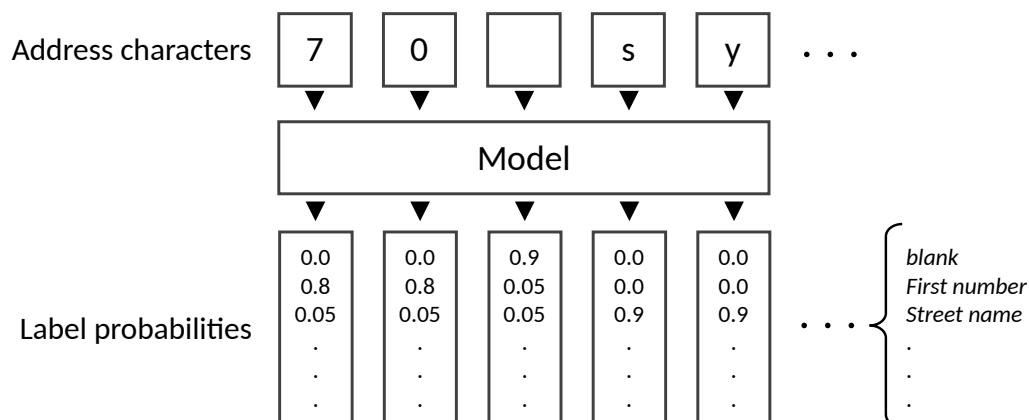


Figure 4.2: RNN Model Outputs

Training

The model is trained using PyTorch, a cross-entropy loss function, the Adam optimiser, and a 0.0001 learning rate.

Cross entropy loss is calculated for each character as:

$$L_{CE} = - \sum_i t_i \ln(p_i)$$

where t_i is the truth label for the i th class (one 1 and the rest 0), and p_i is the softmax probability for the i th class. Therefore, this loss function encourages the model to maximise the probability of the true label for each character.

Postprocessing

Several postprocessing steps were developed to clean the outputs. These are intended to prevent error propagation by removing obvious mistakes made by the model or to improve the output for subsequent steps in the address-matching algorithm. These steps are:

- **Street type normalisation:** Street types are normalised from a lookup table included in Appendix A.
- **Number normalisation:** Cardinal and ordinal forms of numbers are converted to their integer representations, e.g. 1st or first \rightarrow 1. This is applied to levels and units.
- **Number correction:** Non-digit characters are removed from fields that should not have them, e.g. street numbers, level, and postcode.
- **Postcode correction:** Non-digit characters are removed from postcodes. If the postcode is not four digits long, it is removed.

These postprocessing steps allow a parsed address to be used more effectively by subsequent matching models. Table 4.5 shows how a parsed address is more useful

after postprocessing to matching algorithms as the numerical values are correctly represented as integers, and the erroneous postcode has been removed to prevent poor matching.

Address Component	After Parsing	After Postprocessing
level	third floor	3
unit	5th desk	5
street number	70	70
street name	Symonds Street	Symonds Street
postcode	101	None

Table 4.5: Postprocessing for “third floor, 5th desk, 70 Symonds Street, 101”

4.5.2 libpostal

As libpostal already provides a comprehensive address parsing system using CRFs, a statistical method for address parsing was not implemented. Instead, bindings for libpostal are provided. Since libpostal is trained on global OSM data, postprocessing is applied to make the parser outputs more applicable to NZ addresses.

Firstly, the *house number* field needed to be converted. libpostal groups all fields related to the street number into one string. The matching methods that utilise parsing will benefit from having these fields separated.

House Number	Unit	Street Number	Suffix
70		70	
1/70	1	70	
70a		70	A
Apartment 1-70b	1	70	B

Table 4.6: House Number Field Splitting Requirements

Table 4.6 demonstrates some of the conversions that are required. Regex is used to achieve this splitting via the following steps:

1. The *house number* field is split into digits and non-digits (does not include delimiters).
2. For the digits, if there is one group, this is treated as the street number. If there are more than two groups, the first is treated as the unit number, and the second is the street number.
3. For the non-digits, the first group of length one is treated as the suffix.

The aforementioned conversions are made, and the original house number field is retained for debugging purposes.

The remaining labels from libpostal are mapped to their equivalents from the RNN parser. As libpostal is designed to handle addresses in many different countries and languages, several of these labels are either not relevant for NZ or unhelpful for address matching, e.g., *state district* and *near*. These fields are removed during the mapping process. The complete field mapping is demonstrated in Table 4.7.

libpostal Label	Mapped Label
House	Building
Near	-
House number	Unit, first number, and suffix
Road	Street name
Unit	-
Level	Level
Staircase	-
Entrance	-
PO box	-
Postcode	Postcode
Suburb	Suburb town city
City district	Suburb town city
City	Suburb town city
Island	Suburb town city
State district	-
State	-
Country region	-
Country	-
World region	-

Table 4.7: Mapping Table for libpostal Address Labels

The postprocessing module described for the RNN parser is then applied to the outputs of libpostal, completing the parsing process and preparing the addresses for matching.

4.6 Matching

This section describes the implementation of address-matching algorithms—those that take either an unstructured or parsed address and return a most likely match from a reference database.

Matching algorithms typically convert addresses to a numerical representation before performing a distance calculation or nearest neighbour search (NNS) to find the best match. Numerical address representations are often compatible with multiple neigh-

bourhood search methodologies. These are detailed in Table 4.8. Note the inclusion of search trees, which, as mentioned previously, were not mentioned in the literature. Search trees were explored here as it was suspected they could provide significant speed benefits when address matching.

Matching Method	Parser	Compatible NNSs
Fuzzy (address-wise)	✗	Brute force
Fuzzy (component-wise)	✓	Brute force & blocking
TF-IDF	✗	Brute force, classification & search trees
Embedding vectors	✗	Brute force, classification & search trees
Compositional vectors	✓	Brute force, classification & search trees
Similarity vectors (address-wise)	✗	classification
Similarity vectors (component-wise)	✓	classification

Table 4.8: Matching Methods and Associated Neighbourhood Searches

Where possible, subsections for matching methods cover the address representation process, and §4.6.6 covers the general application of nearest neighbour searches to all representations. Where this is not possible, the nearest neighbour search method will be described within the address representation subsection.

4.6.1 Address-Wise Fuzzy Matching

Address-wise fuzzy matching is the process of using a string similarity metric to compare a query address with every address in the lookup database. It is a popular method in the literature and was expected to offer a good baseline performance when analysing other approaches.

The normalised InDel distance was used as the similarity metric. This is a variation on Levenshtein distance that only allows for insertions and deletions (substitutions require an edit distance of two). It was chosen as there is a fast implementation (described in [21]) in C++ available through the Python package RapidFuzz [35]. The InDel distance also considers the order of characters in the two strings. Recall

that this is important as NZ contains many anagrammatical streets, which would not be distinguishable without character-ordering information.

Address-wise fuzzy matching applies the normalised InDel distance to the Cartesian product of query addresses and reference addresses without applying any parser. This is the equivalent of a brute force search and is considered a naive and computationally expensive approach.

During the implementation of this approach, it was noted that the average query time per address exceeded five seconds. This speed is not tractable for larger datasets and does not align with the goals of this project. Therefore, address-wise fuzzy matching was abandoned in favour of component-wise fuzzy matching.

4.6.2 Component-Wise Fuzzy Matching

By making use of a parsed address, address-wise fuzzy matching can be improved by applying the distance function to individual components of the address and their counterparts in the NZSA dataset.

The use of a parser also allows for a variation of blocking to be applied in an iterative approach to continuously reduce the search space. In this variation, filters are applied at each stage in the search to reduce the search space for later steps. There are two kinds of filter steps:

- Exact filters: These filters require an exact match to the search term, e.g., all addresses where the street number is not 70 are removed.
- Fuzzy filters: These filters calculate a similarity score using the normalised InDel distance; scores below a threshold (default 60%) are removed.

Checkpoints are implemented to combat problems with exact filters removing the true address match when typos are present. These checkpoints will reset the search and use fuzzy filters in place of exact filters when no addresses are found.

At the end of the filtering process, a weighted similarity score is calculated based on the importance of each field. If there are no strong contenders remaining, a `NoneType` is returned. The process is visualised in Figure 4.3.

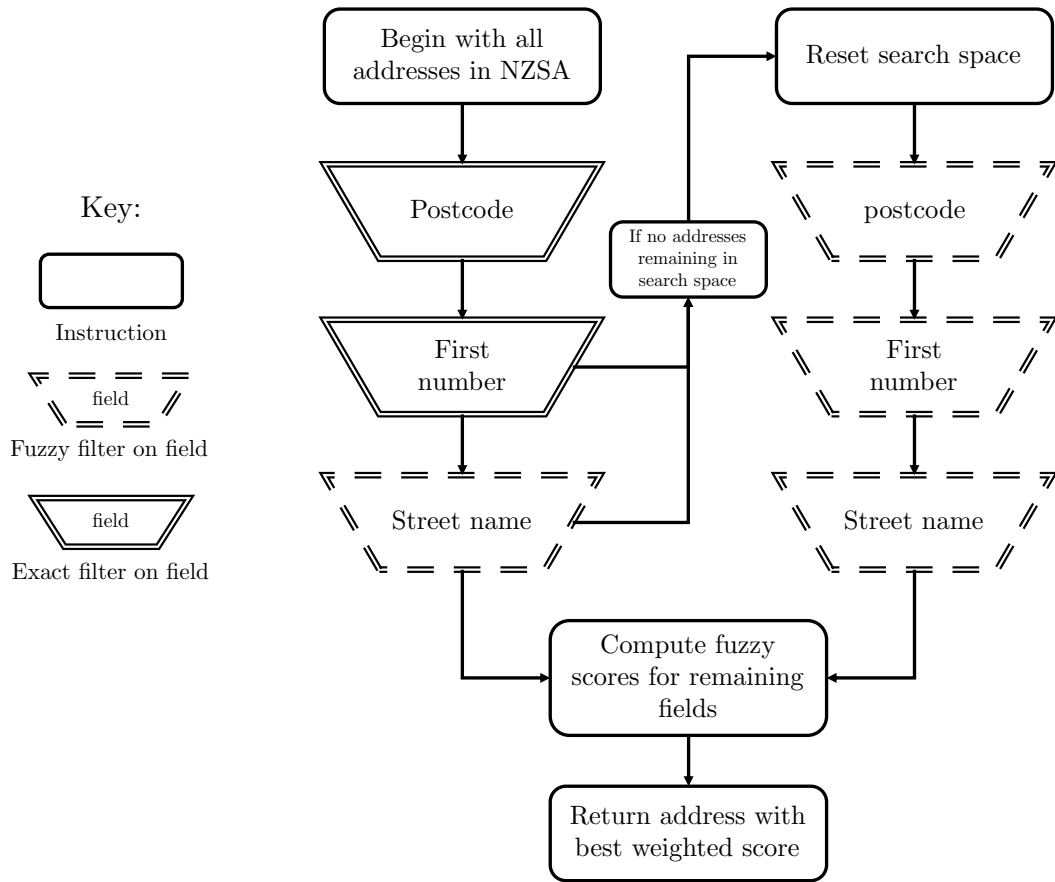


Figure 4.3: Component-Wise Fuzzy Searching Process

This process allows the matcher to iteratively reduce the search space, focusing on the most likely matches while still being robust to typos in important fields such as the number or postcode.

From inspection of the NZSA dataset, it is clear that the suburb/town/city field is the most difficult to match. This is due to the fields in this feature being commonly repeated. For example, ‘Auckland Central, Auckland’, where a human typically writes ‘Auckland’ or ‘Auckland Central’, but not both. Or colloquial and non-standard terms for suburbs being used. To combat this, matching on the suburb is performed later in the process, and less weight is assigned to this match score. In practice, this often results in the matcher finding the best matching suburb for a street name that appears in many different suburbs but also allows the matcher to choose a less favourable street in preference of a significantly well-matching suburb.

4.6.3 TF-IDF

TF-IDF uses a vocabulary of tokens to calculate an importance score for each token based on their frequency in the reference database. Queries are then encoded using these importance scores, and a vector distance calculation is performed to find the closest address.

Hence, TF-IDF is highly dependent on the selected vocabulary. Using words leads to an extremely large vocabulary, so tokens will be used instead. Using an alphabet of lowercase ASCII characters, digits, and the space character, the total number of unique characters used to form tokens is 37.

Bigrams are used as tokens in this implementation, as using either unigrams or trigrams would not be suitable for this project due to the following:

- Using unigrams means each individual character from the vocabulary is used as a token. This means that anagrammatic addresses in the reference dataset will be encoded to identical vectors and will not be able to be matched correctly. Recall that the use of Te Reo Māori in NZ street names is common. As Te Reo Māori contains only 15 letters (including two digraphs), anagrammatic streets are much more likely to be present in NZ addresses. For this reason, unigrams are not suitable for this project.
- Using bigrams *mostly* solves anagrammatic issues. Order is not truly considered, but it would take an extremely contrived example to be problematic. For example, Tamarata Street and Taramata Street contain exactly the same bigrams, just in different orders. While inspection of the NZSA dataset confirmed there were no cases of this, it is also likely that these streets would appear in different areas, providing some distinct bigrams in their vector representation.

Bigrams are, therefore, the minimum viable n-gram size. This conceptually small change has a large impact on efficiency. However, with the vectorised representation of each address increasing to be $37^2 = 1369$ elements long. Using vectors this large with the similarity metrics required by TF-IDF begins to slow down the inference speed as each lookup address needs to be compared to the roughly 2 million addresses in the NZSA dataset. It also requires the use of a sparse matrix, as most computers will not be able to hold the dense array in RAM.

- Using trigrams increases vector lengths to be $37^3 = 50,653$ elements long, resulting in a number of calculations per lookup address that is not feasible within the goals of this project.

4.6.4 Compositional Vectors

Compositional vectors (the proposed method) uses a similar approach to TF-IDF but aim to utilise positional information of characters to reduce the length of vectors, improving inference time while still preserving accuracy.

The design of vectors for this module was a challenging task, and many different methods were explored before the final method was chosen. The key requirements for the vectorisation method were:

- **Uniqueness:** In order to preserve the uniqueness of addresses, vectors had to be checked for collisions. This is when two different addresses are encoded to an identical vector during the vectorisation process. This must be avoided, as the search will not be able to determine a unique match if many addresses are mapped to the same vector.
- **Similarity:** The vectorisation process must map addresses to a unique n -dimensional vector, but it also must contain and represent enough information about the addresses that semantic and structural similarities are preserved in the distance between the vectors. That is, similar addresses such as Symonds Street and Simmonds Street should have vectors that are near each other in the vector space.
- **Speed:** As every address searched will need to be encoded, the process to do this should be simple enough so as not to dominate the overall search time. This means relying on simple rules.
- **Cardinality:** The length of the address vectors should remain as low as possible to keep the distance calculations fast.

Vectorisation

At a high level, the compositional vectors are formatted like:

$$\text{address} = \begin{bmatrix} \text{street number information} \\ \text{street name information} \\ \text{regional information} \\ \text{postcode information} \end{bmatrix},$$

where different components may or may not be included depending on their availability in the address. This leads to embedding vectors of several different lengths. Each component of the address has an individual vectorising function, which typically consists of a rule to convert text to numbers, a transformation function to control the range of values, and a weight/multiplier to represent the importance of the label. The components considered are the house number, street name, suburb/town/city, and postcode.

House Number

The house number segment contains three values:

- Unit: Digits in the unit are summed to give a single value and then logged, giving a range between approx. 0 and 2

$$\log \sum_i d_i$$

where d is the set of digits in the unit.

- First number: The street number is shifted and then logged, giving a range between approx 1 and 3

$$f(x) = \log_{10}(x + 10).$$

- Suffix: The normalised ASCII value of the unit. E.g. A or a = 1/26.

Street name

The street name segment contains 29 values. One for each letter of the alphabet plus a space character, then a digit entry and a length entry.

The first 27 entries in this segment are calculated as the length-normalised geometric mean of character indices (beginning at 1 to avoid multiplying with 0), excluding the corresponding letter/space. This segment represents the positional distribution

of letters within the street name. Any digits in the street name are logged for scale and included in a separate digit entry. This allows the vectors to differentiate between streets, such as ‘12th Street’ and ‘21st Street’, which are identical except for the digits. The length entry is the natural log of the string length.

The geometric mean is an average that uses the product of a set of numbers, as opposed to the arithmetic mean, which uses the sum.

$$gm(\mathbf{x}) = \left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

The geometric mean is used over the arithmetic mean due to its skewed nature. There are multiple ways to arrive at the same result using the arithmetic mean, e.g. $\bar{x} = 5$ for $x = \{4, 6\}$ and $x = \{3, 7\}$. This is not true for the geometric mean, $gm(4, 6) \approx 4.9$ and $gm(3, 7) \approx 4.6$.

This property produces reliably unique numeric representations for the street names. The geometric mean can also be calculated using logs, which is much more efficient by avoiding iterative calculations or integer overflow when calculating the product:

$$gm(\mathbf{x}) = e^{\frac{1}{n} \sum_i \ln x_i}$$

Finally, the values are scaled down by the length of the string. This gives a range of values invariable to the length of the string, and the vectors represent the ‘shape’ of the word being vectorised. For example, the word “glade” would be vectorised like so:

$$\text{glade} \rightarrow \begin{bmatrix} \text{indices of non-a characters} \\ \text{indices of non-b characters} \\ \text{indices of non-c characters} \\ \text{indices of non-d characters} \\ \text{indices of non-e characters} \\ \dots \\ \text{digits entry} \\ \text{length entry} \end{bmatrix} = \begin{bmatrix} gm(1, 2, \cancel{3}, 4, 5) \\ gm(1, 2, 3, 4, 5) \\ gm(1, 2, 3, 4, 5) \\ gm(1, 2, 3, \cancel{4}, 5) \\ gm(1, 2, 3, 4, \cancel{5}) \\ \dots \\ -1 \\ \ln(5) \end{bmatrix} \approx \begin{bmatrix} 2.51 \\ 2.61 \\ 2.61 \\ 2.34 \\ 2.21 \\ \dots \\ -1 \\ 1.61 \end{bmatrix} \approx \begin{bmatrix} 0.502 \\ 0.522 \\ 0.522 \\ 0.468 \\ 0.442 \\ \dots \\ -1 \\ 1.61 \end{bmatrix}$$

The notable benefits of constructing vectors in this way are:

- Independent letter distribution and length representation. This means that insertion or deletion typos near the beginning of the word do not disproportionately affect the representation.
- Non-sparse vectors are resistant to typos due to the flipped index averaging. This works as the address strings are short. If this were applied to longer texts, then the non-inverted indices should be used.
- Semantic preservation.
- Low range of values for consistent scaling during the weighting step.
- Short vectors enabling fast comparisons.
- Retention of digits and their value.
- Simple rules for fast computation.

Suburb/Town/City

Suburbs, towns, and cities are encoded using the same process as street names. They are encoded separately to ensure a better representation, as the inverted index approach works best on short strings. This also allows a lower weight to be assigned to this value, similar to as discussed in the fuzzy §(4.6.2).

Postcode

The postcode is divided by 9999.

Weightings

After all the transforms have been applied, weightings are applied by scaling each component of the address. *Unit* and *suffix* values are assigned a low weight by multiplying of 0.1. Whereas the *first number* and *street name* are assigned high weights of 50 and 100, respectively.

This means that during a distance calculation such as cosine or Euclidean, importance is implied for each component. This allows the algorithm to place more importance on fields such as the number and street name, compared with fields like unit and suffix, which should only be used as tiebreakers between matches.

4.6.5 Address Embedding

Deep learning techniques can be used to create a vectorised representation of addresses with semantic preservation. These are called embeddings. This is essentially the deep learning equivalent of the compositional vectors approach discussed in the previous section.

Dataset Generation

In order to train this deep learning model, the NZSA needed to be augmented and transformed in a similar way to the training data for the RNN parser in §4.5.1. The two common loss functions for training an embedding model are contrastive and triplet loss (see §3.3.3). The performance of these two loss functions can vary depending on the data. A dataset of triplets was generated, as this can easily be reused as pairs for contrastive learning.

Each row in the dataset contains an anchor, positive, and negative case. These triplets were constructed with the following considerations:

- If the triplets are too easy, the model will not fully learn what constitutes a different address.
- If the triplets are too hard, the model will not be able to learn progressively.
- The dataset should support the model's generalisability by including typos and other errors.

The anchor address is taken directly from the NZSA, and positive cases are generated following a similar process to the one described previously in §4.5.1. They are made beginning with raw address components, which are then stitched together with random dropouts, and typos are applied to create a pair of edited but still matching addresses.

The negative cases are generated with a mix of two methods. Twenty percent of the time, negatives are generated by randomly selecting another address from the positive column. The remaining eighty percent of negatives have one of the following edits made, chosen with equal probability:

- unit/number/suffix change;
- street type change;
- street name change; or
- suburb change.

This process generates a dataset composed of 20% easy triples and 80% hard triplets. This balance was intended to provide the embedding model with a sufficient quantity of obvious examples to not stall learning but also include enough difficult addresses that the embeddings will generalise well when applied to real data.

After the address triplets were generated, they were split into pairs of (anchor, positive) and (anchor, negative), with labels of 1 and 0, respectively. These were then concatenated and shuffled to produce a second dataset that can be used with contrastive learning. Tables 4.9 and 4.10 include some examples of the training data generated.

Positive	Negative
70 Symonds St, Gradton	321 Oteha Valkey Road, Albany
70 Symonds Street 1010	70 Symonds St, Christchurch
70 Symonds St 1010	25 Symons Street, Grafton, 1010
70 Symonds Street, Grafton 1010	16 Western Springs Rd, Morningside

Table 4.9: Example Training Triplets Generated for Anchor Address “70 Symonds Street, Grafton 1010”

Address 1	Address 2	Label
70 Symonds Street, Grafton 1010	321 Oteha Valley Road	0
70 Symonds Street, Grafton 1010	70 Symonds St, Gradton	1
70 Symonds Street, Grafton 1010	70 Symonds St, Christchurch	0
70 Symonds Street, Grafton 1010	70 Symonds Street 1010	1
70 Symonds Street, Grafton 1010	25 Symons Street, Grafton, 1010	0
70 Symonds Street, Grafton 1010	70 Symonds St 1010	1
70 Symonds Street, Grafton 1010	16 Western Springs Rd, Morningside	0
70 Symonds Street, Grafton 1010	70 Symonds Street, Grafton 1010	1

Table 4.10: Example Triplets Converted to Training Pairs for Contrastive Loss

Tables 4.9 and 4.10 demonstrate the variety of addresses generated by this process. It also verifies that the training data contains a mix of easy and difficult triplets as intended.

Transformer Encoder Model

To successfully create an address embedding, the embedding model must create a compressed (encoded) representation of the address as a vector. This vector must preserve the semantic and structural meaning of an address and represent it in such a way that similar addresses are near each other in the vector space.

This is a more complicated task than the previous RNN-based deep learning model that was trained for parsing. Hence, a transformer-based approach was chosen, as the attention modules support this form of encoding. Other than this, the approach to designing the address embedding model was similar to the RNN parsing model, described in §4.5.1.

Note that due to the extra complexity of a transformer-based model, more experimentation with model hyperparameters was required (this is covered in §5.2). The generic structure of the models experimented with was:

- Encoding layer: Translates input tokens to a dense encoded vector based on predefined vocabulary.
- Embedding layer: Learns embeddings for each token in the vocabulary. These allow the model to form a meaning behind each token.
- Transformer encoder layers: Applies attention matrix to the sequence of tokens. This takes the variable length input and translates it to a fixed length sequence while considering each pairwise combination of tokens to understand meaning.
- Feed-forward layers: Uses neurons with activation layers to transform the encoding to have a predictable range of values.

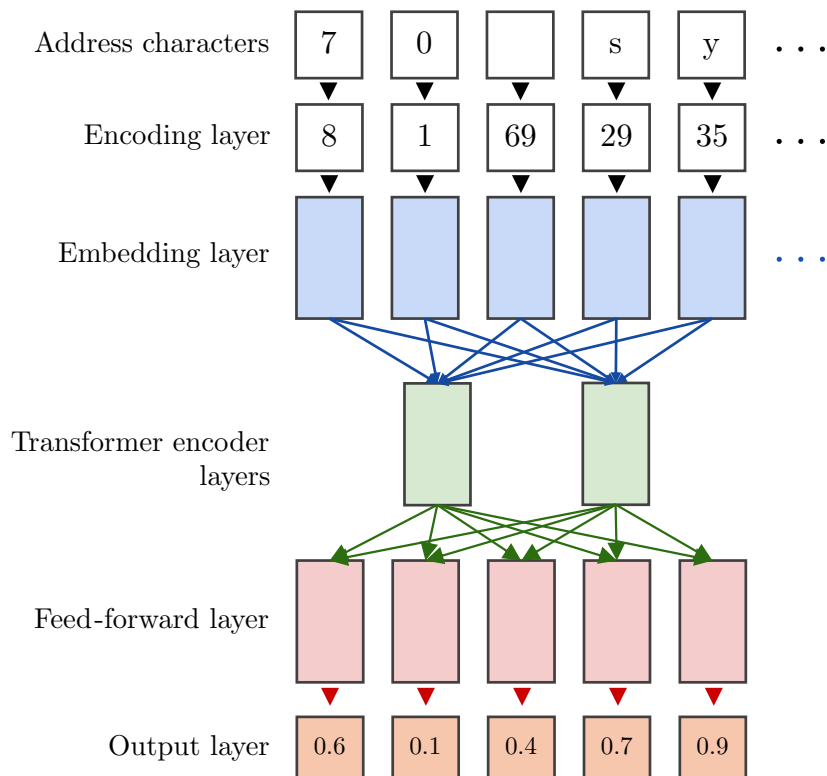


Figure 4.4: Architecture of the Transformer-Based Address Embedding Model

The model is visualised in Figure 4.4. Note the following:

- Unigrams are used as tokens for simplicity in this diagram. Other token sizes were experimented with and could equivalently be shown here.
- Layers preceding the transformer encoder layers are variable length, and layers after the transformer are fixed length.
- The transformer encoder layers are visualised as green boxes. Each layer depends on all input tokens and feeds into each entry in the output. Experiments were not restricted to two transformer encoder layers.
- Output values are controlled to be between 0 and 1 by the feed-forward activation layer.

Training

The model is trained using PyTorch. Tests were conducted using the Adam optimiser for both triplet loss and contrastive loss, with various learning rates.

Batching was implemented to improve training and inference speeds. This requires padding of input vectors within a batch to create equal-length vectors for batch processing. Padding values used are outside of the model vocabulary and are masked out to avoid affecting calculations made inside the model.

4.6.6 Nearest Neighbour Searching

Nearest neighbour searching is an important part of address matching, as it determines how the selected algorithms are applied. This is often where significant computational expense is incurred if the entire address database has to be processed.

Some of the algorithms described in this chapter require a certain nearest neighbour search. For example, component-wise fuzzy matching requires blocking over brute force due to the computational effort required and is not compatible with search trees as the fuzzy distance functions require raw strings. However, the proposed compositional vectors and address embedding models do not have this restriction.

As there was not much information available in the literature for different neighbourhood searching algorithms, this project explored the options available. This section describes the search algorithms considered and how they are applied to vectorised representations of addresses. Due to the precision needed when geocoding, only exact nearest neighbour methods were considered. Approximate methods also exist and are often faster, but these are not guaranteed to return the true nearest neighbour and would result in incorrect address matches. See §5.3 for the experimental results comparing these methods.

Brute Force

Brute force takes the Cartesian product of query addresses and NZSA, then performs a distance calculation. For vectorised addresses, there are two possible approaches:

- **Vector distance:** Euclidean and/or cosine distances are used to compute a vector distance score.
- **Classification model:** A classification model can be trained to compare the vectors and predict a label. This incurs extra overhead on an already computationally expensive approach, so it was not explored for this project.

Search Trees

Search trees spatially index n-dimensional data in order to intelligently navigate the search space. The intuition of a search tree is that for a query, q , if q is far from a point a , and a is known to be near b , then the distance between q and b is not calculated. This avoids having to compare every pair of addresses as in brute force algorithms, though search tree performance can vary depending on the dimensionality of data, as well as the number of data points [2, 22].

KD Tree

KD (K-Dimensional) trees use a binary tree structure to partition data recursively, where K refers to the dimensionality of vectors being organised. This data structure, combined with a traversal algorithm, guarantees an exact nearest neighbour search rather than an approximate nearest neighbour search.

While there are many variations for constructing a KD Tree, they all loosely follow the below process:

1. Choose a starting dimension. This is often the first dimension, but it could be randomly selected or chosen to be the dimension with the largest variance, etc.
2. Generate a splitting hyperplane, e.g. the median of the chosen dimension.
3. Divide data points into each half-space generated by the splitting hyperplane.
4. Select the next dimension. This can simply be the second dimension or selected by another criterion.
5. Generate a new splitting hyperplane in this dimension for each half-space generated by the previous split.

6. Repeat until a stopping condition is met. For example, a maximum depth, minimum size of leaf nodes, etc.

This is demonstrated for 2-dimensional (x, y) vectors like so:

For the set of points:

$$P = \{(1, 9), (2, 3), (3, 7), (4, 1), (5, 4), (6, 8), (7, 2), (7, 9), (8, 8), (9, 6)\}$$

Beginning with the first dimension, x , and all points, calculate the median:

$$x_1 = \text{median}(P) = 5.5$$

Separate points into two half-spaces P_{x_1} and P_{x_2} :

$$P_{x_1} = \{(1, 9), (2, 3), (3, 7), (4, 1), (5, 4)\}, \quad P_{x_2} = \{(6, 8), (7, 2), (7, 9), (8, 8), (9, 6)\}$$

Generate splitting hyperplanes in the second dimension, y , for each set:

$$y_1 = \text{median}(P_{x_1}) = 4 \quad y_2 = \text{median}(P_{x_2}) = 8$$

P_{x_1} and P_{x_2} are then separated into half-spaces based on these values. At this point, the space can be visualised as:

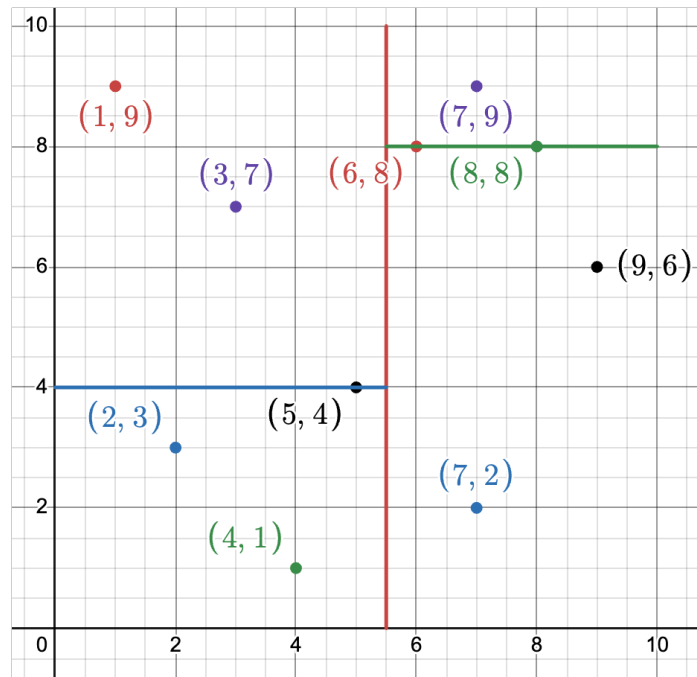
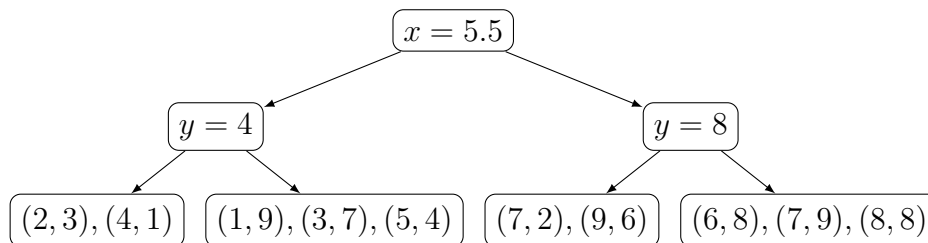


Figure 4.5: 2D Space Partitioned by KD Tree After Two Iterations

with a corresponding KD tree of:



At this point, the tree can continue to grow by splitting each leaf node on the x dimension again, or it can exit as all the leaf nodes contain sufficiently few points.

When querying the KD tree to find an exact nearest neighbour, a comparison is made against each level in the tree to find which leaf node the query vector belongs in. To ensure there are no closer points, the traversal needs to backtrack by checking there are no hyperplanes closer than the closest point within the leaf node; if there are, then these must also be explored to guarantee an exact nearest neighbour is found.

KD Trees have an average-case time complexity of $\mathcal{O}(\log n)$, and a worse-case of $\mathcal{O}(n)$. For high-dimensional data, the curse of dimensionality can result in worst-case

performance due to the increasing sparsity of the data. Exact performance is highly dependent on the data, and since our vectors are often dense and have small distances between them, a KD Tree implementation was tested for this project.

Ball Tree

Ball trees were designed to address the problems KD Trees have with higher dimensions. Instead of using hyperplanes, ball trees partition the data using a series of nested hyperspheres. This results in an increased construction time for the tree but improved query performance as only a single calculation between the query and the hypersphere centroids can give an upper and lower bound for all points contained in the hypersphere.

As with KD trees, exact performance is dependent on data. For this reason, ball trees were tested for comparison to KD Trees in §5.3. As ball tree was not selected for the final implementation, this section is kept brief.

4.6.7 Hybrid Methods

This chapter has discussed four approaches to address matching, they are:

- component-wise fuzzy matching;
- TF-IDF;
- compositional vectors; and
- address embedding.

As mentioned previously in §4.6.6, the compositional vectors and address embedding model approaches are not restricted to brute force searches. For this reason, it was expected that these methods would be significantly faster.

Both of these approaches also rely on compressed representations of the addresses, whereas fuzzy matching and TF-IDF both use raw information to measure similarity. This is likely to cause a reduction in accuracy and/or reliability.

Hybrid approaches use vectorised models to find k nearest neighbours, which are then passed to a raw method (fuzzy or TF-IDF), and may improve matching speed while

maintaining the accuracy and reliability of the raw methods. The exact implementation of a hybrid approach will depend on results from experimentation of nearest neighbour searches. As such, hybrid methods are discussed further in §5.4.

This concludes the Methodology chapter of this thesis. The next chapter (Experimentation) covers the experiments conducted to fine-tune the implemented methods.

Chapter 5

Experimentation

5.1 RNN Parser

The RNN parser includes some hyperparameters that affect the model’s overall performance. In order to produce a parsing model that balances speed with accuracy, combinations of these hyperparameters were tested with a shortened training loop to shortlist promising models. Once the final model was selected, it was trained on the full dataset, and the model weights were saved within the Python package.

5.1.1 Grid Search

A grid search was conducted to search the hyperparameter space. Each model configuration was trained up to 2000 batches (128 batch size) to give an initial indication of the configuration’s learning ability. Table 5.1 includes a list of all hyperparameters searched.

Hyperparameter	Values
Character embedding depth	4, 8, 16, 24
LSTM units	32, 64, 128, 256
LSTM layers	1, 2
Hidden feed-forward layers	0, 1, 2

Table 5.1: RNN Parser Hyperparameter Grid Search

After training, the character accuracy (percentage of characters correctly classified), address accuracy (percentage of addresses with all characters correctly classified), and training iterations per second were calculated using a test dataset of 100,000 unseen addresses. Tables 5.2 and 5.3 include a sample of models trained in the grid search and the resulting scores.

Model ID	Character Accuracy (%)	Address Accuracy (%)	It/s
A	99.33	91.44	4052
B	99.61	94.34	4321
C	99.60	94.34	2930
D	99.67	95.27	3223
E	99.63	95.01	2862
F	99.63	95.04	3500
G	99.71	95.84	821

Table 5.2: Sample of Grid Searched Model Performances

Hyperparameter	A	B	C	D	E	F	G
Character embedding depth	4	8	8	16	16	24	24
LSTM units	32	64	128	64	128	64	256
LSTM layers	1	1	1	1	1	1	2
Hidden feed-forward layers	0	0	0	1	1	1	1

Table 5.3: Sample of Models Made for Grid Search

As expected, the smallest (A) and largest (G) models proved to be the least and most accurate, respectively. However, the range of accuracy scores is very small, with Model

A scoring an address accuracy of 91.44% and Model G scoring 95.84%. The range of speeds, however, is much larger, approximately a factor of five. Of the top-performing models, accuracies were all between 95% and 96%.

The top-performing models also had the longest character embeddings. This indicates that the LSTM layers were able to successfully learn the sequential patterns sufficiently well using the smaller size of 64 units. The most efficient gains for the model are, therefore, in the character embedding depth and the hidden feed-forward layers. Model F was selected for further training, as this provides the best combination of speed and accuracy from the explored configurations.

This logic is confirmed by inspecting the training loss graphs for the models, shown in Figure 5.1, which shows that increasing the model complexity improved the learning speed but plateaued to roughly the same loss.

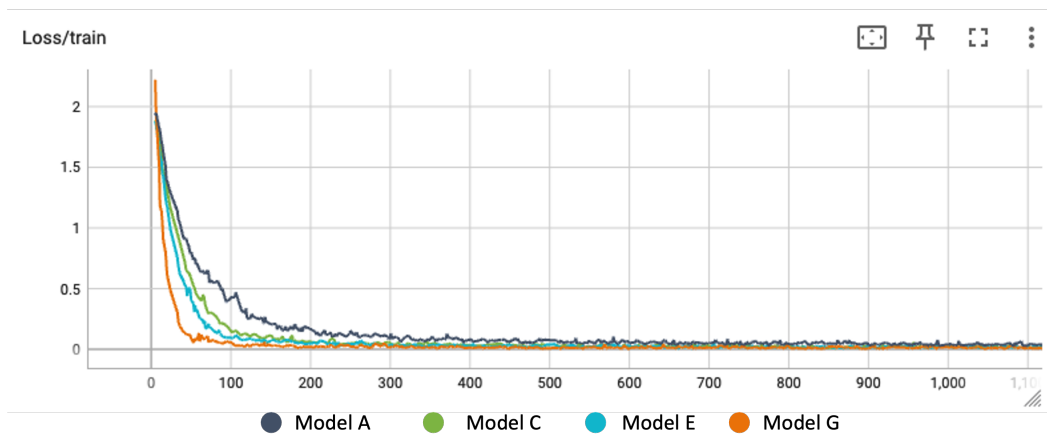


Figure 5.1: Training Loss for Grid Searched Models

5.1.2 Selected Model

Model F was then trained for a full epoch (approx. 14,000 steps) of the training data, using a scheduled learning rate. The final accuracy scores were 99.84 for character accuracy and 97.57 for address accuracy.

5.1.3 Model Inference

An example result from this model for the address “3rd floor 18 viaduct harbour rd” can be visualised as a heatmap (Figure 5.2) using the outputted softmax probabilities for each character. This demonstrates the model’s confidence in parsing each character in the address.

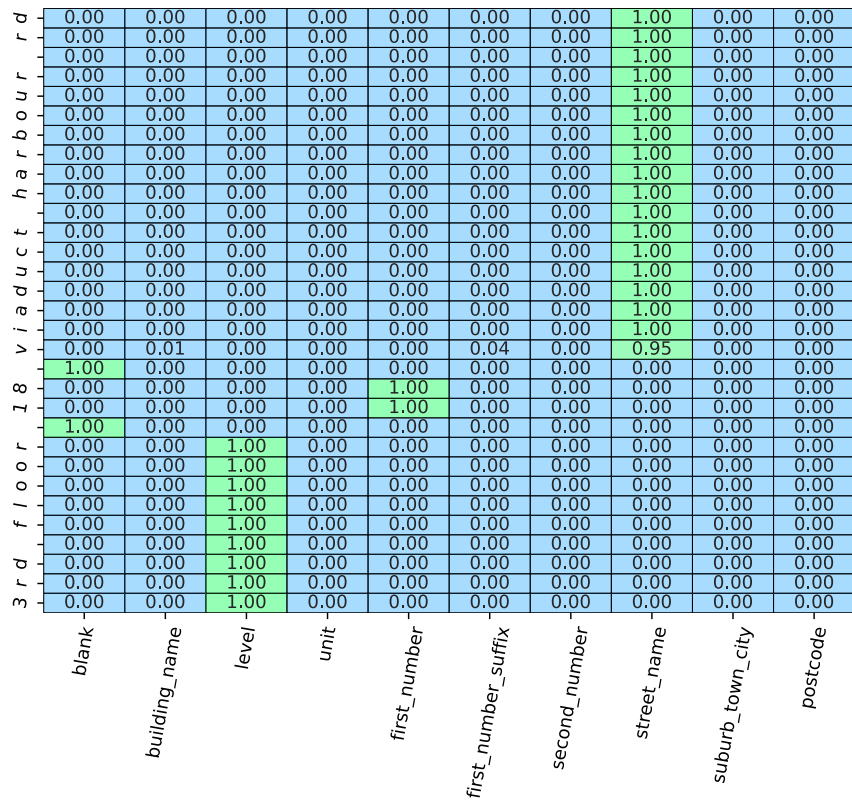


Figure 5.2: Softmax Probabilities/Predictions for Example Input Address

5.2 Address Embedding Model

As with the RNN parsing model, the address embedding model includes several hyperparameters that are expected to affect the model’s accuracy and speed. Table 5.4 includes a list of the hyperparameters for the embedding model and associated potential values.

Hyperparameter	Values
Tokens	Unigrams, bigrams, trigrams
Token embedding size	8, 16, 32
# attention heads	2, 4, 8
# transformer layers	1, 2, 4
Hidden dimension	64, 128, 256
Output dimension	32, 64, 128

Table 5.4: Address Embedding Model Hyperparameter Values

To understand the effects of each hyperparameter, four initial models were trained for a short time (approx. five minutes). Table 5.5 describes the parameters used for each model.

Hyperparameter	A	B	C	D
Token embedding size	8	16	32	32
# attention heads	2	4	8	8
# transformer layers	1	2	2	4
Hidden dimension	64	128	128	256
Output dimension	32	64	128	128

Table 5.5: Address Embedding Models Tested

These models were initially trained on the triplet loss dataset. During this training, it was observed that the training loss of each model was plateauing earlier than expected. Figure 5.3 includes a screenshot taken from TensorBoard (software for tracking learning progress). The screenshot shows training loss over time for a selection of models, as well as some key stats for each training run. Note that “Model A - bigrams - triplets” refers to a training run using Model A, training with bigrams as tokens, and using triplet loss.

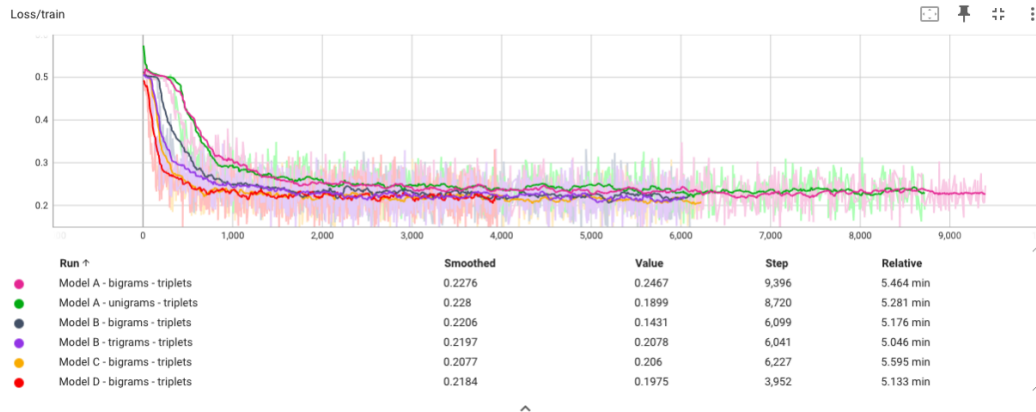


Figure 5.3: Training Loss (Triplet) for Address Embedding Models

The training loss curves show that each variant of the model faced a 'hump' at training losses of around 0.5. This is explained by the margin used for triplet loss, which was set at 0.5. As expected, the larger model sizes demonstrate the ability to learn faster. This is shown by the steeper curves as training begins. However, each model seems to plateau around the same loss value of slightly over 0.2. There also seems to be little difference between the use of unigrams, bigrams, and trigrams as tokens. Smaller models also complete more steps in the same timeframe, as expected.

The loss curves also show a large amount of variation between batches. This is likely due to the mixing of hard triplets described in §4.6.5. For an embedding model to work successfully for address matching, loss values are required to be much nearer to zero. As triplet loss did not appear to be providing the results needed, contrastive loss was also tested.

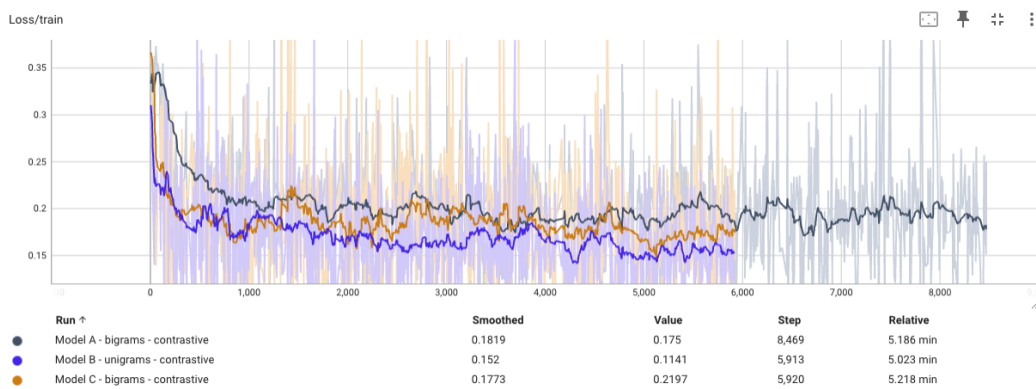


Figure 5.4: Training Loss (Contrastive) for Address Embedding Models

Figure 5.4 demonstrates that training the models with contrastive loss appears to have a similar issue of plateauing training loss. At this point, the following potential issues and resolutions were considered:

- The models tested may not contain enough parameters to learn to distinguish between the training addresses. This is unlikely, as models A, B, C, and D all appear to have similar performance, even though they are significantly different sizes.
- Training data may be too difficult. To combat this, the hard negatives introduced in the data generation step could be removed or their frequency reduced. This would make it easier for the embedding model to distinguish between the training addresses. If the models still fail to converge, they could be too small, or the addresses are still simply too difficult to distinguish between using only cosine distance.
- Learning rate may be too high, preventing the models from converging. A lower learning rate could be tested, or a learning rate scheduler could be used.
- Insufficient training time. It's possible that the extra complexity of transformer models just requires extra training time.

As model size is least likely to be the problem, another test was conducted with the following changes:

- A new dataset was generated, removing all hard negatives and using only easy negatives.
- The optimiser's learning rate was reduced from $1e - 3$ to $1e - 4$.
- Models A and B were trained on the new dataset.

Figure 5.5 shows updated loss curves for the newly generated easy triplets training set. One of the models from the previous round of training is included for reference.

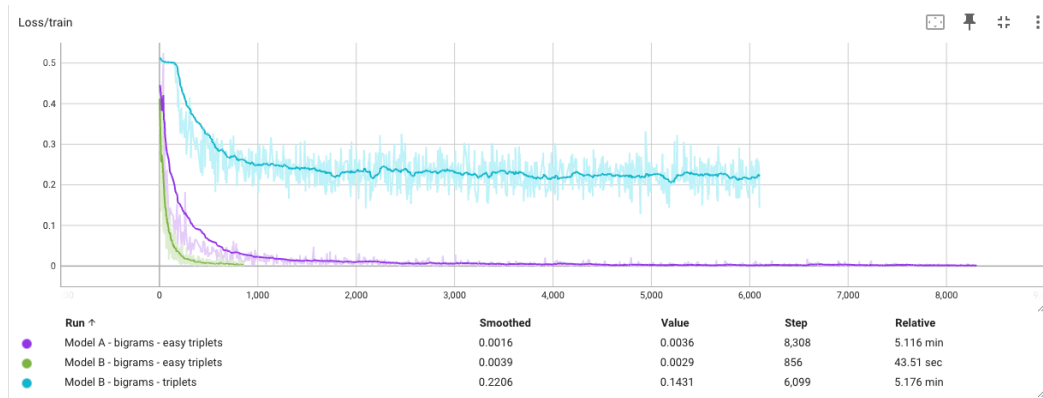


Figure 5.5: Training Loss for Address Embedding Models Using Easy Triplets

The models trained on the easy triplets loss curves trend quickly towards zero. This shows that the problem was with the complexity of the training data. As such, two new training datasets were generated, one with 10% hard triplets and one with 30% hard triplets. Model A was shown to take a longer time to reach zero training loss for the easy triplets, so Model B was chosen to continue development.

The embedding model was then progressively trained on each dataset to gradually increase the number of hard triplets, making the training data more difficult and guiding the model towards good embeddings. Scheduling functions were used to reduce the learning rate, and the margin used in triplet loss was decreased with each training set. This allows more difficult triplets to be closer together in the embedding space.

After completing the progressive training process, anecdotal testing was performed to gauge how well the embedding model separates the addresses. This involved random sampling of 30 addresses from the NZSA; these are presented in Table 5.6.

ID	Address
0	138 Fortescue Street, Mahia 4198
1	34 Glenshea Street, Putaruru 3411
2	154 Kohimarama Road, Kohimarama, Auckland 1071
3	66 Bayly Road, Blagdon, New Plymouth 4310
4	164 Halfway Bush Road, Mount Grand, Dunedin 9076
5	4/32 Avalon Drive, Nawton, Hamilton 3200
6	35 Premier Avenue, Point Chevalier, Auckland 1022
7	4/7 Riversdale Road, Avondale, Auckland 1026
8	961 Kaipara Hills Road, Kaipara Flats 0984
9	3 Cuba Street, Petone, Lower Hutt 5012
10	14 Scotston Avenue, St Albans, Christchurch 8052
11	290 Heatherlea East Road, Levin 5571
12	133 Tanner Street, Grasmere, Invercargill 9810
13	510/22 Herd Street, Te Aro, Wellington 6011
14	70 Devon Street East, New Plymouth 4310
15	38 Beach Parade, Oneroa, Waiheke Island 1081
16	48 JG Wilson Drive, Waipukurau 4281
17	14 Bine Crescent, Orewa 0931
18	4D Coronation Court, Milton 9220
19	12 Honeystone Street, Helensburgh, Dunedin 9010
20	10 Hatfield Way, Huntington, Hamilton 3281
21	64 Archboyd Avenue, Mangere East, Auckland 2024
22	61 William Street, Waihi 3610
23	1/53 Carruth Road, Papatoetoe, Auckland 2025
24	48 Meadowbank Road, Meadowbank, Auckland 1072
25	682 South Titirangi Road, Titirangi, Auckland 0604
26	40 Rennie Drive, Mangere, Auckland 2022
27	20 Ariki Road, Hataitai, Wellington 6021
28	40 Matos Segedin Drive, Leamington, Cambridge 3495
29	24 Tasman Road, Otematata 9412

Table 5.6: Randomly Sampled Addresses for Initial Testing purposes

Addresses were then selected from this sample and adjusted in a variety of ways before being embedded and compared back to the sample database. The results of this testing are as follows:

Test: Untouched addresses should have perfect similarity

✓

Query: '138 Fortescue Street, Mahia 4198'

Matched To: '138 Fortescue Street, Mahia 4198'

Match Similarity: 1.0

Next Best Match Similarity: 0.7029

Average Non-Match Similarity: -0.0163

This is a simple test of the similarity metric and that embeddings have been recorded correctly. This should always result in a similarity of 1, as it has here. This test was successful.

Test: Removing postcodes

✓✓✓

ID: 1

Query: '34 Glenshea Street, Putaruru'

Matched To: '34 Glenshea Street, Putaruru 3411'

Match Similarity: 0.9329

Next Best Match Similarity: 0.6211

Average Non-Match Similarity: 0.02659

ID: 2

Query: '154 Kohimarama Road, Kohimarama, Auckland'

Matched To: '154 Kohimarama Road, Kohimarama, Auckland 1071'

Match Similarity: 0.9498

Next Best Match Similarity: 0.6275

Average Non-Match Similarity: -0.01762

ID: 3

Query: '66 Bayly Road, Blagdon, New Plymouth'

Matched To: '66 Bayly Road, Blagdon, New Plymouth 4310'

Match Similarity: 0.9647

Next Best Match Similarity: 0.4415

Average Non-Match Similarity: -0.04945

As postcodes are relatively unique between addresses (in this database and in a training batch), it was tested that the model does not rely on postcodes to match addresses successfully. In this test, all match similarities remained high, and the next-best matches remained low. This is a good outcome, indicating the model understands what represents an address and does not exploit shortcuts.

Test: Removing area information

✓✓✓

ID: 4

Query: '164 Halfway Bush Road'

Matched To: '164 Halfway Bush Road, Mount Grand, Dunedin 9076'

Match Similarity: 0.7325

Next Best Match Similarity: 0.5997

Average Non-Match Similarity: 0.03778

ID: 5

Query: '4/32 Avalon Drive'

Matched To: '4/32 Avalon Drive, Nawton, Hamilton 3200'

Match Similarity: 0.8272

Next Best Match Similarity: 0.7164

Average Non-Match Similarity: 0.08559

ID: 6

Query: '35 Premier Avenue'

Matched To: '35 Premier Avenue, Point Chevalier, Auckland 1022'

Match Similarity: 0.9004

Next Best Match Similarity: 0.6572

Average Non-Match Similarity: 0.09345

This test measures how well the model handles a large reduction in tokens. The addresses should clearly be matched, but the semantic meaning is different as there is no area information. The similarity scores of the best match have dropped somewhat, but the non-match scores have also dropped a commensurate amount.

Test: Removing a variation of fields

✓✓✓

ID: 7

Query: '4/7 Riversdale Road 1026'

Matched To: '4/7 Riversdale Road, Avondale, Auckland 1026'

Match Similarity: 0.8647

Next Best Match Similarity: 0.6550

Average Non-Match Similarity: 0.05968

ID: 8

Query: '961 Kaipara Hills Road Flats'

Matched To: '961 Kaipara Hills Road, Kaipara Flats 0984'

Match Similarity: 0.9729

Next Best Match Similarity: 0.5032

Average Non-Match Similarity: -0.01369

ID: 9

Query: '3 Cuba 5012'

Matched To: '3 Cuba Street, Petone, Lower Hutt 5012'

Match Similarity: 0.7916

Next Best Match Similarity: 0.5793

Average Non-Match Similarity: 0.02816

This test confirms the model learned to handle information gaps. The similarity scores for matches have remained high, and non-matches are relatively low.

Test: Typo in street name and remove some area information ✓

ID: 10

Query: '14 Scotstn Avenue, St Albans'

Matched To: '14 Scotston Avenue, St Albans, Christchurch 8052'

Match Similarity: 0.8312

Next Best Match Similarity: 0.4038

Average Non-Match Similarity: -0.05224

This test tampers with the street name and city, which are both crucial parts of the address. The match similarity has reduced, but it is still significantly higher than the non-match similarities.

Test: Borrowing street numbers and postcodes from other addresses ✓✓✓

ID: 11

Query: '280 Heatherlea East Road'

Matched To: '34 Glenshea Street, Putaruru 3411'

Match Similarity: 0.7842

Next Best Match Similarity: 0.7421

Average Non-Match Similarity: 0.0880

ID: 12

Query: '133 Tanner Street 9810'

Matched To: '133 Tanner Street, Grasmere, Invercargill 9810'

Match Similarity: 0.6901

Next Best Match Similarity: 0.6694

Average Non-Match Similarity: 0.06077

ID: 13

Query: '133 Herd Street'

Matched To: '510/22 Herd Street, Te Aro, Wellington 6011'

Match Similarity: 0.8021

Next Best Match Similarity: 0.7939

Average Non-Match Similarity: 0.07976

These tests tamper with information by borrowing it from other addresses, this measures how well the embedding model can prioritise parts of the address, and look at the address as a whole. The addresses have successfully matched, but the similarity scores are much closer to the next best matches than previously. This could be problematic when extending to the complete NZSA database.

Due to concerns about the extensibility of this embedding model, this test was repeated using the full NZSA database. In this more complicated test, queries with IDs 1–10 and 12 were matched accurately; however, queries 11 and 13 were mismatched to addresses that share similar tokens:

ID: 11

Query: '280 Heatherlea East Road'
Matched To: '280 Hinuera Road, Matamata'
Match Similarity: 0.9745

ID: 13
Query: '133 Herd Street'
Matched To: '8C/33 Hunter Street, Wellington Central, Wellington'
Match Similarity: 0.9586

Inspection of the NZSA reveals that these specific addresses don't exist on the streets specified (either because they were modified or the address is not included as it is non-residential). However, this was still considered problematic as the preferred behaviour for a geocoder in these instances is to return a match where the street is accurate and the street number is as close as possible.

This indicated that the model was placing too much emphasis on the street number. Significant time and several strategies were used in an effort to combat this. However, they were largely unsuccessful. For this reason, they are covered only briefly here:

- Increasing hyperparameter values to create larger models capable of learning deeper address representations.
- Training the models for longer periods of time and with reduced learning rates and loss margins.
- Introducing a positional encoding layer. Positional encoding layers are used to identify where each token has come from in the address. These are typically used with transformer models but are more impactful for sequence-to-sequence tasks such as language translation. Positional encoding is described in detail in [54].
- Introducing an LSTM layer. This was used in another attempt to capture sequential information due to its success in the parsing model.

As these attempts were unsuccessful, it is likely that the address embedding approach is limited by one of two things:

- How much information can be learned and stored in relatively short embedding vectors. When compared with TF-IDF vectors, for example, these are 10-20 times smaller.
- A more complete dataset may be required for training. This might involve labelled similarities rather than binary labels.

The embedding model was not pursued further, as alternative approaches demonstrated greater promise within the available time frame. It was also decided to only develop the hybrid methods discussed in §4.6.7 based on the compositional vectors method, as this was more promising than the embedding model.

5.3 Neighbourhood Searching

Due to the abandonment of the address embedding model, this section covers only experimentation done for neighbourhood searching on the outputs of the compositional vectors (see §4.6.4) model.

Recall there are four nearest neighbour search methods appropriate for use with these vectors:

- brute force distance calculation;
- brute force classification model;
- KD tree; or
- ball tree

Cosine and Euclidean are the most popular distance metrics in the literature, so these were chosen to test brute force. SciPy [55] and scikit-learn [39] both provide implementations of KD trees, while only scikit-learn provides an implementation of ball tree. These will be used to evaluate the search tree algorithms.

The following process was used to evaluate the speed of these search methods for all $n \in \{1, 5, 50, 100, 500\}$:

1. n addresses were randomly selected from the NZSA.
2. Addresses were vectorised using the compositional vector method.
3. Address vectors queried against the entire NZSA dataset using each nearest neighbour method.
4. Record total query time.
5. Repeat 10 times; measure average duration and standard deviation.

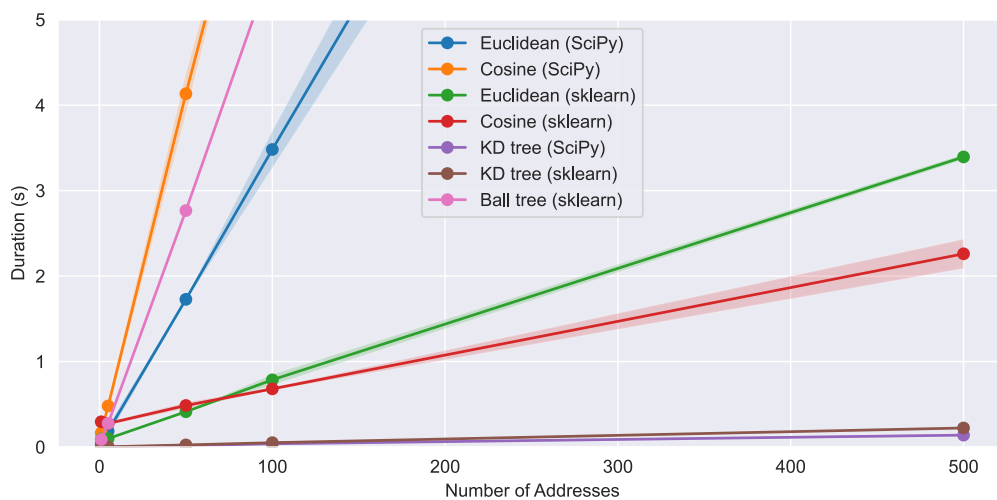


Figure 5.6: Speed Performance Comparison of Nearest Neighbour Search Methods

Figure 5.6 demonstrates the durations achieved for each search method. Ball tree was surprisingly slow when compared to KD tree, as it is intended to deal with higher dimensional data. KD trees significantly outperformed brute force approaches, with the SciPy implementation having a speed advantage compared to the scikit-learn implementation. As each of these methods returns the best match based on a distance calculation, and the cosine distance is equivalent to Euclidean distance on normalised vectors, speed is the only concern here. This makes the SciPy implementation of KD tree the top contender.

The only remaining method to test is classification. This required a different approach, as accuracy and speed needed to be evaluated. The classification models selected for this evaluation were random forest, logistic, and a decision tree classifier, as these were the most common in the literature survey (see §3.3.3).

Two approaches were taken to fit these classification models. Both of these rely on the dataset generated for contrastive loss in the address embedding model. Recall that this dataset contains pairs of addresses and a label of 1 or 0 to indicate whether the addresses match or not. This dataset was used in two ways.

The first test concatenated the vectorised addresses together as raw features for the models. That is, for n dimensional address vectors, the training set has $2n$ columns. This can be visualised using dummy address vectors like so:

$$a_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a_2 = \begin{bmatrix} 0.9 \\ 0 \\ 0 \end{bmatrix}, a_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, a_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Assuming a_1 and a_2 are a match (label of 1), and a_3 and a_4 are not a match (label of 0), the training set would be:

$$\left[\begin{array}{ccc|ccc|c} 1 & 0 & 0 & 0.9 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array} \right].$$

This is essentially providing the models with the raw data. The second test calculated the element-wise absolute difference between address vector pairs. The intuition behind this training set is that the vectors were designed to align in the fashion of a distance calculation, so this should help the classification models understand interactions between the two addresses. Using the same dummy data, this training set would be:

$$\left[\begin{array}{ccc|c} 0.1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array} \right].$$

In each case, 20% of the data was withheld as a testing set, and the classifier models were trained on the remaining data. The testing accuracy was measured for each model and test set, as well as the inference duration. The results from this experiment are presented in Table 5.7.

Classifier	Acc. 1 (%)	Dur. 1 ms	Acc. 2 (%)	Dur. 2 (ms)
Logistic	63	5.55	83	2.67
Random forest	81	1,040	98	324
Decision tree	78	25.2	97	4.90

Table 5.7: Results from Classifier Experimentation

The results in Table 5.7 demonstrate that the random forest classifier is the most accurate, as expected, but it is also significantly slower. The logistic classifier is much faster but has a significantly lower accuracy. The decision tree classifier strikes the best balance between the two metrics.

The second dataset is also clearly the better approach, improving both accuracy and inference duration. The downside of this method is the extra pairwise computation required. Calculation of the absolute differences was completed at approx. 1.7m vector pairs per second. Given the NZSA dataset is over two million rows long, approaches using this training set will not be able to achieve one address match per second.

Given the low accuracy of classifiers using the first dataset and the poor speed performance for generating the second dataset, it was decided not to pursue classification models as a form of nearest neighbour search any further.

The top contender for nearest neighbourhood searching was, therefore, the KD tree implementation from SciPy, and this search strategy was implemented for use with the compositional vectors and associated hybrid methods.

5.4 Hybrid Methods

Recall that hybrid methods were discussed briefly in §4.6.7 as a potential approach to balance the speed of models that compress address data (compositional vectors and address embedding) and the accuracy of those that work on raw address data (fuzzy and TF-IDF).

The speeds demonstrated by KD trees in §5.3 show promise for hybrid methods. Testing was therefore conducted to measure the effective increase in speed from using hybrid methods. Note that the complications with the address embedding model

resulted in a larger-than-expected model. This nullifies the potential speed benefits of hybrid methods. Therefore, this testing was conducted only for the compositional vectors method instead of both as previously planned.

Therefore, two hybrid methods were selected:

- Fuzzy Hybrid: Using KD trees to query the k nearest points for each address, then selecting the corresponding k rows in the NZSA dataset to use as the search space for component-wise fuzzy matching.
- TF-IDF Hybrid: Using KD trees to query the k nearest points for each address, then selecting the corresponding k rows in the TF-IDF matrix. As with the TF-IDF method, cosine similarity can then be used to find the best match between TF-IDF vectors.

The parameter k will determine the balance between speed and accuracy. This is expected as returning more points will both reduce the query speeds of the KD tree and increase the computational effort required for the final similarity step. However, the extra rows returned also produce more addresses to be evaluated in the final similarity step, which should provide more accurate results.

For a hybrid method to have value beyond the individual methods, it must be at least as fast as the slower method (e.g., the fuzzy hybrid method must be at least as fast as the regular fuzzy matching method).

To experiment with the range of acceptable k values, 10,000 addresses were randomly sampled from the NZSA, then fed through the hybrid models to measure matching speed. Note that accuracy is not measured here as it would be 100% due to the use of actual addresses from the reference database.

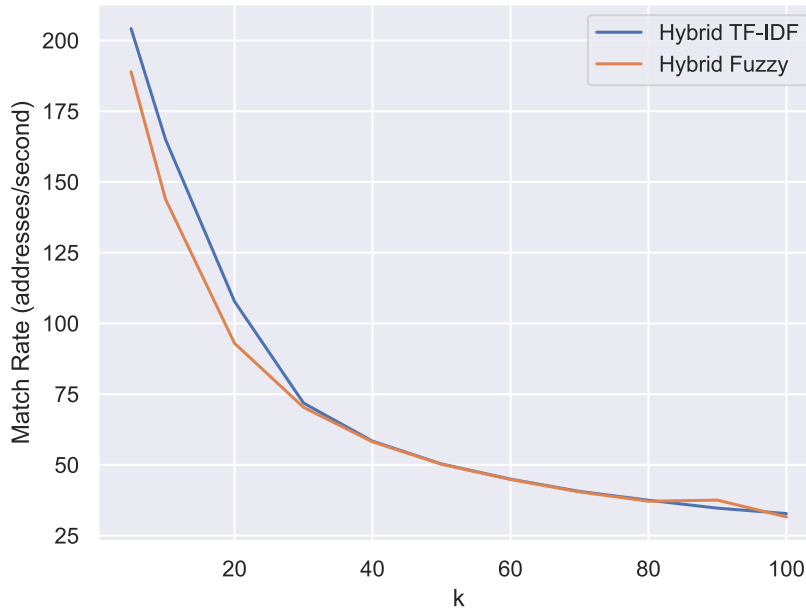


Figure 5.7: Hybrid Method Match Rates for Different k Values

Figure 5.7 demonstrates the decline in matching speeds for larger values of k , as expected. Both hybrid methods display similar performance, which is expected as most of the computation in matching is now offloaded to the KD tree. That is, for a k value of 100, a KD tree returns the 100 best matches (based on compositional vectors) out of approx. 2 million potential matches. Therefore, the fuzzy and TF-IDF similarity measures work on a significantly reduced dataset.

The relationship between k and match rate suggests that values of < 80 are most appropriate for k , but the best value for the parameter is likely to depend on the exact data and user need. Therefore, 80 is used as the default value, and k is exposed as an argument for the user to select.

This concludes the Experimentation chapter. This chapter covered the experiments conducted for the RNN parsing model, the reasons behind abandoning the address embedding approach, nearest neighbour search methods, and hybrid approaches to address matching. The next chapter will discuss the performance of the final methods on real-world data and draw comparisons to alternatives available in the market. The next chapter will also cover potential next steps for this project.

Chapter 6

Results and Discussion

This chapter includes an evaluation of implemented models and comparisons to a selection of alternative solutions available in the market. This chapter also covers potential gaps in the analysis and improvements that could be made in the future.

6.1 Evaluation

This evaluation section includes analyses of the parsing and matching algorithms' performance in isolation and holistically. It also covers the dataset used to evaluate these algorithms.

6.1.1 Evaluation Dataset

NZ Post is a national courier and mail delivery service in NZ. In order to help fairly evaluate the models discussed, NZ Post was able to provide a real dataset of addresses they had recently received for deliveries. This dataset contained 2,500 delivery addresses.

From a visual inspection of the dataset, it was clear that these addresses were of fair quality. While this is indicative of the improvements made in recent years with many online services using address validation, the dataset does not include a good sample

of ‘messy’ addresses. The ability to handle messy addresses was a goal of this project, as many legacy datasets of addresses were created prior to the widespread adoption of address validation services.

As a dataset containing messy addresses was not able to be obtained for this evaluation, a GPT model from OpenAI was used to augment the base dataset. The GPT-4o model was programmatically called with two different prompts, instructing the Large Language Model (LLM) to make ‘realistic’ and ‘aggressive’ adjustments to every address provided from the following list:

- Typographical errors, including substitutions, additions, deletions, transpositions, and duplications.
- Abbreviations where possible.
- Phonetic and transcription errors.
- Randomly including or excluding the city, postcode, and suburb fields.
- Any other common representations expected from human-entered addresses.

The prompt used is included in Appendix D. This was implemented in batches of 50 addresses to avoid token and memory limits for the LLM. The output of these prompts then underwent human review and adjustment. Table 6.1 is an excerpt of this dataset to indicate the level of adjustments made.

Type	Address
Nice	101 Roberts Street Taupo 3330
Realistic	101 Robberts St Taupo 3330
Aggressive	101 Robits St Taopo
Nice	238 Fitzherbert Avenue West End Palmerston North 4410
Realistic	238 Fitzherbert Ave West End Palmerston North
Aggressive	238 Ftzhrbt Av Wst End Plmston N 4410
Nice	96 Symonds Street Grafton Auckland 1010
Realistic	96 Simonds Street Grafton AKL
Aggressive	96 Symnd St Grafton
Nice	214 Queen Street Auckland Central Auckland 1010
Realistic	214 Qeen Street Auckland Central
Aggressive	214 Qn St Akl Cntrl

Table 6.1: Evaluation Dataset Example

This shows:

- Nice addresses are well formatted and include all the appropriate fields.
- Realistic addresses contain 1–2 adjustments that are in line with human-entered addresses and are still clearly understandable.
- Aggressive addresses often contain multiple or stretching adjustments that a human would be unlikely to enter but can still understand.

This approach creates an evaluation dataset comprised of three tiers of difficulty. This enables the following analysis to measure geocoding performance on fair geocoding tasks, as well as measure their robustness to poorly formatted search queries.

6.1.2 Geocoding Errors

When evaluating geocoding results, it is useful to categorise the results beyond just correct or incorrect. This is because different geocoders will be working from different databases and may have slight variations in coordinates, missing addresses, etc. Using

extra categories helps understand how well the geocoder performs and handles these problems. The categories used for this analysis are described below in Table 6.2:

Type	Definition(s)	Considered Correct
Perfect	<ul style="list-style-type: none"> Address matched and coordinates correct 	✓
Near perfect	<ul style="list-style-type: none"> Address matched and coordinates within 500m Street is correct with a small error in street number and coordinates 	
Slight mismatch	<ul style="list-style-type: none"> Address information is partially correct but in the wrong area or street 	✗
Total Mismatch	<ul style="list-style-type: none"> Address incorrectly matched Address is in the wrong country 	

Table 6.2: Evaluation Categories

These categories are ordered from most to least desirable, with the following justifications:

- Perfect match: This is the most desirable as the correct coordinates have been returned, and the address has been matched perfectly. Ideal for data analysis.
- Near perfect: These coordinates are still usable for most applications. The geocoder is likely working off a database with missing addresses or slightly incorrect coordinates.
- Slight mismatch: The geocoder has nearly made the correct match but may be in the wrong area; the results are likely partially usable.
- Total mismatch: There is no clear link between the searched address and the returned one. Unusable for most applications.

6.1.3 Alternative Solutions

There are several commercial and open-source alternatives available for geocoding. This section evaluates their performance using the evaluation dataset, considering key factors such as processing speed, cost structure, accuracy, and API usage limitations.

Table 6.3 lists the alternatives selected for comparison. These were selected to give a range of APIs from large cloud providers, dedicated mapping services, and free and paid services to ensure a comprehensive analysis of the available options.

API	Pricing Model
Google	Commercial
AWS/ESRI	Commercial
Azure	Commercial
Mapbox	Commercial
Nominatim (OpenStreetMap)	Free

Table 6.3: Alternative Geocoders

Usage Limitations

Geocoding services typically impose limitations on the usage of their APIs through the following mechanisms:

- data restrictions;
- API throttling; and
- pricing structures.

This section discusses these limitations and their implications for the alternative geocoders outlined in Table 6.3.

Data Restrictions

Many geocoding APIs restrict what their data may be used for, e.g., the AWS Service Terms prohibit scraping, systematic collection, storing, or duplicating results from

any of their location services [49]. There are also restrictions on using their results alongside company branding or advertisements. Some APIs offer two pricing tiers to allow users to store results. As each API comes with a long list of restrictions, these are not replicated in full here, but the potential limitations are worth mentioning as part of this analysis.

API Throttling

Many APIs employ throttling as a technique to ensure the service does not become overloaded by large volumes of requests. As geocoding often requires processing large volumes of addresses, It's common for these APIs to implement batch geocoding, where a user can submit multiple queries with a single request. This often has no effect on pricing, but will allow users to submit increased volumes of requests, as the rate limits are typically specified in requests per second (r/s), rather than queries per second (q/s). Table 6.4 details the rate limitations documented for the selected APIs, with all rates translated into q/s for comparability.

Provider	Rate Limit (q/s)	Time for 10k (s)
Google Maps [18]	50	200
Mapbox [33]	1000	10
AWS [48]	100	100
Azure Maps [36]	50	200
Nominatim	1	1,000

Table 6.4: Geocoding API Rate Limitations

Table 6.4 shows that there is a large variation in the speed of commercial geocoders, and for Nominatim, the rate limit prohibits geocoding large volumes of data. Mapbox is the fastest (perhaps because it is a dedicated mapping provider), though it's worth noting that this limit has only recently increased from 10 queries per second with the release of their v6 API in April 2024 [33].

It is also worth noting that these rate limits indicate a theoretical maximum speed for their respective services. Full utilisation of these rates may require some form of multithreading, as API responses may not be fast enough to reach the rate limit when calling the APIs sequentially. The speeds achieved in practice will be covered in §6.1.3

Pricing

Each geocoding API has their own pricing model. These are often tiered to reduce the

per-query cost for larger volumes of requests. Some APIs also include a free tier for limited usage. Table 6.5 calculates the cost of using each of these APIs for 10k and 500k addresses to give a quick indication of pricing and scalability for each API.

Provider	Price for 10k	Price for 500k	Currency
Google Maps [18]	\$50	\$2,100	USD
Mapbox (Temporary) [34]	\$0	\$300	USD
Mapbox (Permanent) [34]	\$50	\$2,500	USD
AWS (Core) [47]	\$5	\$250	USD
AWS (Stored) [47]	\$40	\$2,000	USD
Azure Maps [36]	\$22.50	\$2,227.50	USD
Nominatim	\$0	\$0	-

Table 6.5: Pricing Comparison of Alternative Solutions

This shows that AWS and Mapbox provide far cheaper geocoding services (provided that you do not need to store the results). For use cases that require storage of results to perform analysis, the prices are typically much steeper, quickly reaching thousands of dollars.

Results

To evaluate the performance of each geocoding API, their official Python interfaces were used in accordance with their documentation to process the evaluation dataset. The implementation of each of these is included in Appendix C and limits the search space (where possible) to New Zealand addresses. This allows for the best comparison to the methods implemented for this project – recall that only NZ addresses are in scope for this project.

Speed

Table 6.6 details the speeds achieved in practice in comparison to the theoretical max speeds available.

Provider	Rate Limit (q/s)	Rate Achieved (q/s)	Duration (h:mm)
Google Maps	50	3.93	0:38
Mapbox	1000	11.57	0:11
AWS	100	1.10	1:53
Azure Maps	50	1.85	1:08
Nominatim	1	1.00*	2:05

* Nominatim API was artificially limited to respect usage policy

Table 6.6: Geocoding API Rates for 7,500 Evaluation Addresses

The rates achieved for this evaluation are significantly lower than the theoretical maximum for most APIs. This is due to the following:

- The Mapbox Python interface has not yet been updated to use the batch geocoding endpoint. Instead, it uses an older endpoint which has a specified rate limit of 10 q/s [33].
- The Azure Batch Geocoding endpoint does not support the use of country filtering. As such, the regular geocoding endpoint was used to provide a fairer evaluation.
- No special effort was taken to parallelise requests to any APIs. This would have likely significantly increased the speed (although it requires more upfront development effort), particularly for the non-batch endpoints.

Accuracy

To calculate the accuracy of each geocoding API against the evaluation dataset, the 2,500 addresses were initially manually matched to the correct address in the NZSA dataset. The coordinates retrieved via this match are considered to be the truth, and results from the geocoding APIs are compared against this.

In accordance with the definitions previously outlined in Table 6.2, if an address is matched correctly and its coordinates are within 500 m of the true location, the match is considered to be correct. This assumes that the addresses have been matched correctly based on coordinates alone. This means that there is the possibility of ‘slight mismatches’ being considered as ‘near perfect matches’, as the geocoders may return an address in the correct neighbourhood but within the 500 m threshold. As such, this method only provides a rough estimate as to the accuracy of these geocoders.

Provider	No. Incorrect			% Accuracy		
	Nice	Realistic	Aggressive	Nice	Realistic	Aggressive
Google	1	42	224	99.96	98.32	91.04
AWS	7	32	324	99.72	98.72	87.04
Azure	53	194	1284	97.88	92.24	48.64
Mapbox	15	78	478	99.4	96.88	80.88
Nominatim	154	675	2145	93.84	73.00	14.20

Table 6.7: Geocoding API Accuracy Estimations

These results suggest that Google and AWS are the most accurate geocoders, both in terms of plain accuracy as well as robustness. An interesting result is that Azure’s performance declines significantly for the realistic and aggressively modified addresses. Inspection of the match results here shows that despite setting the country filter to NZ, the API has begun to return overseas addresses that are similar to the queried addresses. The freely available geocoder Nominatim performed significantly worse on all address types compared to the commercial solutions. It achieved approximately 94% accuracy on the nicely formatted addresses, but this deteriorates significantly with realistic and aggressively modified addresses.

Manual inspection of the Google and AWS results shows that, as expected, some ‘slight mismatches’ have been classed as ‘near perfect matches’. For example, an address may be matched with the incorrect street within 500 m of the correct coordinates. This should be classified as incorrect. An example of this from the evaluation dataset is:

59 French Street S Eden Terrace Auckland → 59 Mount Eden Rd, Eden Terrace, Auckland

4 Spence Ln Whakatane 3120 → 3120, Whakatane, Bay of Plenty

This demonstrates two addresses may be within a small enough area to be considered a correct match despite the street name and number being clearly incorrect.

This inflates the accuracy measure slightly. As these two geocoders are a step above the rest, they were both selected for further detailed analysis. This involved manual checking of the results, ensuring confidence in the accuracy assessments.

The detailed results presented in tables 6.8 and 6.9 show that the accuracies have reduced slightly for the realistic addresses and more considerably for the aggressive addresses. The Google API is still the top-performing geocoder in terms of accuracy. Additionally, the Google API has a much higher proportion of mismatched addresses in ‘Slight Mismatches’ rather than ‘Total Mismatches’. This means they are more likely to be in the correct vicinity. AWS consistently has six ‘Near Perfect’ matches, which indicates its records may be slightly out of date.

Dataset	Perfect	Near Perfect	Slight Mismatch	Total Mismatch	Acc. (%)
Nice	2499	-	1	-	99.96
Realistic	2411	-	58	1	97.61
Aggressive	2199	1	216	84	88.00

Table 6.8: Detailed Results for Google API

Dataset	Perfect	Near Perfect	Slight Mismatch	Total Mismatch	Acc. (%)
Nice	2490	6	4	-	99.84
Realistic	2461	6	30	3	98.68
Aggressive	2107	6	197	190	84.52

Table 6.9: Detailed Results for AWS API

6.1.4 Implemented Solutions

The same evaluation dataset is used to analyse the performance of the solutions implemented for this project. Algorithms for parsing will be analysed first, as they are the precursor to some of the matching algorithms. Matching algorithms that require a parser are, therefore, limited by the parser’s performance.

Parsing

Two parsing models were implemented for this project. The first is a wrapper around the existing libpostal package, and the second is the custom-built RNN-based parser. The same evaluation dataset is used to estimate accuracy and robustness of the two parsers, however, the approach had to be adjusted slightly for this evaluation as there is no ground truth to use as a baseline.

For nicely formatted addresses, the results previously attained from the Google Geocoding API were used for comparison. These results contain labelled data for the matched address, so comparisons can be drawn between the parsed addresses and this labelled data. Results that differed from the results from the Google API were manually checked to verify incorrect parsing and classify the type of parsing error.

During this classification, how many addresses were incorrectly parsed and the severity of parsing-related errors were tracked, measured by both the number of words and the number of characters misclassified. These are important measures to consider as they have a direct impact on the ability of the matching algorithms to successfully match the address after a parsing error. The results of this evaluation are presented in Table 6.10.

Measurement	libpostal Score	RNN Score
Parsing duration (s)	0.11	0.13
Parsing rate (addresses per s)	23,364	19,380
Total Errant Addresses	56	106
Errant Addresses (%)	2.24	4.24
Total Errant Fields	142	212
Errant Fields (%)	1.40	2.09
Total Errant Characters	567	280
Errant Characters (%)	0.61	0.30

Table 6.10: Parsing Results for Nicely Formatted Addresses

These results demonstrate that both parsing algorithms offer sufficient speed and high precision for parsing addresses. The libpostal parser has a slight edge over the RNN parser in most metrics, except for the number of characters misclassified. This is perhaps the most important metric for address-matching purposes, as smaller errors are more likely to be overcome by the matching algorithm.

During the classification of these errors, the following insights were also noted:

- The libpostal parser classifies words as a whole, whereas the RNN parser classifies individual characters. This is the source of the discrepancy in accuracy measurements between the two parsers.

- The main error present in the results from the libpostal parser was appendment of suburb words to the street.
- The second most common error was interpreting the postcode as a street number.
- The only type of error made by the RNN parser was appending characters from the suburb field to the street.
- 35% of errors made by the RNN parser were on addresses with "quay" in them. This may indicate quays are under-represented in the training data.

For the realistic and aggressively modified addresses, there was no correctly labelled data available for comparison. Instead, results from each parser were compared. Addresses with different parsing results were manually checked and classified as follows: RNN parser was correct, libpostal parser was correct, or neither parser was correct. These results are presented below in Table 6.11

Type	Disagreements	RNN Correct	libpostal Correct	Neither
Realistic	553	284	220	49
Aggressive	1507	1243	99	165

Table 6.11: Error Classifications for Realistically and Aggressively Modified Addresses

Assuming that addresses with no disagreements between the parsers were parsed correctly, error rates can be estimated as:

Metric	libpostal Score	RNN Score
Errant Realistic Addresses	333	269
Errant Realistic Addresses (%)	13.32	10.76
Errant Aggressive Addresses	1408	264
Errant Aggressive Addresses (%)	56.32	10.56

Table 6.12: Parser Error Estimates for Realistically and Aggressively Modified Addresses

These results demonstrate that the performance of the libpostal parser deteriorates similarly to the performance of the RNN parser on addresses with realistic modifications. However, when parsing the aggressively modified addresses, the performance of

the RNN parser does not deteriorate further, while the performance of the libpostal parser deteriorates significantly. The following observations were also made while classifying these errors:

- Both parsers continue to make the same kinds of errors as with nice addresses, i.e., the libpostal parser mistakes suburb info for street names, and the RNN parser misclassifies occasional characters from the suburb into the street name.
- The libpostal parser makes these errors at a significantly higher rate than previously observed in the nicely formatted addresses. The cause of this is not evident in the data.
- The libpostal parser cannot handle typos removing spaces between fields, e.g. “70 Symonds StreetGrafton” will not have the suburb parsed. This is due to the libpostal parser restriction of classifying whole words at a time.
- The RNN parser struggled with abbreviations of “terrace” and “quay”, when the street type was omitted (e.g. 70 Symonds Grafton), and when the address contained ambiguous cardinal information (e.g. 29 Customs Street West Auckland).

The errors observed from the RNN parser were likely due to an under-representation of these issues in the training data. However, the ambiguous cardinal directions may not be overcome without a link to an address database during parsing, as the cardinal directions sometimes belong to the street and sometimes to the suburb. When the separating character (e.g. a comma) is omitted, these addresses may never be parsed accurately with the current methods.

Matching

Finally, the evaluation dataset was used to analyse the performance of each address-matching algorithm. The address-matching algorithms (matchers) evaluated in this section use the methods described in chapters 4 and 5. Recall that some of these methods require an address parser, and utilise different nearest neighbour searches to find the best matching address in the NZSA. Table 6.13 includes a summary of the matchers for convenience. Note that for algorithms requiring a parser, the RNN

parser was used during this evaluation as it demonstrated parsing speeds similar to the libpostal parser and made less significant errors during parsing.

Matcher	Uses Parser	Description
Fuzzy	✓	Uses the component-wise fuzzy matching method described in §4.6.2.
TF-IDF	✗	Uses TF-IDF vectors described in §4.6.3 and a brute force cosine similarity calculation to find the nearest match.
Compositional Vector	✓	Uses compositional vectors described in §4.6.4 to encode addresses, and a KD tree to find the nearest match.
Address Embedding	✗	Uses the address embedding model described in §4.6.5 with a KD tree to find the nearest match.
Fuzzy Hybrid	✓	Initially uses compositional vectors and a KD tree to find the k closest matches, then the component-wise fuzzy matching method is used to determine the final match.
TF-IDF Hybrid	✓	Initially uses compositional vectors and a KD tree to find the k closest matches, then TF-IDF vectors and cosine similarity scores are used to determine the final match.

Table 6.13: Matchers Evaluated

First, the duration taken for each matcher to match the 7,500 evaluation addresses was measured. From this, a rate of addresses per second was calculated to measure scalability. Note that these times include the time taken for the parser to complete parsing before matching begins. The results from this test are displayed in Table 6.14.

Matcher	Duration (mm:ss)	Rate (Addresses/s)
Fuzzy	12:15	10.2
TF-IDF	6:06	20.5
Compositional Vector	0:17	436
Address Embedding	2:25	51.6
Fuzzy Hybrid ($k = 20$)	1:19	95.5
Fuzzy Hybrid ($k = 80$)	3:21	37.4
Fuzzy Hybrid ($k = 120$)	4:10	30.0
TF-IDF Hybrid ($k = 20$)	1:13	103
TF-IDF Hybrid ($k = 80$)	3:15	39.0
TF-IDF Hybrid ($k = 120$)	3:55	31.8

Table 6.14: Matching Algorithm Speeds

These results show that, despite the use of component-wise fuzzy matching over brute force, the Fuzzy matcher is still the slowest method. the TF-IDF matcher was able to match addresses at roughly twice the speed of the Fuzzy matcher, and it does not require a parser. The Compositional Vector matcher was significantly faster than both these methods, averaging speeds of 20-40 times faster. This was expected as it utilises the KD-Tree structure for finding the closest matching address, whereas TF-IDF uses brute force and the Fuzzy matcher uses blocking. Both hybrid methods offer speeds faster than their base algorithm counterparts but still significantly slower than the Compositional Vector matcher.

Repeating the process used for evaluating alternative Geocoding APIs, the results were compared to the true coordinates, and matches farther than 500 m from the true location were considered to be incorrect. The results are summarised in Table 6.15.

Matcher	No. Incorrect			% Accuracy		
	Nice	Real.	Aggr.	Nice	Real.	Aggr.
Fuzzy	72	179	264	97.12	92.84	89.44
TF-IDF	20	28	126	99.20	98.88	94.96
Compositional Vector	167	610	1322	93.32	75.60	47.12
Address Embedding	637	1345	2319	73.08	46.20	7.240
Fuzzy Hybrid ($k = 20$)	127	525	1109	94.92	80.92	55.64
Fuzzy Hybrid ($k = 80$)	118	458	935	95.28	81.68	62.60
Fuzzy Hybrid ($k = 120$)	114	449	890	95.44	82.04	64.40
TF-IDF Hybrid ($k = 20$)	101	448	1027	95.96	82.08	58.92
TF-IDF Hybrid ($k = 80$)	127	377	819	96.48	84.92	67.24
TF-IDF Hybrid ($k = 120$)	118	357	767	96.68	85.72	69.32

Table 6.15: Matching Algorithm Accuracies for Nicely Formatted, Realistically Modified and Aggressively Modified Addresses

The results show that most methods provide highly accurate matching for nicely formatted addresses. However, performance varies significantly when attempting to match realistically and aggressively modified addresses. The TF-IDF matcher was both the most accurate and robust method. This is likely due to the brute force searching and large vectors used to represent the addresses. The Fuzzy matcher was the next most accurate, which utilises blocking and preserves raw strings for matching. Of the methods that rely on compressed address representations, the Compositional Vector matcher significantly outperforms the Address Embedding matcher. Both methods exhibit poor robustness to the modified addresses.

Hybrid methods demonstrated a balance between the speed of the Compositional Vector matcher and the accuracy/robustness of the Fuzzy and TF-IDF matchers. Again, TF-IDF showed a slight edge over the Fuzzy matcher, but both suffered in robustness due to the use of the compositional vectors to reduce the initial search space. Increasing values of k improved accuracy and robustness but not to the level of either base method (TF-IDF and fuzzy).

From these results, it can be concluded that the TF-IDF matcher is the most appropriate method for general purposes, though the Compositional Vector matcher and hybrid matchers could be useful when applied to well-formatted addresses, as these algorithms are orders of magnitude faster than TF-IDF while providing similar accura-

cies. When dealing with extremely messy addresses, TF-IDF is the most appropriate choice, as it is sufficiently robust to missing data fields, typos, abbreviations, and other common address formatting problems.

Inspection of addresses that were mismatched showed that the implemented methods may have been somewhat limited in their accuracy due to address records not included in the NZSA. To illustrate this, the 20 nicely formatted addresses that were mismatched by the TF-IDF matcher are shown in Table 6.16.

Search Address	Matched To	Dist. (m)
Floor 1 11/82 Cable Street Te Aro	11/11 Charlotte Street, Eden Ter*, Akl*	491,354
10 Bethells Road Waitakere Akl*	108 Bethells Road, Waitakere, Akl*	4,968
13 Evergreen Drive Silverdale	13 Arran Drive, Silverdale	3,321
11 Evergreen Drive Silverdale	11 Arran Drive, Silverdale	3,308
9 Evergreen Drive Silverdale	9 Arran Drive, Silverdale	3,283
7 Evergreen Drive Silverdale	7 Arran Drive, Silverdale	3,250
5 Evergreen Drive Silverdale	5 Arran Drive, Silverdale	3,217
58C Milan Road Papatoetoe Akl*	48C Hamilton Road, Papatoetoe, Akl*	2,703
2 Evergreen Drive Silverdale	2 Harris Drive, Silverdale	2,638
20 Main Road South Paraparaumu	20 Menin Road, Raumati South, Para*	2,424
309 Harbour Road Ohope	109 Harbour Road, Ohope	2,137
409 Main South Road Hornby Chch*	209 Main South Road, Hornby, Chch*	1,931
221 Atkinson Road Titirangi Akl*	21 Atkinson Road, Titirangi, Auckland	1,760
181 Walters Road Takanini	3/11 Walters Road, Takanini	1,760
5 Kingfisher Way Tikipunga Whan*	5 Erin Street, Tikipunga, Whangarei	1,483
213 Birkdale Road Birkdale Akl*	21 Birkdale Road, Birkdale, Auckland	1,212
575 Chapel Road East Tamaki Akl*	357 Chapel Road, East Tamaki, Akl*	1,149
146 Wellington Street Opotiki	6 Wellington Street, Opotiki	931
135 Hamlet Street Stratford	35 Hamlet Street, Stratford	912
242 Grenada Street Mount M*	18 Grenada Street, Mount Maunganui	763

* Abbreviations applied to reduce table width

Table 6.16: Addresses Mismatched by TF-IDF Matcher

The first mismatch includes the floor in the address, which has resulted in a mismatch. This could have been avoided by using the address parser first to remove unwanted fields. However, it appears this is the only case.

Many of the remaining incorrect matches share a common theme of having matched the correct street and area but including the wrong street number. Inspection of the copy of NZSA used at the time of matching showed that all 19 mismatched addresses

were not included. That is, either the street does not exist (e.g., Evergreen Drive, Silverdale) or the specific address number is not included (likely due to it being a non-residential address). As TF-IDF works purely based on token distances, differences in street numbers such as 221 vs 21 are perceived as small when a match such as 21 vs 22 would actually be preferred as these are more likely to be close together geographically.

TF-IDF is the only method that can't consider the numerical difference between street numbers. It's likely to have occurred significantly more than the 20 occurrences seen here, as many of these mismatches will be within 500 m. It is also likely these missing addresses are causing problems for the other matching methods, as they may prevent backtracking in the Fuzzy matcher and result in more difficult distance comparisons for the Address Embedding and Compositional Vector matchers.

As the NZSA is currently used as the basis for each matching method implemented in this project, the accuracies are not adjusted to remove or discount errors made when matching with addresses that are not present in the NZSA. However, it is worth noting that accuracies are likely to increase when matching addresses that are strictly residential.

6.1.5 Comparison

This section draws a comparison between the methods implemented for this project and the performance of alternative solutions available.

A range of alternative geocoding APIs was explored in §6.1.3. The main criteria considered were:

- restrictions of use;
- pricing;
- rate limitations; and
- accuracy

These criteria will also be used for comparing the implemented solutions, noting the following:

- Accuracy is compared using the accuracy to within 500 m metric. This was calculated for both implemented and alternative solutions previously and allows for direct and fair comparison.
- Match categories are therefore not used in this comparison. This is due to the sheer volume of addresses that would require manual checking. There was also little variation between accuracy scores when using match categories when compared to the more simple accuracy within 500 m metric in the previous analysis laid out in §6.1.3.
- Due to the nature of the implemented methods matching to specific addresses in the NZSA, it is not possible to return suburb-only or postcode-only matches. For this reason, it is likely the implemented methods return fewer slight mismatches than the alternative geocoders. However, it is also likely that the implemented methods return a greater number of near-perfect matches, as only residential addresses are contained in the NZSA.
- The comparison focuses on TF-IDF and compositional vectors as these were the standout implemented methods. It is therefore expected that these two methods will be the most used.

Accuracy Comparison

Firstly, accuracy statistics are compared in detail in Table 6.17. This shows that the TF-IDF matcher demonstrates accuracies comparable to the best available commercial APIs. This is consistent across nicely formatted addresses, as well as realistically and aggressively modified addresses. Compositional vectors does not perform as well as the commercial APIs, but it has a base accuracy comparable to the Nominatim API (the only free geocoding API available) and slightly better robustness to modifications.

Matcher/API	No. Incorrect			% Accuracy		
	Nice	Real.	Aggr.	Nice	Real.	Aggr.
Google	1	42	224	99.96	98.32	91.04
AWS	7	32	324	99.72	98.72	87.04
Azure	53	194	1284	97.88	92.24	48.64
Mapbox	15	78	478	99.4	96.88	80.88
Nominatim	154	675	2145	93.84	73.00	14.20
TF-IDF	20	28	126	99.20	98.88	94.96
Compositional Vector	167	610	1322	93.32	75.60	47.12

Table 6.17: Comparison of Accuracy Scores for Implemented Methods and Alternative Solutions

To improve the interpretability of these results, a grading scale is applied for base accuracy and robustness. The grading scale used is:

Grade	Percentage Range
A+	95–100%
A	90–94%
A-	85–89%
B+	80–84%
B	75–79%
B-	70–74%
C+	65–69%
C	60–64%
C-	55–59%
F	Below 59%

Note that this is an arbitrary scale, but it aids in providing a relative performance metric for quickly comparing each geocoder. Measuring robustness as the average of the accuracies on realistic and aggressively modified addresses, The resulting grades from applying this rubric are presented in Table 6.18.

Matcher/API	Accuracy		Robustness	
	%	Grade	%	Grade
Google	99.96	A+	94.68	A+
AWS	99.72	A+	92.88	A
Azure	97.88	A+	70.44	B-
Mapbox	99.4	A+	88.88	A-
Nominatim	93.84	A	43.60	F
TF-IDF	99.20	A+	96.92	A+
Compositional Vector	93.32	A	61.36	C

Table 6.18: Accuracy and Robustness Gradings for Each Geocoder

These grades capture the accuracy and robustness of the geocoders on a simple scale and demonstrate that the Nominatim API and Compositional Vectors matcher are a step below in terms of accuracy. The grades also clearly show the variation in robustness between the solutions.

Overall Comparisons

To form a complete comparison between implemented methods and alternative solutions, accuracy needed to be considered alongside several other factors. These are presented in Table 6.19.

Matcher/API	Accuracy	Robustness	Rate (Max)	Rate (Actual)	Pricing	Processing
Google	A+	A+	50	3.93	Paid	Remote
AWS	A+	A	100	11.6	Paid	Remote
Azure	A+	B-	50	1.10	Paid	Remote
Mapbox	A+	A-	1000	1.85	Paid	Remote
Nominatim	A	F	1	1.00	Free	Remote
TF-IDF	A+	A+	-	20.5	Free	Local
CV	A	C	-	436	Free	Local

Table 6.19: Complete Comparison of Implemented and Alternative Geocoders

In terms of accuracy, TF-IDF performs as well as the alternative commercial geocoding APIs. It also demonstrates robustness better than many of these solutions and

is on par with the top-performing Google Geocoding API. The Compositional Vector matcher demonstrates an accuracy on par with the freely available Nominatim API, which is a step below the commercial geocoding APIs but sufficient for many applications.

The theoretical maximum rates are included for each of the alternative APIs, as these show the speeds achievable via parallelisation. These rates are not included for the implemented methods, as these solutions can support any amount of parallelisation. Hence, there is no theoretical maximum. Comparing the actual rates achieved in this project gives a fair comparison of an easy implementation with no parallelisation for each API/method. The Compositional Vector matcher was the fastest method, achieving rates hundreds of times faster than other solutions. The TF-IDF matcher is also faster than the alternative solutions.

Implemented solutions are freely available for installation and use locally, with no restrictions. This is an advantage over the alternative solutions, which require processing of data in the cloud and limit the applications and caching/storage of results from their APIs.

These results demonstrate that the implemented methods are able to perform on par with commercially available geocoding solutions without the restrictions inherent in these APIs (pricing, security, usage rights, etc.), provided the addresses being geocoded are within NZ, and include addresses rather than place of interest names.

6.2 Discussion

The results discussed in §6.1.5 provide a fair evaluation of the geocoding methods implemented for this project by comparing their performance on an evaluation dataset to several alternative solutions available in the market.

This section discusses potential shortcomings of this testing process, as well as other notes made during the testing process that are worth further exploration beyond this project.

6.2.1 Evaluation Shortcomings

While the evaluation process is fairly comprehensive in measuring geocoding performance, there are some shortcomings to be noted:

- The testing dataset does not include much information superfluous to geocoding. That is, there are not many instances of building names and levels. While the parsers were designed to handle this, and the anecdotal testing done during experimentation in §5.1 shows promising results, there was no formal testing of how well the geocoders are able to handle this additional information.
- The evaluation dataset is not split into residential and non-residential addresses. While this is helpful for understanding an approximate accuracy for typical real-world datasets (as these are also unlikely to be organised this way), it places the implemented methods at a disadvantage compared to the alternative solutions evaluated (as they are restricted to residential addresses in the NZSA). It would be helpful to test the implemented methods on residential-only addresses to estimate their accuracies when matching to an address database that includes all possible addresses.

6.2.2 Potential Next Steps

Throughout the duration of this project, potential areas of improvement were identified for the extension of the package. These were not within the scope of this project, but they are noted here as potential next steps in the development of this package.

Parser Improvements

During the evaluation of the parsers in §6.1.4, it was observed that the RNN parser struggled with streets involving "quay", "terrace", and abbreviations of these. This is likely due to the under-representation of these words in the training dataset. Resampling addresses with an even distribution of street types and retraining or fine-tuning the existing model.

Spatial Joins to Complementary Datasets

Using the NZSA dataset as the matching database has some inherent limitations, which are discussed throughout this thesis. One of these is the limitation of matching to “correct” addresses. While it did not appear to be a problem with the evaluation conducted as part of this project, it is possible for addresses to be entered using region names instead of suburbs or city names. For example, using ‘Waikato’ in place of the city ‘Hamilton’. Such addresses are technically incorrect but are still valid representations as they are uniquely identifiable and commonly found in address datasets. This includes issues where addresses only note a suburb rather than a suburb and city, as present in the NZSA.

Statistics New Zealand maintains several shapefiles containing boundaries of regions, suburbs, territorial authorities, etc. These shape files could be spatially joined onto addresses in the NZSA dataset to provide extra context for matching algorithms.

Use of OpenStreetMap Data

Another notable limitation of the NZSA is that only residential addresses are included. This means that non-residential addresses are difficult to match accurately, and places of interest can’t be matched. Replacing the NZSA with OSM data would resolve these issues. However, it would also increase the computation required as there could be significantly many more matching candidates.

Embedding Hybrid Models

During this project, hybrid models that utilise address embeddings were discussed. This approach was discarded due to complications with the address embedding model, which resulted in larger-than-expected embedding vectors. However, an alternative smaller address embedding model could be developed specifically for use in a hybrid model, where the embedding model is only intended for use in determining partial matches, and TF-IDF could be used to make the final match calculation. Using an embedding model specifically trained for approximate accuracy would result in a much smaller and faster model.

TF-IDF Adjustments

There are also a few adjustments that could be made to the TF-IDF model to improve its performance:

- **Use of parsers:** It was seen in §6.1.4 that superfluous information can confuse the TF-IDF matcher. This is because it works directly on the unstructured address. This process could be changed to first use a parser to identify and filter out superfluous information, allowing TF-IDF to focus on key features of the address when matching. This would improve the robustness of the matcher with a minor reduction in matching speeds.
- **Numerical street numbers:** §6.1.4 also revealed a limitation of TF-IDF that numerical distance can't be considered for street numbers. This is because TF-IDF vectors only contain counts of each token. This could be resolved by including an extra entry in the TF-IDF vectors that includes a scaled or transformed version of the street number. This would encourage the cosine similarity to match addresses geographically closer on the street when the actual address isn't available. This would improve accuracy with minimal impacts on computational costs and portability.
- **Multiple match addresses:** The implementation of TF-IDF in this project creates one vector per address in the NZSA. This is a straightforward approach consistent with what's seen in the literature. However, it could be beneficial to create multiple vectorisations per address. This would aid in matching addresses with particularly long area information. For example, '70 Symonds Street, Grafton, Auckland' could be vectorised three times. Once using the raw address, then two more times with '70 Symonds Street, Auckland' and '70 Symonds Street, Grafton'. These are both reasonable representations of the true address but create a distance between the correct match, which can result in false matches to other addresses with similar tokens. This change would improve the accuracy and robustness of the TF-IDF method at the cost of increased computational effort and reduced portability.
- **Token optimisation:** Limiting the number of tokens used would produce shorter TF-IDF vectors and improve computational costs. scikit-learn supports setting a maximum number of tokens to use. However, this selects tokens based

on the number of occurrences in the reference database, which does not necessarily mean that these tokens will divide the database well. Optimisation methods could be used to select tokens that strategically segment the address database.

As many of these potential next steps improve on one aspect at the cost of another (e.g. improved accuracy but reduced speed and portability), combinations of these are likely to yield the best improvements. These adjustments could also be offered as separate models in order to maximise flexibility for end users.


Chapter 7

Software Contributions

The methods tested in the project have been incorporated into a freely available Python package hosted in PyPI. The package allows users to install and use any of the methods discussed in this thesis. The code is also published to GitHub at <https://github.com/lmor152/glam>. A user guide and demo website have also been developed to assist with getting started using the package.

7.1 Demo Website

A demo website is published at <https://glam-demo.lmor152.com>. The demo site allows the visitor to enter an address, and results from parsing and matching the inputted address are presented back to the visitor. Figure 7.1 includes a screenshot of the website.



I created GLAM (Geocoding via LINZ Address Matching) to help with geocoding New Zealand addresses.

GLAM is a tool for *free, offline* geocoding. It works by matching addresses to Land Information New Zealand data, so it only works with residential addresses.

GLAM includes multiple methods for parsing and matching messy unstructured addresses. Find out more here: <https://github.com/lmor152/glam>

GLAM Settings

At this time, this demo site only supports the RNN-based parser, and the TF-IDF matcher.

GLAM Demo ↗

GLAM is designed to handle messy addresses, this includes building names, levels, abbreviations, and typos.

Test it out by entering an address, and seeing how it parses the address components and finds the best match in the LINZ database.

Enter an address to test:

Level three, KPMG, 18 viaduct harbour ave, 1010

Your address was parsed into:

building	value
building	KPMG
level	3
first_number	18
street_name	VIADUCT HARBOUR AVENUE
postcode	1010

Best match found (confidence = 79.01%):

address_id	value
address_id	1313142
address_number	18
full_road_name	Viaduct Harbour Avenue
suburb_locality	Auckland Central
town_city	Auckland
full_address_ascii	18 Viaduct Harbour Avenue, Auckland Central, Auckland
shape_X	174.7577686
shape_Y	-36.8453608833
postcode	1010

You can exact search against the LINZ database to find your address here too:

18 Viaduct Harbour Avenue, Auckland

	address_id	unit_value	address_number	address_number_suffix	address_number_high	suburb_locality	town_city	full_road_name	full_address	
	916,699	1,313,142	None	18	None	None	Auckland Central	Auckland	Viaduct Harbour Avenue	18 Viaduct Harbour Avenue, Auckland

Figure 7.1: Screenshot of Demo Website

7.2 User Guide

The user guide is included with the GitHub repository and includes instructions for installing and using the Python package for geocoding. A copy of the user guide is included here for reference:

Geocoding via LINZ Address Matching (GLAM)

Overview

This package implements methods for performing entity matching on unstructured NZ addresses to the New Zealand Street Address dataset maintained by Land Information New Zealand. This package does not support PO boxes or international addresses.

Installation

Get the latest version of glam by installing from PyPI:

```
pip install glam
```

Usage

To setup glam, the dependencies have to be downloaded first. Alternatively, if this is not done, glam will download dependencies the first time it's used for geocoding.

```
from glam import Geocoder, download_dependencies

# directory to store glam's dependencies
download_dependencies("path/to/glamdeps")

addresses = [
    "16 western springs rd morningside",
    "4 ryelands dr, lincoln",
]

gc = Geocoder(
    "path/to/glamdeps",
    matcher = "tfidf",
    parser = "rnn"
)

matched_addresses = gc.geocode_addresses(addresses)
print(matched_addresses)
```

Outputs:

```
[
  Search address 16 western springs rd morningside -> matched
  ↳ to 16 Western Springs Road, Morningside, Auckland with
  ↳ 0.899795426050709 confidence,

  Search address 4 ryelands dr lincoln -> matched to 4
  ↳ Ryelands Drive, Lincoln with 0.8462000263063917
  ↳ confidence
]
```

Each matched address contains coordinates and other fields from the NZSA dataset:

```
>> matched_addresses[0].matched_address.to_dict()

{
  'address_id': 1076278,
  'unit_value': None,
  'address_number': '16',
  'address_number_suffix': None,
  'address_number_high': None,
  'full_road_name': 'Western Springs Road',
  'suburb_locality': 'Morningside',
  'town_city': 'Auckland',
  'full_address_ascii': '16 Western Springs Road,
  ↪ Morningside, Auckland',
  'shape_X': '174.7345083',
  'shape_Y': '-36.8739952167',
  'postcode': '1021'
}
```

Glam can also be used to parse addresses:

```
>> gc.parse_addresses(addresses)

[
  ParsedAddress{'unit': None, 'building': None, 'level':
  ↪ None, 'first_number': '16', 'first_number_suffix':
  ↪ None, 'second_number': None, 'street_name': 'WESTERN
  ↪ SPRINGS ROAD', 'suburb_town_city': 'MORNINGSIDE',
  ↪ 'postcode': None},

  ParsedAddress{'unit': None, 'building': None, 'level':
  ↪ None, 'first_number': '4', 'first_number_suffix': None,
  ↪ 'second_number': None, 'street_name': 'RYELANDS DRIVE',
  ↪ 'suburb_town_city': 'LINCOLN', 'postcode': None}
]
```


Available Models

Models are divided into parsers and matchers. Some matchers work directly on unstructured address strings, others require a parser to add structure to the addresses before matching.

Matchers

- **TFIDF (default):** Uses term frequency inverse document frequency to match unstructured addresses directly.
- **Fuzzy:** Uses component-wise fuzzy matching on parsed addresses.
- **Vector:** Uses custom address vectorisation logic to find the nearest match.
- **Embedding:** Uses deep-learned embeddings to vectorise addresses and find the nearest match.

Parsers

- **RNN:** Uses an LSTM to parse unstructured address strings.
- **libpostal:** A wrapper around the postal Python package (requires libpostal to be installed).

TFIDF is recommended for most use cases as it's the most accurate method. Vector is the fastest method but has reduced accuracy and robustness compared to TFIDF. Note that by default, dependencies are only downloaded for TFIDF matching. If another method is selected, the dependencies will be built the first time it is used and saved for future use.

Custom Address Datasets

Glam can be used to match to other address datasets by placing your dataset in the dependencies directory under 'nz-street-address.csv'. Dependencies should be rebuilt by deleting the matching directory under the dependency directory to ensure they are referencing the new address database.

Chapter 8

Conclusions

This research project aimed to create a solution for geocoding NZ addresses. The specific priorities of the project were:

- **Speed:** The solution should facilitate a swift turnaround of analytics.
- **Accessibility:** It should be freely available and not require extensive software installations or specialist hardware.
- **Local execution:** The solution should be executable locally to meet potential security requirements surrounding personally identifiable information.
- **Accuracy:** It should maintain enough precision for informed decision-making while balancing speed and portability.
- **Ready to use:** It should not require model training or configuration before geocoding.

A comprehensive literature survey was conducted to understand the current landscape and methodologies used for geocoding. This survey highlighted a variety of strategies and algorithms for address matching. These methods were selectively adapted to align with the goals of this project. Additionally, a novel method (compositional vectors) was proposed.

A dataset consisting of 2,500 real delivery addresses was sourced from NZ Post. This dataset was augmented by creating two additional addresses per source address, one

with realistic modifications and one with aggressive modifications. This produced an evaluation dataset with 7,500 addresses of varying difficulty.

The evaluation dataset was used to assess the performance of the implemented algorithms, and comparisons were made to alternative solutions available in the market. The results demonstrated that the implemented methods exhibit different strengths and are competitive with the best commercial solutions currently available.

Finally, the implemented methods were packaged into a Python library, which has been made publicly available via PyPI. A user guide and demo website were also developed to support the adoption and use of the package.

Appendix

A Lookups

A.1 List of street types

ACCESS	CREEK	GREEN	OAKS	SQUARE
ACCESSWAY	CRESCENT	GROVE	PADDOCK	STATE HIGHWAY
ALLEY	CREST	GULLY	PAKU	STEEP
ANCHORAGE	CUL	HAVEN	PARADE	STEPS
APPROACH	DALE	HEAD	PARK	STRAIGHT
ARCADE	DELL	HEIGHTS	PARKWAY	STRAND
ARCH	DEVIATION	HIGHWAY	PASS	STREET
AVENUE	DOWNS	HILL	PASSAGE	TERRACE
BANK	DRIVE	ISLAND	PATH	TOWERS
BAY	DUNE	JUNCTION	PLACE	TRACK
BEACH	ELM	KEY	PLAZA	TRAIL
BELT	END	KNOB	POINT	TRAMWAY
BEND	ENTRANCE	LADDER	PRIORS	TREES
BLUFF	ESPLANADE	LANDING	PROMENADE	VALE
BOULEVARD	ESTATE	LANE	QUADRANT	VALLEY
BRAE	EXPRESSWAY	LEA	QUAY	VENUS
BRIARS	FAIRWAY	LEADER	REEF	VIEW
BRIDGE	FALL	LEIGH	RESERVE	VIEWS
BYPASS	FARE	LINE	REST	VILLAGE
CENTRE	FARMS	LINK	RETREAT	VILLAS
CHASE	FEN	LOOKOUT	RIDGE	VISTA
CIRCLE	FERN	LOOP	RISE	VUE
CIRCUIT	FREEWAY	MALL	ROAD	WALK
CIRCUS	FLAT	MEAD	ROADS	WATERS
CLAIM	FLATS	MEADOWS	ROADWAY	WAY
CLOSE	GARDEN	MEWS	ROUTE	WHARF
CORNER	GARDENS	MILE	ROW	WYND
COMMON	GATE	MOTORWAY	SERVICE LANE	
COURT	GLADE	MOTU	SLOPE	
COURTS	GLEN	MOUNT	SPA	
COVE	GRANGE	NEAVES	SPUR	

A.2 List of Level Descriptions

BASEMENT	LOWER GROUND FLOOR	PODIUM
GROUND FLOOR	MEZZANINE	ROOFTOP
GROUND LVL	OBSERVATION DECK	SUB-BASEMENT
GROUND	PARKING	UPPER GROUND FLOOR
GROUND LEVEL	PENTHOUSE	
LOBBY	PLATFORM	

A.3 List of Dwelling Types

ANTENNA	COTTAGE	LOT	SHOWROOM	TOWNHOUSE
APARTMENT	DUPLEX	MAISONETTE	SIGN	UNIT
BLOCK	FACTORY	MARINE BERTH	SITE	VAULT
BOATSHED	FLAT	OFFICE	STALL	VILLA
BUILDING	GARAGE	PENTHOUSE	STORE	WARD
BUNGALOW	HALL	REAR	STRATA UNIT	WAREHOUSE
CAGE	HOUSE	RESERVE	STUDIO	WORKSHOP
CARPARK	KIOSK	ROOM	SUBSTATION	
CARSPACE	LEASE	SECTION	SUITE	
CLUB	LOBBY	SHED	TENANCY	
COOLROOM	LOFT	SHOP	TOWER	

A.4 List of Street Abbreviations

ACCS	ACCESS	FAWY	FAIRWAY	PT	POINT
ACCSWY	ACCESSWAY	FRMS	FARMS	PRIORS	PRIORS
ALY	ALLEY	FEN	FEN	PROM	PROMENADE
ANCG	ANCHORAGE	FWY	FREEWAY	QDRT	QUADRANT
APP	APPROACH	FLT	FLAT	QY	QUAY
ARC	ARCADE	FLTS	FLATS	RES	RESERVE
AV	AVENUE	GDN	GARDEN	REST	REST
AVE	AVENUE	GDNS	GARDENS	RTR	RETREAT
BNK	BANK	GTE	GATE	RDGE	RIDGE
BCH	BEACH	GLD	GLADE	RISE	RISE
BND	BEND	GLN	GLEN	RD	ROAD
BLF	BLUFF	GRG	GRANGE	RDS	ROADS
BLVD	BOULEVARD	GRN	GREEN	RDWY	ROADWAY
BR	BRAE	GRV	GROVE	RTE	ROUTE
BRG	BRIDGE	GLY	GULLY	SVLN	SERVICE LANE
BYP	BYPASS	HVN	HAVEN	SLP	SLOPE
CTR	CENTRE	HTS	HEIGHTS	SPUR	SPUR
CH	CHASE	HWY	HIGHWAY	SQ	SQUARE
CIR	CIRCLE	HL	HILL	SH	STATE HIGHWAY
CCT	CIRCUIT	IS	ISLAND	STEEP	STEEP
CRCS	CIRCUS	JCT	JUNCTION	STPS	STEPS
CLM	CLAIM	JNC	JUNCTION	STGT	STRAIGHT
CL	CLOSE	LADR	LADDER	STRD	STRAND
CNR	CORNER	LNDG	LANDING	ST	STREET
CMN	COMMON	LEDR	LEADER	TCE	TERRACE
CRT	COURT	LGH	LEIGH	TWRS	TOWERS
CRTS	COURTS	LKT	LOOKOUT	TRK	TRACK
CV	COVE	MDWS	MEADOWS	TRL	TRAIL
CRK	CREEK	MWY	MOTORWAY	TMWY	TRAMWAY
CR	CRESCENT	MOTU	MOTU	TRS	TREES
CRES	CRESCENT	MT	MOUNT	VALE	VALE
CRST	CREST	NVS	NEAVES	VLY	VALLEY
DLE	DALE	OAKS	OAKS	VNUS	VENUS
DEL	DELL	PADK	PADDOCK	VW	VIEW
DVN	DEVIATION	PDE	PARADE	VWS	VIEWS
DOWNS	DOWNS	PK	PARK	VLG	VILLAGE
DR	DRIVE	PKWY	PARKWAY	VLLS	VILLAS
DUNE	DUNE	PASS	PASS	VIS	VISTA
ENT	ENTRANCE	PSGE	PASSAGE	WLK	WALK
ESP	ESPLANADE	PTH	PATH	WATERS	WATERS
EST	ESTATE	PL	PLACE	WHRF	WHARF
EXP	EXPRESSWAY	PLZ	PLAZA		

B Literature Search Results

Table 8.1: Literature Survey Results

Source	ID	Author	Year	Title	Included	Reason
Google Scholar	G1	Dengyue L. et al.	2014	Approximate address matching	Yes	
Google Scholar	G2	Wasson, J. et al	2008	Real-time travel time estimates using media access control address matching	No	Relevance of title / abstract
Google Scholar	G3	Ramani, K. et al.	2023	Methods for matching English language addresses	Yes	
Google Scholar	G4	Shan, S. et al.	2020	Geographical address representation learning for address matching	Yes	
Google Scholar	G5	Comber S. et al.	2019	Machine learning innovations in address matching: A practical comparison of word2vec and CRFs	Yes	
Google Scholar	G6	Lin Y. et al.	2020	A deep learning architecture for semantic address matching	Yes	
Google Scholar	G7	Yates, D. et al.	2021	PostMatch: A Framework for Efficient Address Matching	Yes	

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Google Scholar	G8	Koumarelas I. et al.	2018	Experience: Enhancing address matching with geocoding and similarity measure selection	Yes	
Google Scholar	G9	Liu, C. et al.	2023	A graph-based method for Chinese address matching	Yes	
Google Scholar	G10	A.B Bashir et al.	2023	A Comparison of Address Matching Techniques to Improve Geocoding: A Case Study of Islamabad, Pakistan	Yes	
Google Scholar	G11	Gupta, V. et al.	2022	Improvement in Semantic Address Matching using Natural Language Processing	Yes	
Google Scholar	G12	Lee K. et al.	2020	Improving a street-based geocoding algorithm using machine learning techniques	Yes	
Google Scholar	G13	Liu, L.	1994	Cache designs with partial address matching	No	Relevance of title / abstract
Google Scholar	G14	Bichler, G. et al.	2007	Address matching bias: Ignorance is not bliss	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Google Scholar	G15	Comber, S.	2020	Demonstrating the utility of machine learning innovations in address matching to spatial socio-economic applications	Yes	
Google Scholar	G16	Xu, L. et al.	2022	Deep Transfer Learning Model for Semantic Address Matching	Yes	
Google Scholar	G17	Lee, J. et al.	2009	GIS-based geocoding methods for area-based addresses and 3D addresses in urban areas	No	Relevance of title / abstract
Google Scholar	G18	Yates, D. et al.	2021	PostMatch: A Framework for Efficient Address Matching	No	Duplicate of G7
Google Scholar	G19	Duarte, V. et al.	2023	Improving Address Matching Using Siamese Transformer Networks	Yes	
Google Scholar	G20	Ranzijn, B. et al.	2013	A Geocoding Algorithm Based On A Comparative Study Of Address Matching Techniques	Yes	

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S1	Choi S.-C.T. et al.	2017	Comparison of public-domain software and services for probabilistic record linkage and address standardization	Yes	
Scopus	S2	Lei J. et al.	2023	CLGLIAM: contrastive learning model based on global and local semantic interaction for address matching	Yes	
Scopus	S3	Kane N.J. et al.	2023	Committing to genomic answers for all kids: Evaluating inequity in genomic research enrollment	No	Relevance of title / abstract
Scopus	S4	Zandbergen P.A. et al.	2012	Error propagation models to examine the effects of geocoding quality on spatial analysis of individual-level datasets	No	Relevance of title / abstract
Scopus	S5	Bertin M. et al.	2015	Socioeconomic Disparities in Adverse Birth Outcomes in Urban and Rural Contexts: A French Mother-Child Cohort	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S6	Pinault L. et al.	2020	Accuracy of matching residential postal codes to census geography	No	Relevance of title / abstract
Scopus	S7	Rashidian S. et al.	2017	Effective Scalable and Integrative Geocoding for Massive Address Datasets	Yes	
Scopus	S8	Hwang M.-J. et al.	2021	Establishment of the Korea National Health and Nutrition Examination Survey air pollution study dataset for the researchers on the health impact of ambient air pollution	No	Relevance of title / abstract
Scopus	S9	Lopes B.L.W. et al.	2017	HIV Care Initiation Delay among Rural Residents in the Southeastern United States, 1996 to 2012	No	Relevance of title / abstract
Scopus	S10	Duncan D.T. et al.	2011	Evaluation of the positional difference between two common geocoding methods	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S11	Tabarcea A. et al.	2011	Ad-hoc georeferencing of web-pages using street-name prefix trees	No	Relevance of title / abstract
Scopus	S12	Jones R.R. et al.	2014	Accuracy of residential geocoding in the Agricultural Health Study	No	Relevance of title / abstract
Scopus	S13	Holman C.D.J. et al.	1999	Population-based linkage of health records in Western Australia: Development of a health services research linked database	No	Relevance of title / abstract
Scopus	S14	Olson S.	2017	Setting the census household into its urban context: Visualizations from 19th-century Montreal	No	Relevance of title / abstract
Scopus	S15	Jacob B.G. et al.	2010	Accounting for autocorrelation in multi-drug resistant tuberculosis predictors using a set of parsimonious orthogonal eigenvectors aggregated in geographic space	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S16	Hampton K.H. et al.	2010	Mapping health data: Improved privacy protection with donut method geomasking	No	Relevance of title / abstract
Scopus	S17	Whitsel E.A. et al.	2004	Accuracy and repeatability of commercial geocoding	Yes	
Scopus	S18	Xiang Y. et al.	2023	A Global-to-Local Algorithm for High-Resolution Optical and SAR Image Registration	No	Relevance of title / abstract
Scopus	S19	Nardone A.L. et al.	2020	Associations between historical redlining and birth outcomes from 2006 through 2015 in California	No	Relevance of title / abstract
Scopus	S20	Steelman T. et al.	2023	The Accuracy of Identifying Constituencies with Geographic Assignment Within State Legislative Districts	No	Relevance of title / abstract
Scopus	S21	Lin Y. et al.	2020	A deep learning architecture for semantic address matching	No	Duplicate of G6
Scopus	S22	Goldberg D.W. et al.	2013	An evaluation framework for comparing geocoding systems	Yes	

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S23	Zhan F.B. et al.	2006	Match Rate and Positional Accuracy of Two Geocoding Methods for Epidemiologic Research	No	Relevance of title / abstract
Scopus	S24	Batterman S. et al.	2014	Spatial resolution requirements for traffic-related air pollutant exposure evaluations	No	Relevance of title / abstract
Scopus	S25	Zhang C. et al.	2022	W-TextCNN: A TextCNN model with weighted word embeddings for Chinese address pattern classification	Yes	
Scopus	S26	Koumarelas I. et al.	2018	Experience: Enhancing address matching with geocoding and similarity measure selection	No	Duplicate of G8
Scopus	S27	Villanueva K. et al.	2013	The impact of the built environment on health across the life course: Design of a cross-sectional data linkage study	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S28	Torok M. et al.	2021	Spatial Errors in Automated Geocoding of Incident Locations in Australian Suicide Mortality Data	No	Relevance of title / abstract
Scopus	S29	Levin-Rector A. et al.	2015	Building-level analyses to prospectively detect influenza outbreaks in long-term care facilities: New York City, 2013-2014	No	Relevance of title / abstract
Scopus	S30	Lindgren A. et al.	2010	Adult asthma and traffic exposure at residential address, workplace address, and self-reported daily time outdoor in traffic: A two-stage case-control study	No	Relevance of title / abstract
Scopus	S31	Mukhopadhyay A. et al.	2023	Neighborhood-Level Socioeconomic Status and Prescription Fill Patterns among Patients with Heart Failure	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S32	Wu C.Y.H. et al.	2019	Influence of the Spatial Resolution of the Exposure Estimate in Determining the Association between Heat Waves and Adverse Health Outcomes	No	Relevance of title / abstract
Scopus	S33	Vutien P. et al.	2019	Utilization of Census Tract-Based Neighborhood Poverty Rates to Predict Non-adherence to Screening Colonoscopy	No	Relevance of title / abstract
Scopus	S34	Jacquemin B. et al.	2013	Impact of geocoding methods on associations between long-term exposure to urban air pollution and lung function	No	Relevance of title / abstract
Scopus	S35	Gilboa S.M. et al.	2006	Comparison of residential geocoding methods in population-based study of air quality and birth defects	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S36	Kroneberg C. et al.	2022	Using police data to measure criminogenic exposure in residential and school contexts: experiences from a data linkage project in Germany	No	Relevance of title / abstract
Scopus	S37	Hyder A. et al.	2021	Opioid Treatment Deserts: Concept development and application in a US Midwestern urban county	No	Relevance of title / abstract
Scopus	S38	Brooks M.S. et al.	2021	Matching participant address with public records database in a US national longitudinal cohort study	Yes	
Scopus	S39	Wu J. et al.	2005	Improving spatial accuracy of roadway networks and geocoded addresses	Yes	
Scopus	S40	Beard K. et al.	2011	Creating residential and tenure histories from multi-year white pages	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S41	Jones R.R. et al.	2014	Farm residence and lymphohematopoietic cancers in the Iowa Women[U+05F3]s Health Study	No	Relevance of title / abstract
Scopus	S42	Yan F. et al.	2021	Association between maternal exposure to gaseous pollutants and atrial septal defect in China: A nationwide population-based study	No	Relevance of title / abstract
Scopus	S43	Cohen S.S. et al.	2022	Validating the use of census data on education as a measure of socioeconomic status in an occupational cohort	No	Relevance of title / abstract
Scopus	S44	Kawai K. et al.	2020	Ultraviolet Radiation Exposure and the Risk of Herpes Zoster in Three Prospective Cohort Studies	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S45	Baldovin T. et al.	2015	Geocoding health data with geographic information systems: A pilot study in northeast Italy for developing a standardized data-acquiring format	No	Relevance of title / abstract
Scopus	S46	Drewnowski A. et al.	2016	Geographic disparities in Healthy Eating Index scores (HEI-2005 and 2010) by residential property values: Findings from Seattle Obesity Study (SOS)	No	Relevance of title / abstract
Scopus	S47	Jiang W. et al.	2018	What3Words Geocoding Extensions	No	Relevance of title / abstract
Scopus	S48	Yu H. et al.	2018	Coarse-to-fine accurate registration for airborne sar images using SAR-FAST and DSP-LATCH	No	Relevance of title / abstract
Scopus	S49	Ng J.Q. et al.	2006	Socioeconomic and rural differences for cataract surgery in Western Australia	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S50	India-Aldana S. et al.	2021	Neighborhood Walkability and Mortality in a	No	Relevance of title / abstract
Scopus	S51	Malaainine M.E.I. et al.	2020	Prospective Cohort of Women Conception of geocoding matching algorithm for casablanca city - Morocco	Yes	
Scopus	S52	Yao X. et al.	2015	A novel fuzzy Chinese address matching engine based on full-text search technology	Yes	
Scopus	S53	Küçük Matci D. et al.	2018	Address standardization using the natural language process for improving geocoding results	Yes	
Scopus	S54	Mendez D.D. et al.	2014	A methodology for combining multiple commercial data sources to improve measurement of the food and alcohol environment: Applications of geographical information systems	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S55	Willis M.D. et al.	2020	Assessing the effectiveness of vehicle emission regulations on improving perinatal health: A population-based accountability study	No	Relevance of title / abstract
Scopus	S56	Nesi P. et al.	2016	Geographical localization of web domains and organization addresses recognition by employing natural language processing, Pattern Matching and clustering	Yes	
Scopus	S57	Rose K.M. et al.	2004	Historical measures of social context in life course studies: Retrospective linkage of addresses to decennial censuses	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S58	Li G. et al.	2022	Associations of combined exposures to ambient temperature, air pollution, and green space with hypertension in rural areas of Anhui Province, China: A cross-sectional study	No	Relevance of title / abstract
Scopus	S59	Brugge D. et al.	2013	Highway proximity associated with cardiovascular disease risk: The influence of individual-level confounders and exposure misclassification	No	Relevance of title / abstract
Scopus	S60	Shusted C.S. et al.	2023	Characterizing Lung Cancer Burden Among Asian-American Communities in Philadelphia	No	Relevance of title / abstract
Scopus	S61	Nieto F.J. et al.	2010	The Survey of the Health of Wisconsin (SHOW), a novel infrastructure for population health research: Rationale and methods	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S62	Das S. et al.	2021	A Spatial Approach for Ending the Human Immunodeficiency Virus Epidemic for the United States - A DC Model	No	Relevance of title / abstract
Scopus	S63	Christen P.	2009	Geocode matching and privacy preservation	No	Relevance of title / abstract
Scopus	S64	Cserbik D. et al.	2020	Fine particulate matter exposure during childhood relates to hemispheric-specific differences in brain structure	No	Relevance of title / abstract
Scopus	S65	Chen X.-Q. et al.	2004	Applying DP to ETL of spatial data warehouse	Yes	
Scopus	S66	Kravets N. et al.	2007	The accuracy of address coding and the effects of coding errors	No	Relevance of title / abstract
Scopus	S67	Gao S. et al.	2017	Research of key technology in the construction of geocoding engine	Yes	

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S68	Ohnmacht T. et al.	2014	Route-recording on high resolution transportation network databases for national transport surveys: An option for valid and reliable distance measures?	No	Relevance of title / abstract
Scopus	S69	Izzi F. et al.	2013	Enhancing the spatial dimensions of open data: Geocoding open PA information using geo platform fusion to support planning process	No	Relevance of title / abstract
Scopus	S70	Fiscella K. et al.	2001	Impact of patient socioeconomic status on physician profiles: A comparison of census-derived and individual measures	No	Relevance of title / abstract
Scopus	S71	Molnár P. et al.	2015	Residential NOx exposure in a 35-year cohort study. Changes of exposure, and comparison with back extrapolation for historical exposure assessment	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S72	Strickland M.J. et al.	2007	Quantifying geocode location error using GIS methods	No	Relevance of title / abstract
Scopus	S73	Grubestic T.H. et al.	2011	Alcohol outlets and clusters of violence	No	Relevance of title / abstract
Scopus	S74	Lee K. et al.	2020	Improving a street-based geocoding algorithm using machine learning techniques	No	Duplicate of G12
Scopus	S75	Ludema C. et al.	2018	Comparing neighborhood and state contexts for women living with and without HIV: Understanding the southern HIV epidemic	No	Relevance of title / abstract
Scopus	S76	Bonner M.R. et al.	2003	Positional accuracy of geocoded addresses in epidemiologic research	No	Relevance of title / abstract
Scopus	S77	Ratcliffe J.H.	2002	Damned if you don't, damned if you do: Crime mapping and its implications in the real world	No	Relevance of title / abstract

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S78	Kuai X. et al.	2020	Spatial context-based local toponym extraction and Chinese textual address segmentation from urban POI data	No	Relevance of title / abstract
Scopus	S79	Byrwa-Hill B.M. et al.	2021	Impact of a pollution breach at a coke oven factory on asthma control in nearby vulnerable adults	No	Relevance of title / abstract
Scopus	S80	Pan K. et al.	2022	Remotely sensed measures of Hurricane Michael damage and adverse perinatal outcomes and access to prenatal care services in the Florida panhandle	No	Relevance of title / abstract
Scopus	S81	Comber S. et al.	2019	Machine learning innovations in address matching: A practical comparison of word2vec and CRFs	No	Duplicate of G5

Continued on next page

Table 8.1 Continued from previous page

Source	ID	Author	Year	Title	Included	Reason
Scopus	S82	Shah T.I. et al.	2014	Geocoding for public health research: Empirical comparison of two geocoding services applied to Canadian cities	No	Relevance of title / abstract
Scopus	S83	Gritta M. et al.	2020	A pragmatic guide to geoparsing evaluation: Toponyms, Named Entity Recognition and pragmatics	No	Relevance of title / abstract
Scopus	S84	Fry R.J. et al.	2017	Using Routinely Collected Administrative Data in Public Health Research: Geocoding Alcohol Outlet Data	No	Relevance of title / abstract
Scopus	S85	Shaposhnikov D. et al.	2019	Labelling for venue visit detection by matching Wi-Fi hotspots with businesses	No	Relevance of title / abstract
Scopus	S86	Kirielle N. et al.	2019	Outlier detection based accurate geocoding of historical addresses	No	Relevance of title / abstract

C Geocoding Services Code

C.1 Google

```
import googlemaps

api_key = os.environ["GOOGLE_API_KEY"]
gmaps = googlemaps.Client(key=api_key)

geocoded = [
    gmaps.geocode(address, components={"country": "NZ"})
    for address in addresses
]
```

C.2 Azure

```
from azure.core.credentials import AzureKeyCredential
from azure.maps.search import MapsSearchClient

credential = AzureKeyCredential(os.environ.get("AZURE_API_KEY"))

search_client = MapsSearchClient(
    credential=credential,
)

geocoded = [
    search_client.get_geocoding(
        address_line=address,
        country_region="NZ",
        top=1
    ) for address in addresses
]
```

C.3 AWS

```
import boto3

client = boto3.client(
    "geo-places",
    region_name="us-east-1",
    aws_access_key_id="",
    aws_secret_access_key=""
)

def gcode(address):
    response = client.geocode(
        QueryText=address,
        MaxResults=1,
        Key=os.environ["AWS_API_KEY"],
        IntendedUse="SingleUse",
        Filter={"IncludeCountries": ["NZ"]},
    )
    return response

geocoded = [gcode(address) for address in addresses]
```

C.4 Mapbox

```
import mapbox

mapbox_token = os.environ.get("MAPBOX_TOKEN")
geocoder = mapbox.Geocoder(access_token=mapbox_token)

def mapbox_geocode(address):
    response = geocoder.forward(
        address,
        country=["nz"],
        limit=1).json()

    return response

geocoded = [mapbox_geocode(address) for address in addresses]
```

C.5 Nominatim

```
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="nz_geocoder")

geocoded = [
    geolocator.geocode(address, country_codes="NZ")
    for address in addresses
]
```


D Address Modifying Prompt

As part of an evaluation process for geocoding algorithms, I am testing how well addresses can be matched to a reference database. To make a complete evaluation, addresses with a variety of quality and matching difficulty are needed. I will provide you with a list of 50 addresses. For each one of these input addresses, I need you to make an address with realistic adjustments, and an address with aggressive adjustments. The adjustments made should represent how a person may choose to write down their address and may include the following:

- Typos, including deletions, additions, substitutions, and transpositions
- Phonetic mistakes
- Abbreviations of areas and common words, e.g. street to st and auckland to akl
- Omittal of various fields within the address, e.g. including the city and not the suburb, removing the postcode, etc.
- Any other modifications that are reasonable in the context of addresses

the realistically modified addresses should make a small amount of adjustments and represent addresses that a human would realistically use, and the aggressively modified addresses should apply more adjustments beyond what a human might typically use. Provide only the modified addresses in your response, use one line per address pair and separate realistic and aggressively modified addresses with a |.

Bibliography

- [1] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitingner. Research-paper recommender systems: A literature survey. *International Journal on Digital Libraries*, pages 1–34, 07 2015.
- [2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.
- [3] Mélanie Bertin, Jean-François Viel, Christine Monfort, Sylvaine Cordier, and Cécile Chevrier. Socioeconomic disparities in adverse birth outcomes in urban and rural contexts: A french mother-child cohort. *Paediatric and Perinatal Epidemiology*, 29(5):426 – 435, 2015.
- [4] Abraham Bookstein, Vladimir Kulyukin, and Timo Raita. Generalized hamming distance. *Information Retrieval*, 5, 10 2002.
- [5] Marquita S. Brooks, Aleena Bennett, Gina S. Lovasi, Philip M. Hurvitz, Natalie Colabianchi, Virginia J. Howard, Jennifer Manly, and Suzanne E. Judd. Matching participant address with public records database in a us national longitudinal cohort study. *SSM - Population Health*, 15, 2021.
- [6] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(36):1109–1135, 2010.
- [7] Davide Chicco. *Siamese Neural Networks: An Overview*, volume 2190, pages 73–94. Springer, 08 2020.

-
- [8] Kyunghyun Cho, Bart Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 06 2014.
- [9] Peter Christen, Tim Churches, Alan Willmore, et al. A probabilistic geocoding system based on a national address file. In *Proceedings of the 3rd Australasian Data Mining Conference, Cairns*. Citeseer, 2004.
- [10] Sam Comber. Demonstrating the utility of machine learning innovations in address matching to spatial socio-economic applications. *Region*, 7(3):17–37, 2020.
- [11] Sam Comber and Daniel Arribas-Bel. Machine learning innovations in address matching: A practical comparison of word2vec and crfs. *Transactions in GIS*, 23(2):334–348, 2019.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [13] André V. Duarte and Arlindo L. Oliveira. Improving address matching using siamese transformer networks. In Nuno Moniz, Zita Vale, José Cascalho, Catarina Silva, and Raquel Sebastião, editors, *Progress in Artificial Intelligence*, pages 413–425, Cham, 2023. Springer Nature Switzerland.
- [14] Samuel Dupond. A thorough review on the current advance of neural network structures. *Annual Reviews in Control*, 14(14):200–230, 2019.
- [15] Python Software Foundation. Python package index (pypi). <https://pypi.org/>, 2024.
- [16] Shan Gao, Si Chen, Minjun Peng, and Chenling Pan. Research of key technology in the construction of geocoding engine. In *International Conference on Geoinformatics*, 2017.
- [17] Benyamin Ghojogh, Milad Sikaroudi, Sobhan Shafiei, Hamid Tizhoosh, Fakhri Karray, and Mark Crowley. Fisher discriminant triplet and contrastive losses

- for training siamese networks. In *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 07 2020.
- [18] Google. Google Maps Usage and Billing. Online, 2025. Accessed: 2025-01-05.
- [19] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006.
- [20] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.
- [21] Heikki Hyyrö, Yoan Pinzon, and Ayumi Shinohara. New bit-parallel indel-distance algorithm. In Sotiris E. Nikolettseas, editor, *Experimental and Efficient Algorithms*, pages 380–390, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [22] International Computer Science Institute and S.M. Omohundro. *Five Balltree Construction Algorithms*. Technical report (International Computer Science Institute). International Computer Science Institute, 1989.
- [23] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.
- [24] Land Information New Zealand. NZ Street Address, Sep 2022.
- [25] Land Information New Zealand. NZ Address, Feb 2025.
- [26] Kangjae Lee, Alexis Richard C. Claridades, and Jiyeong Lee. Improving a street-based geocoding algorithm using machine learning techniques. *Applied Sciences (Switzerland)*, 10(16), 2020.
- [27] Jianjun Lei, Chen Wu, and Ying Wang. Clgiam: contrastive learning model based on global and local semantic interaction for address matching. *Applied Intelligence*, 53(23):29267 – 29281, 2023.
- [28] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, February 1966.
- [29] Dengyue Li, Shengrui Wang, and Zhen Mei. Approximate address matching. In *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 264–269, 2010.

- [30] Yue Lin, Mengjun Kang, Yuyang Wu, Qingyun Du, and Tao Liu. A deep learning architecture for semantic address matching. *International Journal of Geographical Information Science*, 34(3):559–576, 2020.
- [31] Yue Lin, Mengjun Kang, Yuyang Wu, Qingyun Du, and Tao Liu. A deep learning architecture for semantic address matching. *International Journal of Geographical Information Science*, 34(3):559 – 576, 2020.
- [32] Cheng Liu, Xinyang He, Shiliang Su, and Mengjun Kang. A graph-based method for chinese address matching. *Transactions in GIS*, 2023.
- [33] Mapbox. Mapbox Geocoding API. Online, 2025. Accessed: 2025-02-05.
- [34] Mapbox. Mapbox Pricing. Online, 2025. Accessed: 2025-02-05.
- [35] maxbachmann. RapidFuzz Rapid fuzzy string matching in Python and C++ using the Levenshtein Distance.
- [36] Microsoft. Azure Maps Pricing. Online, 2025. Accessed: 2025-02-05.
- [37] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.
- [38] openvenues. Libpostal: international street address NLP.
- [39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [40] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [41] New Zealand Post. Addressing standards. <https://www.nzpost.co.nz/business/shipping-in-nz/addressing-standards>, 2025.
- [42] New Zealand Post. How to address mail. <https://www.nzpost.co.nz/personal/sending-in-nz/how-to-address-mail>, 2025.
- [43] Keshav Ramani and Daniel Borrajo. Methods for matching english language addresses. *Transactions in GIS*, 27(2):347–363, 2023.

- [44] Bas Ranzijn. A geocoding algorithm based on a comparative study of address matching techniques, October 2013. Master’s thesis, Supervisors: Dr. Viorel Milea, Dr. Flavius Frasincar, Stijn Duijzer MSc., Dr. Johan M. M. van Rooij, Dr. Ir. Jacob Jan Paulus CPIM.
- [45] Sina Rashidian, Xinyu Dong, Amogh Avadhani, Prachi Poddar, and Fusheng Wang. Effective scalable and integrative geocoding for massive address datasets. In *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, volume 2017-November, 2017.
- [46] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.
- [47] Amazon Web Services. Amazon Location Service Endpoints and Quotas. Online, 2025. Accessed: 2025-02-05.
- [48] Amazon Web Services. AWS Location Service Pricing. Online, 2025. Accessed: 2025-02-05.
- [49] Amazon Web Services. AWS Service Terms. Online, 2025. Accessed: 2025-02-05.
- [50] Amit Singhal and I. Google. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24, 01 2001.
- [51] Karen Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*, page 132–142. Taylor Graham Publishing, GBR, 1988.
- [52] Auckland Transport. Abbreviating cycle sign content. <https://at.govt.nz/media/1979161/abbreviatingcyclesigncontentapx1.pdf>, 2025.
- [53] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
- [54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [55] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern,

- Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [56] William Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record. In *Proceedings of the Section on Survey Research Methods*, pages 354–359. American Statistical Association, 1990.
- [57] Liuchang Xu, Ruichen Mao, Chengkun Zhang, Yuanyuan Wang, Xinyu Zheng, Xingyu Xue, and Fang Xia. Deep transfer learning model for semantic address matching. *Applied Sciences*, 12(19):10110, 2022.
- [58] Xiaojing Yao, Xiang Li, Ling Peng, and Tianhe Chi. A novel fuzzy chinese address matching engine based on full-text search technology. In *Proceedings of Science*, volume 12-13-September-2015, 2015.
- [59] Darren Yates, Md Zahidul Islam, Yanchang Zhao, Richi Nayak, Vladimir Estivill-Castro, and Salil Kanhere. Postmatch: A framework for efficient address matching. In *Data Mining: 19th Australasian Conference on Data Mining, AusDM 2021, Brisbane, QLD, Australia, December 14-15, 2021, Proceedings 19*, pages 136–151. Springer, 2021.
- [60] Chen Zhang, Renzhong Guo, Xiangyuan Ma, Xi Kuai, and Biao He. W-textcnn: A textcnn model with weighted word embeddings for chinese address pattern classification. *Computers, Environment and Urban Systems*, 95, 2022.